**Interpretation and Satisfiability of ASP**

**Negation as failure**

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

Example

\[
\text{innocent } \leftarrow \text{not guilty}.
\]

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

Rules:
\[ c \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k \]
where \(\{c, b_1, \ldots, b_m, d_1, \ldots, d_k\} \subseteq \mathcal{A}\)

- 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):
  - $\bot \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k$

### Example

\[ \text{fly}(\text{tweety}) \leftarrow \text{bird}(\text{tweety}), \neg \text{abnormal}(\text{tweety}). \]
\[ \text{bird}(\text{tweety}) \leftarrow \text{penguin}(\text{tweety}). \]
\[ \text{abnormal}(\text{tweety}) \leftarrow \text{penguin}(\text{tweety}). \]
Herbrand base and Grounded rules

- The Herbrand Universe, denoted by $U_\Pi$, is the set of ground terms constructed from the function symbols and constants in $\Pi$.
- The Herbrand Base, denoted by $B_\Pi$, 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 $\Pi$ is denoted by $\text{Atoms}(\Pi)$.

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

Example

$$\Pi = \{\text{fly}(X) \leftarrow \text{bird}(X), \text{not abnormal}(X). \}$$

$$U_\Pi = \{t\}$$

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

$$\Gamma_0 = \Gamma(\emptyset) = \{b\}$$

$$\Gamma_1 = \Gamma(\Gamma_0) = \{b,d,a\}$$

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

$$\Gamma_3 = \Gamma(\Gamma_2) = \{b,d,a,c\} = \Gamma_2$$
Definition 1: Gelfond-Lifschitz reduct

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

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

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

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

Complexity: existence of answer sets is NP-complete

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

\[
p \leftarrow \neg \hat{p}. \quad \hat{p} \leftarrow \neg p.
\]

for every proposition \( p \) occurring in the clauses, and

\[
\leftarrow \neg \hat{l}'_1, \neg \hat{l}'_2, \neg \hat{l}'_3
\]

for every clause \( l_1 \lor l_2 \lor l_3 \), where \( \hat{l}'_i = p \) if \( l_i = p \) and \( \hat{l}'_i = \hat{p} \) 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 $(F \cup G)^X$ 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 $F^X$ if 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 $G$.

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) \}.

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
Remember that a literal $l\{b_1,\ldots,b_m\}u$ is true if at least $l$ and at most $u$ atoms (included) are true within the set $\{b_1,\ldots,b_m\}$.

**Example**

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

$$\exists (a(1),a(2),a(3))$$ \iff 2

I want all interpretations over $\{a(1),a(2),a(3)\}$ that do not contain 2 true atoms.

$$\{a(1),a(2),a(3)\}$$ \iff \neg 2

---

The domains of a variable can be set literal-wise, rule-wise or program-wise.

**Example**

For a scope limited to a rule:

$$a(X) := \neg na(X), \text{num}(X).$$

$X$ takes all the values for which $\text{num}(X)$ is stated as a fact.

**Example**

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

$$\text{num}(1..10). a(X) := \neg \text{na}(X), \text{num}(X).$$

$$\text{na}(X) := \neg a(X), \text{num}(X).$$

$$\exists (a(X) : \text{num}(X)).$$

$$\neg 2 \{ a(X) : \text{num}(X) \}.$$
**AnsProlog (Domain restriction)**

- Domains can be restricted thanks to relations. The rule:
  \[- \text{size}(X,Y), X < Y.\]
  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.

\[a(1..10). \quad b(1..10). \quad \text{tuple}(X,Y) :- a(X), b(Y), X < Y.\]

**AnsProlog (optimization)**

- A subset of answer sets can be selected according to some optimization criteria.
  \[\#\text{minimize}\{a,b,c,d\}.\]
  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.

**AnsProlog**

### Preprocessing

**Smodels**

**Solving**

**Solution**

Lparse and smodels II

**Example**

```prolog
#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**

```prolog
#domain node(X). #domain node(Y).
node(1..5). edge(1,2). edge(3,4).
edge(4,5). edge(4,2). edge(1,4).

