Handling Negation in General Deductive Databases: A Program Transformation Method

Weiling Li, Komal Khabya, Ming Fang and Raj Sunderraman

Georgia State University, Atlanta, GA

December 8, 2010

Outline

Introduction

- General deductive databases contain rules with arbitrary negation (negation-recursion) in their bodies.

```
move(1,2).
move(2,3).
move(3,2).
move(1,4).
win(X) :- move(X,Y), not win(Y).
```

- Two popular semantics
  - 3-valued well-founded models
  - 2-valued stable models
- We present a program transformation approach to compute (weak) well-founded model
- Our transformed program eliminates the complex "negation-recursion"
- We then use the (weak) well-founded model as a starting point to compute stable models

Some Deductive Database Terminology

- A *term* is either a variable or a constant.
- An *atom* is of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol and the t_i's are terms.
- A *literal* is either a *positive literal* A or a *negative literal* $\neg A$, where A is an atom.

Definition

A *general deductive database* is a finite set of clauses of the form: $a \leftarrow l_1, l_2, \ldots, l_m$.

Terminology continued...

- A term, atom, literal, or clause is called *ground* if it contains no variables.
- A *ground instance* of a term, atom, literal, or clause Q is the term, atom, literal, or clause, respectively, obtained by replacing each variable in Q by a constant.
- P^* denotes the set of all ground instances of clauses of general deductive database P.
- The *Herbrand Base* of database P is the set of all ground atoms.
- Any subset of the Herbrand Base is termed a *Herbrand interpretation* (atoms in the interpretation are assumed to be true and those outside the interpretation are assumed to be false).
- A Herbrand interpretation is a *model* of the database if all the facts and rules evaluate to true in the interpretation.
- A model is a *minimal model* if none of its proper subsets is a model.

The (weak) well-founded semantics (Fitting model)

- Fitting introduced a semantics for general deductive databases (also called the **weak well-founded semantics**)
- The Fitting semantics is a three-valued semantics
- Fitting was the first to define a semantics that assigned a unique least (partial) model to general deductive databases
- The Fitting semantics is based on **partial interpretations**

Definition

A partial interpretation is a pair $I = \langle I^+, I^- \rangle$, where I^+ and I^- are any subsets of the Herbrand base.

The Fitting Model

Definition

Let I be a partial interpretation and P be a general dedcutive database. Then $T_P^F(I)$ is the partial interpretation given by

$$
\begin{aligned}
T_P^F(I)^+ &= \{a \mid \text{ for some clause } a \leftarrow l_1, l_2, \ldots, l_m \in P^*, \text{ for each } \\
&\quad 1 \leq i \leq m \\
&\quad \text{ if } l_i \text{ is positive } l_i \in I^+ \text{ and,} \\
&\quad \text{ if } l_i \text{ is negative } l_i' \in I^- \} \\
T_P^F(I)^- &= \{a \mid \text{ for every clause } a \leftarrow l_1, l_2, \ldots, l_m \in P^*, \text{ there is some } \\
&\quad 1 \leq i \leq m \\
&\quad \text{ if } l_i \text{ is positive } l_i \in I^- \text{ and,} \\
&\quad \text{ if } l_i \text{ is negative } l_i' \in I^+ \}
\end{aligned}
$$

where l_i' is the complement of the literal l_i.

The least fixed point (lfp) of the above operator is the meaning of P.

Example: Fitting model

Let P be the following general deductive database:

```
move(1,2).
move(2,3).
move(3,2).
move(1,4).
win(X) :- move(X,Y), not win(Y).
```

We start with the empty partial interpretation: $\langle \emptyset, \emptyset \rangle$.Then,

Iteration	I^+	I^-
1	move(1,2), move(2,3), move(3,2), move(1,4)	move(1,1), move(1,3), move(2,1), move(2,2), move(2,4), move(3,1) move(3,3), move(3,4), move(4,1) move(4,2), move(4,3), move(4,4)
2		win(4)
3	win(1)	

Note that in the Fitting model the atom `win(1)` is *true* and the atom `win(4)` is *false*. No truth value is assigned to the atom `win(2)` and `win(3)`.

Stable Model Semantics

- The stable model semantics is a two-valued model for general deductive databases.
- In general, there can be more than one stable model for a given general deductive database.
- Stable models have applications in database repairs as well as search problems.

