Principles of Knowledge Representation and Reasoning

Answer Set Programming

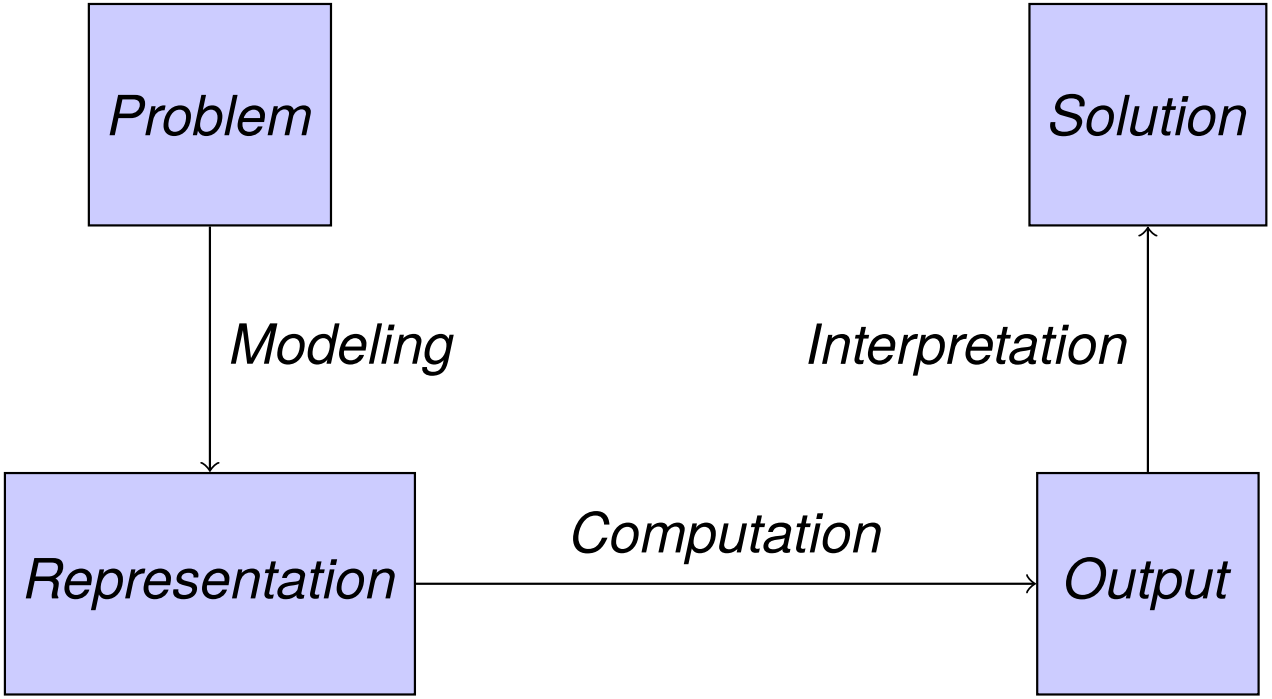
Albert-Ludwigs-Universität Freiburg

Bernhard Nebel, Stefan Wölfl, and Julien Hué

November 25, 2013

Declarative problem Solving

- What is the problem? instead of How to solve the problem?
- Outsourcing the Computation part to an external Solver



Negation as failure

- Another interpretation for negation: $\text{not}\, x \equiv$ "I cannot show that x is true"
- For example, you are innocent until proven guilty

Example

$$\mathit{innocent} \leftarrow \text{not}\, \mathit{guilty}.$$

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

Nonmonotonic logic programs I

Let $\mathcal{A}$ be a set of first-order atoms.

Rules:

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

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

- Meaning similar to default logic:
  If
  1. **we have derived** $b_1, \ldots, b_m$ **and**
  2. **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):

$$\bot \leftarrow b_1, \ldots, b_m, \text{not}\, d_1, \ldots, \text{not}\, d_k$$

Nonmonotonic logic programs II

Let $\mathcal{A}$ be a set of first-order atoms.

