Principles of Knowledge Representation and Reasoning

Answer Set Programming

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
Let $A$ be a set of propositional atoms.

**Rules:**

$$c \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k$$

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

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

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

- Rules without left-hand side (**constraints**):
  $$\bot \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k$$
Let $A$ be a set of propositions.

**Rules:**

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

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

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

Example

fly ← bird, not abnormal.
abnormal ← penguin.
bird ← penguin.

Example

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),
Nonmonotonic logic programs: examples

Example

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

Example

\[ \{ \text{sol}(X, Y, A) : \text{num}(A) \} \]
\[ \leftarrow \text{sol}(X, Y, Z), \text{sol}(X, Y1, Z), Y \neq Y1. \]
\[ \leftarrow \text{sol}(X, Y, Z), \text{sol}(X1, Y, Z), X \neq X1. \]
\[ \leftarrow \text{sol}(W \ast 3 + W2, W1 \ast 3 + W3, Z), \]
\[ \quad \text{sol}(W \ast 3 + W4, W1 \ast 3 + W5, Z), W3 \neq W5. \]
\[ \leftarrow \text{sol}(W \ast 3 + W2, W1 \ast 3 + W3, Z), \]
\[ \quad \text{sol}(W \ast 3 + W4, W1 \ast 3 + W5, Z), W2 \neq W4. \]
The Gelfond-Lifschitz reduct
Logic of here-and-there
SAT translations of ASP
**Definition (Deductive closure)**

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

**Example**

\[
\Pi = \left\{ \begin{array}{c}
a \leftarrow b. \\
d \leftarrow b. \\
f. \\
b. \\
\end{array} \right. \\
\begin{array}{c}
d \leftarrow b. \\
c \leftarrow b, d. \\
e \leftarrow f. \\
\end{array}
\]

\[
\Gamma_0 = \Gamma(\emptyset) = \{b\} \\
\Gamma_1 = \Gamma(\Gamma_0) = \{b, d, a\} \\
\Gamma_2 = \Gamma(\Gamma_1) = \{b, d, a, c\} \\
\Gamma_3 = \Gamma(\Gamma_2) = \{b, d, a, c\} = \Gamma_2
\]
**Definition (Deductive closure)**

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

**Example**

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

\[
\begin{align*}
\Gamma_0 &= \Gamma(\emptyset) = \{b\} \\
\Gamma_1 &= \Gamma(\Gamma_0) = \{b,d,a\} \\
\Gamma_2 &= \Gamma(\Gamma_1) = \{b,d,a,c\} \\
\Gamma_3 &= \Gamma(\Gamma_2) = \{b,d,a,c\} = \Gamma_2
\end{align*}
\]
The Gelfond-Lifschitz reduct
Definition 1: Gelfond-Lifschitz reduct

**Definition (Reduct)**

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

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

**Definition (Answer set)**

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

Definition 1: Gelfond-Lifschitz reduct

**Definition (Reduct)**

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

$$\Pi^X := \{ c \leftarrow b_1, \ldots, b_m \mid (c \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k) \in \Pi, \{d_1, \ldots, d_k\} \cap X = \emptyset\}$$

**Definition (Answer set)**

$X \subseteq \text{Atoms}(\Pi)$ is an answer set of $\Pi$ if $X$ is an answer set of $\Pi^X$. 
Illustration of Gelfond-Lifschitz reduct

Example

\[
\begin{align*}
  a & \leftarrow \text{not}\, b. \\
  b & \leftarrow \text{not}\, a. \\
  d & \leftarrow a. \\
  d & \leftarrow b.
\end{align*}
\]

Example

\[
\begin{align*}
  a & \leftarrow b. \\
  b & \leftarrow a.
\end{align*}
\]

Example

\[
\begin{align*}
  \text{woman} & \leftarrow \text{not}\, \text{n\_woman}. \\
  \text{n\_woman} & \leftarrow \text{not}\, \text{woman}. \\
  \leftarrow \text{woman, n\_woman}. \\
  \text{father} & \leftarrow \text{parent, n\_woman}. \\
  \text{mother} & \leftarrow \text{parent, woman}. \\
  \text{parent}.
\end{align*}
\]

