Nonmonotonic logic programs I

Let $A$ be a set of propositional atoms.

Rules:

\[ c \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k \]

where $\{c, b_1, \ldots, b_m, d_1, \ldots, d_k\} \subseteq A$

- Meaning similar to default logic:
  - **If** we have derived $b_1, \ldots, b_m$ and
  - **cannot derive any of** $d_1, \ldots, d_k$,
  - **then derive** $c$.

- Rules without right-hand side (**facts**): $c \leftarrow \top$

- Rules without left-hand side (**constraints**): $\bot \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k$

---

Nonmonotonic logic programs II

Let $A$ be a set of propositions.

Rules:

\[ c \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k \]

where $\{c, b_1, \ldots, b_m, d_1, \ldots, d_k\} \subseteq A$

- $c$ is called the **head** of the rule (denoted by $\text{head}(r)$);
- $b_1, \ldots, b_m$ is called the **positive body** of the rule (denoted by $\text{body}^+(r)$);
- $\neg d_1, \ldots, \neg d_k$ is called the **negative body** of the rule (denoted by $\text{body}^-(r)$);
- The **body** of the rule consists in its positive and negative part ($\text{body}(r) = \text{body}^+(r) \cup \text{body}^-(r)$).
Nonmonotonic logic programs: examples

Example

\[ \text{fly} \leftarrow \text{bird, not abnormal.} \]
\[ \text{abnormal} \leftarrow \text{penguin.} \]
\[ \text{bird} \leftarrow \text{penguin.} \]

Example

\[ \{\text{sol}(X, Y, A) : \text{num}(A)\} \].
\[ \leftarrow \text{sol}(X, Y, Z), \text{sol}(X, Y_1, Z), Y \neq Y_1. \]
\[ \leftarrow \text{sol}(X, Y, Z), \text{sol}(X, Y, Z), X \neq X_1. \]
\[ \leftarrow \text{sol}(W * 3 + W_2, W_1 * 3 + W_3, Z), \]
\[ \text{sol}(W * 3 + W_4, W_1 * 3 + W_5, Z), W_3 \neq W_5. \]
\[ \leftarrow \text{sol}(W * 3 + W_2, W_1 * 3 + W_3, Z), \]
\[ \text{sol}(W * 3 + W_4, W_1 * 3 + W_5, Z), W_2 \neq W_4. \]

not-free logic programs

Definition (Deductive closure)

Let \( \Pi \) be a logic program without not, \( X \subseteq \text{Atoms}(\Pi) \).
The closure \( \text{dcl}(\Pi) \subseteq \text{Atoms}(\Pi) \) of \( \Pi \) is defined by iterative application of the rules in the obvious way. \( X \) is an answer set of \( \Pi \) if \( X = \text{dcl}(\Pi) \) and there is no constraint in \( \Pi \) violated by \( X \).

Example

\[ \Pi = \{ \begin{array}{l}
   a \leftarrow b, \ d \leftarrow f, \ b, \\
   d \leftarrow b, \ c \leftarrow b, \ d, \ e \leftarrow f
\end{array} \} \]
\[ \Gamma_0 = \Gamma(\emptyset) = \{b\} \]
\[ \Gamma_1 = \Gamma(\Gamma_0) = \{b, d, a\} \]
\[ \Gamma_2 = \Gamma(\Gamma_1) = \{b, d, a, c\} \]
\[ \Gamma_3 = \Gamma(\Gamma_2) = \{b, d, a, c\} = \Gamma_2 \]
**Definition 1: Gelfond-Lifschitz reduct**

**Definition (Reduct)**
The reduct of a program \( \Pi \) with respect to a set of atoms \( X \subseteq \text{Atoms}(\Pi) \) is defined as:

\[
\Pi^X := \{ c \leftarrow b_1, \ldots, b_m \mid (c \leftarrow b_1, \ldots, b_m, \text{not } d_1, \ldots, \text{not } d_k) \in \Pi, \{ d_1, \ldots, d_k \} \cap X = \emptyset \}
\]

**Definition (Answer set)**
\( X \subseteq \text{Atoms}(\Pi) \) is an answer set of \( \Pi \) if \( X \) is an answer set of \( \Pi^X \).

