Slide 1

Suunnittelu logiikkaohjelmointina

- Suunnitteluongelman dynaaminen luonne on aiheuttaa vaikeuksia klassiselle logiikalle (kehysongelma).
- Logiikkaohjelmissa minimaalisuus ja hyvin perustuvuus sisäänrakennettuna, jolloin suunnitteluongelman kuvaaminen yksinkertaistuu.
- Vaaditaan ilmaisuvoimaista logiikkaohjelmointilähestymistapaa (stabiilit mallit).
- Suunnittelun kuvaaminen logiikkaohjelmointiin antanut laskennallisesti lupaavia tuloksia.

Slide 2

Suunnittelun kääntäminen logiikkaohjelmointiin

- Idea:

  Rajoitetulle suunnitteluongelmalle $\langle P, n\rangle$ annetaan käännös logiikkaohjelmaksi $T(P,n)$ ja kyselyksi Q siten, että ongelmalla P on korkeintaan n pituinen ratkaisu joss ohjelmalla $T(P,n)$ on stabiili malli, joka toteuttaa kyselyn Q.
- Jokainen predikaatti ja operaattori varustetaan ylimääräisellä "aika"-argumentilla.

  Predikaateille aika-argumentti: $0,\ldots,n$
  operaattoreille: $0,\ldots,n-1$

Slide 3

- Käännös voidaan tehdä esimerkiksi seuraavasti:

  1. Alkutila otetaan sellaisenaan: $\{p(\vec{t},0) \mid p(\vec{t}) \in S_0\}$
  Lopputilaa ei voida sisällyttää ohjelmaan vaan se muodostaa kyselyn: löydä stabiili malli, joka sisältää joukon $\{p(\vec{t},n) \mid p(\vec{t}) \in S_1\}$.

  2. Operaattorien vaikutukset:
  Jos operaattorin esiehdot pätevät hetkellä I eikä sille ole poikkeusta, operaattoria sovelletaan hetkellä I ja lisättävät atomilauseet pätevät hetkellä $I+1$.

  3. Kehysaksioomat:
  Jos atomilause pätee hetkellä I eikä sen poistavaa operaattoria sovelleta hetkellä I, pätee atomilause hetkellä $I+1$.

  4. Operaattorien poikkeukset:
  Ratkaisevat, saadaanko lineaarisia tai osittaisjärjestyssuunnitelmia.

Slide 4

Esimerkki.
1. Alkutila: $\{on(a,b,0), on(b,c,0), clear(a,0), clear(d,0)\}$
Lopputilakysely: $\{on(b,a,n)\}$.

2. Operaattorien vaikutukset:
$move(X,Y,Z,I) \leftarrow$
$clear(X,I) \wedge on(X,Y,I) \wedge clear(Z,I) \wedge \text{not } blocked\_move(X,Y,Z,I)$
$on(X,Z,I+1) \leftarrow move(X,Y,Z,I)$
$clear(Y,I+1) \leftarrow move(X,Y,Z,I)$
$clear(X,I+1) \leftarrow move(X,Y,Z,I)$

3. Kehysaksioomat jokaiselle predikaatille
$on(X,Y,I+1) \leftarrow on(X,Y,I) \wedge \text{not } change\_on(X,Y,I)$
ja jokaiselle operaattorille, joka poistaa predikaatin
$change\_on(X,Y,I) \leftarrow move(X,Y,Z,I)$.

Slide 5

4. Operaattorin poikkeussäännöillä määrätään suunnitelman laatu.

(i) Lineaariset suunnitelmat: kaikille toimintopareille
$blocked\_move(X,Y,Z,I) \leftarrow op(\vec{X},I)$
$blocked\_move(X,Y,Z,I) \leftarrow move(X',Y',Z',I) \wedge \text{not } X = X'$
...