We say that $X$ satisfies a rule $r$ iff $X \models \text{head}(r) \lor \neg\text{body}(r)$.  
$\Rightarrow X$ can satisfy all rules and not be an answer set.
Illustration of Gelfond-Lifschitz reduct

Example

\begin{align*}
    a & \leftarrow \text{not } b. \\
    b & \leftarrow \text{not } a. \\
    d & \leftarrow a. \\
    d. & \leftarrow b.
\end{align*}

Example

\begin{align*}
    a & \leftarrow b. \\
    b & \leftarrow a.
\end{align*}

Example

\begin{align*}
    \text{woman} & \leftarrow \text{not } \text{n\_woman}. \\
    \text{not } \text{n\_woman} & \leftarrow \text{not } \text{woman}. \\
    \text{woman, n\_woman} & \leftarrow \text{parent, n\_woman}. \\
    \text{father} & \leftarrow \text{parent, n\_woman}. \\
    \text{mother} & \leftarrow \text{parent, woman}. \\
    \text{parent}.
\end{align*}

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

\( \implies \) \( X \) can satisfy all rules and not be an answer set.
We say that $X$ satisfies a rule $r$ iff $X \models \text{head}(r) \lor \neg \text{body}(r)$.  
\[ \Rightarrow X \text{ can satisfy all rules and not be an answer set.} \]
Illustration of Gelfond-Lifschitz reduct

Example

\[
\begin{align*}
a & \leftarrow \neg b. & b & \leftarrow \neg a. \\
d & \leftarrow a. & d. & \leftarrow b.
\end{align*}
\]

Example

\[
\begin{align*}
a & \leftarrow b. & b & \leftarrow a.
\end{align*}
\]

Example

\[
\begin{align*}
\text{woman} & \leftarrow \neg \text{n\_woman}. & \text{n\_woman} & \leftarrow \neg \text{woman}. \\
\leftarrow \text{woman}, \text{n\_woman}. & \text{father} & \leftarrow \text{parent}, \text{n\_woman}. \\
\text{mother} & \leftarrow \text{parent}, \text{woman}. & \text{parent}.
\end{align*}
\]

We say that $X$ satisfies a rule $r$ iff $X \models \text{head}(r) \lor \neg \text{body}(r)$.  
$\Rightarrow X$ can satisfy all rules and not be an answer set.
Based on the Gelfond-Lifschitz reduction, Syrjanen created the ASP solver Smodels.

Example

\begin{verbatim}
 b :- not a.  a :- not b.
 d :- a.  d :- b.
\end{verbatim}
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 literal
\[ l\{b_1,\ldots,b_m\}u \]
is true iff at least \( l \) and at most \( u \) atoms are true within the set \( \{b_1,\ldots,b_m\} \);

#domain encodes the possible values in a given domain:
#domain \( a(X). \ a(1..10). \)
will replace occurrences of \( X \) by integers from 1 to 10.
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.
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)
```
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

```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).

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 $\Pi^X$, compute its closure, compare to $X$ (everything det. polytime).

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

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

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

for every proposition $p$ occurring in the clauses, and

$$\leftarrow \neg l'_1, \neg l'_2, \neg l'_3$$

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

   $p \leftarrow \neg \hat{p}$.  \hspace{1cm} \hat{p} \leftarrow \neg p.$

   for every proposition $p$ occurring in the clauses, and

   $\leftarrow \neg l'_1, \neg l'_2, \neg l'_3$

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

\[
\leftarrow \neg l'_1, \neg l'_2, \neg l'_3
\]

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 \( l_i = \neg p \).
Complexity: existence of answer sets is NP-complete

1. **Membership in NP:** Guess $X \subseteq \text{Atoms}(\Pi)$ (non-deterministic polynomial time), compute $\Pi^X$, compute its closure, compare to $X$ (everything deterministic polynomial time).

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

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

   for every proposition $p$ occurring in the clauses, and

   $$
   \leftarrow \neg l'_1, \neg l'_2, \neg l'_3
   $$

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

November 28, 2012; December 5, 2012 Nebel, Wölfli, Hué – KRR
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:

$$(F \cup G)^X = \begin{cases} F^X \cup G^X, & \text{if } X \models F \cup G \\ \bot, & \text{otherwise} \end{cases}$$
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 $(F \cup G)^X$ is $F^X \cup \bot$ which is equivalent to $F^X$. Consequently $X$ is minimal among the sets satisfying $F^X$ iff it is minimal among the sets satisfying $(F \cup G)^X$.

