Principles of Knowledge Representation and Reasoning
Answer Set Programming

Bernhard Nebel, Stefan Wölfl, and Felix Lindner
January 20 & 25, 2016
1 Introduction
Answer set semantics: a formalization of negation as failure in logic programming (Prolog)

Several formal semantics: well-founded semantics, perfect-model semantics, inflationary semantics, ...

Can be viewed as a simpler variant of default logic
ASP: Negation as failure

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

Example

\[ \text{innocent } \leftarrow \text{not guilty} . \]
ASP: Declarative problem solving

- What is the problem? instead of: How to solve the problem?
- Outsourcing the computation part to an external solver
2 Answer Sets

- Normal logic programs
- Interpretation and Satisfiability
- Definition
- Formal properties
- Stratification
Normal logic programs I

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

**Rules:**

\[
a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_k
\]

where \( \{a, b_1, \ldots, b_m, c_1, \ldots, c_k\} \subseteq \mathcal{A} \)

- Meaning similar to default logic:
  - If
    1. we have derived \( b_1, \ldots, b_m \) and
    2. cannot derive any of \( c_1, \ldots, c_k \),
  - then derive \( a \).
- Rules without right-hand side (facts): \( a \leftarrow \)
- Rules without left-hand side (constraints):
  \[
  \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_k
  \]
Normal logic programs II

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

**Rules:**

\[ a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_k \]

where $\{a, b_1, \ldots, b_m, c_1, \ldots, c_k\} \subseteq \mathcal{A}$

- $a$ is called the **head** of the rule, denoted by $\text{head}(r)$.
- The literals $b_1, \ldots, b_m$ form the **positive body** of $r$, denoted by $\text{body}^+(r)$.
- The literals $\text{not } c_1, \ldots, \text{not } c_k$ form the **negative body** of $r$, denoted by $\text{body}^-(r)$.
- The **body** of $r$ is the union of positive and negative body: $\text{body}(r) = \text{body}^+(r) \cup \text{body}^-(r)$.
Normal logic programs: Example

Example

\[
\begin{align*}
\text{bird}(X) & \leftarrow \text{eagle}(X) \\
\text{bird}(X) & \leftarrow \text{penguin}(X) \\
\text{fly}(X) & \leftarrow \text{bird}(X), \neg \text{nonfly}(X) \\
\text{nonfly}(X) & \leftarrow \text{penguin}(X) \\
\text{eagle}(\text{eddy}) & \leftarrow \\
\text{penguin}(\text{tweety}) & \leftarrow
\end{align*}
\]
Herbrand base and grounded rules

Let \( P \) be a normal logic program, i.e., a finite set of rules as described above.

- The **Herbrand universe** (symb. \( U_P \)) of \( P \) is the set of ground terms constructed from the function symbols and constants in \( P \).
- The **Herbrand base** of \( P \) (symb. \( B_P \)) 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 \( P \) is denoted by \( \text{atoms}(P) \).
Herbrand base and grounded rules

Example

\[
\begin{align*}
\text{bird}(eddy) & \leftarrow \text{eagle}(eddy) \\
\text{bird}(tweety) & \leftarrow \text{eagle}(tweety) \\
\text{bird}(eddy) & \leftarrow \text{penguin}(eddy) \\
\text{bird}(tweety) & \leftarrow \text{penguin}(tweety) \\
\text{fly}(eddy) & \leftarrow \text{bird}(eddy), \text{not}\ \text{nonfly}(eddy) \\
\text{fly}(tweety) & \leftarrow \text{bird}(tweety), \text{not}\ \text{nonfly}(tweety) \\
\text{nonfly}(eddy) & \leftarrow \text{penguin}(eddy) \\
\text{nonfly}(tweety) & \leftarrow \text{penguin}(tweety) \\
\text{eagle}(eddy) & \leftarrow \\
\text{penguin}(tweety) & \leftarrow 
\end{align*}
\]
Satisfaction

A Herbrand interpretation is a subset $X$ of the Herbrand base.

**Satisfaction relation:**

- $X \models a$ if $a \in X$.
- $X \models r$ if $\{b_1, \ldots, b_m\} \not\subseteq X$ or $\{a, c_1, \ldots, c_n\} \cap X \neq \emptyset$, where $r = a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_k$.
- $X \models P$ if $X \models r$ for each $r \in P$.

