PROBLEM SOLVING AND SEARCH IN ARTIFICIAL INTELLIGENCE

Lecture 7 ASP III * slides adapted from Torsten Schaub [Gebser et al.(2012)]

Sarah Gaggl

Dresden
Agenda

1. Introduction
2. Uninformed Search versus Informed Search (Best First Search, A* Search, Heuristics)
3. Local Search, Stochastic Hill Climbing, Simulated Annealing
4. Tabu Search
5. Answer-set Programming (ASP)
6. Constraint Satisfaction (CSP)
7. Structural Decomposition Techniques (Tree/Hypertree Decompositions)
8. Evolutionary Algorithms/ Genetic Algorithms
Overview ASP III

- Language
  - Extended language
- Language Extensions
  - Two kinds of negation
  - Disjunctive logic programs
- Computational Aspects
  - Complexity
Language: Overview

1. Extended language
1 Extended language
1 Extended language
   - Conditional literal
   - Optimization statement
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ \ell : \ell_1, \ldots, \ell_n \]

  where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell \mid \ell_1, \ldots, \ell_n \} \)
Conditional literals

- **Syntax** A conditional literal is of the form

\[ \ell : \ell_1, \ldots, \ell_n \]

where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell \mid \ell_1, \ldots, \ell_n \} \)

- **Note** The expansion of conditional literals is context dependent
Conditional literals

- **Syntax** A conditional literal is of the form

\[ \ell : \ell_1, \ldots, \ell_n \]

where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell \mid \ell_1, \ldots, \ell_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

\[
\begin{align*}
r(X) : p(X), \neg q(X) & : r(X) : p(X), \neg q(X); 1 \{ r(X) : p(X), \neg q(X) \}.
\end{align*}
\]

is instantiated to

\[
\begin{align*}
r(1); \ r(3) & : r(1), \ r(3), 1 \{ r(1); \ r(3) \}.
\end{align*}
\]
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ \ell : \ell_1, \ldots, \ell_n \]

  where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell | \ell_1, \ldots, \ell_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

  \[
  r(X) : p(X), \text{not } q(X) : r(X) : p(X), \text{not } q(X) ; 1 \{ r(X) : p(X), \text{not } q(X) \}. 
  \]

  is instantiated to

  \[
  r(1); r(3) : r(1), r(3), 1 \{ r(1); r(3) \}. 
  \]
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ \ell : \ell_1, \ldots, \ell_n \]

  where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell \mid \ell_1, \ldots, \ell_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

  \[
  r(X) : p(X), \ not \ q(X) \ :- \ r(X) : p(X), \ not \ q(X); \ 1 \ \{ \ r(X) : p(X), \ not \ q(X) \ \}.
  \]

  is instantiated to

  \[
  r(1); \ r(3) \ :- \ r(1), \ r(3) ; \ 1 \ \{ \ r(1); \ r(3) \ \}.
  \]
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ \ell : \ell_1, \ldots, \ell_n \]

  where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell | \ell_1, \ldots, \ell_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3) . \ q(2) . \)’

  \[ r(X) : p(X), \not q(X) :- r(X) : p(X), \not q(X); 1 \{ r(X) : p(X), \not q(X) \} . \]

  is instantiated to

  \[ r(1); r(3) :- r(1), r(3), 1 \{ r(1); r(3) \} . \]
Conditional literals

- **Syntax** A conditional literal is of the form

\[ \ell : \ell_1, \ldots, \ell_n \]

where \( \ell \) and \( \ell_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ \ell \mid \ell_1, \ldots, \ell_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

\[
r(X) : p(X), \text{not } q(X) :- r(X) : p(X), \text{not } q(X); \quad 1 \{ r(X) : p(X), \text{not } q(X) \}.
\]

is instantiated to

\[
r(1); \; r(3) :- r(1), \; r(3), \; 1 \{ r(1); \; r(3) \}.
\]
Outline

1. Extended language
   - Conditional literal
   - Optimization statement
Optimization statement

- **Idea** Express (multiple) cost functions subject to minimization and/or maximization
- **Syntax** A minimize statement is of the form

  \[
  \text{minimize } \{ \ w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n \}.\]

  where each \( \ell_i \) is a literal; and \( w_i \) and \( p_i \) are integers for \( 1 \leq i \leq n \)
Optimization statement

