T-79.179 Rinnakkaiset ja hajautetut digitaaliset järjestelmät

Saavutettavuusanalyysi

Marko Mäkelä

3. maaliskuuta 2003

Saavutettavuusanalyysi

Saavutettavuusanalyysillä tarkoitetaan sitä, että

- tarkastetaan, onko formaalisti kuvatun järjestelmän mahdollista saavuttaa
  - annettujen turvallisuusehtojen vastainen "paha" tila tai
  - annettujen elävyysehtojen vastainen tapahtumasilmukka
- ja näytetään jokin ehtojen vastainen suoritus, jos sellainen on olemassa.

Yleensä saavutettavuusanalyysi edellyttää mallin kaikkien saavutettavissa olevien tilojen tutkimista. Joskus voidaan hyödyntää annettuja ehtoja ja jättää osa tiloista tutkimatta.

Perusratkaisu: saavutettavuusgraafin laskeminen

SAAVUTETTAVUUSGRAAFI($\langle S,T,F,W,M_0\rangle$)

```
1  G : ⟨V,E,v0⟩ ← ⟨{M0},∅,M0⟩;
2  Markings : stack ← ⟨M0⟩;
3  while Markings ≠ ⟨⟩
4  do M ← Markings.pop();
5     for t ∈ enabled(M)
6     do M' ← fire(M,t);
7        if M' ∉ V
8          then V ← V ∪ {M'}
9               Markings.push(M');
10       E ← E ∪ {⟨M,t,M'⟩};
11 return G;
```

Algoritmi käyttää kahta funktiota:

- enabled$(M) := \{t \mid M[t\rangle\}$
- fire$(M,t) := [M\rangle t$

Vaihtamalla pino jonoksi syvyyshaku muuttuu leveyshauksi. Leveyshaku löytää lyhimmän siirtymäpolun alkutilasta virheelliseen (turvattomaan) tilaan. Elävyyden tarkastaminen vaatii syvyyshakua, sillä vastaesimerkit päättyvät silmukoihin, ja silmukat on tehokkainta löytää syvyyshaulla.

Saavutettavuusanalyysin toteuttaminen

Äskeisen algoritmin pahimmat pullonkaulat ovat:

- rivien 5–10 silmukka
- rivit 5 ja 6: seuraajatilojen joukon $\bigcup_{t\in\text{enabled}(M)}\{\text{fire}(M,t)\}$ muodostaminen
- rivit 7 ja 8: tilan etsiminen ja lisääminen käsiteltyjen joukkoon

Rivi 10 on melko harmiton, sillä kaaret voidaan kirjoittaa peräkkäistiedostoon, ellei ole tarvetta tallentaa myös "taaksepäin" vieviä kaaria kohdetilan mukaan järjestettynä. Kaikki kaaret voidaan jättää poiskin ja laskea uudelleen vastaesimerkkiä muodostettaessa.

Hakupino tai -jono $Markings$ voi sisältää joko tiloja tai niiden numeroita, jolloin muistia säästyy mutta rivillä 4 on haettava tila M jostakin "tietokannasta" $DB : \mathbb{N} \to V$.

Tilajoukon V esitystapoja

Joukko V voidaan esittää kahdella täysin erilaisella tavalla. Joukon alkiot $M \in V$ on edullisinta esittää joukon $\bigcup_{t \in \mathsf{enabled}(M)} \{\mathsf{fire}(M,t)\}$ laskennan aikana "avatussa muodossa" siten, että mallinnuskielen lausekkeita on helppo laskea.

Löydettyjen tilojen joukon V on mahdollistettava seuraavat toiminnot:

- $M \in V$: onko tila M jo löydetty?
- $V \leftarrow V \cup \{M\}$: lisää tila M löydettyjen joukkoon (ja anna sille tilanumero n_M)
- (Hae tila numeron perusteella: $DB(n_M) \mapsto M$)

Suluissa olevia toimintoja tarvitaan vain, jos hakupinoon tai -jonoon tallennetaan tilanumeroita. Jos V on esitetty tällaisena dynaamisena taulukkona DB, johon kukin tila tallennetaan erikseen, tilat kannattaa *tiivistää* bittijonoiksi. Jos esimerkiksi tiedetään, että järjestelmän $n = 5$ asiakasta voivat olla $m = 3$ eri tilassa, parien $\langle n, m \rangle$ esittämiseen ei tarvita $32 + 32$ bittiä eikä edes $8 + 8$ bittiä, vaan $\log_2 \lceil nm \rceil = \log_2 \lceil 15 \rceil = 4$ bittiä.

