Declarative problem Solving

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

```
Problem                   Solution
\downarrow                  \uparrow
Modeling                  Interpretation
\downarrow                      \uparrow
Representation              Computation
                                    Output
```
Negation as failure

- Another interpretation for negation: $\neg x \equiv "I cannot show that x is true"$
- 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
    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):
  \( \bot \leftarrow b_1, \ldots, b_m, \neg d_1, \ldots, \neg d_k \)
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

\[
\begin{align*}
\text{fly}(\text{tweety}) & \leftarrow \text{bird}(\text{tweety}), \text{not}\, \text{abnormal}(\text{tweety}). \\
\text{bird}(\text{tweety}) & \leftarrow \text{penguin}(\text{tweety}). \\
\text{abnormal}(\text{tweety}) & \leftarrow \text{penguin}(\text{tweety}).
\end{align*}
\]
1 Interpretation and Satisfiability

- Formal properties of answer sets
- Language and notations
- Computation
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 $Atoms(\Pi)$. 
Herbrand base and Grounded rules

Example

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

\[ U_\Pi = \{ t \} \]
\[ B_\Pi = \{ \text{fly}(t), \text{bird}(t), \text{abnormal}(t), \text{penguin}(t) \} \]

\[ \Pi = \begin{cases} 
  \text{fly}(t) & \leftarrow \text{bird}(t), \text{not abnormal}(t). \\
  \text{bird}(t) & \leftarrow \text{penguin}(t). \\
  \text{abnormal}(t) & \leftarrow \text{penguin}(t). \\
  \text{penguin}(t) & \leftarrow \top. 
\end{cases} \]
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 $\Pi$ be a logic program without $\text{not}$, $X \subseteq \text{Atoms}(\Pi)$. $X$ 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**

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

$$\Gamma_0 = \Gamma(\emptyset) = \{b\}$$
$$\Gamma_1 = \Gamma(\Gamma_0) = \{b,d,a\}$$
$$\Gamma_2 = \Gamma(\Gamma_1) = \{b,d,a,c\}$$
$$\Gamma_3 = \Gamma(\Gamma_2) = \{b,d,a,c\} = \Gamma_2$$
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 $\Pi^X$. 
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 \( X \models \text{head}(r) \lor \neg\text{body}(r) \).

\( \Rightarrow \) \( X \) 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 $\Pi^X$, 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 $l_i = \neg p$. 

November 25, 2013 Nebel, Wölfl, Hué – KRR
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 $F^X$ 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 $G$. \qed
Based on the Gelfond-Lifschitz reduction, Syrjanen created the ASP solver Smodels.
The lparses 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)\}.

\begin{align*}
a(1) & :- \neg na(1). & na(1) & :- \neg a(1). \\
a(2) & :- \neg na(2). & na(2) & :- \neg a(2). \\
a(3) & :- \neg na(3). & na(3) & :- \neg a(3). \\
\end{align*}

\{ a(1), a(2), a(3) \}. 
The literal \{b_1, \ldots, b_m\}
is true iff any subset of the set \{b_1, \ldots, b_m\} is true;

Example

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

\[
\begin{align*}
a(1) & : - \text{ not } na(1). & na(1) & : - \text{ not } a(1). \\
a(2) & : - \text{ not } na(2). & na(2) & : - \text{ not } a(2). \\
a(3) & : - \text{ not } na(3). & na(3) & : - \text{ not } a(3). \\
\end{align*}
\]

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

The \#hide statement can hide literals from the solution.
AnsProlog (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

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 do not contain 2 true atoms.

\[
\{ \ a(1), a(2), a(3) \ \}.\n\]
\[
:- \ 2 \ \{ \ a(1), a(2), a(3) \ \} \ 2.
\]
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:
  \[ a(X) :- \text{not } \text{na}(X), \text{num}(X). \]
  \[ X \text{ takes all the values for which } \text{num}(X) \text{ is stated as a fact.} \]

**Example**

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

\[ \text{num}(1..3). \]
\[ a(X) :- \text{not } \text{na}(X), \text{num}(X). \quad \text{na}(X) :- \text{not } a(X), \text{num}(X). \]
\[ :- 2 \{ a(X) : \text{num}(X) \} 2. \]
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 occurrences 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).
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.

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

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

Queen problem with board of size 5.

\[
\begin{align*}
\text{row}(1..5). & \quad \text{col}(1..5). \\
\{\text{queen}(I,J) : \text{row}(I) : \text{col}(J)\}. \\
:- \text{not} 5 \{\text{queen}(I,J)\} 5. \\
:- \text{queen}(I,J), \text{queen}(I,JJ), J != JJ. \\
:- \text{queen}(I,J), \text{queen}(II,J), I != II. \\
:- \text{queen}(I,J), \text{queen}(II,JJ), (I,J) != (II,JJ), I-J == II-JJ. \\
:- \text{queen}(I,J), \text{queen}(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
\end{align*}
\]
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.
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 (Miscellaneous)
Based on the Gelfond-Lifschitz reduction, Syrjanen created the ASP solver Smodels.