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

November 28, 2012; December 5, 2012 Nebel, Wölfl, Hué – KRR
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 $(F \cup G)^X$ 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 $(F \cup G)^X$.

$\Leftarrow$ $X$ 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 $G$. 

\[\square\]
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 \( (F \cup G)^X \) is \( F^X \cup \neg \perp \) which is equivalent to \( F^X \). Consequently \( X \) is minimal among the sets satisfying \( F^X \) iff it is minimal among the sets satisfying \( (F \cup G)^X \).

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

Smodels is:

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

Example

\[
\begin{align*}
  a & : \neg b. \\
  b & : \neg a. \\
  c & : \neg c, a.
\end{align*}
\]

\[
\begin{array}{c}
  \times \\
  b \quad \times \\
  \times \\
  \times \\
  \sqrt
\end{array}
\]

\[
\begin{array}{c}
  \times \\
  b \quad \times \\
  \times \\
  \times \\
  \sqrt
\end{array}
\]
Algorithm 1 Smodels algorithm

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

Example

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. \( f \leftarrow \neg e \).

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

Case 2:
- \( a \not\subseteq X \) \( \rightarrow \) \( d \subseteq X \);
- \( c \) becomes \( d \);
- \( a \) cannot be fired, \( a \not\subseteq X \);
- \( b \) cannot be fired; \( b \not\subseteq X \);
- Nothing new to be expanded.
Smodels example (I)

**Example**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a \leftarrow \neg b, \neg d$.</td>
<td>2</td>
<td>$d \leftarrow \neg a$.</td>
</tr>
<tr>
<td>3</td>
<td>$b \leftarrow \neg c$.</td>
<td>4</td>
<td>$c \leftarrow \neg a$.</td>
</tr>
<tr>
<td>5</td>
<td>$e \leftarrow \neg f, \neg a$.</td>
<td>6</td>
<td>$f \leftarrow \neg e$.</td>
</tr>
</tbody>
</table>

**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.

---

November 28, 2012; December 5, 2012 Nebel, Wölfl, Hué – KRR 22 / 47
Smodels example (I)

Example

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

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

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

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

### Case 2.1: $e \subseteq X$

After reduction:

- $e \leftarrow \text{not} f. \quad 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
Are the two following logic programs 

\[ \Pi_1 = \quad a \leftarrow \neg b. \quad b \leftarrow \neg a. \]

and

\[ \Pi_2 = \quad a \leftarrow \neg b. \quad b \leftarrow \neg c, \neg a. \]

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)**

Π₁ and Π₂ are *weakly equivalent* if they have the same answer sets.

**Definition (Strong equivalence)**

Π₁ and Π₂ are *strongly equivalent* if for any Π, Π₁ ∪ Π and Π₂ ∪ Π have the same answer sets.

**Example**

Π₁ = a ← notb.  b ← nota.
Π₂ = a ← notb.  b ← notc, nota.

- Do Π₁ and Π₂ have the same answer sets?
- Do Π₁ ∪ {c.} and Π₂ ∪ {c.} have the same answer sets?
Weak equivalence/strong equivalence

**Definition (Weak equivalence)**

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

**Definition (Strong equivalence)**

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

**Example**

\[
\begin{align*}
\Pi_1 &= a \leftarrow \text{not} b. \quad b \leftarrow \text{not} a. \\
\Pi_2 &= a \leftarrow \text{not} b. \quad b \leftarrow \text{not} c, \text{not} a.
\end{align*}
\]

- Do \( \Pi_1 \) and \( \Pi_2 \) have the same answer sets?
- Do \( \Pi_1 \cup \{c.\} \) and \( \Pi_2 \cup \{c.\} \) have the same answer sets?
Weak equivalence/strong equivalence

Definition (Weak equivalence)

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

Definition (Strong equivalence)

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

Example

\[
\begin{align*}
\Pi_1 &= a \leftarrow \text{not}b. \quad b \leftarrow \text{not}a. \\
\Pi_2 &= a \leftarrow \text{not}b. \quad b \leftarrow \text{not}c, \text{not}a.
\end{align*}
\]

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

\[
\begin{align*}
  a & \leftarrow \text{not } b. \\
  b & \leftarrow \text{not } a. \\
  c & \leftarrow a.
\end{align*}
\]

