Stable Models Semantics and Answer Set Programming

CSE 505 – Computing with Logic
Stony Brook University

http://www.cs.stonybrook.edu/~cse505
General Logic Programs

• A program is a collection of rules of the form

\[ a \leftarrow a_1, \ldots, a_n, \text{not } a_{n+1}, \text{not } a_{n+k}. \]

• Let \( \Pi \) be a program and \( X \) be a set of atoms, by \( \Pi^X \) (Gelfond-Lifschitz transformation) we denote the positive program obtained from \( \text{ground}(\Pi) \) by:
  • Deleting from \( \text{ground}(\Pi) \) any rule \( a \leftarrow a_1, \ldots, a_n, \text{not } a_{n+1}, \text{not } a_{n+k}. \) for that \( \{a_{n+1}, \ldots, a_{n+k}\} \cap X \neq \emptyset \), i.e., the body of the rule contains a naf-atom not al and al belongs to \( X \); and
  • Removing all of the naf-atoms from the remaining rules.
General Logic Programs

- A set of atoms $X$ is called an answer set of a program $\Pi$ if $X$ is the minimal model of the program $\Pi^X$.

- Theorem: For every positive program $\Pi$, the minimal model of $\Pi$, $M_\Pi$, is also the unique answer set of $\Pi$.

- Example: Consider $\Pi_2 = \{a \leftarrow \neg b.\ b \leftarrow \neg a.\}$. We will show that its has two answer sets $\{a\}$ and $\{b\}$.

<table>
<thead>
<tr>
<th>$S_1 = \emptyset$</th>
<th>$S_2 = {a}$</th>
<th>$S_3 = {b}$</th>
<th>$S_4 = {a, b}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi_2^{S_1}$ :</td>
<td>$\Pi_2^{S_2}$ :</td>
<td>$\Pi_2^{S_3}$ :</td>
<td>$\Pi_2^{S_4}$ :</td>
</tr>
<tr>
<td>$a \leftarrow$</td>
<td>$a \leftarrow$</td>
<td>$b \leftarrow$</td>
<td></td>
</tr>
<tr>
<td>$b \leftarrow$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{\Pi_2}^{S_1} = {a, b}$</td>
<td>$M_{\Pi_2}^{S_2} = {a}$</td>
<td>$M_{\Pi_2}^{S_3} = {b}$</td>
<td>$M_{\Pi_2}^{S_4} = \emptyset$</td>
</tr>
<tr>
<td>$M_{\Pi_2}^{S_1} \neq S_1$</td>
<td>$M_{\Pi_2}^{S_2} = S_2$</td>
<td>$M_{\Pi_2}^{S_3} = S_3$</td>
<td>$M_{\Pi_2}^{S_4} \neq S_4$</td>
</tr>
<tr>
<td>$NO$</td>
<td>$YES$</td>
<td>$YES$</td>
<td>$NO$</td>
</tr>
</tbody>
</table>

(c) Paul Fodor (CS Stony Brook) and Elsevier
General Logic Programs

- $\Pi = \{p \leftarrow \neg p.\}$ does not have an answer set.
  - $S_1 = \emptyset$, then $\Pi S_1 = \{p \leftarrow \}$ whose minimal model is $\{p\}$. $\{p\} \neq \emptyset$ implies that $S_1$ is not an answer set of $\Pi$.
  - $S_2 = \{p\}$, then $\Pi S_2 = \emptyset$ whose minimal model is $\emptyset$. $\{p\} \neq \emptyset$ implies that $S_2$ is not an answer set of $\Pi 4$. This shows that $\Pi$ does not have an answer set.