**Illustration of Gelfond-Lifschitz reduct**

**Example**

\[
a \leftarrow \text{not } b.\quad b \leftarrow \text{not } a.\quad d \leftarrow a.\quad d. \leftarrow b.
\]

**Example**

\[
a \leftarrow b.\quad b \leftarrow a.
\]

**Example**

\[
\text{woman} \leftarrow \text{not } \text{n\_woman}.\quad \text{n\_woman} \leftarrow \text{not } \text{woman}.\quad \text{father} \leftarrow \text{parent}, \text{n\_woman}.\quad \text{mother} \leftarrow \text{parent}, \text{woman}.
\]

We say that \( X \) satisfies a rule \( r \) iff \( X \models \text{head}(r) \lor \neg \text{body}(r) \).

\( \Rightarrow X \) can satisfy all rules and not be an answer set.

**Lparse and smodels**

- Based on the Gelfond-Lifschitz reduction, Syrjanen created the ASP solver Smodels.

![Lparse](preprocessing) ![Smodels](preprocessing)

- Allow for using variables and cardinality statements.

**Example**

\[
b : \leftarrow \text{not } a.
\quad a : \leftarrow \text{not } b.
\quad d : \leftarrow a.
\quad d : \leftarrow b.
\]

**The lparse format I**

- propositions are any combination of lowercase letters;
- variables are any combination of letters starting with an uppercase letter;
- integers can be used and so can arithmetic operations (+, -, *, /, %);
- negation as failure is denoted by not.
- implication is denoted by "\( \leftarrow \)".
The lparse format II

- The literal \( \{b_1, \ldots, b_m\} \cup \) is true iff at least \( l \) and at most \( u \) atoms are true within the set \( \{b_1, \ldots, b_m\} \);
- \#domain encodes the possible values in a given domain:
  - \#domain a(X). a(1..10).
    will replace occurrences of \( X \) by integers from 1 to 10.

The lparse format III

- Domains can also be set within a cardinality rule:
  - \{clique(X) : num(X)\}.
    num(1..3).
    will be understood as \{clique(1), clique(2), clique(3)\}.
- Domains can be restricted thanks to relations. The rule
  - \( :- \) size(X,Y), X<Y.
    will be instantiated only for value of \( X \) and \( Y \) s.t. \( X<Y \).
- A subset of answer sets can be selected according to some optimization criteria.
  - \#minimize{a,b,c,d}.
    will choose the answer sets with the less number of atoms
    from \{a,b,c,d\}.
    Attention: Does not change the SAT/UNSAT question. You can only optimize one criterion at a time.

The lparse format IV

Example

\#domain a(X). a(1..2).

\begin{align*}
  c(X) & : - not \: d(X). \\
  d(X) & : - not \: c(X).
\end{align*}

\begin{align*}
  & a(1). \quad a(2). \\
  & c : - not \: d(1). \quad c : - not \: d(2). \\
  & d : - not \: c(1). \quad d : - not \: c(2).
\end{align*}


Guess - check - optimize

How to represent a problem in ASP?

- Firstly, define what is a "solution candidate";
- Secondly, verify it fits the constraints
- Finally, keep only the best answer sets

Example

\#domain node(X). \#domain node(Y).

\begin{align*}
  node(1..5). \: & \: edge(1,2). \: edge(3,4). \\
  & \: edge(4,5). \: edge(4,2). \: edge(1,4).
\end{align*}

\begin{align*}
  & \text{uedge}(X,Y) : - \: \text{edge}(X,Y), \: X < Y. \\
  & \text{uedge}(Y,X) : - \: \text{edge}(X,Y), \: Y < X.
\end{align*}

\begin{align*}
  & \{ \text{clique}(X) : \: \text{node}(X) \}. \\
  & :- \: \text{clique}(X), \: \text{clique}(Y), \: \text{not} \: \text{uedge}(X,Y), \: X < Y.
\end{align*}

