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
Handling Negation in General Deductive Databases: A Program Transformation Method

INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

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

\[
\begin{align*}
\text{move}(1,2). \\
\text{move}(2,3). \\
\text{move}(3,2). \\
\text{move}(1,4). \\
\text{win}(X) & :{-} \text{move}(X,Y), \text{not win}(Y).
\end{align*}
\]

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: $a \leftarrow l_1, l_2, \ldots, l_m$. 
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.
Handling Negation in General Deductive Databases: A Program Transformation Method

INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

CONCLUSION

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.
Definition

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

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

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

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:

\[
\begin{align*}
\text{move}(1,2). \\
\text{move}(2,3). \\
\text{move}(3,2). \\
\text{move}(1,4). \\
\text{win}(X) & : \neg \text{move}(X,Y), \neg \text{win}(Y).
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$I^+$</th>
<th>$I^-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>move(1,2), move(2,3), move(3,2), move(1,4)</td>
<td>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)</td>
</tr>
<tr>
<td>2</td>
<td>win(1)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>win(4)</td>
</tr>
</tbody>
</table>

Note that in the Fitting model the atom $\text{win}(1)$ is $true$ and the atom $\text{win}(4)$ is $false$. No truth value is assigned to the atom $\text{win}(2)$ and $\text{win}(3)$. 
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 $\textbf{not} \ B_i$ 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 \( P \):

\[
p(1,2).
q(x) :- p(x,y), \text{not } q(y).
\]

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 \( P \):

\[
p(1,2).
q(1) :- p(1,1), \text{not } q(1).
q(1) :- p(1,2), \text{not } q(2).
q(2) :- p(2,1), \text{not } q(1).
q(2) :- p(2,2), \text{not } q(2).
\]
Let $S_1=\{p(1,2), q(2)\}$. Then $P^{S_1}$:

\[
p(1,2).
\]
\[
q(1) :\neg p(1,1), \neg q(1).
\]
\[
q(1) :\neg p(2,1), \neg q(2).
\]
\[
q(2) :\neg p(2,2), \neg q(2).
\]

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

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 stable.
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:

\[ S_1 = \{ \text{move}(1,2), \text{move}(2,3), \text{move}(3,2), \text{move}(1,4), \]
\[ \quad \text{win}(1), \text{win}(2) \} \]

\[ S_2 = \{ \text{move}(1,2), \text{move}(2,3), \text{move}(3,2), \text{move}(1,4), \]
\[ \quad \text{win}(1), \text{win}(3) \} \]

Note: In the Fitting model, \text{win}(2) and \text{win}(3) both were declared to be "unknown".
Program Transformation

- For each predicate $p$ of $P$, we introduce two predicates $p_{plus}$ and $p_{minus}$ 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
```
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 $p(a_1,\ldots,a_n)$ in the extensional database, output the fact:

$$pplus(a_1,\ldots,a_n).$$

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

$$pminus(X_1,\ldots,X_k) :- \text{dom}(X_1),\ldots,\text{dom}(X_k), \text{not } pplus(X_1,\ldots,X_k).$$

**Example**

