Stable Models Semantics and Answer Set Programming

CSE 595 – Semantic Web

Instructor: Dr. Paul Fodor

Stony Brook University

http://www3.cs.stonybrook.edu/~pfodor/courses/cse595.html

General Logic Programs

- A general program is a collection of rules of the form

$$\mathtt{a} \leftarrow \mathtt{a_1,\ \ldots,\ a_n,\ not\ a_{n+1}, \ldots,\ not\ a_{n+k}.}$$

- Let Π be a program and X be a set of atoms, by Π^X (*Gelfond-Lifschitz transformation*) we denote the positive program obtained from **ground**(Π) by:
  - Deleting from **ground**(Π) any rule for that $\{\mathtt{a_{n+1}, \ldots, a_{n+k}}\} \cap \mathtt{X} \neq \emptyset$, i.e., the body of the rule contains a naf-atom **not** $\mathtt{a_l}$ and $\mathtt{a_l}$ 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 Π if X is the minimal model of the program Π^X
- Theorem: For every positive program Π, the minimal model of Π, M_Π, is also the unique answer set of Π.
- Example: Consider Π_2 = {**`a ← not b. b ← not a.`**}. We will show that its has two answer sets {**`a`**} and {**`b`**}

$S_1 = \emptyset$	$S_2 = \{a\}$	$S_3 = \{b\}$	$S_4 = \{a,b\}$
$\Pi_2^{S_1}$: $a \leftarrow$ $b \leftarrow$	$\Pi_2^{S_2}$: $a \leftarrow$	$\Pi_2^{S_3}$: $b \leftarrow$	$\Pi_2^{S_4}$:
$M_{\Pi_2^{S_1}} = \{a,b\}$	$M_{\Pi_2^{S_2}} = \{a\}$	$M_{\Pi_2^{S_3}} = \{b\}$	$M_{P^{S_4}} = \emptyset$
$M_{\Pi_2^{S_1}} \neq S_1$	$M_{\Pi_2^{S_2}} = S_2$	$M_{\Pi_2^{S_3}} = S_3$	$M_{\Pi_2^{S_4}} \neq S_4$
NO	YES	YES	NO

General Logic Programs

- $\Pi_5 = \{\mathtt{p} \leftarrow \mathtt{not\ p.}\}$ does not have an answer set.
  - $S_1 = \emptyset$, then $\Pi^{S1} = \{\mathtt{p} \leftarrow\}$ whose minimal model is $\{\mathtt{p}\}$. $\{\mathtt{p}\} \neq \emptyset$ implies that S_1 is not an answer set of Π_5.
  - $S_2 = \{\mathtt{p}\}$, then $\Pi^{S2} = \emptyset$ whose minimal model is $\emptyset$. $\{\mathtt{p}\} \neq \emptyset$ implies that S_2 is not an answer set of Π_5.
  - This shows that this program does not have an answer set.
- A program may have zero, one, or more than one answer sets.
  - $\Pi_1 = \{a \leftarrow \text{not } b.\}$ has a unique answer set $\{a\}$.
  - $\Pi_2 = \{a \leftarrow \text{not } b.\ b \leftarrow \text{not } a.\}$ has two answer sets: $\{a\}$ and $\{b\}$.
  - $\Pi_3 = \{p \leftarrow a.\ a \leftarrow \text{not } b.\ b \leftarrow \text{not } a.\}$ has two answer sets: $\{a, p\}$ and $\{b\}$
  - $\Pi_4 = \{a \leftarrow \text{not } b.\ b \leftarrow \text{not } c.\ d \leftarrow .\}$ has one answer set $\{d, b\}$.
  - $\Pi_5 = \{p \leftarrow \text{not } p.\}$ No answer set.
  - $\Pi_6 = \{p \leftarrow \text{not } p, d.\ r \leftarrow \text{not } d.\ d \leftarrow \text{not } r.\}$ has one answer set $\{r\}$.

Entailment w.r.t. Answer Set Semantics

- For a program Π and an atom a, Π *entails* a, denoted by $\Pi \vDash a$, if $a \in S$ for every answer set S of Π.
- For a program Π and an atom a, Π entails $\neg a$, denoted by $\Pi \vDash \neg a$, if $a \notin S$ for every answer set S of Π.
- If neither $\Pi \vDash a$ nor $\Pi \vDash \neg a$, then we say that a is *unknown* with respect to Π.
- Examples:
  - $\Pi_1 = \{a \leftarrow \text{not } b.\}$ has a unique answer set $\{a\}$. $\Pi_1 \vDash a$, $\Pi_1 \vDash \neg b$.
  - $\Pi_2 = \{a \leftarrow \text{not } b.\ b \leftarrow \text{not } a\}$ has two answer sets: $\{a\}$ and $\{b\}$. Both a and b are unknown w.r.t. Π_2.
  - $\Pi_3 = \{p \leftarrow a.\ a \leftarrow \text{not } b.\ b \leftarrow \text{not } a.\}$ has two answer sets: $\{a, p\}$ and $\{b\}$. Everything is unknown.
  - $\Pi_4 = \{p \leftarrow \text{not } p.\}$ 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 Π be a program with constraints.
- Let $\Pi_O = \{r \mid r \in \Pi, r$ has non-empty head$\}$ (Π_O is the set of normal logic program rules in Π)
- Let $\Pi_C = \Pi \setminus \Pi_O$ (Π_C is the set of constraints in Π)
- A set of atoms S is an answer sets of a program Π if it is an answer set of Π_O and satisfies all the constraints in ground (Π_C).

Answer Sets of Programs with Constraints

- Example:
  - $\Pi_1 = \{a \leftarrow \text{not } b.\ b \leftarrow \text{not } a.\}$ has two answer sets $\{a\}$ and $\{b\}$.
  - But, $\Pi_2 = \{$ $a \leftarrow \text{not } b.$

    $b \leftarrow \text{not } a.$

    $\leftarrow \text{not } a\}$

  has only one answer set $\{a\}$.
  - But, $\Pi_3 = \{$ $a \leftarrow \text{not } b.$

    $b \leftarrow \text{not } a.$

    $\leftarrow a\}$

  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.
    - Smodels http://www.tcs.hut.fi/Software/smodels/
    - DLV http://www.dbai.tuwien.ac.at/proj/dlv/
    - deres http://www.cs.engr.uky.edu/ai/deres.html
  - Using SAT solvers: A program Π is translated into a satisfiabilty problem FΠ and a call to a SAT solver is made to compute solution of FΠ. The main task of this approach is to write the program for the conversion from Π to FΠ.
    - Potassco: http://potassco.sourceforge.net/ (clasp, gringo, …)
    - Cmodels http://www.cs.utexas.edu/users/tag/cmodels.html
    - ASSAT http://assat.cs.ust.hk/

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{color(X, red), color(X, yellow), color(X, green)}1 ← node(X).
```

    - or the three rules:

```
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).
```

  - No edge connects two nodes of the same color:

```
← edge(X, Y ), color(X, C), 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 **`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

    `1{cell(I, J) : row(J)}1 ← col(I).`
  - No two queens on the same row

    `← cell(I, J1), cell(I, J2), J1 ≠ J2.`
  - No two queens on the same column (not really needed)

    `← cell(I1, J), cell(I2, J), I1 ≠ I2.`
  - No two queens on the same diagonal

    `← cell(I1, J1), cell(I2, J2),`

    `|I1 - I2| = |J1 - J2|, I1≠ I2.`

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).
```