Rules:

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

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

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

Nonmonotonic logic programs: examples

Example

$$fly(tweety) \leftarrow bird(tweety), \text{not}\, abnormal(tweety).$$
$$bird(tweety) \leftarrow penguin(tweety).$$
$$abnormal(tweety) \leftarrow penguin(tweety).$$

Nonmonotonic logic programs: examples

Example

$$fly(tweety) \leftarrow bird(tweety), \text{not}\, abnormal(tweety).$$
$$bird(tweety) \leftarrow penguin(tweety).$$
$$abnormal(tweety) \leftarrow penguin(tweety).$$

Interpretation and Satisfiability

Herbrand base and Grounded rules

- The Herbrand Universe, denoted by U_Π, is the set of ground terms constructed from the function symbols and constants in Π
- The Herbrand Base, denoted by B_Π, is the set of ground atoms constructed from predicate symbols and ground terms from the Herbrand Universe
- From now on, a program will refer to the set of its grounded rules
- The set of atoms in Π is denoted by $Atoms(\Pi)$

Herbrand base and Grounded rules

Example

$$\Pi = \begin{cases} fly(X) & \leftarrow & bird(X), \text{not}\, abnormal(X). \\ bird(X) & \leftarrow & penguin(X). \\ abnormal(X) & \leftarrow & penguin(X). \\ penguin(t) & \leftarrow & \top. \end{cases}$$

$U_\Pi = \{t\}$

$B_\Pi = \{fly(t), bird(t), abnormal(t), penguin(t)\}$

$$\Pi = \begin{cases} fly(t) & \leftarrow & bird(t), \text{not}\, abnormal(t). \\ bird(t) & \leftarrow & penguin(t). \\ abnormal(t) & \leftarrow & penguin(t). \\ penguin(t) & \leftarrow & \top. \end{cases}$$

Herbrand base and Grounded rules

Example

$$\Pi = \begin{cases} fly(X) & \leftarrow & bird(X), \text{not}\, abnormal(X). \\ bird(X) & \leftarrow & penguin(X). \\ abnormal(X) & \leftarrow & penguin(X). \\ penguin(t) & \leftarrow & \top. \end{cases}$$

$$U_\Pi = \{t\}$$
$$B_\Pi = \{fly(t), bird(t), abnormal(t), penguin(t)\}$$

$$\Pi = \begin{cases} fly(t) & \leftarrow & bird(t), \text{not}\, abnormal(t). \\ bird(t) & \leftarrow & penguin(t). \\ abnormal(t) & \leftarrow & penguin(t). \\ penguin(t) & \leftarrow & \top. \end{cases}$$

Herbrand base and Grounded rules

Example

$$\Pi = \begin{cases} fly(X) & \leftarrow & bird(X), \text{not}\, abnormal(X). \\ bird(X) & \leftarrow & penguin(X). \\ abnormal(X) & \leftarrow & penguin(X). \\ penguin(t) & \leftarrow & \top. \end{cases}$$

$$U_\Pi = \{t\}$$
$$B_\Pi = \{fly(t), bird(t), abnormal(t), penguin(t)\}$$

$$\Pi = \begin{cases} fly(t) & \leftarrow & bird(t), \text{not}\, abnormal(t). \\ bird(t) & \leftarrow & penguin(t). \\ abnormal(t) & \leftarrow & penguin(t). \\ penguin(t) & \leftarrow & \top. \end{cases}$$

Satisfaction

An Herbrand Interpretation is a subset I of the Herbrand Base.

- $I \models a$ if $a \in I$
- $I \models \text{head}(r)$ if $\text{head}(r) \cap I \neq \emptyset$
- $I \models \text{body}(r)$ if $\text{body}^+(r) \subseteq I$ and $\text{body}^-(r) \cap I \neq \emptyset$
- I satisfies a rule r if $I \models \text{head}(r)$ or $I \not\models \text{body}(r)$
- I satisfies a program if it satisfies all its rules

Idea

The idea is that a solution should both satisfying AND justified

Satisfaction

An Herbrand Interpretation is a subset I of the Herbrand Base.