\#maximize \{ \text{clique}(X) : \: \text{node}(X) \}. 

\begin{align*}
  & 1 \: 2 \: 1 \: 1 \: 3 \\
  & 1 \: 4 \: 1 \: 5 \\
  & 1 \: 3 \: 1 \: 2 \\
  & 1 \: 5 \: 1 \: 4 \\
  & 1 \: 6 \: 0 \: 0 \\
  & 1 \: 7 \: 0 \: 0 \\
  & 0 \\
  & 2 \: d(1) \: 3 \: c(1) \: 4 \: d(2) \\
  & 5 \: c(2) \: 6 \: a(1) \: 7 \: a(2)
\end{align*}
Complexity: existence of answer sets is NP-complete

- **Membership in NP**: Guess $X \subseteq \text{Atoms}(\Pi)$ (non-det. polytime), compute $\Pi^X$, compute its closure, compare to $X$ (everything det. polytime).
- **NP-hardness**: Reduction from 3SAT: an answer set exists iff clauses are satisfiable:
  
  $$p \leftarrow \neg \hat{p}.$$  
  $$\hat{p} \leftarrow \neg p.$$  

  for every proposition $p$ occurring in the clauses, and
  
  $$\leftarrow \neg l'_1, \neg l'_2, \neg l'_3$$  

  for every clause $l_1 \lor l_2 \lor l_3$, where $l'_i = p$ if $l_i = p$ and $l'_i = \hat{p}$ if $l_i = \neg p$.

Some properties I

**Proposition**

If an atom $A$ belongs to an answer set of a logic program $\Pi$ then $A$ is the head of one of the rules of $\Pi$.

**Proposition**

Let $F$ and $G$ be sets of rules and let $X$ be a set of atoms. Then the following holds:

$$(F \cup G)^X = \begin{cases} F^X \cup G^X, & \text{if } X \models F \cup G \\ \bot, & \text{otherwise} \end{cases}$$

Some properties II

**Proposition**

Let $F$ be a set of (non-constraint) rules and $G$ be a set of constraints. A set of atoms $X$ is an answer set of $F \cup G$ iff it is an answer set of $F$ which satisfies $G$.

**Proof.**

$\Rightarrow X$ satisfies $F \cup G$. Then $X$ satisfies the constraints in $G$ and $(F \cup G)^X = F^X \cup \bot$ which is equivalent to $F^X$. Consequently $X$ is minimal among the sets satisfying $F^X$ iff it is minimal among the sets satisfying $(F \cup G)^X$.

$\Leftarrow X$ does not satisfy $F \cup G$. Then there exists a rule in $F$ or a rule in $G$ which is not satisfied, then $X$ cannot be a model of $F$ that satisfies $G$.

Smodels: principles

**Smodels** is:

- a Branch and Bound algorithm;
- based on the Gelfond-Lifschitz reduct;
- using reduct as a Forward-Checking procedure.

**Example**

```
| a :- not b. 
| b :- not a. 
| c :- not c, a. |
```

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
```

```
Algorithm 1 Smodels algorithm

1: $A := \text{expand}(P, A)$
2: $A := \text{lookahead}(P, A)$
3: if conflict($P, A$) then
4: return false
5: else if $A$ covers Atoms($P$) then
6: return stable($P, A$)
7: else
8: $x := \text{heuristic}(P, A)$
9: if smodels($P, A \cup \{X\}$) then
10: return true
11: else
12: return smodels($P, A \cup \{\text{not}X\}$)
13: end if
14: end if

Smodels example (I)

Example

(1) $a \leftarrow \text{not}b, \text{not}d$. (2) $d \leftarrow \text{not}a$.
(3) $b \leftarrow \text{not}c$. (4) $c \leftarrow \text{not}a$.
(5) $e \leftarrow \text{not}f, \text{not}a$. (6) $f \leftarrow \text{not}e$.