- A program may have zero, one, or more than one answer sets.
  - $\Pi 1 = \{a \leftarrow \neg b.\}$ has a unique answer set $\{a\}$.
  - $\Pi 2 = \{a \leftarrow \neg b. b \leftarrow \neg a.\}$ has two answer sets: $\{a\}$ and $\{b\}$.
  - $\Pi 3 = \{p \leftarrow a. a \leftarrow \neg b. b \leftarrow \neg a.\}$ has two answer sets: $\{a, p\}$ and $\{b\}$
  - $\Pi 4 = \{a \leftarrow \neg b. b \leftarrow \neg c. d \leftarrow .\}$ has one answer set $\{d, b\}$.
  - $\Pi 5 = \{p \leftarrow \neg p.\}$ No answer set.
  - $\Pi 6 = \{p \leftarrow \neg p, d. r \leftarrow \neg d. d \leftarrow \neg r.\}$ has one answer set $\{r\}$.
Entailment w.r.t. Answer Set Semantics

• For a program $\Pi$ and an atom $a$, $\Pi$ entails $a$, denoted by $\Pi \models a$, if $a \in S$ for every answer set $S$ of $\Pi$.

• For a program $\Pi$ and an atom $a$, $\Pi$ entails $\neg a$, denoted by $\Pi \models \neg a$, if $a \notin S$ for every answer set $S$ of $\Pi$.

• If neither $\Pi \models a$ nor $\Pi \models \neg a$, then we say that $a$ is unknown with respect to $\Pi$.

Examples:

• $\Pi_1 = \{a \leftarrow \neg b\} \,$ has a unique answer set $\{a\}$. $\Pi_1 \models a$, $\Pi_1 \models \neg b$.

• $\Pi_2 = \{a \leftarrow \neg b. \, b \leftarrow \neg a\} \,$ has two answer sets: $\{a\}$ and $\{b\}$. Both $a$ and $b$ are unknown w.r.t. $\Pi_2$.

• $\Pi_3 = \{p \leftarrow a. \, a \leftarrow \neg b. \, b \leftarrow \neg a.\} \,$ has two answer sets: $\{a, p\}$ and $\{b\}$. Everything is unknown.

• $\Pi_4 = \{p \leftarrow \neg p.\} \,$ No answer set. $p$ is unknown.
Answer Sets of Programs with Constraints

- For a set of ground atoms $S$ and a constraint $c$
  
  $$\leftarrow a_1, \ldots, a_n, \text{ not } a_{n+1}, \text{ not } a_{n+k}$$

- we say that $c$ is satisfied by $S$ if $\{a_1, \ldots, a_n\} \setminus S \neq \emptyset$ or $\{a_{n+1}, \ldots, a_{n+k}\} \cap S \neq \emptyset$.

- Let $\Pi$ be a program with constraints.

- Let $\Pi_O = \{r \mid r \in \Pi, r \text{ has non-empty head}\}$ ($\Pi_O$ is the set of normal logic program rules in $\Pi$) and $\Pi_C = \Pi \setminus \Pi_O$ ($\Pi_C$ is the set of constraints in $\Pi$).

- A set of atoms $S$ is an answer sets of a program $\Pi$ if it is an answer set of $\Pi_O$ and satisfies all the constraints in ground($\Pi_C$).
Answer Sets of Programs with Constraints

Example:

\[ \Pi_1 = \{a \leftarrow \text{not } b. \ b \leftarrow \text{not } a.\} \text{ has two answer sets } \{a\} \text{ and } \{b\}. \]

But, \( \Pi_2 = \{a \leftarrow \text{not } b. \ b \leftarrow \text{not } a. \leftarrow \text{not } a\} \) has only one answer set \( \{a\} \).
Computing Answer Sets

- Complexity: The problem of determining the existence of an answer set for finite propositional programs (programs without function symbols) is NP-complete.
- For programs with disjunctions, function symbols, etc. it is much higher.
- A consequence of this property is that there exists no polynomial-time algorithm for computing answer sets.
Answer set solvers

- Programs that compute answer sets of (finite and grounded) logic programs.