- $I \models a$ if $a \in I$
- $I \models \text{head}(r)$ if $\text{head}(r) \cap I \neq \emptyset$
- $I \models \text{body}(r)$ if $\text{body}^+(r) \subseteq I$ and $\text{body}^-(r) \cap I \neq \emptyset$
- I satisfies a rule r if $I \models \text{head}(r)$ or $I \not\models \text{body}(r)$
- I satisfies a program if it satisfies all its rules

Idea

The idea is that a solution should both satisfying AND justified

not-free logic programs

Definition (Answer Set)

Let Π be a logic program without **not**, $X \subseteq \text{Atoms}(\Pi)$.
X is the unique Answer Set of Π if it is the least fixpoint of $\Gamma_\Pi(X) = \{\text{head}(r) \mid X \models \text{body}(r)\}$.

Example

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

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

not-free logic programs

Definition (Answer Set)

Let Π be a logic program without **not**, $X \subseteq \text{Atoms}(\Pi)$.
X is the unique Answer Set of Π if it is the least fixpoint of $\Gamma_\Pi(X) = \{\text{head}(r) \mid X \models \text{body}(r)\}$.

Example

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

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

not-free logic programs

Definition (Answer Set)

Let Π be a logic program without **not**, $X \subseteq \text{Atoms}(\Pi)$.
X is the unique Answer Set of Π if it is the least fixpoint of $\Gamma_\Pi(X) = \{\text{head}(r) \mid X \models \text{body}(r)\}$.

Example

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

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

not-free logic programs

Definition (Answer Set)

Let Π be a logic program without **not**, $X \subseteq \text{Atoms}(\Pi)$.
X is the unique Answer Set of Π if it is the least fixpoint of $\Gamma_\Pi(X) = \{\text{head}(r) \mid X \models \text{body}(r)\}$.

Example

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

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

not-free logic programs

Definition (Answer Set)

Let Π be a logic program without **not**, $X \subseteq \text{Atoms}(\Pi)$.
X is the unique Answer Set of Π if it is the least fixpoint of $\Gamma_\Pi(X) = \{\text{head}(r) \mid X \models \text{body}(r)\}$.

Example

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

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

Definition 1: Gelfond-Lifschitz reduct

- Deleting all rules whose negative part contradicts X
- Removing all negated atoms from the remaining rules

Definition (Reduct)

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

$$\Pi^X := \{\text{head}(r) \leftarrow \text{body}^+(r) \mid r \in \Pi, \text{body}^-(r) \cap X = \emptyset\}$$

Definition (Answer set)

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

Definition 1: Gelfond-Lifschitz reduct

- Deleting all rules whose negative part contradicts X
- Removing all negated atoms from the remaining rules

Definition (Reduct)

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

$$\Pi^X := \{\text{head}(r) \leftarrow \text{body}^+(r) \mid r \in \Pi, \text{body}^-(r) \cap X = \emptyset\}$$

Definition (Answer set)

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

Illustration of Gelfond-Lifschitz reduct

Example

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

Example

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

Example

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

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

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

Example

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

Example

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

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

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

Example

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

Example

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

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

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

Example

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

Example

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

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.

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:

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

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:

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

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:

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

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 $(F \cup G)^X$ whose least fixpoint is the same as $F^X \cup \neg\bot$ 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. $\square$

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 $(F \cup G)^X$ whose least fixpoint is the same as $F^X \cup \neg\bot$ 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. $\square$

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.