AnsProlog and Solvers

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

```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) }.
```

November 25, 2013 Nebel, WölfI, Hué – KRR
Another Example: Sudoku

Example

```prolog
#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.
```
Smmodels: principles

Smmodels is:

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

Example

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

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array}
\begin{array}{c}
\times \\
\times \\
\times
\end{array}
\begin{array}{c}
\neg \text{b} \\
\neg \text{b} \\
\neg \text{b}
\end{array}
\begin{array}{c}
\times \\
\times \\
\times
\end{array}
\begin{array}{c}
\times \\
\checkmark
\end{array}
\]
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 ∪ {X}) then
10: return true
11: else
12: return smodels(P, A ∪ {not X})
13: end if
14: end if
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 \( a \subseteq X \),  
  \( \rightarrow \text{contradiction} \).

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:  
\( e \leftarrow \text{not}f. \)  
\( f \leftarrow \text{not}e. \)

- (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.
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 \( a_0 \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \) in \( \Pi \) with \( a_0 \neq \bot \).

Example

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

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

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

$\text{Comp}(\Pi)$ 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

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

\[
\begin{tikzpicture}
  \node (a) at (0,0) {a};
  \node (b) at (2,0) {b};
  \node (d) at (4,0) {d};
  \node (c) at (2,-2) {c};
  \draw [->] (a) -- (b);
  \draw [->] (b) -- (d);
  \draw [->] (b) -- (c);
\end{tikzpicture}
\]
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

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

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

$\text{Comp}(\Pi)$ 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 $\Pi$. 
Tightness and Clark’s completion (proof)

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. \qed
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 $L$.

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

Example

\[
\Pi = \{a \leftarrow b. b \leftarrow a. a \leftarrow \text{not}c. \ c \leftarrow d. d \leftarrow c. c \leftarrow \text{not}a. \}
\]
\[
R^+(L_1, \Pi) = \{a \leftarrow b. b \leftarrow a.\} \quad R^-(L_1, \Pi) = \{a \leftarrow \text{not}c.\}
\]
\[
R^+(L_2, \Pi) = \{c \leftarrow d. d \leftarrow c.\} \quad R^-(L_2, \Pi) = \{c \leftarrow \text{not}a.\}
\]
### Loop formulas

#### Definition (Loop formulas)

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

\[
p_1 \leftarrow B_{11} \quad \cdots \quad p_1 \leftarrow B_{1k_1}
\]

\[
\vdots
\]

\[
p_n \leftarrow B_{n1} \quad \cdots \quad p_n \leftarrow B_{nk_n}
\]

The loop formula associated with $L$ is the following implication:

\[
\neg[B_{11} \lor \cdots \lor B_{1k_1} \lor \cdots \lor B_{n1} \lor \cdots \lor B_{nk_n}] \rightarrow \bigwedge_{p \in L} \neg p
\]

#### 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

\[
\Pi = \{ \ a \leftarrow b, \ b \leftarrow a, \ a \leftarrow \neg c, \ c \leftarrow d, \ d \leftarrow c, \ c \leftarrow \neg a. \ \}\n\]

\[
\text{Comp}(\Pi) \cup \text{LF}(\Pi) = \{ \begin{array}{c}
    a \equiv \neg c \lor b \\
    b \equiv a \\
    c \equiv \neg a \lor d \\
    d \equiv c \\
    c \rightarrow (\neg a \land \neg b) \\
    a \rightarrow (\neg c \land \neg d)
\end{array} \}
\]
Definition (Body clauses)

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

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

Example

$$\Pi = \{a \leftarrow b, \quad b \leftarrow a, \quad a \leftarrow \neg c, \quad c \leftarrow d, \quad d \leftarrow c, \quad c \leftarrow \neg a\}$$

$$\Pi = \{\beta_1 \lor \neg b, \quad \beta_2 \lor \neg a, \quad \beta_3 \lor c, \quad \beta_4 \lor \neg d, \quad \beta_5 \lor \neg c, \quad \beta_6 \lor a, \quad \neg \beta_1 \lor b, \quad \neg \beta_2 \lor a, \quad \neg \beta_3 \lor \neg c, \quad \neg \beta_4 \lor d, \quad \neg \beta_5 \lor c, \quad \neg \beta_6 \lor \neg a\}$$
CLASP translation II

Definition (Atoms clauses)

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

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

Example

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

\[
\Pi = \begin{cases} 
    a \lor \lnot \beta_1 & b \lor \lnot \beta_2 & a \lor \lnot \beta_3 & c \lor \lnot \beta_4 \\
    d \lor \lnot \beta_5 & c \lor \lnot \beta_6 & \lnot a \lor \beta_1 \lor \beta_3 & \lnot b \lor \beta_2 & \lnot c \lor \beta_4 \lor \beta_6 & \lnot d \lor \beta_5
\end{cases}
\]
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 $p \in U$:

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

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

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

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.