Principles of Knowledge Representation and Reasoning

Answer Set Programming

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Bernhard Nebel, Stefan Wölfl, and Julien Hué

November 28, 2012; December 5, 2012

Nonmonotonic logic programs: background

- Answer set semantics: a formalization of negation-as-failure in logic programming (Prolog)
- Several formalizations: well-founded semantics, perfect-model semantics, inflationary semantics, ...
- Can be viewed as a simpler variant of default logic
- A better alternative to propositional logic in some applications

Nonmonotonic logic programs I

Let A be a set of propositional atoms.

Rules:

$$c \leftarrow b_1, \dots, b_m, \mathrm{not}\, d_1, \dots, \mathrm{not}\, d_k$$

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

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

  then derive c**.**
- Rules without right-hand side (facts): $c \leftarrow \top$
- Rules without left-hand side (constraints):

$$\bot \leftarrow b_1, \dots, b_m, \mathrm{not}\, d_1, \dots, \mathrm{not}\, d_k$$

Nonmonotonic logic programs II

Let A be a set of propositions.

Rules:

$$c \leftarrow b_1, \ldots, b_m, \mathrm{not}\, d_1, \ldots, \mathrm{not}\, d_k$$

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

- c is called the head of the rule (denoted by $\mathrm{head}(r)$);
- $b_1, \ldots, b_m$ is called the positive body of the rule (denoted by $\mathrm{body}^+(r)$);
- $\mathrm{not}\, d_1, \ldots, \mathrm{not}\, d_k$ is called the negative body of the rule (denoted by $\mathrm{body}^-(r)$);
- The body of the rule consists in its positive and negative part ($\mathrm{body}(r) = \mathrm{body}^+(r) \cup \mathrm{body}^-(r)$).

Nonmonotonic logic programs: examples

Example

$$fly \leftarrow bird, \mathrm{not}\, abnormal.$$
$$abnormal \leftarrow penguin.$$
$$bird \leftarrow penguin.$$

Example

$$1\{sol(X,Y,A) : num(A)\}1.$$
$$\leftarrow sol(X,Y,Z), sol(X,Y1,Z), Y \neq Y1.$$
$$\leftarrow sol(X,Y,Z), sol(X1,Y,Z), X \neq X1.$$
$$\leftarrow sol(W*3+W2, W1*3+W3, Z), sol(W*3+W4, W1*3+W5, Z), W3 \neq W5.$$
$$\leftarrow sol(W*3+W2, W1*3+W3, Z), sol(W*3+W4, W1*3+W5, Z), W2 \neq W4.$$

Nonmonotonic logic programs: examples

Example

$$\begin{aligned} fly &\leftarrow bird, \mathrm{not}\, abnormal. \\ abnormal &\leftarrow penguin. \\ bird &\leftarrow penguin. \end{aligned}$$

Example

$$\begin{array}{l} 1\{sol(X,Y,A) : num(A)\}1. \\ \leftarrow sol(X,Y,Z), sol(X,Y1,Z), Y \neq Y1. \\ \leftarrow sol(X,Y,Z), sol(X1,Y,Z), X \neq X1. \\ \leftarrow sol(W*3+W2, W1*3+W3, Z), \\ \qquad\qquad sol(W*3+W4, W1*3+W5, Z), W3 \neq W5. \\ \leftarrow sol(W*3+W2, W1*3+W3, Z), \\ \qquad\qquad sol(W*3+W4, W1*3+W5, Z), W2 \neq W4. \end{array}$$

*

not-free logic programs

Definition (Deductive closure)