Case 1: $a \subseteq X$
- (4) cannot be fired, $\rightarrow c \not\subseteq X$;
- (3) becomes $c$, $\rightarrow b \subseteq X$;
- (1) cannot be fired, $\rightarrow a \not\subseteq X$;
- $a \not\subseteq X$ and $a \subseteq X$, $\rightarrow$ contradiction.

Case 2: $a \not\subseteq X$
- (2) becomes $d$, $\rightarrow d \subseteq X$;
- (4) becomes $c$, $\rightarrow c \subseteq X$;
- (3) cannot be fired, $\rightarrow b \not\subseteq X$;
- $a \not\subseteq X$ and $a \subseteq X$, $\rightarrow$ contradiction.
- Nothing new to be expanded.

Smodels example (II)

Example

(1) $a \leftarrow \text{not}b, \text{not}d$. (2) $d \leftarrow \text{not}a$.
(3) $b \leftarrow \text{not}c$. (4) $c \leftarrow \text{not}a$.
(5) $e \leftarrow \text{not}f, \text{not}a$. (6) $f \leftarrow \text{not}e$.

Case 2.1: $e \subseteq X$
After reduction:

- $e \leftarrow \text{not}f$. $f \leftarrow \text{not}e$.
- (6) cannot be fired, $\rightarrow f \not\subseteq X$;
- (5) becomes $e$, $\rightarrow e \subseteq X$;
- $X$ covers all atoms, there is no contradiction.
Solution: $\{c, d, e\}$ is a stable model.

2 Logic of here-and-there
Equivalence between logic programs

Are the two following logic programs

\[ \Pi_1 = a \leftarrow \text{not} \ b. \ b \leftarrow \text{not} \ a. \]

and

\[ \Pi_2 = a \leftarrow \text{not} \ b. \ b \leftarrow \text{not} \ c, \text{not} \ a. \]

equivalent?

They are weakly equivalent but not strongly equivalent.

Weak equivalence/strong equivalence

**Definition (Weak equivalence)**
\( \Pi_1 \) and \( \Pi_2 \) are weakly equivalent if they have the same answer sets.

**Definition (Strong equivalence)**
\( \Pi_1 \) and \( \Pi_2 \) are strongly equivalent if for any \( \Pi \), \( \Pi_1 \cup \Pi \) and \( \Pi_2 \cup \Pi \) have the same answer sets.

Example

\[ \Pi_1 = a \leftarrow \text{not} \ b. \ b \leftarrow \text{not} \ a. \]
\[ \Pi_2 = a \leftarrow \text{not} \ b. \ b \leftarrow \text{not} \ c, \text{not} \ a. \]

Do \( \Pi_1 \) and \( \Pi_2 \) have the same answer sets?

Do \( \Pi_1 \cup \{c\} \) and \( \Pi_2 \cup \{c\} \) have the same answer sets?

Logic of here-and-there

One can also consider logic programs through the logic of here-and-there.

- A pair of sets of atoms \( (X, Y) \) such that \( X \subseteq Y \) is called an SE-interpretation;
- A SE-interpretation \( (X, Y) \) is called an SE-model iff \( Y \models \Pi \) and \( X \models \Pi^Y \).

Example

\[ a \leftarrow \text{not} \ b. \ b \leftarrow \text{not} \ a. \ c \leftarrow a. \]

\[
\begin{array}{|c|c|c|}
\hline
Y & \Pi^Y & X \\
\hline
\{a,c\} & a \leftarrow a. & \{a,c\} \\
\{b\} & b \leftarrow a. & \{b\} \\
\{b,c\} & b \leftarrow a. & \{b,c\} \\
\{a,b,c\} & c \leftarrow a. & \{\emptyset, \{b\}, \{a,c\}, \{a,b,c\}\} \\
\hline
\end{array}
\]

Thus, there are two answer sets here: \( \{b\} \) and \( \{a,c\} \).

The set of SE-models of \( \Pi \) is denoted by \( SE(\Pi) \).