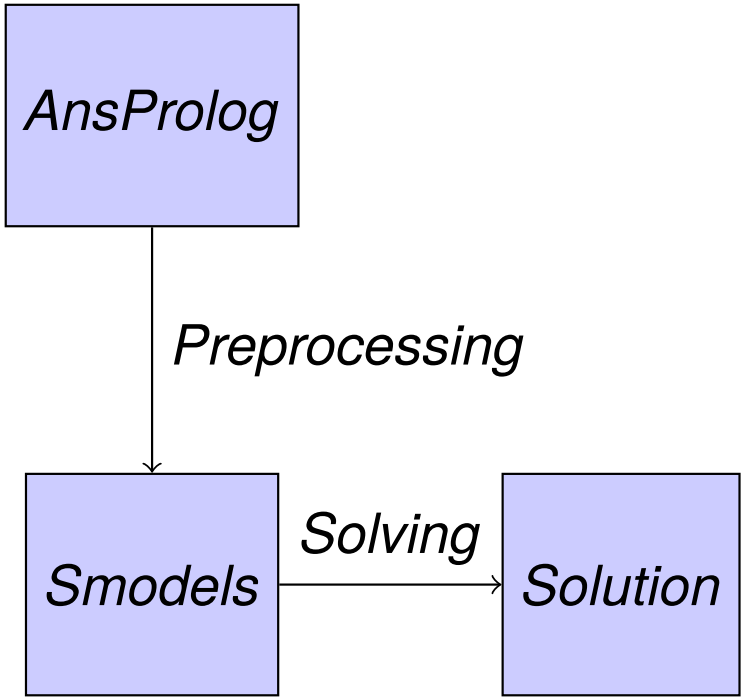
Proof.

$\boxed{\Rightarrow}$ X satisfies $F \cup G$. Then X satisfies the constraints in G and $(F \cup G)^X$ whose least fixpoint is the same as $F^X \cup \neg\bot$ 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. $\square$

AnsProlog and Solvers

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



The lparse format: AnsProlog

- 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 ":-".

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$.

```
a(1) :- not na(1).    na(1) :- not a(1).
a(2) :- not na(2).    na(2) :- not a(2).
a(3) :- not na(3).    na(3) :- not a(3).

{ a(1), a(2), a(3) }.
```

The lparse format: AnsProlog

- 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 ":-".

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$.

```
a(1) :- not na(1).    na(1) :- not a(1).
a(2) :- not na(2).    na(2) :- not a(2).
a(3) :- not na(3).    na(3) :- not a(3).

{ a(1), a(2), a(3) }.
```

AnsProlog (choice functions)

- The literal `{b1,...,bm}`
  is true iff any subset of the set `{b1,...,bm}` is true;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$.

```
a(1) :- not na(1).    na(1) :- not a(1).
a(2) :- not na(2).    na(2) :- not a(2).
a(3) :- not na(3).    na(3) :- not a(3).

{ a(1), a(2), a(3) }.
```

- The `#hide` statement can hide literals from the solution

AnsProlog (choice functions)

- The literal `{b1,...,bm}`
  is true iff any subset of the set `{b1,...,bm}` is true;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$.

```
a(1) :- not na(1).    na(1) :- not a(1).
a(2) :- not na(2).    na(2) :- not a(2).
a(3) :- not na(3).    na(3) :- not a(3).

{ a(1), a(2), a(3) }.
```

- The `#hide` statement can hide literals from the solution

AnsProlog (choice functions)

- The literal `{b1,...,bm}`
  is true iff any subset of the set `{b1,...,bm}` is true;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$.

```
a(1) :- not na(1).    na(1) :- not a(1).
a(2) :- not na(2).    na(2) :- not a(2).
a(3) :- not na(3).    na(3) :- not a(3).

{ a(1), a(2), a(3) }.
```

- The `#hide` statement can hide literals from the solution

AnsProlog (choice functions)

- The literal `{b1,...,bm}`
  is true iff any subset of the set `{b1,...,bm}` is true;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$.

```
a(1) :- not na(1).    na(1) :- not a(1).
a(2) :- not na(2).    na(2) :- not a(2).
a(3) :- not na(3).    na(3) :- not a(3).

{ a(1), a(2), a(3) }.
```

- The `#hide` statement can hide literals from the solution

AnsProlog (cardinality)

- The literal `l{b1,...,bm}u`
  is true iff at least *l* and at most *u* atoms (included) are true within the set `{b1,...,bm}`;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$ that contain 2 true atoms.

```
2 { a(1),a(2),a(3) } 2.
```

