

Suunnittelu stabiiliusongelmana ja stabiilien mallien ominaisuuksia

Käsiteltäviä asioita:

1. Suunnitteluongelma (kertaus)
2. Suunnitteluongelmasta logiikkaohjelmaksi
3. Muuta huomioitavaa
4. Stabiilien mallien laskennallinen vaativuus
5. Suhde lauselogiikan toteutuvuusongelmaan
6. Stabiilien mallien yksikäsitteisyydestä

Esimerkki (kertaus)

- Tyyppi BLOCK : nimivakiot $\{a, b, c, d\}$.
- Operaattori $\text{MOVE}(x, y, z)$:
 1. muuttujien x, y ja z tyyppinä BLOCK,
 2. $\text{Pre}(\text{MOVE}(x, y, z)) = \{\text{clear}(x), \text{on}(x, y), \text{clear}(z)\}$ ja
 3. $\text{Post}(\text{MOVE}(x, y, z)) = \{\text{on}(x, z), \text{clear}(x), \text{clear}(y)\}$.
- Alkutila $S_0 = \{\text{on}(a, b), \text{on}(b, c), \text{clear}(a), \text{clear}(d)\}$.
- Tavoitteen määrittelee lopetusehtojen joukko $S_1 = \{\text{on}(b, a)\}$.

1. Suunnitteluongelma (kertaus)

Suunnitteluongelma on nelikkö $\langle \mathcal{T}, \mathcal{O}, S_0, S_1 \rangle$, missä

- Jokaisen äärellisen tyyppin $T \in \mathcal{T}$ alkiot on nimetty yksikäsitteisesti äärellisellä joukolla $\{a, b, c, \dots\}$ nimivakioita.
- Jokainen operaattori $O(x, y, z, \dots) \in \mathcal{O}$ rakentuu
 1. muuttujiensa tyyppimäärittelyistä $x: T_1, y: T_2, z: T_3, \dots$, ja
 2. esiehtojen ja jälkiehtojen joukoista $\text{Pre}(O)$ ja $\text{Post}(O)$, jotka ovat tyyppitetyistä predikaattisymboleista P, Q, \dots , em. tyyppitetyistä muuttujista x, y, z, \dots ja tyyppien nimivakioista a, b, c, \dots rakentuvien atomikaavojen joukkoja.
- Alkutilanne S_0 ja lopputilanne S_1 ovat (muuttujattomia) atomilauseiden joukkoja.

2. Suunnitteluongelmasta logiikkaohjelmaksi

- Suunnitteluongelman dynaaminen luonne on aiheuttaa vaikeuksia käytettäessä klassista logiikkaa ongelman esittämiseen (kehysongelma).
- Stabiileissa malleissa minimaalisuus ja hyvin perustuvuus sisäänrakennettuna, jolloin suunnitteluongelman kuvaaminen yksinkertaistuu.
- Suunnitteluongelman esittäminen epämonotonisilla säännöillä ja stabiileilla malleilla on antanut laskennallisesti lupaavia tuloksia.

Käännöksen tavoite

- Annetaan rajoitetulle suunnitteluongelmalle $\langle \mathcal{T}, O, S_0, S_1 \rangle$ käännös logiikkaohjelmaksi $PLP(n)$ siten, että
 - ongelmalla $\langle \mathcal{T}, O, S_0, S_1 \rangle$ on enintään n :n mittainen ratkaisu
 - \iff ohjelmalla $PLP(n)$ on stabiili malli.
- Jokainen predikaatti ja operaattori varustetaan ylimääräisellä aika-argumentilla t : $on(x, y, t)$, $MOVE(x, y, z, t)$.
- Predikaateilla aika-argumentit t ovat väliltä $0, \dots, n$ ja operaattoreilla väliltä $0, \dots, n-1$.

Esimerkki: palikkamaailman esitys

1. Alkutila: $on(a, b, 0), on(b, c, 0), clear(a, 0), clear(d, 0)$
2. Lopputila ehtyehdoina: $F \leftarrow not\ on(b, a, n) \wedge not\ F$
3. Operaattorien soveltaminen:

$$MOVE(x, y, z, t) \leftarrow clear(x, t) \wedge on(x, y, t) \wedge clear(z, t) \wedge$$

$$not\ BLOCK_MOVE(x, y, z, t)$$

$$on(x, z, t+1) \leftarrow MOVE(x, y, z, t)$$

$$clear(y, t+1) \leftarrow MOVE(x, y, z, t)$$

Huomio. Viimeisen kaltaista sääntöä ei tarvita predikaatille $clear(x, t+1)$, koska $clear(x, t)$ on operaattorin $MOVE(x, y, z, t)$ esiehtona, ja kehysaksiomat varmistavat, että $clear(x, t+1)$ on tosi.

