Muuttujat logiikkaohjelmissa

- Tähän asti on käsitelty ainoastaan muuttujattomia ohjelmia, jotka ovat siis muotoa:

$$a_1 \leftarrow b_1, \text{not } c_1$$
$$a_2 \leftarrow b_2, \text{not } c_2$$
$$a_3 \leftarrow b_3, \text{not } c_3$$

- Ohjelmien kirjoittaminen tällaisilla säännöillä on huomattavan työlästä. Muuttujia käyttämällä ylläolevat kolme sääntöä voisi supistaa yhdeksi säännöksi:

$$a(X) \leftarrow b(X), \text{not } c(X)$$

Herbrand-universumi ja -instantiaatio

- Ohjelman P *Herbrand-universumi* $\mathbf{HU}(P)$ on siinä esiintyvien muuttujattomien termien joukko.
- Literaalin $p(X_1, \ldots, X_n)$ *Herbrand-instanssi* saadaan korvaamalla muuttujat $X_1, \ldots, X_n$ muuttujattomilla termeillä. Säännön instanssi saadaan instantioimalla kaikki siinä esiintyvät literaalit siten, että sama muuttuja saa aina saman arvon kaikissa paikoissa.
- Ohjelman P *Herbrand-instantiaatio* $\mathbf{HI}(P)$ on sen sääntöjen kaikkien mahdollisten Herbrand-instanssien joukko.

Stabiilit mallit

- Määritelmä: muuttujallisen logiikkaohjelman P stabiilit mallit ovat sen Herbrand-instantiaation $\mathbf{HI}(P)$ stabiilit mallit.
- Mikäli P:ssä ei esiinny funktiosymboleita, on $\mathbf{HI}(P)$ äärellinen, muussa tapauksessa se on ääretön.
- Koska äärettömän kokoisia instantiaatioita on hieman ikävä käsitellä, käytännön toteutuksista jätetään funktiosymbolit pois.

Esimerkki

- Tarkastellaan ohjelmaa P:

$$d(1) \leftarrow; \quad d(2) \leftarrow;$$
$$a(X) \leftarrow \text{not } b(X);$$
$$a(X) \leftarrow \text{not } b(X)$$

- Ohjelman Herbrand-universumi $\mathbf{HU}(P) = \{1, 2\}$ ja Herbrand-instantiaation $\mathbf{HI}(P)$ muodostavat seuraavat säännöt:

$$a(1) \leftarrow \text{not } b(1); \quad a(2) \leftarrow \text{not } b(2);$$
$$b(1) \leftarrow \text{not } a(1); \quad b(2) \leftarrow \text{not } a(2);$$
$$d(1) \leftarrow; \qquad d(2) \leftarrow$$

- Stabiilit mallit ($d(1)$ ja $d(2)$ mukana kaikissa): $M_1 = \{a(1), a(2)\}$, $M_2 = \{a(1), b(2)\}$, $M_3 = \{b(1), a(2)\}$, $M_4 = \{b(1), b(2)\}$

Instantiaatio-ongelma

- Täydellinen Herbrand-instantiaatio on melkein aina aivan liian iso (eksponentiaalinen) suoraan käsiteltäväksi.
- Useimmiten suurin osa säännöistä on turhia, sillä niiden vartaloissa esiintyy atomeita, jotka eivät voi olla koskaan tosia.
- Näin ollen instantiaation koon voi usein pienentää käytännölliseksi ilman, että stabiileja malleja katoaa.
- Joskus kaikki instantiaation säännöt ovat tarpeellisia, joten aina ei voi voittaa.

Esimerkki

- Olkoon P:

$$d(a) \leftarrow; \quad e(b) \leftarrow; \quad e(c) \leftarrow;$$
$$foo(X) \leftarrow d(X), \text{not } bar(X);$$
$$bar(X) \leftarrow d(X), \text{not } foo(X)$$

- Nyt $\mathbf{HI}(P) =$

$$foo(a) \leftarrow d(a), \text{not } bar(a); \quad bar(a) \leftarrow d(a), \text{not } foo(a);$$
$$foo(b) \leftarrow d(b), \text{not } bar(b); \quad bar(b) \leftarrow d(b), \text{not } foo(b);$$
$$foo(c) \leftarrow d(c), \text{not } bar(c); \quad bar(c) \leftarrow d(c), \text{not } foo(c);$$
$$d(a) \leftarrow; \quad e(b) \leftarrow; \quad e(c) \leftarrow;$$

- Koska atomeita $d(b)$ ja $d(c)$ ei voida johtaa, kaikki niistä riippuvat säännöt voidaan poistaa vaikuttamatta stabiileihin malleihin.

Domain-predikaatit

- Turhien sääntöjen karsimiseen käytetään *domain-predikaatteja*. Perusajatuksena on käyttää tiettyjä, yksinkertaisesti määriteltyjä, predikaatteja antamaan kaikki kiinnostavat muuttujasidonnat.
- Predikaatti on domain-predikaatti, mikäli sitä ei ole määritelty rekursiivisesti.
- Olkoon P:

$$a(1) \leftarrow; \quad a(2) \leftarrow$$
$$b(X,Y) \leftarrow a(X), a(Y)$$
$$s(X+1) \leftarrow a(X), s(X)$$
$$d(X) \leftarrow a(X), \text{not } c(X)$$
$$c(X) \leftarrow a(X), \text{not } d(X)$$

Nyt domain-predikaatteja ovat $a/1$ ja $b/2$.

