Sääntöpohjainen päättely

- Sääntöpohjainen päättely:
  logiikkaohjelmointi, deduktiiviset tietokannat, asiantuntijajärjestelmät
- Horn-klausuulit tarjoavat loogisen pohjan sääntöpohjaiselle päättelylle.
- Horn-klausuuleille tehokkaita toteutusmenetelmiä.
- Horn-klausuulit ovat käytetyin lauselogiikan tehokkaasti ratkaistavissa oleva aliluokka.
- Tyypillisesti tarvitaan kuitenkin Horn-klausuuleja ilmaisuvoimaisempi kieli (negaatio/komplementti).

Horn-klausuulit

Määritelmä 1 *Horn-klausuuli on literaalien disjunktio, jossa on korkeintaan* yksi *positiivinen literaali.*

Esimerkki. $\neg P \vee \neg Q \vee \neg R$ ja $\neg P \vee Q \vee \neg R$ ovat Horn-klausuuleja.

$P \vee \neg Q \vee R$ ei ole Horn-klausuuli.

Merkintätapa.
Usein kirjoitetaan Horn-klausuuli $\neg P_1 \vee \cdots \vee \neg P_n \vee Q$ muotoon

$$Q \leftarrow P_1 \wedge \cdots \wedge P_n$$

ja $\neg P_1 \vee \cdots \vee \neg P_n$ muotoon

$$\leftarrow P_1 \wedge \cdots \wedge P_n$$

Ohjelmaklausuulit

Tyypillisesti käytetään ohjelmaklausuuleja (program clauses; definite clauses).

Määritelmä 2 *Ohjelmaklausuuli on literaalien disjunktio, jossa on täsmälleen* yksi *positiivinen literaali.*

Esimerkki. $\neg P \vee \neg Q \vee \neg R$ ei ole ohjelmaklausuuli mutta $\neg P \vee Q \vee \neg R$ $(Q \leftarrow P \wedge Q)$ on.

Propositio 3 *Ohjelmaklausuulijoukko on toteutuva.*

Ohjelmaklausuulijoukolle saadaan malli esim. ottamalla siihen jokaisen klausuulin positiivinen atomi.

Huom! Horn-klausuulijoukko ei ole välttämättä toteutuva (esim. $\{P, \neg P\}$).

Horn-klausuulijoukolle voidaan määritellä vastaava ohjelmaklausuulijoukko ottamalla käyttöön uusi atomi $\perp$ ja kirjoittamalla puhtaasti negatiiviset Horn-klausuulit

$\neg P_1 \vee \cdots \vee \neg P_n$

muotoon

$\perp \vee \neg P_1 \vee \cdots \vee \neg P_n$
$(\perp \leftarrow P_1 \wedge \cdots \wedge P_n)$.

Nyt Horn-klausuulien toteutuvuusongelma voidaan tutkia ohjelmaklausuulien avulla.

Propositio 4 *Joukko Horn-klausuuleja S on toteutuva joss $S' \not\models \perp$, missä S' on joukkoa S vastaava ohjelmaklausuulien joukko.*

Esimerkki $\{P, \neg P\}$ ei ole toteutuva ja $\{P, \perp \leftarrow P\} \models \perp$.

Ohjelmaklausuulien minimimalli

Määritelmä 5 *(i) Malli M_1 (atomilauseiden joukko) on pienempi kuin malli M_2 ($M_1 < M_2$), joss $M_1 \subset M_2$.*

(ii) Lausejoukon mallia M (atomilauseiden joukko) sanotaan minimaaliseksi, jos lausejoukolla ei ole sitä pienempää mallia.

Esimerkki. $M = \{Q, R\}$ ei ole joukon $\{Q \leftarrow R, R \leftarrow P \wedge Q\}$ minimimalli, mutta $M' = \{\}$ on.