- **Idea**: Express (multiple) cost functions subject to minimization and/or maximization
- **Syntax**: A *minimize* statement is of the form

\[
\text{minimize } \{ w_1 @ p_1 : \ell_1, \ldots, w_n @ p_n : \ell_n \}.
\]

where each \( \ell_i \) is a literal; and \( w_i \) and \( p_i \) are integers for \( 1 \leq i \leq n \)

Priority levels, \( p_i \), allow for representing lexicographically ordered minimization objectives
Optimization statement

- **Idea** Express (multiple) cost functions subject to minimization and/or maximization
- **Syntax** A minimize statement is of the form

  \[
  \text{minimize}\ \{\ w_1 p_1 : \ell_1, \ldots, w_n p_n : \ell_n \}.\]

  where each \( \ell_i \) is a literal; and \( w_i \) and \( p_i \) are integers for \( 1 \leq i \leq n \)

  Priority levels, \( p_i \), allow for representing lexicographically ordered minimization objectives
- **Meaning** A minimize statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a weighted sum of elements
Optimization statement

- A maximize statement of the form

\[
\text{maximize} \ \{ w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n \}
\]

stands for \textit{minimize} \ \{ -w_1@p_1 : \ell_1, \ldots, -w_n@p_n : \ell_n \}

Example: When configuring a computer, we may want to maximize hard disk capacity, while minimizing price.

\#maximize \ \{ 250@1:hd(1), 500@1:hd(2), 750@1:hd(3), 1000@1:hd(4) \}.
\#minimize \ \{ 30@2:hd(1), 40@2:hd(2), 60@2:hd(3), 80@2:hd(4) \}.

The priority levels indicate that (minimizing) price is more important than (maximizing) capacity.
Optimization statement

- A maximize statement of the form

\[
\text{maximize } \{ \ w_1 @ p_1 : l_1, \ldots, w_n @ p_n : l_n \ \}
\]

stands for \( \text{minimize } \{ -w_1 @ p_1 : l_1, \ldots, -w_n @ p_n : l_n \} \)

- **Example** When configuring a computer, we may want to maximize hard disk capacity, while minimizing price

```
#maximize { 250@1:hd(1), 500@1:hd(2), 750@1:hd(3), 1000@1:hd(4) }.
#minimize { 30@2:hd(1), 40@2:hd(2), 60@2:hd(3), 80@2:hd(4) }.
```

The priority levels indicate that (minimizing) price is more important than (maximizing) capacity
Language Extensions: Overview

2. Two kinds of negation

3. Disjunctive logic programs
Outline

1. Two kinds of negation
2. Disjunctive logic programs
Motivation

- Classical versus default negation
  - Symbol \( \neg \) and \( \text{not} \)
Motivation

• Classical versus default negation
  – Symbol $\neg$ and $\textit{not}$
  – Idea
    • $\neg a \approx \neg a \in X$
    • $\textit{not} \; a \approx a \notin X$
Motivation

- Classical versus default negation
  - Symbol \( \neg \) and \textit{not}
  - Idea
    - \( \neg a \approx \neg a \in X \)
    - \( \text{not } a \approx a \notin X \)
  - Example
    - \( \text{cross } \leftarrow \neg \text{train} \)
    - \( \text{cross } \leftarrow \text{not train} \)
Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{ \neg a \mid a \in \mathcal{A} \}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program $P$ over $\mathcal{A}$, classical negation is encoded by adding

$$P^\neg = \{ a \leftarrow b, \neg b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A} \}$$
Classical negation

• Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{-a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$

• Given a program $P$ over $\mathcal{A}$, classical negation is encoded by adding

$$P^\neg = \{a \leftarrow b, \neg b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

• A set $X$ of atoms is a stable model of a program $P$ over $\mathcal{A} \cup \overline{\mathcal{A}}$, if $X$ is a stable model of $P \cup P^\neg$
An example