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

Example

$$\Pi = \left\{ \begin{array}{llllllll} a & \leftarrow & b. & d & \leftarrow & f. & b. & \\ d & \leftarrow & b. & c & \leftarrow & b,d. & e & \leftarrow f. \end{array} \right\}$$

$$\begin{aligned} \Gamma_0 &= \Gamma(\emptyset) = \{b\} \\ \Gamma_1 &= \Gamma(\Gamma_0) = \{b,d,a\} \\ \Gamma_2 &= \Gamma(\Gamma_1) = \{b,d,a,c\} \\ \Gamma_3 &= \Gamma(\Gamma_2) = \{b,d,a,c\} = \Gamma_2 \end{aligned}$$

not-free logic programs

Definition (Deductive closure)

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

Example

$$\Pi = \left\{ \begin{array}{llllllllll} a & \leftarrow & b. & d & \leftarrow & f. & b. & & & \\ d & \leftarrow & b. & c & \leftarrow & b,d. & e & \leftarrow & f. \end{array} \right\}$$

$$\begin{aligned} \Gamma_0 &= \Gamma(\emptyset) = \{b\} \\ \Gamma_1 &= \Gamma(\Gamma_0) = \{b,d,a\} \\ \Gamma_2 &= \Gamma(\Gamma_1) = \{b,d,a,c\} \\ \Gamma_3 &= \Gamma(\Gamma_2) = \{b,d,a,c\} = \Gamma_2 \end{aligned}$$

The Gelfond-Lifschitz reduct

Definition 1: Gelfond-Lifschitz reduct

Definition (Reduct)

The reduct of a program Π with respect to a set of atoms $X \subseteq \mathrm{Atoms}(\Pi)$ is defined as:

$$\Pi^X := \{c \leftarrow b_1, \ldots, b_m \quad | \quad (c \leftarrow b_1, \ldots, b_m, \mathrm{not}\, d_1, \ldots, \mathrm{not}\, d_k) \in \Pi, \{d_1, \ldots, d_k\} \cap X = \emptyset\}$$

Definition (Answer set)

$X \subseteq \mathrm{Atoms}(\Pi)$ is an answer set of Π if X is an answer set of Π^X.

Definition 1: Gelfond-Lifschitz reduct

Definition (Reduct)

The reduct of a program Π with respect to a set of atoms $X \subseteq \mathrm{Atoms}(\Pi)$ is defined as:

$$\Pi^X := \{c \leftarrow b_1, \ldots, b_m \quad | \quad (c \leftarrow b_1, \ldots, b_m, \mathrm{not}\, d_1, \ldots, \mathrm{not}\, d_k) \in \Pi, \{d_1, \ldots, d_k\} \cap X = \emptyset\}$$

Definition (Answer set)

$X \subseteq \mathrm{Atoms}(\Pi)$ is an answer set of Π if X is an answer set of Π^X.

Illustration of Gelfond-Lifschitz reduct

Example

$$\begin{array}{rclrcl} a & \leftarrow & \mathrm{not}\, b. & b & \leftarrow & \mathrm{not}\, a. \\ d & \leftarrow & a. & d. & \leftarrow & b. \end{array}$$

Example

$$\begin{array}{rclrcl} a & \leftarrow & b. & b & \leftarrow & a. \end{array}$$

Example

$$\begin{array}{rcl} \mathit{woman} \leftarrow \mathrm{not}\, \mathit{n\_woman}. & & \mathit{n\_woman} \leftarrow \mathrm{not}\, \mathit{woman}. \\ \leftarrow \mathit{woman}, \mathit{n\_woman}. & & \mathit{father} \leftarrow \mathit{parent}, \mathit{n\_woman}. \\ \mathit{mother} \leftarrow \mathit{parent}, \mathit{woman}. & & \mathit{parent}. \end{array}$$

We say that X satisfies a rule r iff $X \models \mathrm{head}(r) \vee \neg \mathrm{body}(r)$.
$\Rightarrow$ X can satisfy all rules and not be an answer set.