<table>
<thead>
<tr>
<th>(Y)</th>
<th>(\Pi^Y)</th>
<th>(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, c}</td>
<td>(a. c \leftarrow a)</td>
<td>{a, c}</td>
</tr>
<tr>
<td>{b}</td>
<td>(b. c \leftarrow a)</td>
<td>{b}</td>
</tr>
<tr>
<td>{b, c}</td>
<td>(b. c \leftarrow a)</td>
<td>{b}, {b, c}</td>
</tr>
<tr>
<td>{a, b, c}</td>
<td>(c \leftarrow a)</td>
<td>{\emptyset}, {b}, {a, c}, {a, b, c}</td>
</tr>
</tbody>
</table>
Answer set definition II

Proposition (Characterization of answer sets)

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

Example

\[ a \rightarrow \neg b. \quad b \leftarrow \neg a. \quad c \leftarrow a. \]

<table>
<thead>
<tr>
<th>( Y )</th>
<th>( \Pi^Y )</th>
<th>( X )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, c}</td>
<td>a. \quad c \leftarrow a.</td>
<td>{a, c}</td>
</tr>
<tr>
<td>{b}</td>
<td>b. \quad c \leftarrow a.</td>
<td>{b}</td>
</tr>
<tr>
<td>{b, c}</td>
<td>b. \quad c \leftarrow a.</td>
<td>{b}, {b, c}</td>
</tr>
<tr>
<td>{a, b, c}</td>
<td>c \leftarrow a.</td>
<td>{\emptyset}, {b}, {a, c}, {a, b, c}</td>
</tr>
</tbody>
</table>

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

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

*The logic programs* $\Pi_1$ *and* $\Pi_2$ *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* $\Pi_1$ *and* $\Pi_2$.

Proof.

$\Pi_1$ *and* $\Pi_2$ *have the same SE-models. Consider* $\Pi$. *By lemma 2:* $\Pi_1 \cup \Pi$ *and* $\Pi_2 \cup \Pi$ *have the same SE-models. By lemma 1:* $\Pi_1 \cup \Pi$ *and* $\Pi_2 \cup \Pi$ *are weakly equivalent.*
Strong equivalence: properties

**Proposition**

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

**Proof.**

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

**Case** $Y \nsubseteq \Pi_2$ $Y \nsubseteq \Pi_2 \cup Y$ 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 $\Pi = X \cup \{L \leftarrow L' : L, L' \in Y \setminus X\}$. $Y \models \Pi_2 \cup \Pi$, 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 $X \nsubseteq \Pi_2^Y$ so $X \neq Z$. There is some $L \in Y \setminus X$ that belongs to $Z$. It follows that $Y \setminus X \subseteq Z$. Thus, $Z \models Y$, 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 $\Pi_1 \cup \Pi$. $\square$
SAT translations of ASP
Definition (Dependency graph)

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

Example

\[ \Pi = \{ \begin{array}{c}
  a \leftarrow b. \\
  b \leftarrow a. \\
  c \leftarrow d. \\
  d \leftarrow c. \\
  c \leftarrow \neg a. \\
\end{array} \} \]

- $d \rightarrow c$  
- $b \rightarrow a$
Clark’s completion

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

- If $\leftarrow B$ is a constraint in $\Pi$, then $\neg B$ is in $\text{Comp}(\Pi)$.

Example

\[
\Pi = \left\{ \begin{array}{l}
a \leftarrow b. \\
b \leftarrow a. \\
a \leftarrow \text{not} \ c. \\
c \leftarrow d. \\
d \leftarrow c. \\
c \leftarrow \text{not} \ a. \\
\end{array} \right\}
\]

\[
\text{Comp}(\Pi) = \left\{ \begin{array}{l}
a \equiv \neg c \lor b \\
b \equiv a \\
c \equiv \neg a \lor d \\
d \equiv c \\
\end{array} \right\}
\]

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

Definition (Tight program)

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

Example

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

\[
\Pi = \left\{ \begin{array}{c}
  a \leftarrow b. \\
  b \leftarrow a. \\
  a \leftarrow \neg c. \\
  c \leftarrow d. \\
  d \leftarrow c. \\
  c \leftarrow \neg a. \\
\end{array} \right. 
\]

\[
\text{Comp}(\Pi) = \left\{ \begin{array}{c}
  a \equiv \neg c \lor b \\
  b \equiv a \\
  c \equiv \neg a \lor d \\
  d \equiv c \\
\end{array} \right. 
\]