Käännöksen rakenne

Suunnitteluongelman esitys jakautuu seuraaviin osiin:

1. Alkutilan määrittely: $\{P(\vec{c}, 0) \mid P(\vec{c}) \in S_0\}$.
2. Lopputilan määrittely: $\{F \leftarrow not\ P(\vec{c}, n) \wedge not\ F \mid P(\vec{c}) \in S_1\}$.
3. *Operaattorien soveltaminen*: jos operaattorin $O\sigma$ esiehdot ovat voimassa hetkellä t eikä sille ole poikkeusta, operaattoria $O\sigma$ sovelletaan ajanhetkellä t ja lisättävät atomilauseet $(Post(O)\sigma - Pre(O)\sigma)$ tulevat voimaan hetkellä $t+1$.
4. *Kehysaksiomat*: jos atomilause pätee hetkellä t eikä sitä poistavaa operaattoria sovelleta hetkellä t , se pätee myös hetkellä $t+1$.
5. Operaattorien poikkeukset ratkaisevat, saadaanko lineaarisia suunnitelmia vaiko osittaisjärjestysuunnitelmia.

4. Kehysaksiomat:

- (i) Jokaiselle atomiselle lauseelle:

$$on(x, y, t+1) \leftarrow on(x, y, t) \wedge not\ remove_on(x, y, t)$$

- (ii) Atomisen lauseen poistaville operaattoreille:

$$remove_on(x, y, t) \leftarrow MOVE(x, y, z, t)$$

5. Suunnitelmien laadun määrittäminen:

- (i) *Lineaariset suunnitelmat*: sallitaan ainoastaan yhden operaattori-instanssin suorittaminen kerrallaan.

$$BLOCK_MOVE(x, y, z, t) \leftarrow MOVE(x', y', z', t) \wedge not\ x = x'$$

$$BLOCK_MOVE(x, y, z, t) \leftarrow MOVE(x', y', z', t) \wedge not\ y = y'$$

$$BLOCK_MOVE(x, y, z, t) \leftarrow MOVE(x', y', z', t) \wedge not\ z = z'$$

$$BLOCK_MOVE(x, y, z, t) \leftarrow O(\vec{x}, t) \dots$$

5. Suunnitelmien laadun määrittäminen (jatkoa):

- (ii) *Osittaisjärjestysuunnitelmat*: operaattori-instanssien samanaikainen suorittaminen kielletään ainoastaan konfliktitapauksissa.

$$\text{BLOCK_MOVE}(x,y,z,t) \leftarrow \text{MOVE}(x,y',z',t) \wedge \text{not } z = z'$$

$$\text{BLOCK_MOVE}(x,y,z,t) \leftarrow \text{MOVE}(x',y',z,t) \wedge \text{not } x = x'$$

6. Operaattorin NOOP toteuttaminen poikkeuksilla:

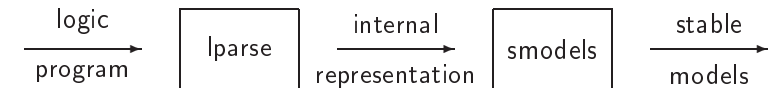
- (i) Operaattorin oma poikkeus (eli *valinta* jättää suorittamatta):

$$\text{BLOCK_MOVE}(x,y,z,t) \leftarrow \text{not } \text{MOVE}(x,y,z,t)$$

- (ii) Tai lopputilan saavuttamisesta aiheutuvat poikkeukset:

$$\text{BLOCK_MOVE}(x,y,z,t) \leftarrow \text{GOAL_REACHED}(t)$$

$$\text{GOAL_REACHED}(t) \leftarrow \text{on}(b,a,t)$$

smodels

- Käytössä Prologin mukainen syntaksi: predikaatit kirjoitetaan pienellä alkukirjaimella ja muuttujat isolla.