• The program

\[ P = \{ a \leftarrow \text{not } b, \ b \leftarrow \text{not } a \} \cup \{ c \leftarrow b, \ \neg c \leftarrow b \} \]
An example

- The program

\[ P = \{ a \leftarrow \text{not } b, \ b \leftarrow \text{not } a \} \cup \{ c \leftarrow b, \ \neg c \leftarrow b \} \]

induces

\[ P^- = \begin{cases} 
  a & \leftarrow a, \neg a \\
  \neg a & \leftarrow a, \neg a \\
  b & \leftarrow a, \neg a \\
  \neg b & \leftarrow a, \neg a \\
  c & \leftarrow a, \neg a \\
  \neg c & \leftarrow a, \neg a \\
  a & \leftarrow b, \neg b \\
  \neg a & \leftarrow b, \neg b \\
  b & \leftarrow b, \neg b \\
  \neg b & \leftarrow b, \neg b \\
  c & \leftarrow b, \neg b \\
  \neg c & \leftarrow b, \neg b \\
  a & \leftarrow c, \neg c \\
  \neg a & \leftarrow c, \neg c \\
  b & \leftarrow c, \neg c \\
  \neg b & \leftarrow c, \neg c \\
  c & \leftarrow c, \neg c \\
  \neg c & \leftarrow c, \neg c 
\end{cases} \]
An example

• The program

\[ P = \{ a \leftarrow \text{not } b, \ b \leftarrow \text{not } a \} \cup \{ c \leftarrow b, \ \neg c \leftarrow b \} \]

induces

\[ P^- = \{ a \leftarrow a, \neg a \quad a \leftarrow b, \neg b \quad a \leftarrow c, \neg c \\
\neg a \leftarrow a, \neg a \quad \neg a \leftarrow b, \neg b \quad \neg a \leftarrow c, \neg c \\
 b \leftarrow a, \neg a \quad b \leftarrow b, \neg b \quad b \leftarrow c, \neg c \\
\neg b \leftarrow a, \neg a \quad \neg b \leftarrow b, \neg b \quad \neg b \leftarrow c, \neg c \\
c \leftarrow a, \neg a \quad c \leftarrow b, \neg b \quad c \leftarrow c, \neg c \\
\neg c \leftarrow a, \neg a \quad \neg c \leftarrow b, \neg b \quad \neg c \leftarrow c, \neg c \} \]

• The stable models of \( P \) are given by the ones of \( P \cup P^- \), viz \( \{ a \} \)
Properties

- The only inconsistent stable “model” is $X = \mathcal{A} \cup \overline{\mathcal{A}}$
Properties

- The only inconsistent stable “model” is $X = \mathcal{A} \cup \overline{\mathcal{A}}$
- Note Strictly speaking, an inconsistent set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model
Properties

- The only inconsistent stable “model” is $X = A \cup \overline{A}$
- **Note** Strictly speaking, an inconsistent set like $A \cup \overline{A}$ is not a model
- For a logic program $P$ over $A \cup \overline{A}$, exactly one of the following two cases applies:
  1. All stable models of $P$ are consistent or
  2. $X = A \cup \overline{A}$ is the only stable model of $P$
Train spotting

- $P_1 = \{\text{cross} \leftarrow \neg \text{train}\}$

- $P_2 = \{\text{cross} \leftarrow \neg \text{train}\}$

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$

- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{not train}\}$

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{not train}, \neg \text{cross} \leftarrow\}$
Train spotting

- $P_1 = \{\text{cross} \leftarrow \text{not train}\}$
  - stable model: $\{\text{cross}\}$
Train spotting

- $P_2 = \{\text{cross } \leftarrow \neg \text{train}\}$
Train spotting

- $P_2 = \{\text{cross } \leftarrow \neg \text{train}\}$
  - stable model: $\emptyset$
Train spotting

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$
Train spotting

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$
Train spotting

• $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
Train spotting

- \( P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\} \)
  - stable model: \( \{\text{cross}, \neg \text{cross}, \text{train}, \neg \text{train}\} \) inconsistent as \( \mathcal{A} \cup \bar{\mathcal{A}} \)
Train spotting

