Toteutustekniikoita

Käsiteltäviä asioita:

1. Horn-klausuulien ratkaisumenetelmät
2. Dowling-Gallier-algoritmi
3. Stabiilien mallien hakutekniikkaa
4. Approksimaatioiden toteuttaminen
5. Hakualgoritmi stabiileille malleille

1. Horn-klausuulien ratkaisumenetelmät

- Horn-klausuulit muodostavat keskeisimmän tehokkaasti toteutettavissa olevan osajoukon lauselogiikasta.
- Ratkaisuongelma: onko äärellinen joukko propositionaalisia Horn-klausuuleja S toteutuva?
- Kun tämä osataan ratkaista tehokkaasti, saamme myös tehokkaan tavan ratkaista ongelman

  $$S \models Q_1 \vee \cdots \vee Q_n$$

  missä S on Horn-klausuulijoukko ja jokainen Q_i on literaalien konjunktio, jossa esiintyy korkeintaan yksi negatiivinen literaali.

  Huomio. $S \models Q \iff S \cup \{\neg Q\}$ on toteutumaton.

Neliöllinen ratkaisumenetelmä

1. Joukko Horn-klausuuleja S on toteutuva $\iff S' \not\models \bot$ (S' on joukkoa S vastaava ohjelmaklausuulien joukko).
2. $S' \not\models \bot \iff \bot \notin M_{S'}$.
3. Malli $M_{S'}$ voidaan laskea operaattorilla $\mathrm{T}_{S'}$:

   $$\mathrm{T}_{S'} \uparrow 0,\ \mathrm{T}_{S'} \uparrow 1,\ \mathrm{T}_{S'} \uparrow 2,\ \ldots$$

   kunnes saavutetaan kiintopiste ($\mathrm{T}_{S'} \uparrow n = \mathrm{T}_{S'} \uparrow n-1$).
4. Iteraatioita tarvitaan korkeintaan $|\mathrm{HB}(S)|+1$ kappaletta.
5. Joukon $\mathrm{T}_{S'}(M)$ laskenta voidaan suorittaa mille tahansa $M \subseteq \mathrm{HB}(S')$ lineaarisessa ajassa $||S'||$:n suhteen.
6. Aikakompleksisuus: $O(|\mathrm{HB}(S')| \times ||S'||)$.

2. Dowling-Gallier-algoritmi

- W.F. Dowling ja J.H. Gallier [1984]: *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae*
- Idea: toteutetaan T_S-iteraatio tehokkaalla tietorakenteella siten, että kutakin atomia käsitellään vain kerran.
- Käytetään taulukoita, joita varten tarvitaan kullekin atomille yksikäsitteinen indeksi $\mathrm{HB}(S) = \{P_0, P_1, \ldots, P_{n-1}\}$.
- Jos indeksejä ei tiedetä etukäteen ja syötejoukko joudutaan jäsentämään, vaatii indeksien muodostaminen $O(||S|| \times \log |\mathrm{HB}(S)|)$ työn (symbolitaulu).
- Tämän jälkeen algoritmi toimii *lineaarisessa ajassa* $||S||$:n suhteen.

Tietorakenteet

Olkoon syötejoukossa n atomilausetta ja m ohjelmaklausuulia.

Otetaan käyttöön seuraavat taulukot:

- val[0...n-1]: atomien $P_0, P_1, \ldots, P_{n-1}$ totuusarvot (aluksi "true", jos P_i esiintyy ainoana (positiivisena) literaalina jossain ohjelmaklausuulissa, muutoin "false").
- occurs[0...n-1]: niiden ohjelmaklausuulien indeksien lista, joissa atomi esiintyy negatiivisena.
- negcnt[0...m-1]: klausuulin negatiivisten literaalien lukumäärä.
- poslit[0...m-1]: klausuulin positiivinen literaali.