Tilajoukon symboliset esitystavat (1/2)

Tilajoukko V voidaan myös esittää yhtenä "klönttinä", johon voidaan vain lisätä tiloja ja jolta voidaan kysyä, onko annettu tila jo lisätty joukkoon.

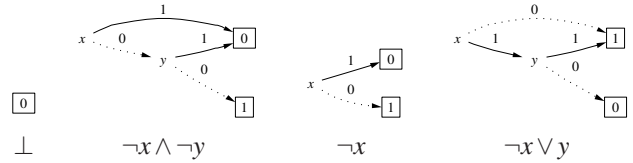
Eräs tapa toteuttaa tällainen rakenne on kasata suunnattua graafia, jonka esittämä tilakone tunnistaa sellaista kieltä, jonka sanoja ovat täsmälleen ne tilat, jotka on jo lisätty.

Aina, kun joukkoon lisätään tila, graafia esittävää osoitinviidakkoa muutetaan sieltä täältä. Tällainen *symbolinen* tietorakenne on käytännössä pakko toteuttaa keskusmuistissa.

Symboliset tietorakenteet ovat erittäin läheisesti sidoksissa kielen operaatioihin ja tietotyyppeihin. Tallennettaessa jokainen tila erikseen ei tarvitse kiinnittää operaatioihin mitään huomiota; riittää, että kukin yksittäinen tila pystytään esittämään bittijonona.

Tilajoukon symboliset esitystavat (2/2)

Tunnetuin symbolinen tietorakenne lienee BDD *(Binary Decision Diagram)* , joka soveltuu Boolen piirien tilojen tallentamiseen. BDD:t ovat suunnattuja silmukattomia graafeja (laskostettuja puita), joissa on enintään kaksi lehtisolmua, 0 ja 1, sekä kutakin muuttujaa vastaavat sisäsolmut. Muuttujasolmuista lähtee kaksi kaarta, joista toinen esittää muuttujan arvoa 0 ja toinen arvoa 1. BDD esittää sellaista lauseketta, joka pätee täsmälleen siihen jo lisätyille muuttujasidonnoille. Esimerkki: tallennetaan tyhjään tilajoukkoon tilat $\langle x \mapsto \bot, y \mapsto \bot \rangle$, $\langle x \mapsto \bot, y \mapsto \top \rangle$ ja $\langle x \mapsto \top, y \mapsto \top \rangle$ tässä järjestyksessä:



Tilan lisääminen voi sekä pienentää että suurentaa joukon esitystä. Tämänkaltaiset menetelmät ovat hyvin herkkiä siitä, missä järjestyksessä muuttujat esitetään ja tilat lisätään.

Tilajoukon kutistaminen

Perinteiset tilajoukon esitystavat ovat usein ainoa järkevä vaihtoehto, jos tutkittavassa järjestelmässä on runsaasti korkean tason operaatioita. Ongelmaksi muodostuu silloin tilojen ja siirtymien suuri määrä. Mitä, jos säästetään tilaa käyttämällä enemmän aikaa? Järjestelmän kaikkea mahdollista käyttäytymistä ei ole välttämättä pakko tutkia, jos tutkittavaa ominaisuutta voidaan hyödyntää. Tärkeimmät kutistusmenetelmät ovat:

- tilojen samastaminen:
  - symmetrioiden tunnistaminen ja tilojen kuvaaminen toisikseen permutoinneilla
  - kuolleiden muuttujien tunnistaminen ja alustaminen oletusarvoihin
  - viipalointi (tarkasteltavaan ominaisuuteen vaikuttamattomien muuttujien ja siirtymien poistaminen mallista ennen saavutettavuusanalyysin aloittamista)
- osittaisjärjestysmenetelmät: haarautumisen vähentäminen

Symmetriat (1/2)

Paikka–siirtymä-verkon $\langle S,T,F,W\rangle$ *symmetria* $\sigma : (S \to S) \cup (T \to T)$ on sellainen *automorfismi* (verkon isomorfismi itsensä kanssa), joka kunnoittaa kaaripainoja:

$$W(a,b) = W(\sigma(a),\sigma(b)).$$

Verkon symmetria σ voidaan laajentaa merkintöjen symmetriaksi $\sigma : (S \to \mathbb{N}) \to (S \to \mathbb{N})$ määrittelemällä merkinnöille $M : S \to \mathbb{N}$

$$(\sigma(M))(\sigma(s)) = M(s).$$

Tästä seuraa, että $M' = [M\rangle t$ jos ja vain jos $\sigma(M') = [\sigma(M)\rangle\sigma(t)$.