- \( P_5 = \{ \text{cross } \leftarrow \neg \text{train}, \neg \text{train } \leftarrow \text{not train} \} \)
Train spotting

- $P_5 = \{\text{cross } \leftarrow \neg \text{train}, \neg \text{train } \leftarrow \text{not train}\}$
  - stable model: $\{\text{cross, } \neg \text{train}\}$
Train spotting

- $P_1 = \{\text{cross} \leftarrow \neg \text{train} \}$
- $P_2 = \{\text{cross} \leftarrow \neg \text{train} \}$
- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \}$
- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \}$
- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train} \}$
- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{cross} \leftarrow \}$
Train spotting

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \text{not train}, \neg \text{cross} \leftarrow\}$
  - no stable model
Train spotting

- $P_1 = \{\text{cross } \leftarrow \text{not train}\}$
  - stable model: $\{\text{cross}\}$

- $P_2 = \{\text{cross } \leftarrow \neg\text{train}\}$
  - stable model: $\emptyset$

- $P_3 = \{\text{cross } \leftarrow \neg\text{train, } \neg\text{train } \leftarrow\}$
  - stable model: $\{\text{cross, } \neg\text{train}\}$

- $P_4 = \{\text{cross } \leftarrow \neg\text{train, } \neg\text{train } \leftarrow, \neg\text{cross } \leftarrow\}$
  - stable model: $\{\text{cross, } \neg\text{cross, } \neg\text{train, } \neg\text{train}\}$ inconsistent as $A \cup \bar{A}$

- $P_5 = \{\text{cross } \leftarrow \neg\text{train, } \neg\text{train } \leftarrow \text{not train}\}$
  - stable model: $\{\text{cross, } \neg\text{train}\}$

- $P_6 = \{\text{cross } \leftarrow \neg\text{train, } \neg\text{train } \leftarrow \text{not train, } \neg\text{cross } \leftarrow\}$
  - no stable model
We consider logic programs with default negation in rule heads.
Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{\tilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$
Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{\tilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$
- Given a program $P$ over $\mathcal{A}$, consider the program

$$
\tilde{P} = \{r \in P \mid \text{head}(r) \neq \text{not } a\}
\cup \{\leftarrow \text{body}(r) \cup \{\text{not } \tilde{a}\} \mid r \in P \text{ and } \text{head}(r) = \text{not } a\}
\cup \{\tilde{a} \leftarrow \text{not } a \mid r \in P \text{ and } \text{head}(r) = \text{not } a\}
$$
Default negation in rule heads

- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{ \tilde{a} \mid a \in \mathcal{A} \}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$
- Given a program $P$ over $\mathcal{A}$, consider the program

$$\tilde{P} = \{ r \in P \mid \text{head}(r) \neq \text{not } a \}$$
$$\cup \{ \leftarrow \text{body}(r) \cup \{ \text{not } \tilde{a} \} \mid r \in P \text{ and head}(r) = \text{not } a \}$$
$$\cup \{ \tilde{a} \leftarrow \text{not } a \mid r \in P \text{ and head}(r) = \text{not } a \}$$

- A set $X$ of atoms is a **stable model** of a program $P$ (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model $Y$ of $\tilde{P}$ over $\mathcal{A} \cup \tilde{\mathcal{A}}$
Outline

2 Two kinds of negation

3 Disjunctive logic programs
Disjunctive logic programs

- A disjunctive rule, \( r \), is of the form

\[
\begin{align*}
  a_1 & ; \ldots ; a_m \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o 
\end{align*}
\]

where \( 0 \leq m \leq n \leq o \) and each \( a_i \) is an atom for \( 0 \leq i \leq o \)

- A disjunctive logic program is a finite set of disjunctive rules
Disjunctive logic programs

- A disjunctive rule, \( r \), is of the form

\[
a_1 ; \ldots ; a_m \leftarrow a_{m+1}, \ldots, a_n, \text{not } a_{n+1}, \ldots, \text{not } a_o
\]

where \( 0 \leq m \leq n \leq o \) and each \( a_i \) is an atom for \( 0 \leq i \leq o \)

