INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

CONCLUSION

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
INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

CONCLUSION
Problem Statement

- 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: $a \leftarrow l_1, l_2, \ldots, l_m$. 
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.
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 deductive database. Then $T^F_P(I)$ is the partial interpretation given by

$$
T^F_P(I)^+ = \{ a | \text{ for some clause } a \leftarrow l_1, l_2, \ldots, l_m \in P^*, \text{ for each } 1 \leq i \leq m
$$

if $l_i$ is positive $l_i \in I^+$ and,

if $l_i$ is negative $l'_i \in I^-$

$$
T^F_P(I)^- = \{ a | \text{ for every clause } a \leftarrow l_1, l_2, \ldots, l_m \in P^*, \text{ there is some } 1 \leq i \leq m
$$

if $l_i$ is positive $l_i \in I^-$ and,

if $l_i$ 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 $P$. 

59x204}
Example: Fitting model

Let $P$ be the following general deductive database:

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

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

\begin{table}
\begin{tabular}{|c|c|c|}
\hline
Iteration & $I^+$ & $I^-$ \\
\hline
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) & \\
\hline
\end{tabular}
\end{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 $\text{win}(3)$.
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 \( \text{not } B_i \) 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 $P$:

\[
\begin{align*}
& p(1,2).
& q(x) := p(x,y), \text{ not } q(y).
\end{align*}
\]

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 $P$:

\[
\begin{align*}
& p(1,2).
& q(1) := p(1,1), \text{ not } q(1).
& q(1) := p(1,2), \text{ not } q(2).
& q(2) := p(2,1), \text{ not } q(1).
& q(2) := p(2,2), \text{ not } q(2).
\end{align*}
\]
Let $S_1 = \{ p(1,2), q(2) \}$. Then $P^{S_1}$:

\[
\begin{align*}
    p(1,2). \\
    q(1) & :\ p(1,1), \ \text{not} \ q(1). \\
    q(1) & :\ p(1,2), \ \text{not} \ q(2). \\
    q(2) & :\ p(2,1), \ \text{not} \ q(1). \\
    q(2) & :\ p(2,2), \ \text{not} \ q(2). \\
\end{align*}
\]

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

\[
\begin{align*}
    p(1,2). \\
    q(1) & :\ p(1,2). \\
    q(2) & :\ p(2,2). \\
\end{align*}
\]

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:

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

has 2 stable models:

\[
S_1 = \{ \text{move}(1,2), \text{move}(2,3), \text{move}(3,2), \text{move}(1,4), \text{win}(1), \text{win}(2) \} \\
S_2 = \{ \text{move}(1,2), \text{move}(2,3), \text{move}(3,2), \text{move}(1,4), \text{win}(1), \text{win}(3) \}
\]

Note: In the Fitting model, win(2) and win(3) both were declared to be "unknown".
For each predicate \( p \) of \( P \), we introduce two predicates \( p_{\text{plus}} \) and \( p_{\text{minus}} \) in the transformed general deductive database \( tr(P) \).

Transformation proceeds in 4 steps.

**Example**

<table>
<thead>
<tr>
<th>%% Extensional Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0(1).</td>
</tr>
<tr>
<td>g(1,2,3).</td>
</tr>
<tr>
<td>g(2,5,4).</td>
</tr>
<tr>
<td>g(2,4,5).</td>
</tr>
<tr>
<td>g(5,3,6).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>%% Intensional Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>t(Z) :- t0(Z).  ( \text{%% rule 1} )</td>
</tr>
<tr>
<td>t(Z) :- g(X,Y,Z), t(X). ( \text{%% rule 2} )</td>
</tr>
<tr>
<td>t(Z) :- g(X,Y,Z), not t(Y). ( \text{%% rule 3} )</td>
</tr>
</tbody>
</table>
Step 1: Domain Predicate: Introduce a unique unary predicate \textit{dom}. For each constant symbol, \( a \), present in \( P \), output the fact: \textit{dom}(a).