Teoreema 6 *Joukolla ohjelmaklausuuleja S on yksikäsitteinen minimimalli M_S (pienin malli), joka on joukon S kaikkien mallien leikkaus.*

Teoreema 7 *Olkoon S joukko ohjelmaklausuuleja. Tällöin $M_S = \{P \mid P$ on atomilause ja $S \models P\}$.*

Minimimallin konstruointi

Määritelmä 8 *Olkoon S (mahd. ääretön) joukko ohjelmaklausuuleja. Määritellään kuvaus T_S atomilausejoukoilta atomilausejoukoille:*

$$\mathrm{T}_S(M) = \{P \mid P \leftarrow P_1 \wedge \cdots \wedge P_n \in S, \{P_1, \ldots, P_n\} \subseteq M\}$$

Esimerkki. $S = \{P \leftarrow P, Q, R_1 \leftarrow Q, R_2 \leftarrow Q \wedge P\}$.
Nyt $\mathrm{T}_S(\{P\}) = \{P, Q\}$ ja $\mathrm{T}_S(\{P, Q\}) = \{P, Q, R_1, R_2\}$.

- M on kuvauksen T_S *kiintopiste*, joss $\mathrm{T}_S(M) = M$.
  Esimerkki. $M_1 = \{P, Q, R_1, R_2\}$ on kuvauksen T_S kiintopiste: $\mathrm{T}_S(M_1) = \{P, Q, R_1, R_2\} = M_1$
- Kiintopiste M on pienin, jos muille kiintopisteille M', $M \subseteq M'$.
  Esimerkki. $M_1 = \{Q, R_1\}$ on kuvauksen T_S pienin kiintopiste.

- T_S on monotoninen: kaikille $M \subseteq M'$ pätee $\mathrm{T}_S(M) \subseteq \mathrm{T}_S(M')$.
- Knaster-Tarski -teoreema: jokaisella monotonisella kuvauksella on yksikäsitteinen pienin (suurin) kiintopiste.

Teoreema 9 *Kuvauksella T_S on pienin kiintopiste* $\mathrm{lfp}(\mathrm{T}_S) = \bigcap\{M \mid \mathrm{T}_S(M) \subseteq M\}$.

Propositio 10 *M on ohjelmaklausuulijoukon S malli, joss $\mathrm{T}_S(M) \subseteq M$.*

Teoreema 11 $M_S = \mathrm{lfp}(\mathrm{T}_S)$.

Määritelmä 12
$\mathrm{T}_S \uparrow 0 = \emptyset$
$\mathrm{T}_S \uparrow \alpha = \mathrm{T}_S(\mathrm{T}_S \uparrow \alpha - 1)$
$\mathrm{T}_S \uparrow \omega = \bigcup\{\mathrm{T}_S \uparrow \alpha \mid \alpha < \omega\}$

Teoreema 13 $\mathrm{lfp}(\mathrm{T}_S) = \mathrm{T}_S \uparrow \omega$.

Esimerkki. $S = \{P \leftarrow P, Q, R_1 \leftarrow Q, R_2 \leftarrow Q \wedge P\}$.

$\mathrm{T}_S \uparrow 0 = \emptyset$
$\mathrm{T}_S \uparrow 1 = \mathrm{T}_S(\mathrm{T}_S \uparrow 0) = \mathrm{T}_S(\emptyset) = \{Q\}$
$\mathrm{T}_S \uparrow 2 = \mathrm{T}_S(\mathrm{T}_S \uparrow 1) = \mathrm{T}_S(\{Q\}) = \{Q, R_1\}$
$\mathrm{T}_S \uparrow 3 = \mathrm{T}_S(\mathrm{T}_S \uparrow 2) = \mathrm{T}_S(\{Q, R_1\}) = \{Q, R_1\}$
$\vdots$
$\mathrm{T}_S \uparrow \omega = \bigcup\{\mathrm{T}_S \uparrow \alpha \mid \alpha < \omega\} = \{Q, R_1\}$

Johdettavuus ohjelmalauseilla

Määritelmä 14 *Atomilauseen P on johto ohjelmalauseista S on äärellinen jono atomilauseita $P_0, \ldots, P_n$ siten, että $P = P_n$ ja*

(i) $P_0 \in S$ ja (ii) jokaisella $i > 0$, on olemassa ohjelmaklausuuli $P_i \leftarrow Q_1 \wedge \cdots \wedge Q_n \in S$, jossa jokainen Q_j esiintyy aiemmin jonossa.