Answer set definition II

**Proposition (Characterization of answer sets)**
\( Y \) is an answer set of \( \Pi \) iff \( (Y, Y) \) is an SE-model of \( \Pi \) and there is no \( (X, Y) \) within the SE-models of \( \Pi \) such that \( X \subseteq Y \).

Example

\[ a \rightarrow \text{not} \ b. \ b \leftarrow \text{not} \ a. \ c \leftarrow a. \]

\[
\begin{array}{|c|c|c|}
\hline
Y & \Pi^Y & X \\
\hline
\{a,c\} & a \leftarrow a. & \{a,c\} \\
\{b\} & b \leftarrow a. & \{b\} \\
\{b,c\} & b \leftarrow a. & \{b,c\} \\
\{a,b,c\} & c \leftarrow a. & \{\emptyset, \{b\}, \{a,c\}, \{a,b,c\}\} \\
\hline
\end{array}
\]

Thus, there are two answer sets here: \( \{b\} \) and \( \{a,c\} \).
Strong equivalence: properties

- Proposition
  The logic programs $\Pi_1$ and $\Pi_2$ are strongly equivalent if and only if they have the same set of SE-models.

- Lemma
  1. Programs with the same SE-models are weakly equivalent.
  2. The SE-models of $\Pi_1 \cup \Pi_2$ are exactly the SE-models common to $\Pi_1$ and $\Pi_2$.

- Proof.
  Assume $\Pi_1$ and $\Pi_2$ have the same SE-models. Consider $\Pi$. By lemma 2: $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same SE-models. By lemma 1: $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are weakly equivalent.

- Proof.
  \[ \Rightarrow \] Assume $\exists (X, Y) \in SE(\Pi_1) \text{ and } (X, Y) \notin SE(\Pi_2)$. Two cases:
  - contrary, $Y \models \Pi_1$ thus $Y \models \Pi_1 \cup Y$. Follows, $Y \models (\Pi_1 \cup Y)^Y$. No subset of $Y$ satisfies $(\Pi_1 \cup Y)^Y$ and thus $Y$ is a model of $\Pi_1 \cup Y$.

Dependency graph

- Definition (Dependency graph)
  The dependency graph of a program $\Pi$ is the directed graph $G$ such that the vertexes of $G$ are the atoms in $\Pi$ and $G$ has an edge from $a_0$ to $a_1, \ldots, a_m$ for each rule of the form $a_0 \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$ in $\Pi$ with $a_0 \neq \bot$.

- Example
  $\Pi = \{ a \leftarrow b, b \leftarrow a, a \leftarrow \text{not.c. } \}$

3 SAT translations of ASP

- Positive-order consistent logic programs
- Clark’s completion
- CLASP solver

Clark’s completion

- For each $p \in \text{Atoms}(\Pi)$, let $p \leftarrow B_1, \ldots, p \leftarrow B_n$ be all the rules about $p \in \Pi$, then $p \equiv B_1 \lor \ldots \lor B_n$ is in $\text{Comp}(\Pi)$. In particular, if $n = 0$ then the equivalence is $p \equiv \bot$, which is equivalent to $\neg p$.
- If $\neg B$ is a constraint in $\Pi$, then $\neg B$ is in $\text{Comp}(\Pi)$.

- Example
  $\Pi = \{ a \leftarrow b, b \leftarrow a, a \leftarrow \text{not.c. } \}$
  $\text{Comp}(\Pi) = \{ a \equiv \neg c \lor b, b \equiv a, c \equiv \neg a \lor d, d \equiv c \}$
  $\text{Comp}(\Pi)$ has 3 models: $\{a, b\}, \{c, d\}$ and $\{a, b, c, d\}$. 
Tight programs

Definition (Tight program)
A logic program \( \Pi \) is said to be tight (or positive-order consistent) if its dependency graph is cycle-free.

Example
\[
\Pi = \{ 
  d \leftarrow b, 
  b \leftarrow a, 
  a \leftarrow \text{not}c, 
  d \leftarrow b, 
  b \leftarrow c, 
  c \leftarrow \text{not}a. 
\}
\]

Tightness and Clark’s completion

Proposition
If \( \Pi \) is a positive-order consistent logic program, then \( X \) is an answer set of \( \Pi \) if and only if \( X \) is a model of \( \text{Comp}(\Pi) \).