Illustration of Gelfond-Lifschitz reduct

Example

$$\begin{array}{llllll} a & \leftarrow & \mathrm{not}\, b. & b & \leftarrow & \mathrm{not}\, a. \\ d & \leftarrow & a. & d. & \leftarrow & b. \end{array}$$

Example

$$a \leftarrow b. \quad b \leftarrow a.$$

Example

$$\begin{array}{cc} woman \leftarrow \mathrm{not}\, n\_woman. & n\_woman \leftarrow \mathrm{not}\, woman. \\ \leftarrow woman, n\_woman. & father \leftarrow parent, n\_woman. \\ mother \leftarrow parent, woman. & parent. \end{array}$$

We say that X satisfies a rule r iff $X \models \mathrm{head}(r) \vee \neg\mathrm{body}(r)$.

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

Illustration of Gelfond-Lifschitz reduct

Example

$$\begin{array}{rclrcl} a & \leftarrow & \text{not}\, b. & b & \leftarrow & \text{not}\, a. \\ d & \leftarrow & a. & d. & \leftarrow & b. \end{array}$$

Example

$$a \leftarrow b. \quad b \leftarrow a.$$

Example

$$\begin{array}{rr} \mathit{woman} \leftarrow \text{not}\, \mathit{n\_woman}. & \mathit{n\_woman} \leftarrow \text{not}\, \mathit{woman}. \\ \leftarrow \mathit{woman}, \mathit{n\_woman}. & \mathit{father} \leftarrow \mathit{parent}, \mathit{n\_woman}. \\ \mathit{mother} \leftarrow \mathit{parent}, \mathit{woman}. & \mathit{parent}. \end{array}$$

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

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

Illustration of Gelfond-Lifschitz reduct

Example

$$a \leftarrow \mathrm{not}\, b. \quad b \leftarrow \mathrm{not}\, a.$$
$$d \leftarrow a. \quad d. \leftarrow b.$$

Example

$$a \leftarrow b. \quad b \leftarrow a.$$

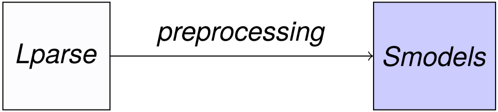
Example

$$woman \leftarrow \mathrm{not}\, n\_woman. \qquad n\_woman \leftarrow \mathrm{not}\, woman.$$
$$\leftarrow woman, n\_woman. \qquad father \leftarrow parent, n\_woman.$$
$$mother \leftarrow parent, woman. \qquad parent.$$

We say that X satisfies a rule r iff $X \models \mathrm{head}(r) \vee \neg\mathrm{body}(r)$.
$\Rightarrow$ 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.



- Allow for using variables and cardinality statements.

Example

```
b :- not a.     a :- not b.
d :- a.         d :- b.
```


The Gelfond-Lifschitz reduct
Language and notations
Formal properties of answer sets
Computation
Logic of here-and-there
SAT translations of ASP

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 ":-".

The lparse format II

- The literal
  `l{b1,...,bm}u`
  is true iff at least *l* and at most *u* atoms are true within the set `{b1,...,bm}`;
- `#domain` encodes the possible values in a given domain:
  `#domain a(X). a(1..10).`
  will replace occurences 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).
c(X) :- not d(X). d(X) :- not c(X).

a(1).  a(2).
c :- not d(1).  c :- not d(2).
d :- not c(1).  d :- not c(2).

1 2 1 1 3
1 4 1 1 5
1 3 1 1 2
1 5 1 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)
```

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).
node(1..5). edge(1,2). edge(3,4).
edge(4,5). edge(4,2). edge(1,4).

uedge(X,Y) :- edge(X,Y), X < Y.
uedge(Y,X) :- edge(X,Y), Y < X.

{ clique(X) : node(X) }.
:- clique(X), clique(Y), not uedge(X,Y), X < Y.

#maximize { clique(X) : node(X) }.
```

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).
node(1..5). edge(1,2). edge(3,4).
edge(4,5). edge(4,2). edge(1,4).

uedge(X,Y) :- edge(X,Y), X < Y.
uedge(Y,X) :- edge(X,Y), Y < X.