- A disjunctive logic program is a finite set of disjunctive rules

- Notation

\[
\text{head}(r) = \{a_1, \ldots, a_m\} \\
\text{body}(r) = \{a_{m+1}, \ldots, a_n, \text{not } a_{n+1}, \ldots, \text{not } a_o\} \\
\text{body}(r)^+ = \{a_{m+1}, \ldots, a_n\} \\
\text{body}(r)^- = \{a_{n+1}, \ldots, a_o\} \\
\text{atom}(P) = \bigcup_{r \in P} \left( \text{head}(r) \cup \text{body}(r)^+ \cup \text{body}(r)^- \right) \\
\text{body}(P) = \{\text{body}(r) \mid r \in P\}
\]
Disjunctive logic programs

- A disjunctive rule, \( r \), is of the form

\[
a_1 ; \ldots ; a_m \leftarrow a_{m+1}, \ldots, a_n, not a_{n+1}, \ldots, not a_o
\]

where \( 0 \leq m \leq n \leq o \) and each \( a_i \) is an atom for \( 0 \leq i \leq o \)

- A disjunctive logic program is a finite set of disjunctive rules

- Notation

\[
\begin{align*}
\text{head}(r) &= \{a_1, \ldots, a_m\} \\
\text{body}(r) &= \{a_{m+1}, \ldots, a_n, not a_{n+1}, \ldots, not a_o\} \\
\text{body}(r)^+ &= \{a_{m+1}, \ldots, a_n\} \\
\text{body}(r)^- &= \{a_{n+1}, \ldots, a_o\} \\
\text{atom}(P) &= \bigcup_{r \in P} \left( \text{head}(r) \cup \text{body}(r)^+ \cup \text{body}(r)^- \right) \\
\text{body}(P) &= \{\text{body}(r) \mid r \in P\}
\end{align*}
\]

- A program is called positive if \( \text{body}(r)^- = \emptyset \) for all its rules
Stable models

- Positive programs
  - A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $\text{head}(r) \cap X \neq \emptyset$ whenever $\text{body}(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program $P$ is denoted by $\text{min}_{\subseteq}(P)$
  - $\text{min}_{\subseteq}(P)$ corresponds to the $\subseteq$-minimal models of $P$ (ditto)
Stable models

- **Positive programs**
  - A set $X$ of atoms is **closed under** a positive program $P$ iff for any $r \in P$, $\text{head}(r) \cap X \neq \emptyset$ whenever $\text{body}(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program $P$ is denoted by $\text{min}_{\subseteq}(P)$
    - $\text{min}_{\subseteq}(P)$ corresponds to the $\subseteq$-minimal models of $P$ (ditto)

- **Disjunctive programs**
  - The **reduct**, $P^X$, of a disjunctive program $P$ relative to a set $X$ of atoms is defined by

$$P^X = \{\text{head}(r) \leftarrow \text{body}(r)^+ \mid r \in P \text{ and } \text{body}(r)^- \cap X = \emptyset\}$$
Stable models

- Positive programs
  - A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $\text{head}(r) \cap X \neq \emptyset$ whenever $\text{body}(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program $P$ is denoted by $\min_{\subseteq}(P)$
  - $\min_{\subseteq}(P)$ corresponds to the $\subseteq$-minimal models of $P$ (ditto)

- Disjunctive programs
  - The reduct, $P^X$, of a disjunctive program $P$ relative to a set $X$ of atoms is defined by
    $$P^X = \{\text{head}(r) \leftarrow \text{body}(r)^+ \mid r \in P \text{ and } \text{body}(r)^- \cap X = \emptyset\}$$
  - A set $X$ of atoms is a stable model of a disjunctive program $P$, if $X \in \min_{\subseteq}(P^X)$
A “positive” example

\[ P = \{ a \leftarrow b ; c \leftarrow a \} \]
A “positive” example

\[
P = \left\{ \begin{array}{c}
a \\ b \\ c \\ a \\ \end{array} \right. \left\uparrow \quad \left\downarrow \quad a \right. \right\}
\]