**Idea**

Idea: “models” as interpretations that are satisfying, stable, and supported.
Positive *(not-free)* logic programs

**Definition (Answer set)**

Let $P$ be a logic program without *not*, $X \subseteq \text{atoms}(P)$. $X$ is the (unique) answer set of $P$ if it is the least fixpoint of the operator:

$$\Gamma_P(X) = \{a : \exists r = a \leftarrow b_1, \ldots, b_m \in P \text{ with } \{b_1, \ldots, b_m\} \subseteq X\}.$$  

**Example**

$$P = \left\{ \begin{array}{c} a \leftarrow b, \\ d \leftarrow f, \\ b \leftarrow, \\ d \leftarrow b, \\ c \leftarrow d, \\ e \leftarrow f \end{array} \right\}$$

$$\Gamma^0 = \emptyset, \quad \Gamma^1 = \Gamma(\emptyset) = \{b\}, \quad \Gamma^2 = \Gamma(\Gamma^1) = \{b, d, a\}, \quad \Gamma^3 = \Gamma(\Gamma^2) = \{b, d, a, c\}, \quad \Gamma^4 = \Gamma(\Gamma^3) = \{b, d, a, c\} = \Gamma^3$$
Gelfond-Lifschitz reduct

Definition (Reduct)

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

$$P^{X} := \{ \text{head}(r) \leftarrow \text{body}^{+}(r) : r \in P, \quad c \notin X \text{ for each not } c \in \text{body}^{-}(r) \}$$

That is, given $X$,

- ... delete all rules whose negative part contradicts $X$
- ... remove all negated atoms from the remaining rules

Definition (Answer set)

$X \subseteq \text{atoms}(P)$ is an answer set of $P$ if $X$ is an answer set of $P^{X}$. 
Answer sets: Examples

Example

\[ a \leftarrow \neg b, \quad b \leftarrow \neg a, \quad c \leftarrow a, \quad d \leftarrow b. \]

Example

\[ a \leftarrow \neg b, \quad b \leftarrow \neg a, \quad b \leftarrow a, \quad c \leftarrow b \]

Example

\[ a \leftarrow b, \quad b \leftarrow a \]
Some properties I

Proposition

*If an atom* \( a \) *belongs to an answer set of a logic program* \( P \), *then* \( a \) *is the head of one of the rules of* \( P \).*

Proposition

*Each answer set of a normal logic program* \( P \) *is a minimal model of* \( P \), *i.e., it satisfies all rules in* \( P \) *and there is no proper subset of* \( P \) *satisfying all rules in* \( P \).*

**Notice:** The converse is not true: not each minimal model is an answer set.
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 $F$ that satisfies $G$.

Proof.

$F \subseteq F \cup G$ implies $F^X \subseteq (F \cup G)^X$ and hence $\text{lfp}_\Gamma(F^X) \subseteq \text{lfp}_\Gamma((F \cup G)^X)$.

$\Rightarrow$: Assume $X$ is an answer set of $F \cup G$, hence $X = \text{lfp}_\Gamma((F \cup G)^X)$ and $X \models G$. Since $G$ contains constraints only, it follows that each $a \in X$ is the head of some rule in $F$. Hence, $X \subseteq \text{lfp}_\Gamma(F^X)$, and thus $X$ is an answer set of $F$ that satisfies $G$.

$\Leftarrow$: Similar.
Complexity: Existence of answer sets is NP-complete

1. **Membership in NP:** Guess $X \subseteq \text{atoms}(P)$ (nondet. polytime), compute $P^X$, compute its closure, compare to $X$ (everything det. polytime).

2. **NP-hardness:** Reduction from 3SAT: an answer set exists iff the following clauses are satisfiable:

   $$p \leftarrow \neg \hat{p}. \quad \hat{p} \leftarrow \neg p.$$ 

   for every propositional variable $p$ occurring in the clauses, and

   $$\leftarrow \neg l_1', \neg l_2', \neg l_3'$$

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

---

The **ancestor** relation is the transitive closure of the **parent** relation.

Transitive closure **cannot be** (concisely) represented in propositional/predicate logic.

\[
par(X, Y) \rightarrow anc(X, Y) \\
par(X, Z) \land anc(Z, Y) \rightarrow anc(X, Y)
\]

The above formulae only guarantee that **anc** is a superset of the transitive closure of **par**.

For transitive closure one needs the **minimality condition** in some form: nonmonotonic logics, fixpoint logics, ...
The reason for multiple answer sets is the fact that $a$ may depend on $b$ and simultaneously $b$ may depend on $a$. The lack of this kind of circular dependencies makes reasoning easier.

**Definition**

A logic program $P$ is **stratified** if $P$ can be partitioned to $P = P_1 \cup \cdots \cup P_n$ so that for all $i \in \{1, \ldots, n\}$ and $(a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_k) \in P_i$,

1. there is no $\neg a$ in $P_i$ and
2. there are no occurrences of $a$ anywhere in $P_1 \cup \cdots \cup P_{i-1}$. 
Stratification

Theorem

A stratified program $P$ has exactly one answer set. The unique answer set can be computed in polynomial time.