Marko Mäkelä

Symmetriat (2/2)

Symmetrioiden joukko Σ on *symmetriaryhmä*, jos ja vain jos se on refleksiivinen, symmetrinen ja transitiivinen:

1. $id \in \Sigma$ siten, että $id(M) = M$,
2. jos $\sigma \in \Sigma$, niin $\sigma^{-1} \in \Sigma$, ja
3. jos $\sigma \in \Sigma$ ja $\sigma' \in \Sigma$, niin $(\sigma' \circ \sigma) \in \Sigma$.

Symmetriaryhmä määrittelee merkinnöille (ja paikoille ja siirtymille) ekvivalenssirelaation:

$$M \equiv M' :\Leftrightarrow (\exists \sigma \in \Sigma : M = \sigma(M')).$$

Merkintää M kutsutaan *symmetriseksi*, jos ja vain jos

$$\forall \sigma \in \Sigma : M = \sigma(M).$$

Marko Mäkelä

Symmetriakutistusmenetelmä

SYMMETRINEN($\langle S,T,F,W,M_0,\Sigma\rangle$)

```
1  G : ⟨V,E,v0⟩ ← ⟨{M0},∅,M0⟩;
2  Markings : stack ← ⟨M0⟩;
3  while Markings ≠ ⟨⟩
4  do M ← Markings.pop();
5     for t ∈ enabled(M)
6     do M' ← fire(M,t);
7        if ∃M'' ∈ V : ∃σ ∈ Σ : M'' = σ(M')
8          then E ← E ∪ {⟨M,t,M''⟩}
9          else if M' ∉ V
10                then V ← V ∪ {M'}
11                     Markings.push(M')
12               E ← E ∪ {⟨M,t,M'⟩}
13 return G;
```

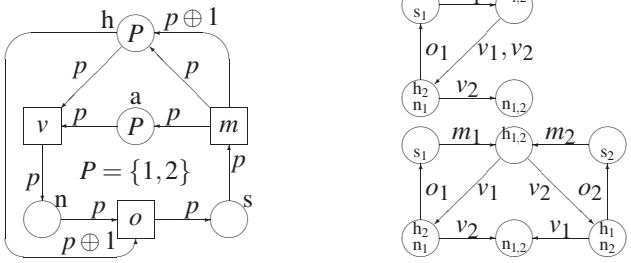
Jos alkumerkintä M_0 on symmetrinen, saavutettavat merkinnät ja lukkiumat säilyvät. Siirtymien elävyys ei aina säily, kuten ei kaikkien LTL-kaavojen totuusarvokaan.

Pahimmat pullonkaulat ovat symmetriaryhmien löytäminen ja symmetristen tilojen hakeminen.

Menetelmä voidaan yleistää muihinkin kuvauskieliin, kuten korkean tason verkkoihin. Aikaa voi säästyäkin, sillä kokonaisia tila-avaruuden alipuita voi jäädä käsittelemättä.

Marko Mäkelä

Symmetriakutistusmenetelmä: esimerkki



Lounastavien filosofien järjestelmän saavutettavuusgraafi on selvästi symmetrinen. Eräs symmetria vaihtaa filosofit ja haarukat keskenään:

$$\sigma = \{h_1 \mapsto h_2, h_2 \mapsto h_1, a_1 \mapsto a_2, a_2 \mapsto a_1, n_1 \mapsto n_2, n_2 \mapsto n_1, s_1 \mapsto s_2, s_2 \mapsto s_1\}.$$

Itse asiassa tämä σ *generoi* symmetriaryhmän: esimerkiksi $id = \sigma \circ \sigma$ ja $\sigma^{-1} = \sigma$.

Marko Mäkelä

Eräs osittaisjärjestysmenetelmä: itsepäiset joukot

Itsepäisten joukkojen menetelmä pyrkii siihen, että seuraajatiloja laskettaessa otetaan huomioon vain osa vireessä olevista siirtymistä, jolloin saavutettavuusgraafi haarautuu vähemmän ja sen koko pienenee. Menetelmä säilyttää saavutettavuusgraafin kaikki lukkiumat. Eräs sen muunnelma säilyttää myös kaikkien $\bigcirc$-konnektiivia sisältämättömien LTL-kaavojen totuusarvon.

Menetelmä perustuu siirtymäjoukon T jakamiseen kahtia osiin T_s ja $T \setminus T_s$ siten, ettei minkään joukkoon T_s kuuluvan siirtymän laukeaminen vaikuta joukon $T \setminus T_s$ virittyneisyyteen.