```
t0plus(1).
t0minus(X) :- \text{dom}(X), \text{not } t0plus(X).
gplus(1,2,3).
gplus(2,5,4).
gplus(2,4,5).
gplus(5,3,6).
gminus(X,Y,Z) :- \text{dom}(X),\text{dom}(Y),\text{dom}(Z), \text{not } gplus(X,Y,Z).
```
Step 3: Intensional Database:
Consider a rule of the form:

\[ p(W_1, \ldots, W_l) :- q_1(X_1), \ldots, q_n(X_n), \text{not } r_1(Y_1), \ldots, \text{not } r_m(Y_m). \]

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

Step 3a. Output “plus” rule:
Output the following rule for \( p_{\text{plus}} \):

\[ p_{\text{plus}}(W_1, \ldots, W_l) :- q_{1\text{plus}}(X_1), \ldots, q_{n\text{plus}}(X_n), r_{1\text{minus}}(Y_1), \ldots, r_{m\text{minus}}(Y_m). \]

Example

\[
\begin{align*}
t_{\text{plus}}(Z) & :- t_{0\text{plus}}(Z). \\
t_{\text{plus}}(Z) & :- g_{\text{plus}}(X,Y,Z), t_{\text{plus}}(X). \\
t_{\text{plus}}(Z) & :- g_{\text{plus}}(X,Y,Z), t_{\text{minus}}(Y).
\end{align*}
\]
Transformation Algorithm continued...

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

Step 3b-1:
For each positive subgoal in rule, $q_i(X_i)$, output:

$$temp_{p_j}(V_1, \ldots, V_k) :- \text{dom}(U_1), \ldots, \text{dom}(U_a), q_{iminus}(X_i).$$

Step 3b-2:
For each negative subgoal in rule, not $r_i(Y_i)$, output:

$$temp_{p_j}(V_1, \ldots, V_k) :- \text{dom}(U_1), \ldots, \text{dom}(U_a), r_{iplus}(Y_i).$$

Note: $V_1, \ldots, V_k$ are variables in body and $U_1, \ldots, U_a$ are variables present in the body that are not present in the subgoal.

Step 3b-3:
Output the following two rules:

$$temp_{p_j \ 2}(W_1, \ldots, W_l) :- \text{dom}(V_1), \ldots, \text{dom}(V_k),$$
$$\text{not } temp_{p_j}(V_1, \ldots, V_k).$$

$$p_{minus_j}(W_1, \ldots, W_l) :- \text{dom}(W_1), \ldots, \text{dom}(W_l), \text{not } temp_{p_j \ 2}(W_1, \ldots, W_l).$$
Example

%% rule 1: $t(Z) :- t0(Z)$.
$\text{temp}_{t.1}(Z) :- t0\text{-}minus(Z)$.  
$\text{temp}_{t.1.2}(Z) :- \text{dom}(Z), \text{not}\ \text{temp}_{t.1}(Z)$.  
$t\text{-}minus_{1}(Z) :- \text{dom}(Z), \text{not}\ \text{temp}_{t.1.2}(Z)$.  

%% rule 2: $t(Z) :- g(X,Y,Z), t(X)$.  
$\text{temp}_{t.2}(X,Y,Z) :- g\text{-}minus(X,Y,Z)$.  
$\text{temp}_{t.2}(X,Y,Z) :- \text{dom}(Y), \text{dom}(Z), t\text{-}minus(X)$.  
$\text{temp}_{t.2.2}(Z) :- \text{dom}(X), \text{dom}(Y), \text{dom}(Z), \text{not}\ \text{temp}_{t.2}(X,Y,Z)$.  
$t\text{-}minus_{2}(Z) :- \text{dom}(Z), \text{not}\ \text{temp}_{t.2.2}(Z)$.  

%% rule 3: $t(Z) :- g(X,Y,Z), \text{not} t(Y)$.  
$\text{temp}_{t.3}(X,Y,Z) :- g\text{-}minus(X,Y,Z)$.  
$\text{temp}_{t.3}(X,Y,Z) :- \text{dom}(X), \text{dom}(Z), t\text{+}plus(Y)$.  
$\text{temp}_{t.3.2}(Z) :- \text{dom}(X), \text{dom}(Y), \text{dom}(Z), \text{not}\ \text{temp}_{t.3}(X,Y,Z)$.  
$t\text{-}minus_{3}(Z) :- \text{dom}(Z), \text{not}\ \text{temp}_{t.3.2}(Z)$.
Transformation Algorithm continued...

Step 4. Output “minus” rules:
For each IDB predicate $p$ defined in rules numbered $i_1, \ldots, i_n$, output the following rule:

$$p_{\text{minus}}(W_1, \ldots, W_l) :- \text{dom}(W_1), \ldots, \text{dom}(W_l),$$

$$p_{\text{minus}_{i_1}}(W_1, \ldots, W_l), \ldots,$$

$$p_{\text{minus}_{i_n}}(W_1, \ldots, W_l).$$

Example

$$t_{\text{minus}}(Z) :- \text{dom}(Z), t_{\text{minus}_{1}}(Z), t_{\text{minus}_{2}}(Z),$$

$$t_{\text{minus}_{3}}(Z).$$
A bottom-up evaluation of the output program produces:
\{ tplus(1), tplus(3), tminus(2) \}

We introduce unknown values via rules of the form:
\[
punknown(X_1, \ldots, X_k) :- \text{dom}(X_1), \ldots, \\
\text{dom}(X_k), \text{not } pplus(X_1, \ldots, X_k), \text{not } pminus(X_1, \ldots, X_k).
\]

for each IDB predicate.

For the example, the following "unknown" rule is generated:
\[
tunknown(Z) :- \text{dom}(Z), \text{not } tplus(Z), \text{not } tminus(Z).
\]

A bottom-up evaluation of the output program produces:
\[
\text{\{ tunknown(4), tunknown(5), tunknown(6) \}}
\]
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.
- $p(a_1, ..., a_n)$ belongs to the positive component of the Fitting model of $P$ if and only if $pplus(a_1, ..., a_n)$ belongs to the well-founded model of $tr(P)$.
- $p(a_1, ..., a_n)$ belongs to the negative component of the Fitting model of $P$ if and only if $pminus(a_1, ..., a_n)$ belongs to the well-founded model of $tr(P)$. 

Theorem
Handling Negation in General Deductive Databases: A Program Transformation Method

Weiling Li, Komal Khabya, Ming Fang and Raj Sunderraman

INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

CONCLUSION

Computing Stable Models: Naive approach
Computing Stable Models: Our approach

DB: Database  E: EDB  I: IDB
P: Positive Values  U: Unknown Values
Experiments

- Database with varying EDBs:
  
  ```prolog
  %%%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),...
• 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
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