Suunnittelu stabiileilla malleilla

- 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.

Suunnitteluongelma (kertaus)

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

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

Esimerkki (kertaus)

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

Suunnittelun esittäminen logiikkaohjelmana

- 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: $\mathrm{on}(x, y, t)$, $\mathrm{MOVE}(x, y, z, t)$.
- Predikaateilla aika-argumentit t ovat väliltä $0, \ldots, n$ ja operaattoreilla väliltä $0, \ldots, n-1$.

Käännöksen rakenne

Suunnitteluongelman esitys jakautuu seuraaviin osiin:

1. Alkutilan määrittely: $\{P(\vec{c},0) \mid P(\vec{c}) \in S_0\}$.
2. Lopputilan määrittely: $\{F \leftarrow \text{not } P(\vec{c},n) \wedge \text{not } F \mid P(\vec{c}) \in S_1\}$.
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 $(\text{Post}(O)\sigma - \text{Pre}(O)\sigma)$ 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 osittaisjärjestyssuunnitelmia.

Esimerkki: palikkamaailman esitys

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

$$\begin{aligned} \text{MOVE}(x,y,z,t) \leftarrow\ & \text{clear}(x,t) \wedge \text{on}(x,y,t) \wedge \text{clear}(z,t) \wedge \\ & \text{not BLOCK\_MOVE}(x,y,z,t) \\ \text{on}(x,z,t+1) \leftarrow\ & \text{MOVE}(x,y,z,t) \\ \text{clear}(y,t+1) \leftarrow\ & \text{MOVE}(x,y,z,t) \end{aligned}$$

Huomio. Vimeisen kaltaista sääntöä ei tarvita predikaatille $\text{clear}(x,t+1)$, koska $\text{clear}(x,t)$ on operaatorin $\text{MOVE}(x,y,z,t)$ esiehtona, ja kehysaksiomat varmistavat, että $\text{clear}(x,t+1)$ on tosi.

4. Kehysaksiomat:
   (i) Jokaiselle atomiselle lauseelle:
   $$\text{on}(x,y,t+1) \leftarrow \text{on}(x,y,t) \wedge \text{not remove\_on}(x,y,t)$$
   (ii) Atomisen lauseen poistaville operaattoreille:
   $$\text{remove\_on}(x,y,t) \leftarrow \text{MOVE}(x,y,z,t)$$
5. Suunnitelmien laadun määrääminen:
   (i) *Lineaariset suunnitelmat*: sallitaan ainoastaan yhden operaattori-instanssin suorittaminen kerrallaan.
   $$\begin{aligned} \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow \text{MOVE}(x',y',z',t) \wedge \text{not } x = x' \\ \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow \text{MOVE}(x',y',z',t) \wedge \text{not } y = y' \\ \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow \text{MOVE}(x',y',z',t) \wedge \text{not } z = z' \\ \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow O(\vec{x},t) \ \ldots \end{aligned}$$

5. Suunnitelmien laadun määrääminen (jatkoa):
   (ii) *Osittaisjärjestyssuunnitelmat*: operaattori-instanssien samanaikainen suorittaminen kielletään ainoastaan konfliktitapauksissa
   $$\begin{aligned} \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow \text{MOVE}(x,y',z',t) \wedge \text{not } z = z' \\ \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow \text{MOVE}(x',y',z,t) \wedge \text{not } x = x' \end{aligned}$$
6. Operaattorin NOOP toteuttaminen poikkeuksilla:
   (i) Operaattorin oma poikkeus (eli *valinta* jättää suorittamatta):
   $$\text{BLOCK\_MOVE}(x,y,z,t) \leftarrow \text{not MOVE}(x,y,z,t)$$
   (ii) Tai lopputilan saavuttamisesta aiheutuvat poikkeukset:
   $$\begin{aligned} \text{BLOCK\_MOVE}(x,y,z,t) &\leftarrow \text{GOAL\_REACHED}(t) \\ \text{GOAL\_REACHED}(t) &\leftarrow \text{on}(b,a,t) \end{aligned}$$