Definition

For any set S of atoms from the Herbrand base of a general deductive database P, let P^S be the program obtained from P^* by deleting:

1. each rule with a negative literal **not** B_i in body with $B_i \in S$, and
2. all negative literals from bodies of remaining rules.

If S is a minimal model of P^S, then S is a stable model of P.

Example : Stable models

Consider program P:

```
p(1,2).
q(x) :- p(x,y), not q(y).
```

The set of constants (Herbrand Universe) is

$$\{1,2\}$$

The set of ground atoms (Herbrand Base) is

$$\{q(1),\ q(2),\ p(1,1),\ p(1,2),\ p(2,1),\ p(2,2)\}.$$

The following is P^*, the ground instances of the rules of P:

```
p(1,2).
q(1) :- p(1,1), not q(1).
q(1) :- p(1,2), not q(2).
q(2) :- p(2,1), not q(1).
q(2) :- p(2,2), not q(2).
```

Stable Models Example continued...

Let S_1={p(1,2),q(2)}. Then P^{S_1}:

```
p(1,2).
q(1) :- p(1,1), ~~not q(1).~~
~~q(1) :- p(1,2), not q(2).~~
q(2) :- p(2,1), ~~not q(1).~~
~~q(2) :- p(2,2), not q(2).~~
```

The minimal Herbrand model of this program is {p(1,2)}, which is different from S_1; thus S_1 is <u>not stable</u>.

Let S_2={p(1,2),q(1)}. In this case, P^{S_1} is

```
p(1,2).
q(1) :- p(1,2).
q(2) :- p(2,2).
```

The minimal Herbrand model of this program is {p(1,2), q(1)}, i.e., S_2. Hence S_2 is <u>stable</u>.

Stable Models - win example

The win-program:

```
move(1,2).
move(2,3).
move(3,2).
move(1,4).
win(X) :- move(X,Y), not win(Y).
```

has 2 stable models:

S_1 = { move(1,2),move(2,3),move(3,2),move(1,4), win(1),win(2) }

S_2 = { move(1,2),move(2,3),move(3,2),move(1,4), win(1),win(3) }

Note: In the Fitting model, win(2) and win(3) both were declared to be "unknown".

Program Transformation

- For each predicate p of P, we introduce two predicates pplus and pminus in the transformed general deductive database $tr(P)$.
- Transformation proceeds in 4 steps.

Example

```
%% Extensional Database
t0(1).
g(1,2,3).
g(2,5,4).
g(2,4,5).
g(5,3,6).
%% Intensional Database
t(Z) :- t0(Z). %% rule 1
t(Z) :- g(X,Y,Z), t(X). %% rule 2
t(Z) :- g(X,Y,Z), not t(Y). %% rule 3
```

Transformation Algorithm

Step 1: Domain Predicate: Introduce a unique unary predicate `dom`. For each constant symbol, a, present in P, output the fact: `dom(a).`

Example

```
dom(1).
dom(2).
dom(3).
dom(4).
dom(5).
dom(6).
```

Transformation Algorithm continued...

Step 2: Extensional Database:

For each fact `p(a1,...,an)` in the extensional database, output the fact:

```
pplus(a1,...,an).
```

For each predicate *p* with arity *k* in the extensional database, output the rule:

```
pminus(X1,...,Xk) :- dom(X1),...,dom(Xk), not
   pplus(X1,...,Xk).
```

Example

```
t0plus(1).
t0minus(X) :- dom(X), not t0plus(X).
gplus(1,2,3).
gplus(2,5,4).
gplus(2,4,5).
gplus(5,3,6).
gminus(X,Y,Z) :- dom(X),dom(Y),dom(Z), not
   gplus(X,Y,Z).
```

Transformation Algorithm continued...

Step 3: Intensional Database:

Consider a rule of the form:

```
p(W1,...,Wl) :- q1(X1),..., qn(Xn), not
    r1(Y1),..., not rm(Ym).
```

For each such rule, perform Steps 3a and 3b.

Step 3a. Output "plus" rule:

Output the following rule for pplus:

```
pplus(W1,...,Wl) :- q1plus(X1),...,qnplus(Xn),
    r1minus(Y1),...,rmminus(Ym).
```

Example

```
tplus(Z) :- t0plus(Z).
tplus(Z) :- gplus(X,Y,Z), tplus(X).
tplus(Z) :- gplus(X,Y,Z), tminus(Y).
```

Transformation Algorithm continued...