Jos lauseelle P on olemassa johto joukosta S, merkitään tätä $S \vdash P$.

Teoreema 15 *(i) $S \models P$ joss $S \vdash P$.*
(ii) $M_S = \{P \mid P$ on atomilause ja $S \vdash P\}$.

Esimerkki. $S = \{P \leftarrow P, Q, R_1 \leftarrow Q, R_2 \leftarrow Q \wedge R_1, R_2 \leftarrow P\}$.

Q, R_1, R_2 on lauseen R_2 johto joukosta S ($S \vdash R_2$).

Q, P, R_2 ei ole lauseen R_2 johto joukosta S.

Logiikkaohjelmat ja deduktiiviset tietokannat

- Logiikkaohjelmat (ilman ei-loogisia operaatioita: cut, not, assert, retract) ja deduktiiviset tietokannat voidaan nähdä joukkona ohjelmaklausuuleja, joissa literaalit sisältävät muuttujia sekä vakio- ja funktiosymboleja.
- Mikä on tällaisen logiikkaohjelman semantiikka: milloin annettuun kyselyyn Q pitää antaa myönteinen vastaus?
  ☞ Joss Q on tosi ohjelman pienimässä Herbrand-mallissa.
- Ohjelman pienin Herbrand-malli on yksikäsitteinen:
  (i) Ohjelmaklausuulijoukon Herbrand-mallit ovat sen Herbrand-instanssien (propositionaalisia klausuuleja) malleja.
  (ii) Ohjelman Herbrand-instanssien joukko on propositionaalinen ohjelmaklausuulien joukko, jolle on yksikäsitteinen pienin malli.

Esimerkki. Tietokanta S

```
v(k,j).
v(t,j).
l(X) :- v(X,Y).
```

Herbrand-instanssit:

$\{v(k,j), v(t,j),$
$l(j) \leftarrow v(j,j)$
$l(j) \leftarrow v(j,k)$
$l(j) \leftarrow v(j,t)$
$l(k) \leftarrow v(k,j)$
$l(k) \leftarrow v(k,k)$
$l(k) \leftarrow v(k,t)$
$l(t) \leftarrow v(t,j)$
$l(t) \leftarrow v(t,k)$
$l(t) \leftarrow v(t,t)\}$

Pienin Herbrand-malli:
$\{v(k,j), v(t,j), l(k), l(t)\}$
antaa odotetut vastaukset:

```
?- v(k,j).
yes
?- v(k,p).
no
?- l(k).
yes
?- l(j).
no
```


Esimerkki.
Olkoon tietokanta S

```
vanhempi(kaija,jussi).
vanhempi(tauno,jussi).
vanhempi(kaija,teija).
vanhempi(saija,kaija).
vanhempi(raija,saija).
sisarus(X,Y)  :- vanhempi(Z,X),
                 vanhempi(Z,Y).
esivanh(X,Y)  :- vanhempi(X,Y).
esivanh(X,Y)  :- vanhempi(X,Z),
                 esivanh(Z,Y).
```

S:n pienin Herbrand-malli antaa odotetut vastaukset:

```
?- vanhempi(kaija,jussi)
yes
?- vanhempi(raija,jussi)
no
?- sisarus(teija,jussi)
yes
?- sisarus(teija,jukka)
no
?- esivanh(raija,jussi)
yes
?-
esivanh(raija,jalmari)
no
```