Example

Our earlier examples with more than one or no answer sets:

$$P_3 = \{ p \leftarrow \text{not } p \}$$

$$P_4 = \{ p \leftarrow \text{not } q, \quad q \leftarrow \text{not } p \}$$
3 AnsProlog and ASP Tools

- Language and notations
Programs for Reasoning with Answer Sets

- **smodels** (Niemelä & Simons), **dlv** (Eiter et al.), **clasp** (Schaub et al.), ...

- **Schematic input:**

  ```prolog
  p(X) :- not q(X).
  q(X) :- not p(X).
  r(a).
  r(b).
  r(c).
  anc(X,Y) :- par(X,Y).
  anc(X,Y) :- par(X,Z), anc(Z,Y).
  par(a,b). par(a,c). par(b,d).
  female(a).
  male(X) :- not(female(X)).
  forefather(X,Y) :-
    anc(X,Y), male(X).
  ```
Propositions are any combination of lowercase letters.

Variables are any combination of letters starting with an uppercase letter.

Write ":-" instead of ←.

Integers can be used and so can arithmetic operations (+, −, ∗, /, %).

Negation as failure is denoted by not.

Strong negation is denoted by −.

#const n = ... statements can be used to define constants.

The #hide/#show statements can be used to influence which iterals are shown in the solution.
AnsProlog: Choice functions

- The literal \{b_1; \ldots ; b_m\}
  is true iff any subset of the set \{b_1, \ldots, b_m\} is true.

Example

Generate all interpretations over the atoms \(a(1), a(2), a(3)\):

\{ a(1); a(2); a(3) \}.

With strong negation:

\[-a(X) :- \text{not } a(X), X=1..3.\]
\{ a(1..3) \}.
AnsProlog: Choice with cardinality

- The literal \( l \{ b_1; \ldots; b_m \} u \) is true iff at least \( l \) and at most \( u \) atoms (included) are true within the set \( \{ b_1, \ldots, b_m \} \).

Example

Generate all interpretations over the atoms \( a(1), a(2), a(3), b(1), b(2) \) that contain exactly 2 true atoms:

\[
2 \{ a(1..3); b(1..2) \} 2.
\]

Generate all interpretations over the atoms \( a(1), a(2), a(3), b(1), b(2), b(3) \) that do not contain exactly 2 or more true atoms for the same predicate:

\[
\{ a(1..3); b(1..3) \}.
\]

\[
:- 2 \{ a(1..3) \} 3.
\]

\[
:- 2 \{ b(1..3) \} 3.
\]
AnsProlog: Domains of variables

- The domain of a variable must be known in order to avoid “unsafe”-error while the program is grounded.
- The domain can be set literal-wise, rule-wise, or program wise.
- For limiting the scope within a literal use the syntax:
  \[ a(X) : \text{dom}(X) \quad \text{or} \quad a(X) : X=1..3 \]

**Example**

\[
\text{num}(0..10).
\text{even}(2*X) :- \text{num}(X), 2*X \leq 10.
1 \{ a(X) : \text{even}(X) \} 1.
\]

#show a/1.
Example: Graph coloring

Example

#const n = 2.
c(1..n).
1 {color(X,I) : c(I)} 1 :- v(X).
:- color(X,I), color(Y,I), e(X,Y), c(I).

% Instance
v(1..4).
e(1,2).
e(1,3).
e(2,4).
e(3,4).
% e(2,3).

#show color/2.
Generate and test

ASP programs are often organized in a “generate-and-test” style: first describe candidate solutions, then rule out possible solutions by stating constraints.

Example

% n-Queens encoding %
#const n = 4.

% Generate possible positions %
1 { q(I,1..n) } 1 :- I = 1..n.

% Rule out attacking positions %
:- q(I1,J), q(I2,J), I1 != I2.
:- q(I,J), q(I+D,J+D), D = 1..n.
:- q(I,J), q(I+D,J-D), D = 1..n.
Generate and test: Further example

**Problem:** In a graph find cliques of size $\geq n$

**Example**

```prolog
#const n = 3.

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

n {clique(X) : node(X)}.

:- clique(X), clique(Y), node(X), node(Y), X!=Y, not edge(X,Y).

% Instance %
node(1..5).
edge(1,2;4).
edge(2,3;4).
edge(3,4).
edge(4,2;5).

#show clique/1.
```

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
- ...

(More on that in the exercises!)
Literature

Michael Gelfond and Vladimir Lifschitz.  
**The stable models semantics for logic programming.**  

François 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.**  
Literature

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

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