{ clique(X) : node(X) }.
:- clique(X), clique(Y), not uedge(X,Y), X < Y.

#maximize { clique(X) : node(X) }.
```

Complexity: existence of answer sets is NP-complete

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

$$p \leftarrow \text{not}\,\hat{p}. \qquad \hat{p} \leftarrow \text{not}\,p.$$

for every proposition p occurring in the clauses, and

$$\leftarrow \text{not}\,l'_1, \text{not}\,l'_2, \text{not}\,l'_3$$

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

Complexity: existence of answer sets is NP-complete

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

$$p \leftarrow \text{not}\,\hat{p}. \qquad \hat{p} \leftarrow \text{not}\,p.$$

for every proposition p occurring in the clauses, and

$$\leftarrow \text{not}\,l'_1, \text{not}\,l'_2, \text{not}\,l'_3$$

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

Complexity: existence of answer sets is NP-complete

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

$$p \leftarrow \text{not}\, \hat{p}. \qquad \hat{p} \leftarrow \text{not}\, p.$$

for every proposition p occurring in the clauses, and

$$\leftarrow \text{not}\, l_1', \text{not}\, l_2', \text{not}\, l_3'$$

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

Complexity: existence of answer sets is NP-complete

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

$$p \leftarrow \text{not}\, \hat{p}. \qquad \hat{p} \leftarrow \text{not}\, p.$$

for every proposition p occurring in the clauses, and

$$\leftarrow \text{not}\, l_1', \text{not}\, l_2', \text{not}\, l_3'$$

for every clause $l_1 \vee l_2 \vee l_3$, 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 Π then A is the head of one of the rules of Π.

Proposition

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

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

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.

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

$\boxed{\Leftarrow}$ 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. □

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.

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

$\boxed{\Leftarrow}$ 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. □

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.

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

$\boxed{\Leftarrow}$ 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. □

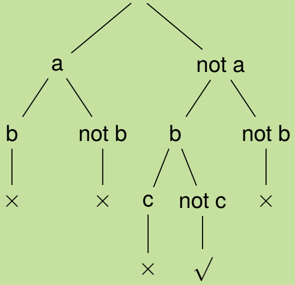
Smodels: principles

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.
```



Smodels I

Algorithm 1 Smodels algorithm

A := $expand(P,A)$
A := $lookahead(P,A)$
if $conflict(P,A)$ **then**
    return false
else if A covers $Atoms(P)$ **then**
    return $stable(P,A)$
else
    x := heuristic(P,A)
    if $smodels(P,A \cup \{X\})$ **then**
        return true
    else
        return $smodels(P,A \cup \{\text{not}\,X\})$
    end if
end if

Smodels example (I)

Example

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

Smodels example (I)

Example

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

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.

Smodels example (I)

Example

$$\begin{array}{llllllll}
(1) & a & \leftarrow & \mathrm{not}\, b, \mathrm{not}\, d. & (2) & d & \leftarrow & \mathrm{not}\, a. \\
(3) & b & \leftarrow & \mathrm{not}\, c. & (4) & c & \leftarrow & \mathrm{not}\, a. \\
(5) & e & \leftarrow & \mathrm{not}\, f, \mathrm{not}\, a. & (6) & f & \leftarrow & \mathrm{not}\, e.
\end{array}$$

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, $\rightarrow b \not\subseteq X$;
- (1) cannot be fired, $\rightarrow a \not\subseteq X$;
- Nothing new to be expanded.

Smodels example (II)

Example

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

Case 2.1: $e \subseteq X$

After reduction:

$$e \leftarrow \text{not}\, f. \qquad f \leftarrow \text{not}\, e.$$

- (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.

