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 general program is a collection of rules of the form:

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

- Variables are placeholders for constants
- **Grounding**: “Replace variables by constants in all possible ways”
- Example:

  ```prolog
  isInterestedinASP(X) :- attendsASP(X).
  attendsASP(john).
  attendsASP(mary).
  ```

- After grounding:

  ```prolog
  isInterestedinASP(john) :-
    attendsASP(john).
  isInterestedinASP(mary) :-
    attendsASP(mary).
  ```
Gelfond-Lifschitz transformation

- 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 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 $a_1$ and $a_1$ belongs to $X$; and
  - Removing all of the naf-atoms from the remaining rules
A set of atoms $X$ is called an **answer set** of a program $\Pi$ if $X$ is the **minimal** model of the program $\Pi^X$.

**Example:** Consider $\Pi_2 = \{a \leftarrow \text{not } b. \ b \leftarrow \text{not } a.\}$. 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>$\Pi_{S_1}^{\Pi_2}$:</td>
<td>$\Pi_{S_2}^{\Pi_2}$:</td>
<td>$\Pi_{S_3}^{\Pi_2}$:</td>
<td>$\Pi_{S_4}^{\Pi_2}$:</td>
</tr>
<tr>
<td>$a \leftarrow$</td>
<td>$a \leftarrow$</td>
<td>$b \leftarrow$</td>
<td>$b \leftarrow$</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><strong>NO</strong></td>
<td><strong>YES</strong></td>
<td><strong>YES</strong></td>
<td><strong>NO</strong></td>
</tr>
</tbody>
</table>

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

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

- $\Pi_5 = \{ p \leftarrow \textbf{not} \hspace{1mm} p. \} \text{ does not have an answer set.}$
  - $S_1 = \emptyset$, then $\Pi^{S_1} = \{ p \leftarrow \}$ whose minimal model is $\{ p \}$. $\{ p \} \neq \emptyset$ implies that $S_1$ is not an answer set of $\Pi_5$.
  - $S_2 = \{ p \}$, then $\Pi^{S_2} = \emptyset$ whose minimal model is $\emptyset$. $\{ p \} \neq \emptyset$ implies that $S_2$ is not an answer set of $\Pi_5$.
  - This shows that this program does not have an answer set.

- A program may have zero, one, or more than one answer sets:
  - $\Pi_1 = \{ a \leftarrow \textbf{not} \hspace{1mm} b. \} \text{ has a unique answer set } \{ a \}$.
  - $\Pi_2 = \{ a \leftarrow \textbf{not} \hspace{1mm} b. \ b \leftarrow \textbf{not} \hspace{1mm} a. \} \text{ has two answer sets: } \{ a \} \text{ and } \{ b \}$.
  - $\Pi_3 = \{ p \leftarrow a. \ a \leftarrow \textbf{not} \hspace{1mm} b. \ b \leftarrow \textbf{not} \hspace{1mm} a. \} \text{ has two answer sets: } \{ a, p \} \text{ and } \{ b \}$.
  - $\Pi_4 = \{ a \leftarrow \textbf{not} \hspace{1mm} b. \ b \leftarrow \textbf{not} \hspace{1mm} c. \ d \leftarrow . \} \text{ has one answer set } \{ d, b \}$.
  - $\Pi_5 = \{ p \leftarrow \textbf{not} \hspace{1mm} p. \} \text{ No answer set.}$
  - $\Pi_6 = \{ p \leftarrow \textbf{not} \hspace{1mm} p, \ d. \ r \leftarrow \textbf{not} \hspace{1mm} d. \ d \leftarrow \textbf{not} \hspace{1mm} r. \} \text{ has one answer set } \{ r \}$. 

6
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 \( \Pi_O = \{r \mid r \in \Pi, \text{r has non-empty head}\} \) (\( \Pi_O \) is the set of normal logic program rules in \( \Pi \))

• Let \( \Pi_C = \Pi \setminus \Pi_O \) (\( \Pi_C \) 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:
  - \( \Pi_1 = \{ a \leftarrow \neg b. b \leftarrow \neg a. \} \) has two answer sets \{a\} and \{b\}
  - But, \( \Pi_2 = \{ a \leftarrow \neg b. b \leftarrow \neg a. \leftarrow \neg a. \} \) has only one answer set \{a\}.
  - But, \( \Pi_3 = \{ a \leftarrow \neg b. b \leftarrow \neg a. \leftarrow a. \} \) has only one answer set \{b\}.
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: \texttt{node}(1). \ldots \texttt{node}(n).
  • The edges: \texttt{edge}(i, j).

• Each node is assigned one color:
  • the weighted rule

\[ 1\{\text{color}(X, \text{red}), \text{color}(X, \text{yellow}), \text{color}(X, \text{green})\}1 \leftarrow \texttt{node}(X). \]
  • or the three rules:

\[
\begin{align*}
\text{color}(X, \text{red}) & \leftarrow \texttt{node}(X), \text{not color}(X, \text{green}), \text{not color}(X, \text{yellow}). \\
\text{color}(X, \text{green}) & \leftarrow \texttt{node}(X), \text{not color}(X, \text{red}), \text{not color}(X, \text{yellow}). \\
\text{color}(X, \text{yellow}) & \leftarrow \texttt{node}(X), \text{not color}(X, \text{green}), \text{not color}(X, \text{red}).
\end{align*}
\]

• No edge connects two nodes of the same color:

\[ \leftarrow \texttt{edge}(X, Y), \text{color}(X, C), \text{color}(Y, C). \]
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 × 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 can not be on the same column, we know that each column has to have one and only one queen
  \[
  \{\text{cell}(I, J) : \text{row}(J)\} \leftarrow \text{col}(I).
  \]
- 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
  \[
  \leftarrow \text{cell}(I1, J1), \text{cell}(I2, J2),
  \quad |I1 - I2| = |J1 - J2|, I1 \neq I2.
  \]
Example: n-queens

%%% representing the board, using n as a constant

\texttt{col(1..n). \% n column}
\texttt{row(1..n). \% n row}

%%% generating solutions

1 \{\texttt{cell(I,J) : row(J)}\}:- \texttt{col(I)}.

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

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