I want all interpretations over $\{a(1), a(2), a(3)\}$ that donot contain 2 true atoms.

```
{ a(1), a(2), a(3) }.
:- 2 { a(1),a(2),a(3) } 2.
```

AnsProlog (cardinality)

- The literal `l{b1,...,bm}u`
  is true iff at least *l* and at most *u* atoms (included) are true within the set `{b1,...,bm}`;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$ that contain 2 true atoms.

```
2 { a(1),a(2),a(3) } 2.
```

I want all interpretations over $\{a(1), a(2), a(3)\}$ that donot contain 2 true atoms.

```
{ a(1), a(2), a(3) }.
:- 2 { a(1),a(2),a(3) } 2.
```

AnsProlog (cardinality)

- The literal `l{b1,...,bm}u`
  is true iff at least *l* and at most *u* atoms (included) are true within the set `{b1,...,bm}`;

Example

I want all interpretations over $\{a(1), a(2), a(3)\}$ that contain 2 true atoms.

```
2 { a(1),a(2),a(3) } 2.
```

I want all interpretations over $\{a(1), a(2), a(3)\}$ that donot contain 2 true atoms.

```
{ a(1), a(2), a(3) }.
:- 2 { a(1),a(2),a(3) } 2.
```

AnsProlog (domains and conditionals)

- The domains of a variable can be set literal-wise, rule-wise or program wise.
- For a scope limited to a literal:
  `clique(X) : num(X). num(1..3).`
  will be understood as
  `clique(1). clique(2). clique(3).`

Example

```
num(1..3).
{ a(X) : num(X) }.
:- 2 { a(X) : num(X) } 2.
```

AnsProlog (domains and conditionals)

- The domains of a variable can be set literal-wise, rule-wise or program-wise.
- For a scope limited to a rule:
  `a(X) :- not na(X), num(X).`
  X takes all the values for which $num(X)$ is stated as a fact.

Example

Interpretations over $\{a,b,c\}$ that does not contain 2 true atoms.

```
num(1..3).
a(X) :- not na(X), num(X).    na(X) :- not a(X), num(X).
:- 2 { a(X) : num(X) } 2.
```

AnsProlog (domains)

- Values can be associated to a variable within the scope of the whole logic program
- `#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

Example

I want all tuples (x,y) (x and y integers between 1 and 10).

```
a(1..10).        b(1..10).
tuple(X,Y) :- a(X), b(Y).

#domain a(X).  a(1..10).
#domain a(Y).  tuple(X,Y).
```

AnsProlog (domains)

- Values can be associated to a variable within the scope of the whole logic program
- `#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

Example

I want all tuples (x,y) (x and y integers between 1 and 10).

```
a(1..10).       b(1..10).
tuple(X,Y) :- a(X), b(Y).

#domain a(X).  a(1..10).
#domain a(Y).  tuple(X,Y).
```

AnsProlog (domains)

- Values can be associated to a variable within the scope of the whole logic program
- `#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

Example

I want all tuples (x,y) (x and y integers between 1 and 10).

```
a(1..10).       b(1..10).
tuple(X,Y) :- a(X), b(Y).

#domain a(X).  a(1..10).
#domain a(Y).  tuple(X,Y).
```

AnsProlog (Domain restriction)

- 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`.

Example

I want all tuples (x,y) (x and y integers between 1 and 10) s.t. x<y.

```
a(1..10).        b(1..10).
tuple(X,Y) :- a(X), b(Y), X<Y.

#domain a(X).  a(1..10).
#domain a(Y).  tuple(X,Y) :- X<Y.
```

AnsProlog (Domain restriction)

- 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`.

Example

I want all tuples (x,y) (x and y integers between 1 and 10) s.t. x<y.

```
a(1..10).        b(1..10).
tuple(X,Y) :- a(X), b(Y), X<Y.

#domain a(X).  a(1..10).
#domain a(Y).  tuple(X,Y) :- X<Y.
```

AnsProlog (Domain restriction)

- 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`.