- Ohjelman muuttujien arvoalueiden on oltava rajattuja:

Esimerkki. `step(0..100). eq(X,X) :- step(X).`

- Tyypillinen komentorivi, kun halutaan laskea n stabiilia mallia:

```
lparse file.lp | smodels n
```

(Arvolla $n = 0$ lasketaan kaikki stabiilit mallit).

3. Muuta huomioitavaa

- Käännöksen kokoa voidaan pienentää jakamalla operaattoreita:

$$\text{MOVE}(x,y,z,t) \leftrightarrow \text{OBJ}(x,t) \wedge \text{SRC}(y,t) \wedge \text{DST}(z,t).$$

Tämä idea toimii sellaisenaan lineaaristen suunnitelmien tapauksessa, mutta johtaa ongelmiin osittaisjärjestysuunnitelmissa.

- Stabiilien mallien laskennallinen tehokkuus riippuu instantioitujen ohjelman not-literaalien määrästä.

- Eheyshdoilla voidaan antaa suunnitelmille lisärajoitteita.

Esimerkki. Kielto liikuttaa samaa objektia kahdesti peräkkäin:

$$F \leftarrow \text{MOVE}(x,y,z,t) \wedge \text{MOVE}(x,z,v,t+1) \wedge \text{not } F.$$

4. Laskennallisesta vaativuudesta

- On olemassa useita merkittäviä laskennallisia ongelmia, joille ei tunneta yleistä polynomisen ajan ratkaisualgoritmia:

suunnitteluongelma, graafien värittäminen, ...

☞ Kysymyksessä NP-*vaikeat* (tai NP-*täydelliset*) ongelmat.

- Ongelma O on NP-täydellinen, jos

- O kuuluu luokkaan NP (ongelman O ratkaisut voidaan tarkistaa polynomisessa ajassa) ja
- O on NP-vaikea (muut NP-ongelmat ovat redusoitavissa ongelmaksi O polynomisessa ajassa).

Esimerkki. Lauselogiikan toteutuvuusongelma SAT on NP-täydellinen.

Stabiilin mallin olemassaolon tarkastaminen

- Jos annettuna on logiikkaohjelma P ja atomilausejoukko $M \subseteq \text{HB}(P)$, voidaan selvittää lineaarisessa ajassa, onko M logiikkaohjelman P stabiili malli.
 - ☞ Logiikkaohjelmien stabiilin mallin olemassaolon tarkistamista vastaava päätösongelma SM kuuluu luokkaan NP .
- SAT voidaan muuntaa SM -ongelmaksi polynomisessa ajassa käyttäen aiemmalla luennolla esitettyä muunnosta.
 - ☞ SM on NP -täydellinen.

Todistus: Olkoon O NP -ongelma. Koska SAT on NP -täydellinen, O voidaan muuntaa SAT-ongelmaksi O' pol. ajassa ja O' voidaan muuntaa SM -ongelmaksi O'' pol. ajassa, joten yhdistämällä muunnokset O voidaan muuntaa SM -ongelmaksi O'' pol. ajassa.

- Logiikkaohjelmat ovat tietämyksen esittämisen kannalta ilmaisuvoimaisempia kuin lauselogiikka.
- SAT voidaan muuntaa SM -ongelmaksi *modulaarisesti*: paikalliset muutokset lausejoukossa johtavat ainoastaan paikalliseen muutokseen logiikkaohjelmassa.
- SM -ongelmaa ei voida muuntaa SAT-ongelmaksi modulaarisesti! Kuvauksella $\text{Tr}(\cdot)$ on modulaarinen, jos jokaiselle atomijoukolle A :

ohjelmalla $P \cup A$ on stabiili malli

$\iff \text{Tr}(P) \cup A$ on toteutuva.

Todistus: Olkoon $\text{Tr}(\cdot)$ modulaarinen. Ohjelmalla $P = \{F \leftarrow \text{not } F\}$ ei ole stabiilia mallia, joten $\text{Tr}(P)$ ei ole toteutuva. Siis $\text{Tr}(P) \cup \{F\}$ ei toteutuva, joten ohjelmalla $P \cup \{F\}$ ei stabiilia mallia, ristiriita.