Logic of here-and-there

Equivalence between logic programs

Are the two following logic programs

$$\Pi_1 = \quad a \leftarrow \text{not}\, b. \qquad b \leftarrow \text{not}\, a.$$

and

$$\Pi_2 = \quad a \leftarrow \text{not}\, b. \qquad b \leftarrow \text{not}\, c, \text{not}\, a.$$

equivalent?

They are weakly equivalent but not strongly equivalent.

Equivalence between logic programs

Are the two following logic programs

$$\Pi_1 = \quad a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.$$

and

$$\Pi_2 = \quad a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, c, \text{not}\, a.$$

equivalent?

They are weakly equivalent but not strongly equivalent.

Weak equivalence/strong equivalence

Definition (Weak equivalence)

Π_1 and Π_2 are weakly equivalent if they have the same answer sets.

Definition (Strong equivalence)

Π_1 and Π_2 are strongly equivalent if for any Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets.

Example

$$\Pi_1 = \quad a \leftarrow \text{not}\, b. \qquad b \leftarrow \text{not}\, a.$$
$$\Pi_2 = \quad a \leftarrow \text{not}\, b. \qquad b \leftarrow \text{not}\, c, \text{not}\, a.$$

- Do Π_1 and Π_2 have the same answer sets?
- Do $\Pi_1 \cup \{c.\}$ and $\Pi_2 \cup \{c.\}$ have the same answer sets?

Weak equivalence/strong equivalence

Definition (Weak equivalence)

Π_1 and Π_2 are weakly equivalent if they have the same answer sets.

Definition (Strong equivalence)

Π_1 and Π_2 are strongly equivalent if for any Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets.

Example

$$\begin{array}{llllllll} \Pi_1 = & a & \leftarrow & \mathrm{not}\, b. & b & \leftarrow & \mathrm{not}\, a. \\ \Pi_2 = & a & \leftarrow & \mathrm{not}\, b. & b & \leftarrow & \mathrm{not}\, c, \mathrm{not}\, a. \end{array}$$

- Do Π_1 and Π_2 have the same answer sets?
- Do $\Pi_1 \cup \{c.\}$ and $\Pi_2 \cup \{c.\}$ have the same answer sets?

Weak equivalence/strong equivalence

Definition (Weak equivalence)

Π_1 and Π_2 are weakly equivalent if they have the same answer sets.

Definition (Strong equivalence)

Π_1 and Π_2 are strongly equivalent if for any Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets.

Example

$$\begin{array}{lllllll} \Pi_1 = & a & \leftarrow & \mathrm{not}\, b. & b & \leftarrow & \mathrm{not}\, a. \\ \Pi_2 = & a & \leftarrow & \mathrm{not}\, b. & b & \leftarrow & \mathrm{not}\, c, \mathrm{not}\, a. \end{array}$$

- Do Π_1 and Π_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

$$a \leftarrow \mathrm{not}\, b. \quad b \leftarrow \mathrm{not}\, a. \quad c \leftarrow a.$$

Y	Π^Y	X
$\{a,c\}$	$a. \quad c \leftarrow a.$	$\{a,c\}$
$\{b\}$	$b. \quad c \leftarrow a.$	$\{b\}$
$\{b,c\}$	$b. \quad c \leftarrow a.$	$\{b\}, \{b,c\}$
$\{a,b,c\}$	$c \leftarrow a.$	$\{\emptyset\}, \{b\}, \{a,c\}, \{a,b,c\}$

Answer set definition II

Proposition (Characterization of answer sets)

Y is an answer set of Π iff (Y,Y) is an SE-model of Π and there is no (X,Y) within the SE-models of Π such that $X \subsetneq Y$.