Example

\begin{verbatim}
  dom(1).
  dom(2).
  dom(3).
  dom(4).
  dom(5).
  dom(6).
\end{verbatim}
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**

$\text{t0plus}(1).$
$\text{t0minus}(X) :- \text{dom}(X), \text{not} \ \text{t0plus}(X).$
$\text{gplus}(1,2,3).$
$\text{gplus}(2,5,4).$
$\text{gplus}(2,4,5).$
$\text{gplus}(5,3,6).$
$\text{gminus}(X,Y,Z) :- \text{dom}(X),\text{dom}(Y),\text{dom}(Z), \text{not} \ \text{gplus}(X,Y,Z).$
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
\[
\begin{align*}
\text{tplus}(Z) & :- \text{t0plus}(Z). \\
\text{tplus}(Z) & :- \text{gplus}(X,Y,Z), \text{tplus}(X). \\
\text{tplus}(Z) & :- \text{gplus}(X,Y,Z), \text{tminus}(Y).
\end{align*}
\]
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:

\[
temp_{p_j}(V_1, \ldots, V_k) :- \text{dom}(U_1), \ldots, \text{dom}(U_a), q_i^{\text{minus}}(X_i).
\]

**Step 3b-2:**
For each negative subgoal in rule, \( \neg r_i(Y_i) \), output:

\[
temp_{p_j}(V_1, \ldots, V_k) :- \text{dom}(U_1), \ldots, \text{dom}(U_a), r_i^{\text{plus}}(Y_i).
\]

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:

\[
temp_{p_j^{\text{2}}}(W_1, \ldots, W_l) :- \text{dom}(V_1), \ldots, \text{dom}(V_k), \neg temp_{p_j}(V_1, \ldots, V_k).
\]
\[
p_{\text{minus}}^{\text{j}}(W_1, \ldots, W_l) :- \text{dom}(W_1), \ldots, \text{dom}(W_l), \neg temp_{p_j^{\text{2}}}(W_1, \ldots, W_l).
\]
Example

%% rule 1: t(Z) :- t0(Z).

\[
\text{temp}_t.1(Z) :- t0\text{minus}(Z).
\]

\[
\text{temp}_t.1.2(Z) :- \text{dom}(Z), \text{not temp}_t.1(Z).
\]

\[
\text{tminus}_1(Z) :- \text{dom}(Z), \text{not temp}_t.1.2(Z).
\]

%% rule 2: t(Z) :- g(X,Y,Z), t(X).

\[
\text{temp}_t.2(X,Y,Z) :- g\text{minus}(X,Y,Z).
\]

\[
\text{temp}_t.2(X,Y,Z) :- \text{dom}(Y), \text{dom}(Z), \text{tminus}(X).
\]

\[
\text{temp}_t.2.2(Z) :- \text{dom}(X), \text{dom}(Y), \text{dom}(Z), \text{not temp}_t.2(X,Y,Z).
\]

\[
\text{tminus}_2(Z) :- \text{dom}(Z), \text{not temp}_t.2.2(Z).
\]

%% rule 3: t(Z) :- g(X,Y,Z), \text{not} t(Y).

\[
\text{temp}_t.3(X,Y,Z) :- g\text{minus}(X,Y,Z).
\]

\[
\text{temp}_t.3(X,Y,Z) :- \text{dom}(X), \text{dom}(Z), \text{tplus}(Y).
\]

\[
\text{temp}_t.3.2(Z) :- \text{dom}(X), \text{dom}(Y), \text{dom}(Z), \text{not temp}_t.3(X,Y,Z).
\]

\[
\text{tminus}_3(Z) :- \text{dom}(Z), \text{not temp}_t.3.2(Z).
\]
Step 4. Output “minus” rules:

For each IDB predicate $p$ defined in rules numbered $i_1, \ldots, i_n$, output the following rule:

\[ p_{\text{minus}}(W_1, \ldots, W_l) :\neg \text{dom}(W_1), \ldots, \text{dom}(W_l), \]
\[ p_{\text{minus}i_1}(W_1, \ldots, W_l), \ldots, \]
\[ p_{\text{minus}i_n}(W_1, \ldots, W_l). \]

Example

\[ t_{\text{minus}}(Z) :\neg \text{dom}(Z), t_{\text{minus}1}(Z), t_{\text{minus}2}(Z), \]
\[ t_{\text{minus}3}(Z). \]
A bottom-up evaluation of the output program for the example database results in the following values for tplus and tminus:

\[
\begin{align*}
\{ & \text{tplus}(1), \text{tplus}(3) \} \\
\{ & \text{tminus}(2) \}
\end{align*}
\]

We introduce unknown values via rules of the form:

\[
\text{punknown}(X_1,\ldots,X_k) :- \text{dom}(X_1),\ldots, \\
\text{dom}(X_k), \neg \text{pplus}(X_1,\ldots,X_k), \neg \text{pminus}(X_1,\ldots,X_k).
\]

for each IDB predicate.