Ilmaisuvoima

- Unioni:
  $u(X_1, \ldots, X_n) \leftarrow r_1(X_1, \ldots, X_n)$
  $u(X_1, \ldots, X_n) \leftarrow r_2(X_1, \ldots, X_n)$
- Leikkaus:
  $l(X_1, \ldots, X_n) \leftarrow r_1(X_1, \ldots, X_n) \wedge r_2(X_1, \ldots, X_n)$
- Projektio:
  $isa(X) \leftarrow isa(X, Y)$
- Valinta:
  $miljonaari(X) \leftarrow tulot(X, Y) \wedge (Y > 100000)$
- Join:
  $isoisa(X, Y, Z) \leftarrow isa(X, Y) \wedge vanhempi(Y, Z)$

- Verrattuna relaatioalgebraan (SQL): komplementti puuttuu.
- Toisaalta ilmaisuvoimaisempi kuin relaatioalgebra (SQL): mahdollistaa rekursiiviset määrittelyt.
  Esim. transitiivinen sulkeuma:
  $ts_r(X_1, X_2) \leftarrow r(X_1, X_2)$
  $ts_r(X_1, X_2) \leftarrow r(X_1, Y) \wedge ts_r(Y, X_2)$
- Tyypillisesti lisätään komplementti jossain muodossa.
  Ad hoc -ratkaisut johtavat ongelmiin:
  $b(X) \leftarrow eng(X) \wedge \text{not } a(X)$
  $a(X) \leftarrow mba(X) \wedge \text{not } b(X)$
  $eng(liisa)$
  $mba(liisa)$

Toteutustekniikat

- Prologin toteutus ei ole täydellinen.
  Esim. Ohjelma `p :- p.` ja kysely `?- p.`
  Vastaus pitäisi olla kieltävä mutta Prolog jää silmukkaan.
- Bottom-up: magic sets
  Deduktiiviset tietokannat
- Top-down: SLG-resoluutio (tabling-tekniikka)
  XSB-Prolog
- Muuttujaton ohjelma: minimimalli laskettavissa lineaarisessa ajassa [Dowling-Gallier, 1984].

Ratkaisumenetelmä prop. Horn-klausuuleille

- Horn-klausuulit keskeisin tehokkaasti toteutettavissa oleva osajoukko lauselogiikkaa.
- Ratkaisuongelma: onko äärellinen joukko propositionaalisia Horn-klausuuleja toteutuva.
- Kun tämä osataan ratkaista tehokkaasti, saamme tehokkaan ratkaisun myös ongelmalle
  $S \models Q$, missä
  S on Horn-klausuulijoukko ja Q disjuntiivisessa normaalimuodossa, jossa kussakin konjunktissa esiintyy korkeintaan yksi negatiivinen literaali.
  (Huom.: $S \models Q$ joss $S \cup \{\neg Q\}$ on toteutumaton.)

Polynominen ratkaisumenetelmä saadaan suoraviivaisesti:

(i) Joukko Horn-klausuuleja S on toteutuva joss $S' \not\models \perp$, missä S' on joukkoa S vastaava ohjelmaklausuulien joukko.
(ii) $S' \not\models \perp$ joss $\perp \notin M_{S'}$
(iii) $M_{S'}$ laskettavassa iteroimalla operaattoria $\mathrm{T}_{S'}$ $(\mathrm{T}_{S'} \uparrow 0, \mathrm{T}_{S'} \uparrow 1, \ldots)$, kunnes saavutetaan kiintopiste $(\mathrm{T}_{S'} \uparrow n = \mathrm{T}_{S'} \uparrow n-1)$.
(iv) Iteraatioita korkeintaan $n+1$ kappaletta, missä n on S':n atomien määrä, sillä $\mathrm{T}_{S'}$ on monotoninen ja jos ei olla vielä kiintopisteessä, saadaan vähintään yksi uusi atomi.
(v) Tämä kiintopiste on pienin, sillä $\mathrm{T}_{S'}$ on monotoninen.
(vi) Kukin iteraatio toteutettavissa lineaarisessa ajassa.

☞ Saadaan neliöllinen ratkaisumenetelmä.