- Two main approaches:
  - Direct implementation: Due to the complexity of the problem, most solvers implement a variation of the generate-and-test algorithm.
    - DLV [http://www.dbai.tuwien.ac.at/proj/dlv/](http://www.dbai.tuwien.ac.at/proj/dlv/)
    - deres [http://www.cs.engr.uky.edu/ai/deres.html](http://www.cs.engr.uky.edu/ai/deres.html)

- Using SAT solvers: A program $\Pi$ is translated into a satisfiability problem $F\Pi$ and a call to a SAT solver is made to compute solution of $F\Pi$. The main task of this approach is to write the program for the conversion from $\Pi$ to $F\Pi$.
Example: Graph Coloring

- Given a (bi-directed) graph and three colors red, green, and yellow. Find a color assignment for the nodes of the graph such that no edge of the graph connects two nodes of the same color.

- Graph representation:
  - The nodes: node(1), ... node(n).
  - The edges: edge(i, j).

- Each node is assigned one color:
  - the weighted rule
    \[ 1 \{ \text{color}(X, \text{red}), \text{color}(X, \text{yellow}), \text{color}(X, \text{green}) \} 1 \leftarrow \text{node}(X). \]
  - or the three rules:
    \[ \text{color}(X, \text{red}) \leftarrow \neg \text{color}(X, \text{green}), \neg \text{color}(X, \text{yellow}). \]
    \[ \text{color}(X, \text{green}) \leftarrow \neg \text{color}(X, \text{red}), \neg \text{color}(X, \text{yellow}). \]
    \[ \text{color}(X, \text{yellow}) \leftarrow \neg \text{color}(X, \text{green}), \neg \text{color}(X, \text{red}). \]

- No edge connects two nodes of the same color:
  \[ \leftarrow \text{edge}(X, Y), \text{color}(X, C), \text{color}(Y, C). \]
Example: Graph Coloring

%%% representing the graph
node(1). node(2). node(3). node(4). node(5).
edge(1,2). edge(1,3). edge(2,4). edge(2,5). edge(3,4). edge(3,5).
%%% each node is assigned a color
color(X,red):- node(X), not color(X,green), not color(X, yellow).
color(X,green):- node(X), not color(X,red), not color(X, yellow).
color(X,yellow):- node(X), not color(X,green), not color(X, red).
%%% constraint checking
:- edge(X,Y), color(X,C), color(Y,C).

• Try with clingo –n 0 color.lp and see the result.
Example: n-Queens

- Place \( n \) queens on a \( n \times n \) chess board so that no queen is attacked (by another one).
- The chess board can be represented by a set of cells \( \text{cell}(i, j) \) and the size \( n \).
- Since two queens cannot be on the same column, we know that each column has to have one and only one queen

\[
1 \{ \text{cell}(I, J) : \text{row}(J) \} 1 \leftarrow \text{col}(I).
\]

- No two queens on the same row

\[
\leftarrow \text{cell}(I, J1), \text{cell}(I, J2), J1 \neq J2.
\]

- No two queens on the same column (not really needed)

\[
\leftarrow \text{cell}(I1, J), \text{cell}(I2, J), I1 \neq I2.
\]

- No two queens on the same diagonal

\[
\leftarrow \text{cell}(I1, J1), \text{cell}(I2, J2), |I1 - I2| = |J1 - J2|
\]
Example: n-Queens

%%% representing the board, using n as a constant
col(1..n). % n column
row(1..n). % n row

%%% generating solutions
1 {cell(I,J) : row(J)} :- col(I).

% two queens cannot be on the same row/column
:- col(I), row(J1), row(J2), neq(J1,J2), cell(I,J1), cell(I,J2).
:- row(J), col(I1), col(I2), neq(I1,I2), cell(I1,J), cell(I2,J).

% two queens cannot be on a diagonal
:- row(J1), row(J2), J1 > J2, col(I1), col(I2), I1 > I2,
cell(I1,J1), cell(I2,J2), eq(I1 - I2, J1 - J2).
:- row(J1), row(J2), J1 > J2, col(I1), col(I2), I1 < I2,
cell(I1,J1), cell(I2,J2), eq(I2 - I1, J1 - J2).

• Command line: lparse -c n=?? prog2 | smodels