Example

$$a \rightarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a. \quad c \leftarrow a.$$

Y	Π^Y	X
$\{a,c\}$	$a. \quad c \leftarrow a.$	$\{a,c\}$
$\{b\}$	$b. \quad c \leftarrow a.$	$\{b\}$
$\{b,c\}$	$b. \quad c \leftarrow a.$	$\{b\},\{b,c\}$
$\{a,b,c\}$	$c \leftarrow a.$	$\{\emptyset\},\{b\},\{a,c\},\{a,b,c\}$

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

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

Strong equivalence: properties

Proposition

The logic programs Π_1 and Π_2 are strongly equivalent iff 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 Π_1 and Π_2.*

Proof.

$\boxed{\Leftarrow}$ Π_1 and Π_2 have the same SE-models. Consider Π. 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. □

Strong equivalence: properties

Proposition

The logic programs Π_1 and Π_2 are strongly equivalent iff they have the same set of SE-models.

Proof.

$\boxed{\Rightarrow}$ Assume $\exists(X,Y) \in SE(\Pi_1)$ and $(X,Y) \notin SE(\Pi_2)$. Two cases:

Case $Y \not\models \Pi_2$ $Y \not\models \Pi_2 \cup Y$ which means Y is not an answer set of $\Pi_2 \cup Y$. On 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 $\Pi_1 \cup Y$.

Case $Y \models \Pi_2$ Take $\Pi = X \cup \{L \leftarrow L' : L, L' \in Y \backslash X\}$. $Y \models \Pi_2 \cup \Pi$, follows $Y \models (\Pi_2 \cup \Pi)^Y$. Let Z be a subset of Y s.t. $Z \models (\Pi_2 \cup \Pi)^Y$ $(= \Pi_2^Y \cup \Pi)$. We know that $X \subseteq Z$ and by assumption $X \not\models \Pi_2^Y$ so $X \neq Z$. There is some $L \in Y \backslash X$ that belongs to Z. It follows that $Y \backslash X \subseteq Z$. Thus, $Z = Y$, and so Y is an answer set $\Pi_2 \cup \Pi$. On contrary, X is proper subset of Y and satisfies $\Pi_1^Y \cup \Pi = (\Pi_1 \cup \Pi)^Y$. Y is not an answer set of $\Pi_1 \cup \Pi$. □

SAT translations of ASP

Dependency graph

Definition (Dependency graph)

The dependency graph of a program Π is the directed graph G such that the vertexes of G are the atoms in Π, and G has an edge from a_0 to $a_1, \ldots, a_m$ for each rule of the form $a_0 \leftarrow a_1, \ldots, a_m, \mathrm{not}\, a_{m+1}, \ldots, \mathrm{not}\, a_n$ in Π with $a_0 \neq \bot$.

Example

$$\Pi = \left\{ \begin{array}{lllllllll} a & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \mathrm{not}\, c. \\ c & \leftarrow & d. & d & \leftarrow & c. & c & \leftarrow & \mathrm{not}\, a. \end{array} \right\}$$

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 $p \equiv B_1 \vee \ldots \vee B_n$ is in $\text{Comp}(\Pi)$. In particular, if $n = 0$ then the equivalence is $p \equiv \bot$, which is equivalent to $\neg p$.
- If $\leftarrow B$ is a constraint in Π, then $\neg B$ is in $\text{Comp}(\Pi)$.

Example

$$\Pi = \left\{ \begin{array}{lllllllll} a & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \text{not}\, c. \\ c & \leftarrow & d. & d & \leftarrow & c. & c & \leftarrow & \text{not}\, a. \end{array} \right\}$$

$$\text{Comp}(\Pi) = \left\{ \begin{array}{llllll} a & \equiv & \neg c \vee b & b & \equiv & a \\ c & \equiv & \neg a \vee d & d & \equiv & c \end{array} \right\}$$

$\text{Comp}(\Pi)$ has 3 models: $\{a, b\}$, $\{c, d\}$ and $\{a, b, c, d\}$.