(ii) Osittaisjärjestyssuunnitelmat: poikkeus vain konfliktitapauksissa
$blocked\_move(X,Y,Z,I) \leftarrow move(X,Y',Z',I) \wedge \text{not } Z = Z'$
$blocked\_move(X,Y,Z,I) \leftarrow move(X',Y',Z,I) \wedge \text{not } X = X'$

(iii) Operaattori $noop$ korvattavissa operaattorin omalla poikkeuksella
$blocked\_move(X,Y,Z,I) \leftarrow \text{not } move(X,Y,Z,I)$

tai poikkeuksella, kun lopputila on saavutettu
$blocked\_move(X,Y,Z,I) \leftarrow goal\_reached(I)$

Slide 6

Muuta huomioitavaa

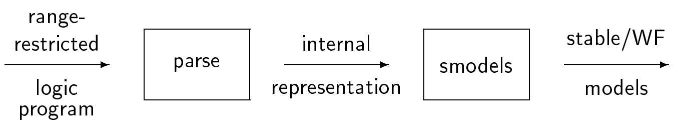
- Käännöksen kokoa voidaan pienentää käyttämällä operaattorin jakamistekniikkkaa: $move(X,Y,Z,I) \leftrightarrow o(X,I) \wedge s(Y,I) \wedge d(Z,I)$.
- Stabiilien mallien laskennallinen vaativuus riippuu ohjelman not -literaalien määrästä (huom. instantantioitu ohjelma).
- Eheysehdoilla voidaan antaa suunnitelmalle lisärajoitteita.

  Esim. älä liikuta samaa objektia kahta kertaa peräkkäin:

  $false \leftarrow move(X,Y,Z,I) \wedge move(X,Z,Z',I+1)$
  (haetaan stabiilia malli, joka ei sisällä atomia $false$)

Slide 7

smodelsin käyttö



- Prolog-syntaksi (ei funktioita eikä kirjasto-predikaatteja); ohjelman muuttujien oltava rajattuja. Esim. `eq(X,X) :- dom(X).`
- smodels laskee halutun määrän stabiileja malleja tai WF-mallin.
- Kysely voidaan antaa `compute`-komennolla:
  `compute 2 {not false, on(a,b,c,n)}`
  (hae 2 stabiilia malli, jossa ei ole `false` mutta on `on(a,b,c,n)`).

Slide 8

Esimerkki. demo.pl:

```
pickup(X,I) :- time(I), obj(X), not blocked(X,I).
blocked(X,I) :- time(I), obj(X), not pickup(X,I).
clear(X,I1) :- next(I1,I0), clear(X,I0).
false :- next(I1,I0), pickup(X,I0), pickup(X,I1).
clear(o1,t0). % initial situation
compute 1 {not false, clear(o1,t3)} % goal
next(t1,t0). next(t2,t1). next(t3,t2).
time(t0). time(t1). time(t2). time(t3).
obj(o1). obj(o2). obj(o3).
```

```
UNIX> parse < demo.pl | smodels     Answer: 1
UNIX> parse -text < demo.pl         Stable Model: obj(o3)...blocked(o3,t3)
UNIX> parse < demo.pl > demo.sm     Full set: pickup(o3,t3)...change(o1,t0)
UNIX> smodels -w < demo.sm          True
UNIX> smodels 0 < demo.sm           ...
```

Slide 9

Laskennallisesta kompleksisuudesta

- Suuri joukko merkittäviä laskennallisia ongelmia, joille ei tunneta yleistä polynomisen ajan ratkaisualgoritmia (suunnittelu, aikataulujen laadinta, optimointi, ...)

  ☞ NP-täydelliset ongelmat

- Ongelma on NP-täydellinen, jos
  - se kuuluu luokkaan NP: ratkaisu voidaan tarkistaa polyn. ajassa
  - jokainen NP-ongelma voidaan muuntaa ko. ongelmaksi (polyn. ajassa)

Esimerkki. Lauselogiikan toteutuvuusongelma (**SAT**) on NP-täydellinen.

Slide 10

Stabiilien mallien löytäminen

- Annetulle atomilausejoukolle voidaan lineaarisessa ajassa tarkistaa, onko se annetun logiikkaohjelman stabiili malli.

  ☞ Logiikkaohjelman stabiilien mallin löytämisongelma ($\mathbf{LP_{SM}}$) kuuluu luokkaan NP.

- **SAT** voidaan muuntaa $\mathbf{LP_{SM}}$-ongelmaksi polynomisessa ajassa.