Olkoon oheisessa kommutoivassa diagrammissa valittu $t \in T_s$ ja siirtymäjono $\tau = (T \setminus T_s)^*$. Merkinnässä M''' pätevien ominaisuuksien kannalta on yhdentekevää, saavutaanko merkintään polkua $t\tau$ vai polkua τt pitkin.

$$\begin{array}{ccc} M & \xrightarrow{\tau} & M' \\ \downarrow t & & \downarrow t \\ M'' & \xrightarrow{\tau} & M''' \end{array}$$

Marko Mäkelä

Itsepäiset joukot (1/4)

Olkoon $\langle S, T, F, W \rangle$ paikka–siirtymä-järjestelmä ja M jokin sen merkintä. Tällöin $T_s \subseteq T$ on *itsepäinen* merkinnässä M, jos ja vain jos siinä on M-virittyneitä siirtymiä ja $\forall t \in T_s$:

1. Jos $M[t\rangle$, niin $({}^\bullet t)^\bullet \subseteq T_s$.

2. Jos $\neg M[t\rangle$, niin $\exists s \in {}^\bullet t : M(s) < W(s,t) \wedge {}^\bullet s \subseteq T_s$.

Kaikkien siirtymien joukko T on triviaalisti itsepäinen.

On edullisinta muodostaa mahdollisimman pieniä itsepäisiä joukkoja, sillä silloin saavutettavuusgraafi haarautuu vähiten.

Marko Mäkelä

Itsepäiset joukot (2/4)

Jokin merkinnässä M itsepäinen joukko T_s voidaan (tehottomasti) laskea seuraavasti:

1. Valitaan M-virittynyt siirtymä $t \in T$: $M[t\rangle$. Olkoon $T_s = \{t\}$.

2. Valitaan siirtymä $t' \in T_s$ (jota ei ole vielä tutkittu):

   (a) $M[t'\rangle$: Määritelmä toteutuu, kun asetetaan $T_s \leftarrow T_s \cup ({}^\bullet t')^\bullet$. Palataan kohtaan 2.

   (b) $\neg M[t'\rangle$: Määritelmän mukaan etsitään sellainen paikka $s \in {}^\bullet t'$, joka estää t':n laukeamisen, ja asetetaan $T_s = T_s \cup {}^\bullet s$. Palataan kohtaan 2.

Algoritmi päättyy, kun joukko T_s ei enää muutu iteraatiokierroksen aikana.

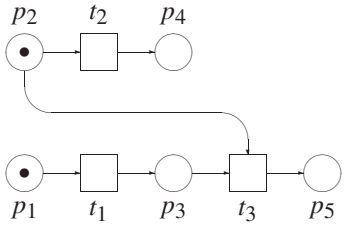
Marko Mäkelä

Itsepäiset joukot (3/4)

ITSEPÄINEN($\langle S, T, F, W, M_0 \rangle$)

```
1  G : ⟨V, E, v0⟩ ← ⟨{M0}, ∅, M0⟩ ;
2  Markings : stack ← ⟨M0⟩ ;
3  while Markings ≠ ⟨⟩
4  do M ← Markings.pop();
5     for t ∈ stubborn(M)
6     do M' ← fire(M, t);
7        if M' ∉ V
8          then V ← V ∪ {M'}
9               Markings.push(M');
10       E ← E ∪ {⟨M, t, M'⟩};
11 return G;
```

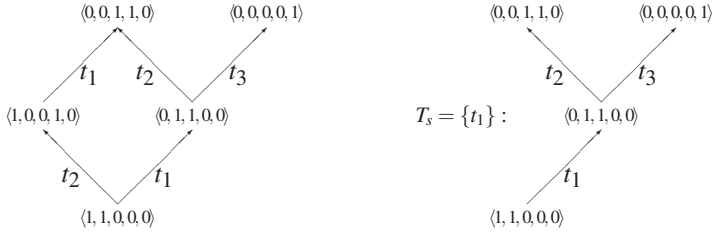
Tarkastellaan seuraavaa järjestelmää:



Jos valitaan aluksi $T_s \supseteq \{t_2\}$, joukkoon on lisättävä $({}^\bullet t_2)^\bullet = \{t_3\}$ ja ${}^\bullet p_3 = \{t_1\}$, koska $p_3 = {}^\bullet t_3$. Saadaan $T_s = T$.
Jos valitaan $T_s = \{t_1\}$, joukko on sellaisenaan itsepäinen.