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

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

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

Another Example: Sudoku

**Example**

```prolog
#domain num(X). #domain num(X1). #domain num(Y).
#domain num(Y1). #domain num(Z).
#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).
```

```prolog
1 { sol(X,Y,Z) : num(A) } 1.
:- sol(X,Y,Z), sol(X,Y1,Z), Y != Y1.
:- sol(X,Y,Z), sol(X1,Y,Z), X != X1.
```
Smodels: principles

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

Example

\[
\begin{array}{c}
a \leftarrow \neg b. \\
b \leftarrow \neg a. \\
c \leftarrow \neg c, a.
\end{array}
\]

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

Smodels example (I)

Example

\[
\begin{array}{c}
(1) a \leftarrow \neg b, \neg d. \\
(2) d \leftarrow \neg a. \\
(3) b \leftarrow \neg c. \\
(4) c \leftarrow \neg a. \\
(5) e \leftarrow \neg f, \neg a. \\
(6) f \leftarrow \neg e.
\end{array}
\]

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

\[
\begin{array}{c}
(1) a \leftarrow \neg b, \neg d. \\
(2) d \leftarrow \neg a. \\
(3) b \leftarrow \neg c. \\
(4) c \leftarrow \neg a. \\
(5) e \leftarrow \neg f, \neg a. \\
(6) f \leftarrow \neg e.
\end{array}
\]

Case 2.1: \( e \subseteq X \)

After reduction:
- \( e \leftarrow \neg f \).
- \( f \leftarrow \neg e \).
- (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.
Interpretation and Satisfiability
SAT Translations of ASP
Positive-order consistent logic programs
Clark’s completion
CLASP solver

2 SAT translations of ASP

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

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 $a_0 ← a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$ in $\Pi$ with $a_0 \not= \bot$.

Example
$$\Pi = \{ a ← b. \ b ← a. \ a ← \neg c. \ c ← d. \ d ← c. \ c ← \neg a. \}$$

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
$$\Pi = \{ d ← b. \ b ← a. \ a ← \neg c. \ c ← \neg a. \}$$
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

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

\[
\text{Comp}(\Pi) = \{ a \equiv \neg c \lor b \quad b \equiv a \}\quad\{ a \equiv \neg c \lor b \quad d \equiv c \}
\]

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

Tightness and Clark’s completion (proof)

Proof.

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

\( \Leftarrow \) 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 \( \exists x \in \Pi \) 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.

Tightness and Clark’s completion (proof) (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 \( < \) 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 \( \Pi \) is a tight logic program then the model of \( \text{Comp}(\Pi) \) are exactly the answer sets of \( \Pi \).

Loops

Definition (Loop)

A loop of \( \Pi \) 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 \( \Pi \) whose intermediate nodes belong to \( L \).

\[
R^+(L,\Pi) = \{ p \leftarrow G \mid (p \leftarrow G) \in \Pi, p \in L, (\exists q) s.t. q \in G \land q \in L \} \\
R^-(L,\Pi) = \{ p \leftarrow G \mid (p \leftarrow G) \in \Pi, p \in L, (\exists q) s.t. q \in G \land q \in L \}
\]

Example

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

\[
R^+(L_1,\Pi) = \{ a \leftarrow b. \quad b \leftarrow a. \} \quad R^-(L_1,\Pi) = \{ a \leftarrow \text{not.c.} \}
\]

\[
R^+(L_2,\Pi) = \{ c \leftarrow d. \quad d \leftarrow c. \} \quad R^-(L_2,\Pi) = \{ c \leftarrow \text{not.a.} \}
\]
Loop formulas

Definition (Loop formulas)
Let $R^- (L, \Pi)$ be the following rules:
\[
p_1 \leftarrow B_{11} \ldots p_1 \leftarrow B_{1k_1} \\
\vdots \\
p_n \leftarrow B_{n1} \ldots p_n \leftarrow B_{nk_n}
\]
The loop formula associated with $L$ is the following implication:
\[
\neg [B_{11} \lor \ldots \lor B_{1k_1} \lor \ldots \lor B_{n1} \lor \ldots \lor B_{nk_n}] \rightarrow \bigwedge_{p \in L} \neg p
\]