Algoritmi

```
Luetaan syötejoukko;
Alustetaan taulukot val[0...n-1], occurs[0...n-1],
  negcnt[0...m-1] ja poslit[0...m-1];
consistent := true;
falseatom := atomin ⊥ indeksi;
queue := atomit, joille val[i] aluksi true;
minmodel;
/* Laskee syötteenä annetun ohjelmaklausuulijoukon pienimmän mallin
taulukkoon val paitsi, jos ⊥ saa arvon tosi, jolloin keskeytetään. */
if consistent then
  print('satisfying model: '); printmodel
else print('unsat set of clauses')
endif
```



```
procedure minmodel;
while not empty(queue) and consistent do
  a := get(queue);
  clauses := occurs[a] ;
  while not empty(clauses) and consistent do
    clause := get(clauses); negcnt[clause] := negcnt[clause]-1;
    if negcnt[clause]=0 then
      n := poslit[clause];
      if val[n]=false then
        val[n] := true;
        if n=falseatom then consistent := false else push(n, queue);
      endif
    endif
  done
done
```


Esimerkki. Olkoon $S =$

$$\{0: P_0,\ 1: P_1 \leftarrow P_0,\ 2: P_2 \leftarrow P_1 \wedge P_0,\ 3: P_3 \leftarrow P_1,\ 4: \bot_5 \leftarrow P_2 \wedge P_4\}.$$

Algoritmi laskee mallin $M_S = \{P_0, P_1, P_2, P_3\}$ seuraavasti:

val[0] := true	a := 1	a := 2
queue := ⟨0⟩	clauses := [2,3]	clauses := [4]
a := 0	clause := 2	clause := 4
clauses := [1,2]	negcnt[2] := 0	negcnt[4] := 1
clause := 1	val[2] := true	a := 3
negcnt[1] := 0	queue := ⟨2⟩	clauses := []
val[1] := true	clause := 3	
queue := ⟨1⟩	negcnt[3] := 0	
clause := 2	val[3] := true	
negcnt[2] := 1	queue := ⟨2,3⟩	

Huomioita Dowling-Gallier-algoritmista

➤ Oikeellisuus: val[i]=true $\iff$ $P_i \in \mathrm{lfp}(\mathrm{T}_S)$.

➤ Lineaarinen aikavaativuus suhteessa $||S||$:een.

1. Kukin atomi päätyy jonoon (queue) korkeintaan kerran.
2. Sisemmässä **while**-silmukassa tehdään työtä kokonaisuudessaan korkeintaan negatiivisten literaalien esiintymien lukumäärän verran.

➤ Algoritmin suorittaman hakutyyppi voidaan valita:

– Jono $\Rightarrow$ leveyshaku

– Pino $\Rightarrow$ syvyyshaku

➤ Samalla periaatteella voidaan laatia lineaarisen ajan toteutus yhden literaalin säännöstä (esim. Davis-Putnam -menetelmässä).

3. Stabiilien mallien hakutekniikkaa

➤ Tavoiteena laskea mahdollisimman tehokkaasti logiikkaohjelman P annetut kriteerit täyttävä(t) stabiili(t) malli(t).

➤ Täysiin joukkoihin perustuvan stabiilien mallien karakterisoinnin perusteella hakuavaruus muodostuu olennaisesti joukon $\mathrm{NA}(P)$ (ohjelmassa P esiintyvät not-literaalit) osajoukoista.

➤ Rakennetaan asteittain stabiileja malleja rajaavaa osittaismallia (literaalijoukko FS) ja pyritään karsimaan hakuavaruutta.

(i) Malleja rajaavat oletukset tehdään yksi kerrallaan.

(ii) Jokaisessa hakupisteessä lasketaan approksimaatio kaikille stabiileille malleille, jotka toteuttavat senhetkiset oletukset ja tutkitaan, onko mahdollisesti saavutettu konflikti.

☞ Saadaan Davis-Putnam -menetelmää muistututtava algoritmi stabiilien mallien laskemista varten.

➤ Toteutustekniikkaa analysoidaan mm. seuraavassa artikkelissa:

P. Simons, I. Niemelä, and T. Soininen [2002, AIJ]: *Extending and Implementing the Stable Model Semantics*

➤ Lisäksi on toteutettu muuttujia sisältävien ohjelmien instantiointi:

T. Syrjänen [2004]: *Cardinality Constraint Programs*

T. Syrjänen [1998]: *Implementation of Local Grounding for Logic Programs with Stable Model Semantics*

➤ Ohjelmat `smodels` ja `lparse` ovat saatavilla verkossa osoitteella

`http://www.tcs.hut.fi/Software/smodels/`

Approksimointiperiaatteet

➤ Laskettavat stabiilit mallit rajataan literaalijoukolla FS:

(i) Jos $A \in FS$, niin $A \in M$ laskettaville stabiileille malleille M.