Marko Mäkelä

Itsepäiset joukot (4/4)



Itsepäisen joukon suuruus riippuu suuresti siitä, mikä vireessä oleva siirtymä valitaan ensin, sekä siitä, miten siirtymien laukeamisen estävät paikat s valitaan.

Pienin mahdollinen itsepäinen joukko saadaan käyttämällä *poistoalgoritmia*. Tällöin aloitetaan joukosta $T_s = T$. Siitä poistetaan jokin siirtymä, jonka jälkeen poistetaan siirtymiä, kunnes joukko on taas itsepäinen. Edellä esitettyä *lisäysalgoritmia* voidaan parantaa hyödyntämällä Tarjanin vahvasti kytkettyjen komponenttien algoritmia.

Marko Mäkelä

Kuvausten väliset käännökset

Saavutettavuusanalyysi näyttää helpolta toteuttaa mille tahansa laskentajärjestelmälle, mutta käytännössä monia järjestelmiä ei ole suunniteltu niin, että niiden tila voitaisiin tallentaa tehokkaasti ja palauttaa myöhemmin. On parasta olla keksimättä pyörää uudelleen ja käyttää saavutettavuusanalyysiin suunniteltuja työkaluja.

Jos tarkasteltavat järjestelmät on jo kuvattu jollakin tietokoneen tulkitsemalla kielellä ja jos tarkasteltavia järjestelmäkuvauksia on paljon, ei ole mielekästä kääntää kuvauksia käsin. Se, kuinka työlästä automaattisen kääntäjän tekeminen on, riippuu lähdekielen laajuudesta. Usein riittää tarkastella lähdekielen pientä osaa.

Käännöstä tehtäessä kuvauksia on myös syytä yksinkertaistaa ja jättää niistä kaikki tarkasteltavien ominaisuuksien kannalta turha pois. Joitain rakenteita voidaan ilmaista Petriverkoilla lyhemmin kuin ohjelmointikielellä. Käännöksen laadulla on erittäin suuri vaikutus järjestelmän saavutettavuusgraafin kokoon.

Marko Mäkelä

Vastaesimerkkien esittäminen (1/2)

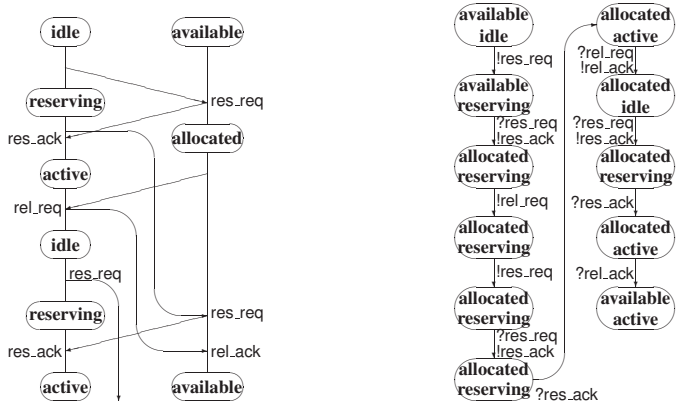
Kun työkalu havaitsee järjestelmän toimivan vaatimusten vastaisesti, sen on jotenkin ilmoitettava asiasta. Käyttäjän kannalta olisi sangen turhauttavaa saada vain tietää, että jossakin päin järjestelmää on virhe. Paljon hyödyllisempää on esittää *vastaesimerkki* eli jokin vaaditun ominaisuuden vastainen suoritus.

Suoritus voidaan esittää tila–siirtymä-ketjuna, jossa tiloihin on merkitty muuttujien arvot ja siirtymiin tapahtumien nimet. Jos saavutettavuusanalysaattorin syöte on käännetty muusta kuvauksesta, nimien ja merkintätapojen on parasta seurata alkuperäistä kuvauskieltä. Toisin sanottuna vastaesimerkitkin on käännettävä!

Suoritusten esittäminen tila–siirtymä-ketjuina ei aina ole luontevaa. Joskus on mukavampaa tarkastella suoritusta debuggeria muistuttavassa työkalussa, joka esittää suorituksen animaationa korostaen yhtä koodiriviä tai rakennetta kerrallaan. Sanomakaaviot sopivat sanomanvälitykseen perustuvien suoritusten esittämiseen.

Marko Mäkelä

Vastaesimerkkien esittäminen (2/2)



Sanomakaaviosta tilanne hahmottuu siirtymäketjua paremmin. Sanomakaaviot ovat korkeamman tason kieltä, sillä yksi sanomakaavio vastaa useaa erilaista siirtymäketjua.

Marko Mäkelä