- The sets \{a, b\}, \{a, c\}, and \{a, b, c\} are closed under \(P\)
A “positive” example

\[ P = \left\{ \begin{array}{c} a \\ b ; c \leftrightarrow a \end{array} \right\} \]

- The sets \( \{a, b\} \), \( \{a, c\} \), and \( \{a, b, c\} \) are closed under \( P \)
- We have \( \min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\} \)
Graph coloring (reloaded)

node(1..6).

edge(1, (2; 3; 4)). edge(2, (4; 5; 6)). edge(3, (1; 4; 5)).
edge(4, (1; 2)). edge(5, (3; 4; 6)). edge(6, (2; 3; 5)).

color(X, r) ; color(X, b) ; color(X, g) :- node(X).

:- edge(X, Y), color(X, C), color(Y, C).
Graph coloring (reloaded)

node(1..6).

edge(1, (2;3;4)). edge(2, (4;5;6)). edge(3, (1;4;5)).
edge(4, (1;2)). edge(5, (3;4;6)). edge(6, (2;3;5)).

col(r). col(b). col(g).

color(X,C) : col(C) :- node(X).

:- edge(X,Y), color(X,C), color(Y,C).
More Examples

- $P_1 = \{a; b; c \leftarrow\}$
More Examples

- $P_1 = \{a \, ; \, b \, ; \, c \leftarrow\}$
  - stable models $\{a\}, \{b\}, \text{and} \{c\}$
More Examples

- $P_2 = \{a \mid b \mid c \leftarrow, \leftarrow a\}$
More Examples

• \( P_2 = \{ a ; b ; c \leftarrow , \leftarrow a \} \)
  – stable models \( \{ b \} \) and \( \{ c \} \)
More Examples

- $P_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \}$
More Examples

- $P_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \}$
  - stable model $\{b, c\}$
More Examples

- $P_4 = \{a ; b \leftarrow c, b \leftarrow \text{not } a, \text{not } c, a ; c \leftarrow \text{not } b\}$
More Examples

- $P_4 = \{a ; b \leftarrow c , b \leftarrow not a, not c , a ; c \leftarrow not b\}$
  - stable models $\{a\}$ and $\{b\}$
More Examples

• $P_1 = \{a ; b ; c \leftarrow\}$
  – stable models $\{a\}$, $\{b\}$, and $\{c\}$

• $P_2 = \{a ; b ; c \leftarrow , \leftarrow a\}$
  – stable models $\{b\}$ and $\{c\}$

• $P_3 = \{a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b\}$
  – stable model $\{b, c\}$

• $P_4 = \{a ; b \leftarrow c , b \leftarrow not a, not c , a ; c \leftarrow not b\}$
  – stable models $\{a\}$ and $\{b\}$
Some properties

- A disjunctive logic program may have zero, one, or multiple stable models.
- If $X$ is a stable model of a disjunctive logic program $P$, then $X$ is a model of $P$ (seen as a formula).
- If $X$ and $Y$ are stable models of a disjunctive logic program $P$, then $X \nsubseteq Y$. 

Some properties

- A disjunctive logic program may have zero, one, or multiple stable models.
- If $X$ is a stable model of a disjunctive logic program $P$, then $X$ is a model of $P$ (seen as a formula).
- If $X$ and $Y$ are stable models of a disjunctive logic program $P$, then $X \not\subseteq Y$.
- If $A \in X$ for some stable model $X$ of a disjunctive logic program $P$, then there is a rule $r \in P$ such that
  \[\text{body}(r)^+ \subseteq X, \text{body}(r)^- \cap X = \emptyset, \text{and head}(r) \cap X = \{A\}\]
An example with variables

\[ P = \left\{ \begin{array}{l}
  a(1, 2) \\
  b(X) ; c(Y) \leftarrow a(X, Y), \text{not } c(Y)
\end{array} \right\} \]
An example with variables

\[ P = \{ \begin{align*} a(1, 2) & \leftarrow a(X, Y), \text{not } c(Y) \\ b(X) ; c(Y) & \leftarrow a(X, Y) \end{align*} \} \]