(ii) Jos $\mathrm{not}\,A \in FS$, niin $A \notin M$ laskettaville stabiileille malleille M (näitä vastaaville täysille joukoille F pätee $\mathrm{not}\,A \in F$).

➤ Tämä ehdot voidaan ymmärtää ohjelman P stabiiliin mallin M ja literaalijoukon FS välisenä *yhteensopivuutena*.

Esimerkki. Tarkastellaan normaalia logiikkaohjelmaa

$$P = \{\ A \leftarrow \mathrm{not}\,B,\ B \leftarrow \mathrm{not}\,A,\ C \leftarrow \mathrm{not}\,D,$$
$$D \leftarrow \mathrm{not}\,C,\ E \leftarrow \mathrm{not}\,F,\ F \leftarrow \mathrm{not}\,E\ \}.$$

Nyt esim. literaalijoukko $FS = \{A, \mathrm{not}\,C\}$ on yhteensopiva ohjelman P stabiilien mallien $M_1 = \{A, D, E\}$ ja $M_2 = \{A, D, F\}$ kanssa.

- Approksimointi perustuu stabiilien mallien ala- ja ylärajaan:
  1. *Alaraja* $\mathrm{LB}(P, FS) \supseteq FS$ on literaalijoukko, joka on yhteensopiva jokaisen literaalijoukon FS kanssa yhteensopivan ohjelman P stabiilin mallin $M \subseteq \mathrm{HB}(P)$ kanssa.
  2. *Yläraja* $\mathrm{UB}(P, FS) \subseteq \mathrm{HB}(P)$ on atomilauseiden joukko, joka sisältää jokaisen literaalijoukon FS kanssa yhteensopivan ohjelman P stabiilin mallin $M \subseteq \mathrm{HB}(P)$.
- Approksimaatio **expand**(P, FS) on pienin joukko FS', joka sisältää FS:n ja on suljettu seuraavien sääntöjen suhteen:
  (i) Jos literaali $L \in \mathrm{LB}(P, FS')$, niin $L \in FS'$.
  (ii) Jos atomi $A \notin \mathrm{UB}(P, FS')$, niin $\mathrm{not}\, A \in FS'$.
- Approksimaatio **expand**(P, FS) saadaan laskemalla ala- ja ylärajoja toistuvasti ja täydentämällä literaalijoukkoa FS (kunnes saavutetaan kiintopiste periaatteiden (i) ja (ii) suhteen).

4. Approksimaatioiden toteuttaminen

- Tavoitteena on, että alaraja $\mathrm{LB}(P, FS)$ ja yläraja $\mathrm{UB}(P, FS)$ voidaan laskea lineaarisessa ajassa.
- Vähintään yhtä tarkat rajat kuin WF-mallilla.
- Hakutilanne (osittaismalli FS) hyödynnettävä tehokkaasti.
- Oletuksilla FS vielä (mahdollisesti) käytettävissä olevat säännöt:

$$P_{FS} = \{\, H \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \mathrm{not}\, B_{m+1} \wedge \cdots \wedge \mathrm{not}\, B_n \in P \mid FS \cap \{\mathrm{not}\, B_1, \ldots, \mathrm{not}\, B_m, B_{m+1}, \ldots, B_n\} = \emptyset \,\}.$$

- Määritellään literaalin L komplementti $\overline{L}$ normaaliin tapaan:

$$\overline{A} = \mathrm{not}\, A \text{ ja } \overline{\mathrm{not}\, A} = A.$$

- Nyt $P_{FS} = \{H \leftarrow L_1 \wedge \cdots \wedge L_n \in P \mid FS \cap \{\overline{L_1}, \ldots, \overline{L_n}\} = \emptyset\}$.