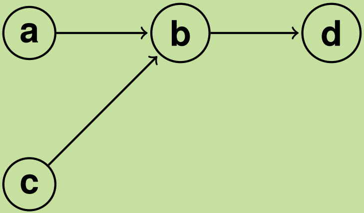
Tight programs

Definition (Tight program)

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

Example

$$\Pi = \left\{ \begin{array}{llllllllllll} d & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \mathrm{not}\, c. \\ d & \leftarrow & b. & b & \leftarrow & c. & c & \leftarrow & \mathrm{not}\, a. \end{array} \right\}$$



Tightness and Clark's completion

Proposition

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

Example

$$\Pi = \left\{ \begin{array}{lllllllll} a & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \mathrm{not}\, c. \\ c & \leftarrow & d. & d & \leftarrow & c. & c & \leftarrow & \mathrm{not}\, a. \end{array} \right\}$$

$$\mathrm{Comp}(\Pi) = \left\{ \begin{array}{llllll} a & \equiv & \neg c \vee b & b & \equiv & a \\ c & \equiv & \neg a \vee d & d & \equiv & c \end{array} \right\}$$

$\mathrm{Comp}(\Pi)$ 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 Π if there exists a grounding sequence for M, i.e., there exists an order $<$ between rules such that for every rule $r \in \Pi$ with $a = \text{head}(r)$ and $M \models \text{body}(r)$, then $\forall b \in \text{body}^+(r), b < a$.

Theorem

If Π is a tight logic program then the model of $Comp(\Pi)$ are exactly the answer sets of Π.

Tightness and Clark's completion (proof)

Proof.

$\boxed{\Rightarrow}$ If X is an answer set of Π, then it is a well-supported model of Π, then it is a minimal Herbrand model of Π, then it is a model of $Comp(\Pi)$.

$\boxed{\Leftarrow}$ Assume that M is model of $Comp(\Pi)$ but not a well-supported model of Π. $\exists x \in M$ that cannot be finitely justified. M being a supported model of (Π), 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 Π is a set L 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 Π whose intermediate nodes belong to L.

$$\begin{array}{lcl} R^+(L,\Pi) & = & \{p \leftarrow G \mid (p \leftarrow G) \in \Pi, p \in L, (\exists q)\ s.t.\ q \in G \wedge q \in L\} \\ R^-(L,\Pi) & = & \{p \leftarrow G \mid (p \leftarrow G) \in \Pi, p \in L, \neg(\exists q)\ s.t.\ q \in G \wedge q \in L\} \end{array}$$

Example

$$\Pi = \left\{ \begin{array}{lllllllll} a & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \mathrm{not}\, c. \\ c & \leftarrow & d. & d & \leftarrow & c. & c & \leftarrow & \mathrm{not}\, a. \end{array} \right\}$$

$$\begin{array}{lcllcl} R^+(L_1,\Pi) & = & \{a \leftarrow b. \quad b \leftarrow a.\} & R^-(L_1,\Pi) & = & \{a \leftarrow \mathrm{not}\, c.\} \\ R^+(L_2,\Pi) & = & \{c \leftarrow d. \quad d \leftarrow c.\} & R^-(L_2,\Pi) & = & \{c \leftarrow \mathrm{not}\, a.\} \end{array}$$

Loop formulas

Definition (Loop formulas)

Let $R^{-}(L,\Pi)$ be the following rules:

$$\begin{array}{ccc} p_1 \leftarrow B_{11} & \cdots & p_1 \leftarrow B_{1k_1} \\ & \vdots & \\ p_n \leftarrow B_{n1} & \cdots & p_n \leftarrow B_{nk_n} \end{array}$$

The loop formula associated with L is the following implication:

$$\neg[B_{11} \vee \ldots \vee B_{1k_1} \vee \ldots \vee B_{n1} \vee \ldots \vee B_{nk_n}] \rightarrow \bigwedge_{p \in L} \neg p$$