Muuta huomioitavaa

- Käännöksen kokoa voidaan pienentää jakamalla operaattoreita:

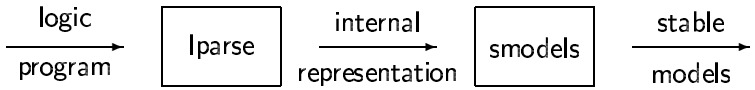
$$\mathrm{MOVE}(x,y,z,t) \leftrightarrow \mathrm{OBJ}(x,t) \wedge \mathrm{SRC}(y,t) \wedge \mathrm{DST}(z,t).$$

- Stabiilien mallien laskennallinen tehokkuus riippuu instantioidun ohjelman not-literaalien määrästä.
- Eheysehdoilla voidaan antaa suunnitelmille lisärajoitteita.

  Esimerkki. Kielto liikuttaa samaa objektia kahdesti peräkkäin:

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

smodelsin käyttö



- 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).`
- Typillinen komentorivi, kun halutaan laskea n stabiilia mallia:

  `lparse file.lp | smodels` n

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

Stabiilien mallien ominaisuuksia

- Stabiilien mallien laskennallinen vaativuus.
- Suhde lauselogiikan toteutuvuusongelmaan.
- Stabiilien mallien yksikäsitteisyydestä.

Laskennallisesta vaativuudesta

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

  suunnitteluogelma, 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.

Stabiilien mallien löytäminen

- Jos annettuna on logiikkaohjelma P ja atomilausejoukko $M \subseteq \mathrm{HB}(P)$, voidaan selvittää lineaarisessa ajassa, onko M logiikkaohjelman P stabiili malli.

  ☞ Logiikkaohjelmien stabiilien mallin löytämisongelma SM kuuluu luokkaan NP.

- SAT voidaan muuntaa SM-ongelmaksi polynomisessa ajassa (muunnos esitettiin aikaisemmalla luennolla).

  ☞ 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

Ilmaisuvoima

- 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.

- 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 kuvata SAT-ongelmaan modulaarisesti!

  Kuvaus $\mathrm{Tr}(\cdot)$ on modulaarinen, jos jokaiselle atomijoukolle A:

  ohjelmalla $P \cup A$ on stabiili malli

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

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

Stabiilien mallien yksikäsitteisyys

- 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 $\mathrm{DG}(P)$ seuraavasti:
  - (i) Graafin solmuina ovat ohjelmassa esiintyvät atomit.
  - (ii) Kaari (A, B) kuuluu graafiin $\mathrm{DG}(P)$ $\iff$ ohjelmassa P on sääntö $A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \mathrm{not}\, A_{m+1} \wedge \cdots \wedge \mathrm{not}\, A_n$ siten, että $B \in \{A_1, \ldots, A_n\}$.
  - (iii) Em. kaari (A, B) on *negatiivinen* $\iff$ $B \in \{A_{m+1}, \ldots, A_n\}$.

- Graafin $G = \langle V, E \rangle$ *vahvasti kytketty komponentti* K on mikä tahansa solmujen osajoukko $K \subseteq V$ siten, että jokaisesta solmusta $v \in K$ pääsee jokaiseen solmuun $u \in K$ jotain graafin G polkua (suunnattu kaarten sekvenssi) pitkin.
- Graafin $\mathrm{DG}(P)$ vahvasti kytketyt komponentit määräävät toistensa suhteen rekursiivisesti määritelyt 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 $\mathrm{WFM}(P)$ kanssa).
- Ohjelman kerrostuneisuus voidaan selvittää lineaarisessa ajassa, koska graafin $\mathrm{DG}(P)$ vahvasti kytketyt komponentit voidaan laskea lineaarisessa ajassa graafin koon suhteen.

Esimerkki. Olkoon ohjelma $P =$

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

1. Ohjelma P ei ole kerrostunut:
   riippuvuusgraafissa $\mathrm{DG}(P)$ 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 $\mathrm{DG}(P')$ 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:

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

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