Alaraja

Määritelmä. Alaraja $\mathrm{LB}(P, FS)$ on pienin joukko FS', joka sisältää joukon FS ja on suljettu seuraavien periaatteiden suhteen:

S1: Jos $H \leftarrow L_1 \wedge \cdots \wedge L_n \in P_{FS'}$ ja $\{L_1, \ldots, L_n\} \subseteq FS'$, $H \in FS'$.

S2: Jos atomi H ei esiinny minkään ohjelman $P_{FS'}$ säännön seurauksena, niin $\mathrm{not}\, H \in FS'$.

S3: Jos $H \in FS'$ on täsmälleen yhden säännön $H \leftarrow L_1 \wedge \cdots \wedge L_n \in P_{FS'}$ seurauksena, $\{L_1, \ldots, L_n\} \subseteq FS'$.

S4: Jos $\mathrm{not}\, H \in FS'$, sääntö $H \leftarrow L_1 \wedge \cdots \wedge L_n \in P_{FS'}$ ja $\{L_1, \ldots, L_{i-i}, L_{i+1}, \ldots, L_n\} \subseteq FS'$, niin $\overline{L_i} \in FS'$.

S5: Jos jollekin atomille H sekä $H \in FS'$ että $\mathrm{not}\, H \in FS'$, niin kaikki literaalit kuuluvat joukkoon FS'.

Esimerkki. Tarkastellaan ohjelmaa

$$P_1 = \{\, A \leftarrow \mathrm{not}\, B \text{ (R1)},\ C \leftarrow A \text{ (R2)},\ B \leftarrow A \wedge \mathrm{not}\, C \wedge \mathrm{not}\, D \text{ (R3)},\ D \leftarrow C \wedge \mathrm{not}\, E \text{ (R4)} \,\}.$$

- Alarajan $\mathrm{LB}(P_1, \{\mathrm{not}\, B\})$ laskenta:

Literaalit	Periaate	Käytettävissä olevat säännöt
$\mathrm{not}\, B$	oletus	R1, R2, R3, R4
A	S1	R1, R2, R3, R4
C	S1	R1, R2, R4
$\mathrm{not}\, E$	S2	R1, R2, R4
D	S1	R1, R2, R4

- Approksimaatio **expand**$(P_1, \{\mathrm{not}\, B\}) = \{\mathrm{not}\, B, A, C, \mathrm{not}\, E, D\}$ määrää jo stabiilin mallin $\{A, C, D\}$ (johon atomi B ei kuulu).

Esimerkki. Tarkastellaan edelleen ohjelmaa

$$P_1 = \{\ A \leftarrow \text{not}\, B\ \text{(R1)},\ C \leftarrow A\ \text{(R2)},$$
$$B \leftarrow A \wedge \text{not}\, C \wedge \text{not}\, D\ \text{(R3)},\ D \leftarrow C \wedge \text{not}\, E\ \text{(R4)}\ \}.$$

- Alarajan $\mathrm{LB}(P_1, \{B\})$ laskenta:

Literaalit	Periaate	Käytettävissä olevat säännöt
B	oletus	R2, R3, R4
$A, \text{not}\, C, \text{not}\, D$	S3	R2, R3
C	S1	R2
kaikki literaalit	S5	

- Täten myös **expand**$(P_1, \{B\})$ antaa kaikki literaalit
  ☞ ohjelmalla P_1 ei ole stabiilia mallia M siten, että $B \in M$.

Yläraja

Määritelmä. Yläraja $\mathrm{UB}(P, FS)$ on pienin mallin ohjelmalle P', joka saadaan poistamalla not-literaalit ohjelmasta P_{FS}.

Esimerkki. Tarkastellaan ohjelmaa

$$P_2 = \{\ A \leftarrow \text{not}\, B,\ B \leftarrow \text{not}\, A,\ C \leftarrow \text{not}\, A,$$
$$D \leftarrow \text{not}\, C,\ E \leftarrow \text{not}\, D\ \}$$

Lasketaan ohjelmalle P_2 esim. seuraavat ylärajat:

1. $\mathrm{UB}(P_2, \emptyset) = \{A, B, C, D, E\}$
   (koska lisäksi myös $\mathrm{LB}(P_2, \emptyset) = \emptyset$, saadaan **expand**$(P_2, \emptyset) = \emptyset$).
2. $\mathrm{UB}(P_2, \{A\}) = \{A, D, E\}$.
3. $\mathrm{UB}(P_2, \{\text{not}\, A\}) = \{A, B, C, D, E\}$.