Example
\[
R^+ (L_1, \Pi) = \{a \leftarrow b, b \leftarrow a\} \quad R^- (L_1, \Pi) = \{a \leftarrow \text{notc.}\} \\
R^+ (L_2, \Pi) = \{c \leftarrow d, d \leftarrow c\} \quad R^- (L_2, \Pi) = \{c \leftarrow \text{nota.}\}
\]
\[
LF (L_1): c \rightarrow (\neg a \land \neg b) \quad LF (L_2): a \rightarrow (\neg c \land \neg d)
\]

Clark + loop formulae

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

Example
\[
\Pi = \{a \leftarrow b, b \leftarrow a, a \leftarrow \text{notc.}\}
\]
\[
\text{Comp}(\Pi) \cup LF(\Pi) = \{a \equiv \neg c \lor b, b \equiv a, c \equiv \neg a \lor d, d \equiv c, c \rightarrow (\neg a \land \neg b), a \rightarrow (\neg c \land \neg d)\}
\]

CLASP translation I

Definition (Body clauses)
Let $\beta$ be a body of a rule $\beta = \{\rho_1, ..., \rho_m, \text{not} \rho_{m+1}, ..., \text{not} \rho_n\}$, then:
\[
\delta(\beta) = \{\beta \lor \neg \rho_1 \lor \ldots \lor \neg \rho_{m+1} \lor \ldots \lor \neg \rho_n\}
\]
\[
\Delta(\beta) = \{\neg \beta \lor \rho_1, \ldots, \neg \beta \lor \rho_m, \neg \beta \lor \neg \rho_{m+1}, \ldots, \neg \beta \lor \neg \rho_n\}
\]

Example
\[
\Pi = \{a \leftarrow b, b \leftarrow a, a \leftarrow \text{notc.}\}
\]
\[
\Pi = \{\beta_1 \lor \neg b, \beta_2 \lor \neg a, \beta_2 \lor c, \beta_4 \lor \neg d, \beta_5 \lor \neg c, \beta_6 \lor a, \neg \beta_1 \lor b, \neg \beta_2 \lor a, \neg \beta_2 \lor c, \neg \beta_4 \lor d, \neg \beta_5 \lor c, \neg \beta_6 \lor a\}
\]

CLASP translation II

Definition (Atoms clauses)
Let $p$ be an atom appearing as head of rules whose body are $\{\beta_1, ..., \beta_k\}$, then:
\[
\Delta(p) = \{p \lor \neg \beta_1, ..., p \lor \neg \beta_k\}
\]
\[
\delta(p) = \{\neg p \lor \beta_1 \lor \ldots \lor \beta_k\}
\]

Example
\[
\Pi = \{a \leftarrow b, b \leftarrow a, a \leftarrow \text{notc.}\}
\]
\[
\Pi = \{a \lor \neg \beta_1, b \lor \neg \beta_2, a \lor \neg \beta_3, c \lor \neg \beta_4, a \lor \neg \beta_3 \land c \lor \neg \beta_4 \land d \lor \neg \beta_5, a \lor \neg \beta_1 \land \neg b \lor \beta_2, a \lor \neg \beta_3 \lor \beta_4 \lor \beta_5 \land \neg d \lor \beta_6\}
\]
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
\[
\{\text{body}(r) \mid r \in \Pi, \text{head}(r) \in U, \text{body}(r) \cap U = \emptyset\}
\]

Definition (Loop clause)
For a set $U \subseteq \text{Atoms}(\Pi)$ and an atom $p \in U$:
\[
\lambda(p, U) = \{\beta_1 \lor \ldots \lor \beta_k \lor \neg p\}
\]
where $EB_\Pi(U) = \{\beta_1, \ldots, \beta_k\}$.

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

CLASP translation IV

Proposition
$X$ is an answer set of $\Pi$ 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