$\text{Comp}(\Pi)$ has 3 models: $\{a, b\}$, $\{c, d\}$ and $\{a, b, c, d\}$. 
Tightness and Clark’s completion (proof)

Definition (Well-supported model)

$M$ is a well-supported model of $\Pi$ if there exists a grounding sequence for $M$, i.e., there exists an order $<$ 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 $\Pi$. 
Proof.

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

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

Definition (Loop)

A loop of \( \Pi \) is a set \( L \) 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 \( L \).

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

Example

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

\[
\begin{align*}
R^+(L_1, \Pi) &= \{ a \leftarrow b. \ b \leftarrow a. \} \\
R^-(L_1, \Pi) &= \{ a \leftarrow \text{not} c. \} \\
R^+(L_2, \Pi) &= \{ c \leftarrow d. \ d \leftarrow c. \} \\
R^-(L_2, \Pi) &= \{ c \leftarrow \text{not} a. \}
\end{align*}
\]
Definition (Loop formulas)

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

$$p_1 \leftarrow B_{11} \quad \ldots \quad p_1 \leftarrow B_{1k_1}$$

$$\vdots$$

$$p_n \leftarrow B_{n1} \quad \ldots \quad p_n \leftarrow B_{nk_n}$$

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

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

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

$$LF(L_1) : c \rightarrow (\neg a \land \neg b) \quad LF(L_2) : a \rightarrow (\neg c \land \neg d)$$
Clark + loop formulae

Theorem

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

Example

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

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

Definition (Body clauses)

Let \( \beta \) be a body of a rule \( \beta = \{ p_1, \ldots, p_m, \neg p_{m+1}, \ldots, \neg p_n \} \), then:

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

Example

\[ \Pi = \left\{ \begin{array}{ccc} a & \leftarrow & b \quad b & \leftarrow & a \quad a & \leftarrow & \neg c \quad c & \leftarrow & d \quad d & \leftarrow & c \quad c & \leftarrow & \neg a \end{array} \right\} \]

\[ \Pi = \left\{ \begin{array}{ccccccc} \beta_1 \lor \neg b & \beta_2 \lor \neg a & \beta_3 \lor c & \beta_4 \lor \neg d & \beta_5 \lor \neg c & \beta_6 \lor a & \neg \beta_1 \lor b & \neg \beta_2 \lor a & \neg \beta_3 \lor \neg c & \neg \beta_4 \lor d & \neg \beta_5 \lor c & \neg \beta_6 \lor \neg a \end{array} \right\} \]
Definition (Atoms clauses)

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

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

Example

\[\Pi = \{\begin{array}{llll}
  a & \leftarrow & b. & b \leftarrow a. \\
  c & \leftarrow & d. & d \leftarrow c. \\
  & & a \leftarrow & \text{not } c. \\
  & & c \leftarrow & \text{not } a. \\
\end{array}\} \]

\[\Pi = \{\begin{array}{l}
  a \lor \neg \beta_1 \\
  b \lor \neg \beta_2 \\
  a \lor \neg \beta_3 \\
  c \lor \neg \beta_4 \\
  d \lor \neg \beta_5 \\
  c \lor \neg \beta_6 \\
  \neg a \lor \beta_1 \lor \beta_3 \\
  \neg b \lor \beta_2 \\
  \neg c \lor \beta_4 \lor \beta_6 \\
  \neg d \lor \beta_5 \\
\end{array}\} \]
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 $\Lambda_\Pi = \bigcup_{U \subseteq \text{Atoms}(\Pi), U \neq \emptyset} \{\lambda(p, U) \mid p \in U\}$. 
Proposition

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

\[ \Lambda_\Pi \cup \Delta(p) \cup \delta(p) \cup \delta(\beta) \cup \Delta(\beta) \]
Literature

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

Francois Fages.  
Consistency of Clark’s completion and existence of stable models.  
Meth. of Logic in CS, p51-60, 1994.

Hudson Turner.  
Strong equivalence made easy: nested expressions and weight constraints.  
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.