  ☞ $\mathbf{LP_{SM}}$ 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 $\mathbf{LP_{SM}}$-ongelmaksi O'' pol. ajassa, joten yhdistämällä muunnokset O voidaan muuntaa $\mathbf{LP_{SM}}$-ongelmaksi O'' pol. ajassa.)

Slide 11

Ilmaisuvoima

- NP-täydellisyys:

  ☞ epädeterministinen polynominen laskenta on kuvattavissa:

  Jokaiselle epädeterministiselle Turing-koneelle T, syötteelle x ja polynomille p on olemassa lausejoukko (logiikkaohjelma) P siten, että koneella T on syötteelle x hyväksyvä laskenta, jonka pituus on pienempi kuin $p(|x|)$ joss lausejoukolla (logiikkaohjelmalla) P on (stabiili) malli.

  Idea: Olkoon syöte x ja $t = p(|x|)$.
  Kuvataan lauseina kullakin laskennan hetkellä (t hetkeä)
  (i) nauhan tila (t paikkaa) ja
  (ii) mahdolliset tilasiirtymät
  ja lisätään lopputilaehto.

Slide 12

- Logiikkaohjelmat ovat tietämyksen esittämisen kannalta ilmaisuvoimaisempia kuin lauselogiikka!
- **SAT** voidaan muuntaa $\mathbf{LP_{SM}}$-ongelmaksi modulaarisesti: paikalliset muutokset lausejoukossa johtavat paikalliseen muutokseen ohjelmassa.
- $\mathbf{LP_{SM}}$-ongelmaa ei voida kuvata **SAT**-ongelmaan modulaarisesti.

  Sanotaan kuvausta $\mathrm{Tr}(\cdot)$ modulaariseksi jos jokaiselle atomilausejoukolle F pätee:
  ohjelmalla $P \cup F$ on stabiili malli joss $F \cup \mathrm{Tr}(P)$ on toteutuva.

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

Slide 13

Mistä löytyvät vaikeat ongelmat?

NP-täydellisyys:
(i) Jos yhdelle NP-täydelliselle ongelmalle pol. ratkaisualgoritmi, tällöin löytyy kaikille.
(ii) Kaikki tunnetut ratkaisualgoritmi huonoimmassa tapauksessa eksponentiaalisia.

☞ Miten löydetään vaikeita instansseja NP-täydellisistä ongelmista?

Esimerkki. Satunnainen **SAT**
(i) Generoidaan lauseita satunnaisesti.
(ii) Lauseet literaalien disjunktioita (Esim. $p \vee \neg q \vee r$).
(iii) Kukin lause generoidaan valitsemalla siihen literaalit satunnaisesti annetusta literaalijoukosta.

☞ Keskimäärin helppoja sekä paikallisille että DP-menetelmille.

Slide 14

Esimerkki. Satunnainen k-**SAT**

- Lauseet literaalien disjunktioita, joissa kaikissa täsmälleen k literaalia (esimerkiksi 3).
- Kukin lause generoidaan valitsemalla siihen satunnaisesti k atomilausetta ja negatoimalla kukin niistä todennäköisyydellä 0.5.

☞ Laskennallisesti haastavia esimerkkejä **faasimuutosalueella**.

Slide 15

Faasimuutos

- Faasimuutos: ongelma muuttuu nopeasti alimääritellystä ylimääritellyksi.
- Satunnainen 3-**SAT** (Crawford/Auton, 1996):

  Vaikeimmat ongelmat pisteessä, jossa puolet satunnaisesti generoiduista lauseista on toteutuvia.

  Tämä piste saavutetaan, kun

$$\frac{\text{lauseiden määrä}}{\text{atomilauseiden määrä}} \approx 4.26$$

Slide 16

Nykytilanne

- Stokastisilla menetelmillä (WSAT) hämmästyttävän hyvä suorituskyky (pystyvät vaikeisiin 2000 atomilauseen ongelmiin)
- Suurimmat ongelmat: 50k atomil., 500k-1M lausetta
- Stokastisten menetelmien haittana epätäydellisyys.

  Ei voida soveltaa optimaalisuustarkasteluihin, teoreemantodistukseen.
- Täydelliset menetelmät (DP): 400 atomilauseeseen asti faasimuutosalueella.