Example
\[
\Pi = \{ 
  a \leftarrow b, 
  b \leftarrow a, 
  a \leftarrow \text{not}c, 
  c \leftarrow d, 
  d \leftarrow c, 
  c \leftarrow \text{not}a. 
\}
\]
\[
\text{Comp}(\Pi) = \{ 
  a \equiv \neg c \lor b 
  b \equiv a 
  c \equiv \neg a \lor d 
  d \equiv c 
\}
\]
\( \text{Comp}(\Pi) \) has 3 models: \( \{a, b\} \), \( \{c, d\} \) and \( \{a, b, c, d\} \).

Tightness and Clark’s completion (proof)

Definition (Well-supported model)
\( M \) is a well-supported model of \( \Pi \) if there exists a grounding sequence for \( M \), i.e., there exists an order \( \prec \) between rules such that for every rule \( r \in \Pi \) with \( a = \text{head}(r) \) and \( M \models \text{body}(r) \), then \( \forall b \in \text{body}^+(r), b \prec a \).

Theorem
If \( \Pi \) is a tight logic program then the model of \( \text{Comp}(\Pi) \) are exactly the answer sets of \( \Pi \).

Proof.
If \( X \) is an answer set of \( \Pi \), then it is a well-supported model of \( \Pi \), then it is a minimal Herbrand model of \( \Pi \), then it is a model of \( \text{Comp}(\Pi) \).
Assume that \( M \) is model of \( \text{Comp}(\Pi) \) but not a well-supported model of \( \Pi \). \( \exists x \in M \) that cannot be finitely justified. \( M \) being a supported model of \( (\Pi) \), then \( \exists r \in \Pi \) with \( x = \text{head}(r) \) and \( M \models \text{body}(r) \). Thus, there exists \( y \in M \) which is upper in the dependency graph that cannot be justified and thus, there exists a \( z \in M \) such that, etc... There is an infinite chain in the dependency graph which is contradictory with the tightness hypothesis.
Loops

Definition (Loop)
A loop of $\Pi$ is a set of atoms such that for each pair $A, A'$ of atoms in $L$ there is a path from $A$ to $A'$ in the dependency graph of $\Pi$ whose intermediate nodes belong to $L$.

$R^+(L, \Pi) = \{ p \leftarrow G \mid (p \leftarrow G) \in \Pi, p \in L, (\exists q) s.t. q \in G \wedge q \in L \}$

$R^-(L, \Pi) = \{ p \leftarrow G \mid (p \leftarrow G) \in \Pi, p \in L, (\exists q) s.t. q \in G \wedge q \in L \}$

Example
$\Pi = \{ a \leftarrow b. b \leftarrow a. a \leftarrow \text{notc.} \}$

$R^+(L_1, \Pi) = \{ a \leftarrow b. b \leftarrow a. \}$ $R^-(L_1, \Pi) = \{ a \leftarrow \text{notc.} \}$

$R^+(L_2, \Pi) = \{ c \leftarrow d. d \leftarrow c. \}$ $R^-(L_2, \Pi) = \{ c \leftarrow \text{nota.} \}$

CLASP translation I

Definition (Body clauses)
Let $\beta$ be a body of a rule $\beta = \{ p_1, ..., p_m, \text{not} p_{m+1}, ..., \text{not} p_n \}$, then:

$\delta(\beta) = \{ \beta \vee \neg p_1 \vee ... \vee \neg p_m \vee \neg p_{m+1} \vee ... \vee \neg p_n \}$

$\Delta(\beta) = \{ \neg \beta \vee p_1, ..., \neg \beta \vee p_m, \neg \beta \vee p_{m+1}, ..., \neg \beta \vee p_n \}$