Example

I want all tuples (x,y) (x and y integers between 1 and 10) s.t. x<y.

```
a(1..10).        b(1..10).
tuple(X,Y) :- a(X), b(Y), X<Y.

#domain a(X).  a(1..10).
#domain a(Y).  tuple(X,Y) :- X<Y.
```

AnsProlog (Domain restriction)

- 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`.

Example

Queen problem with board of size 5.

```
row(1..5).                col(1..5).
{queen(I,J) : row(I) : col(J)}.
:- not 5 {queen(I,J)} 5.
:- queen(I,J), queen(I,JJ), J != JJ.
:- queen(I,J), queen(II,J), I != II.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J == II-JJ.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
```

AnsProlog (Domain restriction)

- 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`.

Example

Queen problem with board of size 5.

```
row(1..5).                col(1..5).
{queen(I,J) : row(I) : col(J)}.
:- not 5 {queen(I,J)} 5.
:- queen(I,J), queen(I,JJ), J != JJ.
:- queen(I,J), queen(II,J), I != II.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J == II-JJ.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
```

AnsProlog (optimization)

- 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 lesser number of atoms from `{a,b,c,d}`.
  Attention: Does not change the SAT/UNSAT question, just the answer sets themselves.

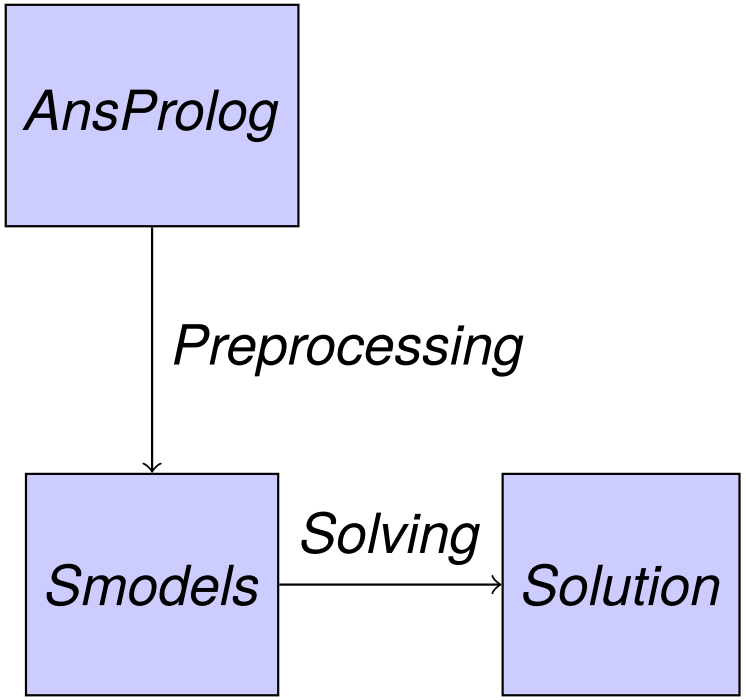
AnsProlog (Miscellaneous)

The language is even bigger than that! It includes

- Disjunction in the head
- Other operators: `#sum`,`#min`,`#max`,`#even`,`#odd`,`#avg`, ...
- Multi-criteria optimizations
- Heuristic optimizations
- ...

AnsProlog and Solvers

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



Lparse and smodels II

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)
```

Lparse and smodels II

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)
```

Lparse and smodels II

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?

- First, define what is a "solution candidate"
- Second, verify it fits the constraints
- Then, 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?