Example

$$\begin{array}{lclcl} R^{+}(L_1,\Pi) & = & \{a \leftarrow b. \quad b \leftarrow a.\} & R^{-}(L_1,\Pi) & = & \{a \leftarrow \mathrm{not}\, c.\} \\ R^{+}(L_2,\Pi) & = & \{c \leftarrow d. \quad d \leftarrow c.\} & R^{-}(L_2,\Pi) & = & \{c \leftarrow \mathrm{not}\, a.\} \end{array}$$

$$LF(L_1) : c \rightarrow (\neg a \wedge \neg b) \qquad LF(L_2) : a \rightarrow (\neg c \wedge \neg d)$$

Clark + loop formulae

Theorem

Let Π be a logic program, then the models of $Comp(\Pi) \cup LF(\Pi)$ are exactly the answer sets of Π.

Example

$$\Pi = \left\{ \begin{array}{lllllllll} a & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \mathrm{not}\, c. \\ c & \leftarrow & d. & d & \leftarrow & c. & c & \leftarrow & \mathrm{not}\, a. \end{array} \right\}$$

$$\mathrm{Comp}(\Pi) \cup LF(\Pi) = \left\{ \begin{array}{llllll} a & \equiv & \neg c \vee b & b & \equiv & a \\ c & \equiv & \neg a \vee d & d & \equiv & c \\ c & \rightarrow & (\neg a \wedge \neg b) & a & \rightarrow & (\neg c \wedge \neg d) \end{array} \right\}$$

CLASP translation I

Definition (Body clauses)

Let β be a body of a rule $\beta = \{p_1, \ldots, p_m, \mathrm{not}\, p_{m+1}, \ldots, \mathrm{not}\, p_n\}$, then:

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

Example

$$\Pi = \left\{ \begin{array}{llllll} a \leftarrow b. & b \leftarrow a. & a \leftarrow \mathrm{not}\, c. \\ c \leftarrow d. & d \leftarrow c. & c \leftarrow \mathrm{not}\, a. \end{array} \right\}$$

$$\Pi = \left\{ \begin{array}{cccccc} \beta_1 \vee \neg b & \beta_2 \vee \neg a & \beta_3 \vee c & \beta_4 \vee \neg d & \beta_5 \vee \neg c & \beta_6 \vee a \\ \neg\beta_1 \vee b & \neg\beta_2 \vee a & \neg\beta_3 \vee \neg c & \neg\beta_4 \vee d & \neg\beta_5 \vee c & \neg\beta_6 \vee \neg a \end{array} \right\}$$

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:

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

Example

$$\Pi = \left\{ \begin{array}{lllllllll} a & \leftarrow & b. & b & \leftarrow & a. & a & \leftarrow & \mathrm{not}\, c. \\ c & \leftarrow & d. & d & \leftarrow & c. & c & \leftarrow & \mathrm{not}\, a. \end{array} \right\}$$

$$\Pi = \left\{ \begin{array}{cccc} a \vee \neg\beta_1 & b \vee \neg\beta_2 & a \vee \neg\beta_3 & c \vee \neg\beta_4 \\ & d \vee \neg\beta_5 & c \vee \neg\beta_6 & \\ \neg a \vee \beta_1 \vee \beta_3 & \neg b \vee \beta_2 & \neg c \vee \beta_4 \vee \beta_6 & \neg d \vee \beta_5 \end{array} \right\}$$

CLASP translation III

Definition (External body)

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

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

Definition (Loop clause)

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

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

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

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

CLASP translation IV

Proposition

X is an answer set of Π iff $X \cap Atoms(\Pi)$ 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.
ICLP/SLP, p.1070-1080, 1988.

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.
TPLP, p609-622, 2003.

Literature

Martin Gebser and Benjamin Kaufmann and André Neumann and Torsten Schaub.
Conflict-Driven Answer Set Solving.
IJCAI, p.386-393, 2007.

Ilkka Niemelä and Patrik Simons
Efficient Implementation of the Well-founded and Stable Model Semantics.
JICSLP, p.289-303, 1996.