Esimerkki. Lasketaan samaiselle ohjelmalle

$$P_2 = \{\ A \leftarrow \text{not}\, B,\ B \leftarrow \text{not}\, A,\ C \leftarrow \text{not}\, A,$$
$$D \leftarrow \text{not}\, C,\ E \leftarrow \text{not}\, D\ \}$$

approksimaatiot **expand**$(P_2, \{A\})$ ja **expand**$(P_2, \{\text{not}\, A\})$.

1. $\mathrm{LB}(P_2, \{A\}) = \{A, \text{not}\, B, \text{not}\, C, D, \text{not}\, E\} =$ **expand**$(P_2, \{A\})$.
2. $\mathrm{LB}(P_2, \{\text{not}\, A\}) = \{\text{not}\, A, B, C, \text{not}\, D, E\} =$ **expand**$(P_2, \{\text{not}\, A\})$.

Esimerkki. Ohjelmalle $P_3 = \{A \leftarrow \text{not}\, B, B \leftarrow B\}$ alaraja $\mathrm{LB}(P_3, \emptyset) = \emptyset$, mutta yläraja $\mathrm{UB}(P_3, \emptyset) = \{A\}$.

Näin voidaan päätellä not B (kts. **expand**). Tästä seuraa, että **expand**$(P_3, \emptyset) = \{A, \text{not}\, B\}$, koska $\mathrm{LB}(P_3, \{\text{not}\, B\}) = \{A, \text{not}\, B\}$.

Ala- ja ylärajojen toteuttaminen

- Ala- ja ylärajoille voidaan antaa lineaarisen ajan toteutukset (käyttämällä Dowling-Gallier -tyyppisiä tietorakenteet).
- Tarkkuus on vähintään sama kuin WF-mallilla: **expand**$(P, \emptyset)$ antaa ohjelman P WF-mallin.
- Hakutilanne (osittaismalli FS) hyödynnetään dynaamisesti.

Esimerkki. Ohjelman $P = \{A \leftarrow \text{not}\, B,\ B \leftarrow \text{not}\, A,\ A \leftarrow B\}$ tapauksessa approksimaatioksi **expand**$(P, \{B\})$ saadaan $\{A, B, \text{not}\, A, \text{not}\, B\}$.

Ohjelman $P \cup \{B\}$ WF-malli on $\{A, B\}$ (konflikti ei tule esille)!

5. Hakualgoritmi stabiileille malleille

function smodels(P, FS, φ): boolean;
$FS' :=$ **expand**(P, FS);
if conflict(P, FS') returns true **then** return false
else if $\mathrm{NA}(P)$ is covered by FS' **then** return **test**(P, FS', φ)
else
  $\chi :=$ **choose**(P, FS');
  if smodels$(P, FS' \cup \{\chi\}, \varphi)$ returns true **then**
    return true
  else return **smodels**$(P, FS' \cup \{\overline{\chi}\}, \varphi)$
  end if
end if

Funktioiden tehtävät:

- Funktiota **test** käytetään löytyneiden mallien valintaan (parametri φ antaa valintakriteerit).
- Funktio **conflict**(P, FS) tarkistaa, sisältääkö saatu approksimaatio FS konfliktin ($\{A, \mathrm{not}\,A\} \subseteq FS$ jollekin $A \in \mathrm{HB}(P)$).
- Funktio **choose** toteuttaa hakuheuristiikan (paluuarvona literaali).
- Funktio **smodels** suorittaa *fokusoitua mallinetsintää*:

  smodels(P, FS, φ) palauttaa arvon true
  $\Longleftrightarrow$ $\exists$ P:n stabiili malli $M = \mathrm{Dcl}(P, \{\mathrm{not}\,A \mid \mathrm{not}\,A \in FS'\})$,
  joka on yhteensopiva FS:n kanssa ($FS \subseteq FS'$) ja
  jolle **test**(P, FS', φ) palauttaa arvon true.