Lineaarinen ratkaisumenetelmä

[Dowling-Gallier, 1984]

- Idea: toteutetaan T_S-iteraatio tehokkaalla tietorakenteella siten, että kutakin atomia käsitellään vain kerran.
- Käytetään taulukoita, joita varten tarvitaan kullekin atomille yksikäsitteinen indeksi.
- Jos indeksejä ei tiedetä etukäteen ja syöttöjoukko joudutaan jäsentämään, vaatii indeksien muodostaminen $\mathcal{O}(n \log n)$ työn (symbolitaulu).
- Tämän jälkeen algoritmi toimii lineaarisessa ajassa.

Tietorakenteet

Syötejoukossa K atomilausetta, M klausuulia.
Otetaan käyttöön seuraavat taulukot.

- val[K]
  val[p]: atomin p totuusarvo (aluksi positiiviset yksikköliteraalit true, muut false).
- clauselist[K]
  clauselist[p]: lista klausuuleja, joissa atomi p esiintyy negatiivisena.
- numargs[M]
  numargs[n]: klausuulin n niiden negatiivisten literaalien p määrä, joille val[p] = false.
- poslit[M]
  poslit[n]: klausuulin n positiivinen atomi

Algoritmi

Luetaan syötejoukko;
Alustetaan taulukot clauselist, numargs, poslit, val;
consistent := true;
falseatom := atomin $\perp$ indeksi;
queue := positiiviset atomit yksikköklausuuleista;
minmodel;
/* Laskee syötteenä annetun ohjelmaklausuulijoukon pienimmän mallin taulukkoon val paitsi, jos $\perp$ saa arvon tosi, jolloin keskeytetään. */
if val[falseatom] = true **then** print('unsat set of clauses')
else
  print('satisfying model: '); printmodel
endif

```
procedure minmodel;
while queue is not empty and consistent do
  atom1 := pop(queue) ;
  clist := clauselist[atom1] ;
  while clist not empty and consistent do
    clause2 := get(clist); numargs[clause2] := numargs[clause2] - 1;
    if numargs[clause2] = 0 then
      n:= poslit[clause2];
      if val[n] = false then
        val[n] := true;
        if n = falseatom then consistent := false;
        else queue := push(n, queue);
        endif
      endif
    endif
  endwhile
endwhile
```


Esimerkki.

$S = \{1 : P_0$

$2 : P_1 \leftarrow P_0$

$3 : P_2 \leftarrow P_1 \wedge P_0$

$4 : P_3 \leftarrow P_1$

$5 : \perp_5 \leftarrow P_2 \wedge P_4\}$

Algoritmi laskee seuraavan mallin:

P_0	true
P_1	true
P_2	true
P_3	true
P_4	false
$\perp_5$	false

val[0] := true
queue := $\langle 0 \rangle$
atom1 := 0
clist := [2,3]
  clause2 := 2
  numargs[2] := 0
  val[1] := true
  queue := $\langle 1 \rangle$
  clause2 := 3
  numargs[3] := 1
atom1 := 1
clist := [3,4]
  clause2 := 3
  numargs[3] := 0
  val[2] := true
  queue := $\langle 2 \rangle$
  clause2 := 4
  numargs[4] := 0
  val[3] := true
  queue := $\langle 2, 3 \rangle$
atom1 := 2
clist := [5]
  clause2 := 5
  numargs[5] := 1
atom1 := 3
clist := []

Huomioita

- Oikeellisuus:
  (i) Päättyy T_S-operaattorin kiintopisteeseen $\{P_i \mid$ val[i] = true$\}$.
  (ii) Jos val[i] = true, niin $P_i \in \mathrm{lfp}(\mathrm{T}_S)$.
- Lineaarinen aikavaativuus:
  (i) Kukin atomi jonoon (queue) korkeintaan kerran.
  (ii) Uloimmassa **while**-silmukassa työtä tehdään kokonaisuudessaan korkeintaan negatiivisten literaalien esiintymien lukumäärän verran.
- Jono ⇒ leveyshaku (Pino ⇒ syvyyshaku)
- Samaa ideaa voidaan käyttää yksikköresoluution toteuttamiseen lineaarisessa ajassa (esim. Davis-Putnam -menetelmässä).