Step 3b. Output temporary "minus" rules (j: rule number in P)

Step 3b-1:

For each positive subgoal in rule, `qi(Xi)`, output:

```
temp_p_j(V1,...,Vk) :- dom(U1),..., dom(Ua),
   qiminus(Xi).
```

Step 3b-2:

For each negative subgoal in rule, `not ri(Yi)`, output:

```
temp_p_j(V1,...,Vk) :- dom(U1),..., dom(Ua),
   riplus(Yi).
```

Note: V1,...,Vk are variables in body and U1,...Ua are variables present in the body that are not present in the subgoal.

Step 3b-3:

Output the following two rules:

```
temp_p_j_2(W1,...,Wl) :- dom(V1),..., dom(Vk),
   not temp_p_j(V1,...,Vk).
pminus_j(W1,...,Wl) :- dom(W1),..., dom(Wl), not
   temp_p_j_2(W1,...,Wl).
```

Transformation Algorithm continued...

Example

```
%% rule 1:  t(Z) :- t0(Z).
temp_t_1(Z) :- t0minus(Z).
temp_t_1_2(Z) :- dom(Z), not temp_t_1(Z).
tminus_1(Z) :- dom(Z), not temp_t_1_2(Z).

%% rule 2:  t(Z) :- g(X,Y,Z), t(X).
temp_t_2(X,Y,Z) :- gminus(X,Y,Z).
temp_t_2(X,Y,Z) :- dom(Y), dom(Z), tminus(X).
temp_t_2_2(Z) :- dom(X), dom(Y), dom(Z), not
   temp_t_2(X,Y,Z).
tminus_2(Z) :- dom(Z), not temp_t_2_2(Z).

%% rule 3:  t(Z) :- g(X,Y,Z), not t(Y).
temp_t_3(X,Y,Z) :- gminus(X,Y,Z).
temp_t_3(X,Y,Z) :- dom(X), dom(Z), tplus(Y).
temp_t_3_2(Z) :- dom(X), dom(Y), dom(Z), not
   temp_t_3(X,Y,Z).
tminus_3(Z) :- dom(Z), not temp_t_3_2(Z).
```

Transformation Algorithm continued...

Step 4. Output "minus" rules:

For each IDB predicate p defined in rules numbered i1,...,in, output the following rule:

```
pminus(W1,...,Wl) :- dom(W1),...,dom(Wl),
   pminus_i1(W1,...,Wl),...,
   pminus_in(W1,...,Wl).
```

Example

```
tminus(Z) :- dom(Z), tminus_1(Z), tminus_2(Z),
   tminus_3(Z).
```

Transformation Algorithm continued...

- A bottom-up evaluation of the output program produces:

  `{ tplus(1), tplus(3), tminus(2) }`

- We introduce unknown values via rules of the form:

```
punknown(X1,...,Xk) :- dom(X1),...,
    dom(Xk), not pplus(X1,...,Xk), not
    pminus(X1,...,Xk).
```

  for each *IDB* predicate.

- For the example, the following "unknown" rule is generated:

```
tunknown(Z) :- dom(Z), not tplus(Z), not
    tminus(Z).
```

  A bottom-up evaluation of the output program produces:

  `{ tunknown(4), tunknown(5), tunknown(6) }`

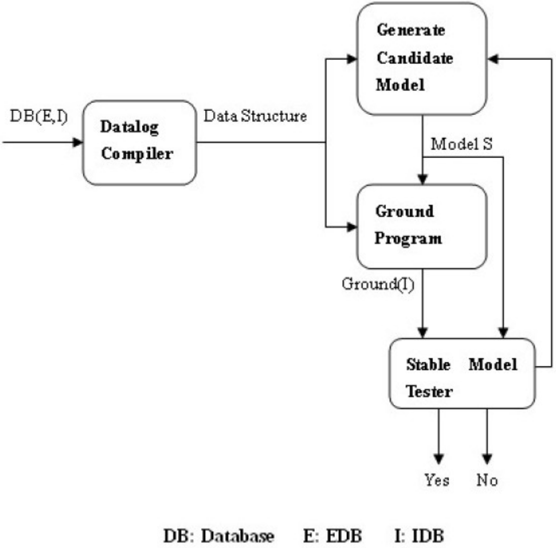
Correctness of Algorithm

Theorem

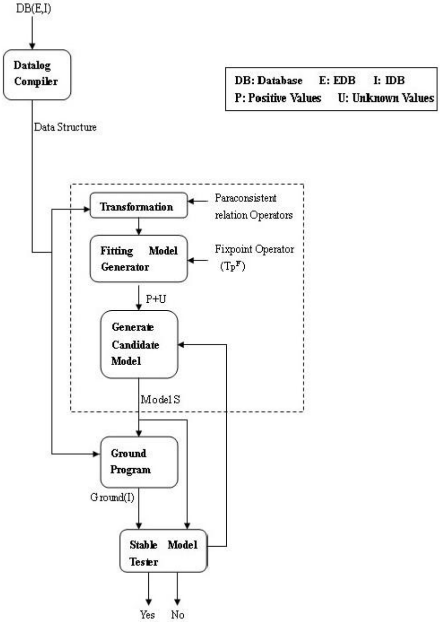
Let P be a general deductive database and let $tr(P)$ be the output of the transformation algorithm. Then,

- *$tr(P)$ has a complete well-founded model.*
- *$p(a1, ..., an)$ belongs to the positive component of the Fitting model of P if and only if $pplus(a1, ..., an)$ belongs to the well-founded model of $tr(P)$.*
- *$p(a1, ..., an)$ belongs to the negative component of the Fitting model of P if and only if $pminus(a1, ..., an)$ belongs to the well-founded model of $tr(P)$.*

Computing Stable Models: Naive approach



Computing Stable Models: Our approach



Experiments

- Database with varying EDBs:

```
%%generate EDB facts of t0
%%generate EDB facts of g
t(Z) :- t0(Z).
t(Z) :- g(X,Y,Z), t(X).
t(Z) :- g(X,Y,Z), not t(Y).
```

- Facts in the EDB are randomly generated from constant values.
- We vary the following parameters:
  - number of constants (#constants).
  - size of EDB (#facts = number of t0_facts + number of g facts).
- The above two parameters can be used as measures of "problem size" in graph problems; e.g. constants = nodes, facts = edges; node(1), node(2),... edge(1,2), edge(1,3),...

Experiments continued...

- Intelligent Grounding: technique used to reduce size of ground program
- 2 versions of our approach:
  - V1.0 - without intelligent grounding
  - V1.1 - with intelligent grounding

Experiment 1

Vary the number of constants present in the program (with fixed size of EDB).

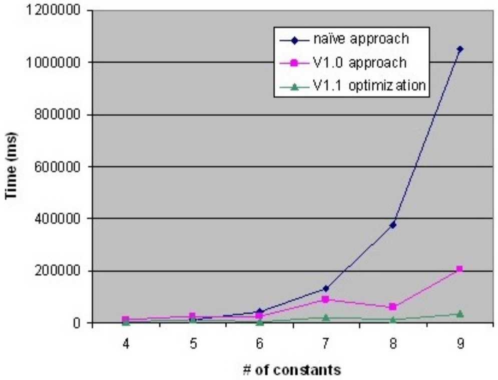


Figure: Vary number of constants

Experiment 2

Vary the size of EDB (with fixed number of constants).

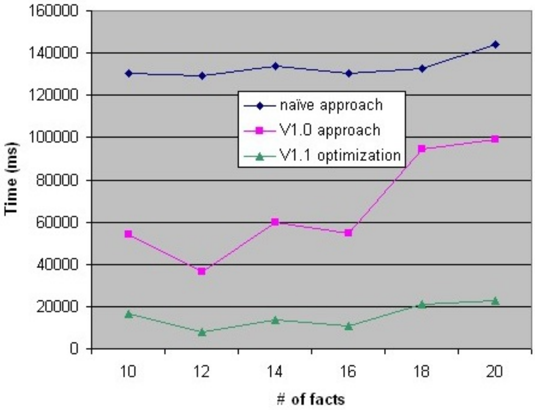


Figure: Vary number of facts

Concluding Remarks

- Program transformation method introduced to compute well-founded model
- Transformed program has many desirable properties including the amenability to traditional bottom-up computation.
- Future Work:
  - Compare with "alternating fixed point" and other approaches to compute stable models.
  - Program transformation to detect "positive loops" to compute well-founded model
  - Applications - graph problems