5. Suhde lauselogiikan toteutuvuusongelmaan

- SM -ongelman NP -täydellisyydestä johtuen epädeterministinen polynominen laskenta palautuu stabiilien mallien laskemiseen.
- Jokaiselle epädeterministiselle Turing-koneelle T , syötteelle x ja polynomille p voidaan muodostaa polynomisessa ajassa logiikkaohjelma P siten, että
 - T :llä on syötteelle x korkeintaan $p(|x|)$:n pituinen hyväksyvä laskenta \iff logiikkaohjelmalla P on stabiili malli.
- Käännöksen idea: olkoon syöte x ja $n = p(|x|)$. Kuvataan n laskenta-askelta: (i) nauhan tila (n paikkaa), (ii) mahdolliset T :n tilasiirtymät ja (iii) lisätään lopputilaehto.

6. Stabiilien mallien yksikäsitteisyydestä

- Logiikkaohjelmalla saattaa olla useampia stabiileja malleja tai ei yhtään stabiilia mallia.
- Milloin stabiileja malleja on täsmälleen yksi?
- Keskeisin tällainen ohjelmaluokka: *kerrostuneet ohjelmat* (engl. *stratified programs*).
- Määritellään ohjelmalle P riippuvuusgraafi $DG(P)$ seuraavasti:
 - Graafin solmuina ovat ohjelmassa esiintyvät atomit.
 - Kaari (A, B) kuuluu graafiin $DG(P) \iff$ ohjelmassa P on sääntö $A \leftarrow A_1 \wedge \dots \wedge A_m \wedge \text{not } A_{m+1} \wedge \dots \wedge \text{not } A_n$ siten, että $B \in \{A_1, \dots, A_n\}$.
 - Em. kaari (A, B) on *negatiivinen* $\iff B \in \{A_{m+1}, \dots, A_n\}$.

- Graafin $G = \langle V, E \rangle$ vahvasti kytketty komponentti K on mikä tahansa *maksimaalinen* solmujen osajoukko $K \subseteq V$ siten, että jokaisesta solmusta $v \in K$ pääsee jokaiseen solmuun $u \in K$ joltain graafin G polkua (suunnattu kaarten sekvenssi) pitkin.
- Graafin $DG(P)$ vahvasti kytketyt komponentit määräävät toistensa suhteen rekursiivisesti määritellyt atomit.
- Ohjelma on *kerrostunut*, joss sen riippuvuusgraafin vahvasti kytketyissä komponenteissa ei ole negatiivisia kaaria.
- Kerrostuneella ohjelmalla P on yksikäsitteinen stabiili malli M , (joka on myös yhtenevä mallin $WFM(P)$ kanssa).
- Ohjelman kerrostuneisuus voidaan selvittää lineaarisessa ajassa, koska graafin $DG(P)$ vahvasti kytketyt komponentit voidaan laskea lineaarisessa ajassa graafin koon suhteen.

Oppimistavoitteet

- Osaat ratkaista laajemman suunnitteluongelman esittämällä sen logiikkaohjelman ja hakemalla stabiileja malleja (3. kotitehtävä).
- Tiedät, millainen stabiilin mallin olemassaolon tarkistaminen on laskennalliselta vaativuudeltaan.
- Tiedät, mistä stabiilien mallien ja klassisten mallien ilmaisuvoimaero syntyy.
- Osaat muodostaa ohjelmalle riippuvuusgraafin ja hyödyntää sitä stabiilien mallien hakemisessa.

Esimerkki. Olkoon ohjelma $P =$

$$\{ A \leftarrow B \wedge \text{not } C, \quad B \leftarrow A \wedge \text{not } D, \\ C \leftarrow \text{not } E, \quad D \leftarrow E, \quad E \leftarrow A \}.$$

1. Ohjelma P ei ole kerrostunut: riippuvuusgraafissa $DG(P)$ on yksi vahvasti kytketty komponentti $\{A, B, C, D, E\}$, jossa negatiivisia kaaria.
2. Jos sääntö $E \leftarrow A$ jätetään pois, ohjelma P' on kerrostunut: riippuvuusgraafissa $DG(P')$ on vahvasti kytketyt komponentit $\{E\}$, $\{D\}$, $\{C\}$ ja $\{A, B\}$, joissa ei negatiivisia kaaria.
3. Stabiilin mallin haku voidaan suorittaa komponenteittain jossain riippuvuusgraafin kanssa yhteensopivassa järjestyksessä:

$$\text{not } E, \text{ not } D, C, \text{ not } A, \text{ not } B.$$

☞ Ohjelman P' yksikäsitteinen stabiili malli on $M = \{C\}$.