For the example, the following unknown rules are generated:

\[
\text{tunknown}(Z) :- \text{dom}(Z), \neg \text{tplus}(Z), \neg \text{tminus}(Z).
\]

The output program for the example database results in the following values for tunknown:

\[
\{ \text{tunknown}(4), \text{tunknown}(5), \text{tunknown}(6) \}
\]
Correctness of Algorithm

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

- $\text{tr}(P)$ has a complete well-founded model.
- $p(a_1, \ldots, a_n)$ belongs to the positive component of the Fitting model of $P$ if and only if $p_{\text{plus}}(a_1, \ldots, a_n)$ belongs to the well-founded model of $\text{tr}(P)$.
- $p(a_1, \ldots, a_n)$ belongs to the negative component of the Fitting model of $P$ if and only if $p_{\text{minus}}(a_1, \ldots, a_n)$ belongs to the well-founded model of $\text{tr}(P)$. 
Computing Stable Models: Naive approach

INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

CONCLUSION

Weiling Li, Komal Khabya, Ming Fang and Raj Sunderraman

Handling Negation in General Deductive Databases: A Program Transformation Method

DB: Database  E: EDB  I: IDB
Computing Stable Models: Our approach

INTRODUCTION

BACKGROUND

PROGRAM TRANSFORMATION ALGORITHM

STABLE MODEL COMPUTATION

EXPERIMENTS

CONCLUSION
Experiments

- We use the IDB from Example we discussed above with various EDBs as our logic program.

  ```
  %%generate EDB facts of t
  %%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).
  ```

- Note that the facts in the EDB would be generated randomly from constant values in the experiments. In the experiments we keep vary the following parameters:
  - number of constants (#constants).
  - size of EDB (#facts = the number of t0_facts (#t0_facts) + the number of g facts (#g_facts))
  - the percentage of minus values (minus%) in the total number of t values We use tables as well as graphs to show the results.
Given the IDB rules we keep \#t0\_facts fixed to 2 and \#g\_facts fixed to 10, and vary the number of constants present in the program in increments of 1, starting from 4 and going up to 9.

![Naive approach vs. our approach with variable number of constants](image)

**Figure**: Naive approach vs. our approach with variable number of constants
Experiment 2

Given the IDB rules we keep \#constants fixed to 7 and vary \#facts, in increments of 2, starting from 10 and going up to 20.

Figure: Naive approach vs. our approach with variable number of facts
Given the IDB rules we keep \#constants fixed to 10, \#t0\_facts fixed to 1, and \#g\_facts fixed to 15, then we check how the percent of minus values affects the running time.

**Figure:** Naive approach vs. our approach with different percentages of minus values
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

- Considering the example when explaining the Stable Models

**Example**

\[
\begin{align*}
p(1,2). \\
q(x) &\leftarrow p(x,y), \text{not } q(y).
\end{align*}
\]

- **Intelligent Grounding**
  Assume that the number of the set of constants is \( n \), and the number of constants is \( 2n \). Since the second rule has two different variables, its *Herbrand Instantiation* contains \( (2n)^2 \) ground instances of the second. While using intelligent grounding, it has only \( n \) ground instances of the second rule.
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

Redo Experiment 3 with intelligent grounding

<table>
<thead>
<tr>
<th>minus%</th>
<th>Naive Approach</th>
<th>Our V1.0 Approach</th>
<th>Our V1.1 optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2905656</td>
<td>1651750</td>
<td>121422</td>
</tr>
<tr>
<td>20</td>
<td>2906328</td>
<td>856078</td>
<td>91688</td>
</tr>
<tr>
<td>30</td>
<td>2932235</td>
<td>474890</td>
<td>50656</td>
</tr>
<tr>
<td>40</td>
<td>2898391</td>
<td>272187</td>
<td>40266</td>
</tr>
<tr>
<td>50</td>
<td>2878047</td>
<td>156671</td>
<td>24812</td>
</tr>
<tr>
<td>60</td>
<td>2905781</td>
<td>133188</td>
<td>22484</td>
</tr>
</tbody>
</table>

![Graph showing time vs. percentage of minus value](image-url)
Concluding Remarks

Weiling Li, Komal Khabya, Ming Fang and Raj Sunderraman

Handling Negation in General Deductive Databases: A Program Transformation Method

INTRODUCTION
BACKGROUND
PROGRAM TRANSFORMATION ALGORITHM
STABLE MODEL COMPUTATION
EXPERIMENTS
CONCLUSION