- First, define what is a "solution candidate"
- Second, verify it fits the constraints
- Then, 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) }.
```

Another Example: Sudoku

Example

```
#domain num(X). #domain num(X1). #domain num(Z).
#domain num(Y). #domain num(Y1).
#domain three(W). #domain three(W1). #domain three(W2).
#domain three(W3). #domain three(W4). #domain three(W5).
num(1..9). three(1..3).

sol(2,6,3).  sol(2,8,8).  sol(2,9,5).
sol(3,3,1).  sol(3,5,2).  sol(4,4,5).
sol(4,6,7).  sol(5,3,4).  sol(5,7,1).
sol(6,2,3).  sol(7,1,5).  sol(7,8,7).
sol(7,9,3).  sol(8,3,2).  sol(8,5,1).
sol(9,5,4).  sol(9,9,9).
1 { sol(X,Y,A) : num(A) } 1.
:- sol(X,Y,Z), sol(X,Y1,Z), Y != Y1.
:- sol(X,Y,Z), sol(X1,Y,Z), X != X1.
:- sol(W*3+W2,W1*3+W3,Z), sol(W*3+W4,W1*3+W5,Z), W3 != W5.
:- sol(W*3+W2,W1*3+W3,Z), sol(W*3+W4,W1*3+W5,Z), W2 != W4.
```

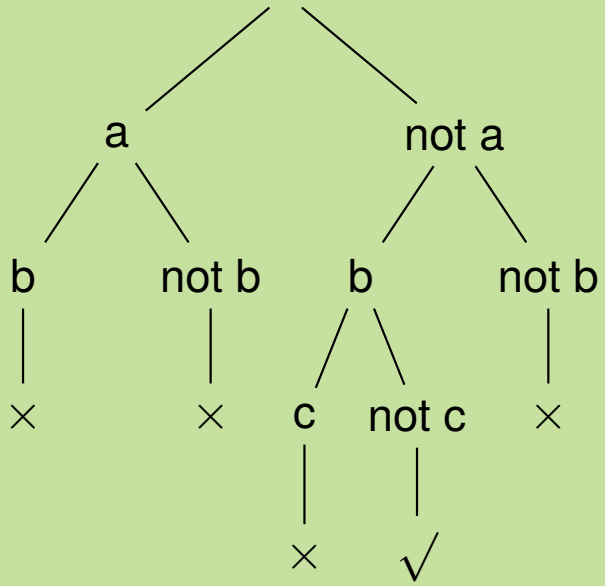
Smodels: principles

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 ∪ {X} then
        return true
    else
        return smodels(P,A ∪ {not X}
    end if
end if
```

Smodels example (I)

Example

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

Smodels example (I)

Example

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

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

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

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.

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

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

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.

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 $a_0 \leftarrow a_1, \ldots, a_m, \text{not}\, a_{m+1}, \ldots, \text{not}\, a_n$ in Π with $a_0 \neq \bot$.

Example

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

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

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

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

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

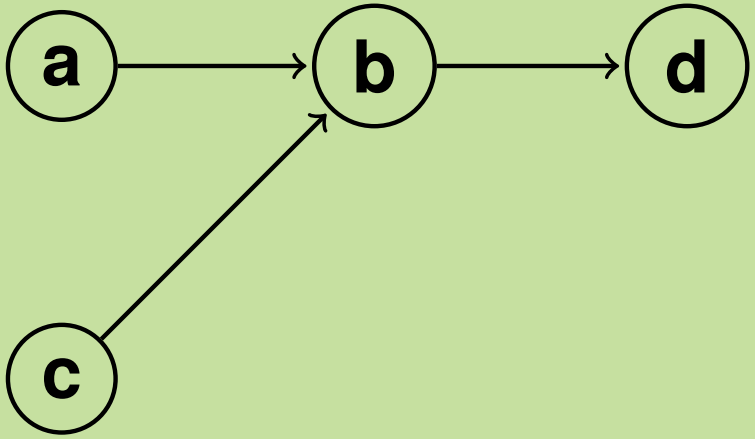
Tight programs

Definition (Tight program)

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

Example

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



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

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

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

Comp(Π) 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. $\square$

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.

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

Example

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

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

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

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

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

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

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

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

CLASP translation I

Definition (Body clauses)

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

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

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

CLASP translation II

Definition (Atoms clauses)

Let p be an atom appearing as head of rules whose body are $\{\beta_1, ..., \beta_k\}$, then:

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

Example

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

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

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.