Esimerkki. Tutkitaan, kuuluuko atomi E ohjelman P kaikkiin stabiileihin malleihin etsimällä vastamalli, johon E ei kuulu.

Aloitetaan osittaismallista $FS = \{\mathrm{not}\,E\}$
(valintafunktio **test** palauttaa aina arvon true).

$$
\begin{aligned}
P = \{\, & A \leftarrow \mathrm{not}\,B, \\
& B \leftarrow \mathrm{not}\,A, \\
& C \leftarrow A \wedge \mathrm{not}\,B, \\
& C \leftarrow B \wedge \mathrm{not}\,A, \\
& D \leftarrow \mathrm{not}\,C, \\
& E \leftarrow \mathrm{not}\,D \,\}
\end{aligned}
$$

not E	
D	
not C	
A	not A
not B	B
C	C
$\times$	$\times$

☞ Kyllä kuuluu.

Esimerkki. Etsitään ohjelman

$$
\begin{aligned}
P = \{\, & A \leftarrow C \wedge \mathrm{not}\,B, \\
& B \leftarrow \mathrm{not}\,A, \\
& C \leftarrow \mathrm{not}\,D, \\
& D \leftarrow \mathrm{not}\,A \,\}
\end{aligned}
$$

stabiilit mallit (alussa $FS = \emptyset$ ja **test** palauttaa aina true):

A	not A
not D	B
C	D
not B	not C

☞ Stabiilit mallit $M_1 = \mathrm{Dcl}(P, \{\mathrm{not}\,B, \mathrm{not}\,D\}) = \{A, C\}$ ja $M_2 = \mathrm{Dcl}(P, \{\mathrm{not}\,A, \mathrm{not}\,C\}) = \{B, D\}$.

Heuristiikka

- Lookahead-tekniikka: jokaiselle atomille $A \in \mathrm{NA}(P)$, johon osittaismalli FS ei ota kantaa ($A \notin FS$ ja $\mathrm{not}\, A \notin FS$), lasketaan $FS_1 = \textbf{expand}(P, FS \cup \{A\})$ ja $FS_2 = \textbf{expand}(P, FS \cup \{\mathrm{not}\, A\})$
- Saadaan tarkempi alaraja:
  Jos FS_1 sisältää konfliktin, lisätään $\mathrm{not}\, A$ joukkoon FS ja
  jos FS_2 sisältää konfliktin, lisätään A joukkoon FS.
- FS_1 and FS_2 antavat arvion jäljellä olevasta hakuavaruudesta.
- Heuristiikka: valitaan atomi, jolla on pienin arvioitu jäljellä oleva hakuavaruus.

Arviointia

- Logiikkaohjelmat ja stabiilit mallit ilmaisuvoimainen rajoiteohjelmointiparadigma.
  (Esim. Boolen rajoitteet/lauselogiikka erikoistapaus).
- Hakuavaruuden rakenne poikkeaa lauselogiikan toteutuvuusongelmasta:
  (i) Vain not-literaalit lisäävät hakuavaruutta.
  (ii) Säännöt suunnattuja: kerrostuneisuus rajaa hakuavaruutta.
  ☞ Enemmän vapautta tietämyksen esittämiseen
- Toteutustekniikat parantuneet ratkaisevasti: mm. edellä kuvatun algoritmin tilavaativuus on lineaarinen ohjelman kokoon nähden.

Oppimistavoitteet

- Tunnet Dowling-Gallier-algoritmin toimintaperiaatteen ja osaat tarvittaessa simuloida algoritmin toimintaa annetulle ohjelmalle.
- Osaat sovittaa Davis-Putnam-tyyppisen hakualgoritmin stabiilien mallien tapaukseen.
- Osaat hakea annetulle logiikkaohjelmalle stabiilit mallit.
- Tunnet joitain approksimointiperiaatteita ja pystyt hyödyntämään niitä vaihtoehtojen karsimiseen laskiessasi stabiileja malleja.