Vahvasti tyypitetyt ohjelmat

- Sääntö on *vahvasti tyypitetty* (*strongly domain restricted*), mikäli kaikki siinä esiintyvät muuttujat esiintyvät myös jossain säännön hännässä olevassa positiivisessa domain-predikaatissa.
- Olkoon $d/1$ ainoa domain-predikaatti. Tarkastellaan sääntöjä:

$$b(Y,X) \leftarrow d(X), d(Y), \text{not } a(X,Y)$$
$$e(X,Y) \leftarrow c(X), d(Y)$$
$$e(X) \leftarrow \text{not } d(X)$$

Ainoastaan ensimmäinen sääntö on vahvasti tyypitetty. Toinen ei ole, koska muuttuja X ei esiinny domain-predikaatissa ja kolmas ei ole, koska not $d(X)$ on negatiivinen literaali.

- Ohjelma on vahvasti tyypitetty, jos kaikki sen säännöt ovat.

Tarvittavien sääntöjen laskeminen (1/2)

- Kunkin säännön tarvittavat instanssit saadaan laskettua käyttäen relaatioalgebraa.
- Tarkastellaan ohjelmaa

$$P = \{d_1(a,b); d_1(c,b); d_2(a,b); d_2(b,c); d_3(a);$$
$$p(X,Z) \leftarrow d_1(X,Y), d_2(Y,Z), q(X,Y), \text{not } d_3(X)\}$$

- Säännön mahdolliset muuttujasidonnat lasketaan hakemalla ensin kaikki $d_2(Y,Z)$:n instanssit ja etsimällä sitten niihin sopivat $d_1(X,Y)$:n instanssit. Lopuksi poistetaan ne instanssit, joille $d_3(X)$ on tosi.

Tarvittavien sääntöjen laskeminen (2/2)

1. askel			2. askel		
X	Y	Z	X	Y	Z
	a	b	a	b	c
	b	c	c	b	c

- Vastaava muuttujaton ohjelma on siten:

$$P_G = \{d_1(a,b); d_1(c,b); d_2(a,b); d_3(a)$$
$$p(c,b) \leftarrow d_1(c,b), d_2(b,c), \text{not } d_3(c), q(c,b)\}$$

Smodels ja lparse

- Smodels on Tietojenkäsittelyteorian laboratoriossa kehitetty toteutus stabiliin mallien semantiikalle.
- Smodels laskee malleja muuttujattomille ohjelmille.
- Lparse on esikäsittelijä, joka poistaa ohjelmasta muuttujat ja lisää muutenkin pari kerrosta syntaktista sokeria ohjelmien käyttämiseksi.

Smodelsin ja lparsen käyttäminen

- Yleensä smodelsia ja lparsea käytetään yhdessä:

```
% ./lparse -Wall foo.lp | smodels
smodels version 2.26. Reading...done
Answer: 1
Stable Model: a
True
Duration: 0.000
```

- Komentoriviargumenteilla voidaan parametrisoida ohjelmia ja vaikuttaa laskettavien mallien määrään.

Monimutkaisempia esimerkkejä

- Graafin väritysongelma voidaan ratkaista seuraavalla smodels-ohjelmalla:

```
1 { has_color(N,C) : color(C) } 1 :- node(N).
 :- edge(X,Y),color(C),has_color(X,C),has_color(Y,C).
```

- Tässä muotoa

$$L \ \{l_1, \ldots, l_n\} \ U$$

oleva rakenne on *painorajoite* (weight constraint), joka on tosi täsmälleen silloin, kun vähintään L ja korkeintaan U literaaleista l_i ovat tosia, ja muotoa

$$a(X) : b(X)$$

oleva rakenne tarkoittaa joukkoa $\{a(x_1), a(x_2), \ldots, a(x_n)\}$, missä kaikilla $1 \leq i \leq n$, $b(x_i)$ on tosi.

Toinen esimerkki: rehdit ja retkut

- Rehtien ja retkujen saarella on kahdenlaisia asukkaita: rehtejä, jotka puhuvat aina totta, ja retkuja, jotka valehtelevat aina. Eräänä päivänä saarelle tullut turisti tapasi kolme saaren asukasta, nimeltään A, B ja C. Asukkaat kertoivat itsestään turistille seuraavasti:

1. A kertoi, että B ja C ovat molemmat rehtejä.
2. B kertoi, että A on retku ja C on rehti.

Mitä tyyppiä kolme asukasta olivat?

Arvoitus logiikkaohjelmana

- Arvoituksen voi kirjoittaa smodels-ohjelmaksi seuraavasti:

```
henkilo(a; b; c).  % maarittelee 3 henkilo-instanssia
1 { rehti(H), retku(H) } 1 :- henkilo(H).

2 { rehti(b), rehti(c) } 2 :- rehti(a). % 1. vihje
 :- retku(a), rehti(b), rehti(c).

2 { retku(a), rehti(c) } 2 :- rehti(b). % 2. vihje
  :- retku(b), retku(a), rehti(c).
```


Vastaus

- Ajetaan ohjelma komennolla

```
% lparse -d none -Wall puzzle.lp | smodels 0
smodels version 2.26. Reading...done
Answer: 1
Stable Model: retku(a) retku(c) retku(b)
False
Duration: 0.020
```

- Ainoa oikea vastaus on siis se, että kaikki olivat retkuja.

Lisää syntaktista sokeria

- Lparsen sisältämästä syntaktisesta sokerista tärkeimmät ovat numeeriset funtiot, vakiomäärittelyt ja intervallit:

```
const n=5.
a(1..n). % määrittelee faktat a(1), ..., a(5)
b(2*X) :- a(X).
```

- Funktiot eivät ole aitoja funktiosymboleita vaan ne tulkitaan lyhennysmerkinnöiksi (eli $a(2+1)$ on vain hassu tapa kirjoittaa $a(3)$).