Example
$\Pi = \{ a \leftarrow b. b \leftarrow a. a \leftarrow \text{notc.} \}$

$\text{Comp}(\Pi) \cup LF(\Pi) = \{ a \equiv \neg c \vee b \}
\quad b \equiv a
\quad c \equiv \neg a \vee d
\quad d \equiv c
\quad c \rightarrow (\neg a \wedge \neg b)
\quad a \rightarrow (\neg c \wedge \neg d) \}$

Loop formulas

Definition (Loop formulas)
Let $R^-(L, \Pi)$ be the following rules:

\[
p_1 \leftarrow B_{11} \quad \cdots \quad p_1 \leftarrow B_{1k_1} \\
\vdots \\
p_n \leftarrow B_{nk_1} \quad \cdots \quad p_n \leftarrow B_{nk_n}
\]

The loop formula associated with $L$ is the following implication:

\[
\neg [B_{11} \lor \cdots \lor B_{1k_1} \lor \cdots \lor B_{nk_1}] \lor \bigwedge_{p \in L} \neg p
\]

Example

$R^+(L_1, \Pi) = \{ a \leftarrow b. b \leftarrow a. \}$ $R^-(L_1, \Pi) = \{ a \leftarrow \text{notc.} \}$

$R^+(L_2, \Pi) = \{ c \leftarrow d. d \leftarrow c. \}$ $R^-(L_2, \Pi) = \{ c \leftarrow \text{nota.} \}$

$LF(L_1) : c \rightarrow (\neg a \wedge \neg b)$ $LF(L_2) : a \rightarrow (\neg c \wedge \neg d)$
CLASP translation II

Definition (Atoms clauses)
Let $p$ be an atom appearing as head of rules whose body are 
$\{\beta_1, \ldots, \beta_k\}$, then:

- $\Delta(p) = \{p \lor \neg \beta_1, \ldots, p \lor \neg \beta_k\}$
- $\delta(p) = \{\neg p \lor \beta_1 \lor \ldots \lor \beta_k\}$

Example

$\Pi = \{a \leftarrow b. b \leftarrow a. a \leftarrow \text{not.c.} \}$

CLASP translation III

Definition (External body)
For a program $\Pi$ and some $U \subseteq \text{Atoms}(\Pi)$, we define the external bodies of $U$ for $\Pi$, $EB_{\Pi}(U)$ as

$\{\text{body}(r) | r \in \Pi, \text{head}(r) \in U, \text{body}(r) \cap U = \emptyset\}$

Definition (Loop clause)
For a set $U \subseteq \text{Atoms}(\Pi)$ and an atom $p \in U$:

$\lambda(p, U) = \{\beta_1 \lor \ldots \lor \beta_k \lor \neg p\}$

where $EB_{\Pi}(U) = \{\beta_1, \ldots, \beta_k\}$.

We define $\Lambda_{\Pi} = \bigcup_{U \subseteq \text{Atoms}(\Pi), U \neq \emptyset} \{\lambda(p, U) | p \in U\}$.

CLASP translation IV

Proposition
$X$ is an answer set of $\Pi$ iff $X \cap \text{Atoms}(\Pi)$ is a model of the following CNF:

$\Lambda_{\Pi} \cup \Delta(p) \cup \delta(p) \cup \delta(\beta) \cup \Delta(\beta)$

Literature

- Michael Gelfond and Vladimir Lifschitz.
  The stable models semantics for logic programming.
- Francois Fages.
  Consistency of Clark’s completion and existence of stable models.
  Meth. of Logic in CS, p51-60, 1994.
- Hudson Turner.
  Strong equivalence made easy: nested expressions and weight constraints.
The Gelfond-Lifschitz reduct
Logic of here-and-there
SAT
Translations of ASP
Positive-order consistent logic programs
Clark's completion
CLASP solver

Literature

- Martin Gebser and Benjamin Kaufmann and André Neumann and Torsten Schaub. 
  Conflict-Driven Answer Set Solving. 

- Ilkka Niemelä and Patrik Simons 
  Efficient Implementation of the Well-founded and Stable Model Semantics. 