\[ \text{ground}(P) = \{ \begin{align*} a(1, 2) & \leftarrow a(1, 1), \text{not } c(1) \\ b(1) ; c(1) & \leftarrow a(1, 2), \text{not } c(2) \\ b(2) ; c(1) & \leftarrow a(2, 1), \text{not } c(1) \\ b(2) ; c(2) & \leftarrow a(2, 2), \text{not } c(2) \end{align*} \} \]
An example with variables

\[ P = \begin{\{ } 
  a(1, 2) & \leftarrow a(X, Y), \text{not } c(Y) \\
  b(X) ; c(Y) & \leftarrow \ 
\end{\{ } \]

\[ \text{ground}(P) = \begin{\{ } 
  a(1, 2) & \leftarrow a(1, 1), \text{not } c(1) \\
  b(1) ; c(1) & \leftarrow a(1, 2), \text{not } c(2) \\
  b(1) ; c(2) & \leftarrow a(2, 1), \text{not } c(1) \\
  b(2) ; c(1) & \leftarrow a(2, 2), \text{not } c(2) \\
  b(2) ; c(2) & \leftarrow a(2, 2), \text{not } c(2) \\
\end{\{ } \]

For every stable model \( X \) of \( P \), we have

- \( a(1, 2) \in X \) and
- \( \{a(1, 1), a(2, 1), a(2, 2)\} \cap X = \emptyset \)
An example with variables

\[ \text{ground}(P) = \{\begin{array}{l}
a(1, 2) \leftarrow \\
b(1); c(1) \leftarrow a(1, 1), \text{not } c(1) \\
b(1); c(2) \leftarrow a(1, 2), \text{not } c(2) \\
b(2); c(1) \leftarrow a(2, 1), \text{not } c(1) \\
b(2); c(2) \leftarrow a(2, 2), \text{not } c(2) \\
\end{array} \} \]
An example with variables

\[ \text{ground}(P) = \begin{cases} 
    a(1, 2) & \leftarrow a(1, 1), \text{not } c(1) \\
    b(1) ; c(1) & \leftarrow a(1, 2), \text{not } c(2) \\
    b(2) ; c(2) & \leftarrow a(2, 1), \text{not } c(1) \\
    b(2) ; c(2) & \leftarrow a(2, 2), \text{not } c(2) 
\end{cases} \]

- Consider \( X = \{a(1, 2), b(1)\} \)
An example with variables

\[
ground(P)^X = \begin{cases} 
    a(1, 2) & \leftarrow \ a(1, 1) \\
    b(1); c(1) & \leftarrow \ a(1, 2) \\
    b(1); c(2) & \leftarrow \ a(2, 1) \\
    b(2); c(1) & \leftarrow \ a(2, 2) \\
    b(2); c(2) & \leftarrow \ a(2, 2) 
\end{cases}
\]

- Consider \( X = \{a(1, 2), b(1)\} \)
An example with variables

\[
\text{ground}(P)^X = \{ 
\begin{align*}
    a(1, 2) & \leftarrow a(1, 1) \\
    b(1); c(1) & \leftarrow a(1, 1) \\
    b(1); c(2) & \leftarrow a(1, 2) \\
    b(2); c(1) & \leftarrow a(2, 1) \\
    b(2); c(2) & \leftarrow a(2, 2)
\end{align*}
\}
\]

- Consider \( X = \{a(1, 2), b(1)\} \)
- We get \( \text{min}_{\subseteq} (\text{ground}(P)^X) = \{ \{a(1, 2), b(1)\}, \{a(1, 2), c(2)\} \} \)
An example with variables

\[
\text{ground}(P)^X = \begin{cases} 
  a(1, 2) & \leftarrow \\
  b(1) ; c(1) & \leftarrow a(1, 1) \\
  b(1) ; c(2) & \leftarrow a(1, 2) \\
  b(2) ; c(1) & \leftarrow a(2, 1) \\
  b(2) ; c(2) & \leftarrow a(2, 2) 
\end{cases}
\]

- Consider \( X = \{a(1, 2), b(1)\} \)
- We get \( \text{min} \subseteq (\text{ground}(P)^X) = \{ \{a(1, 2), b(1)\}, \{a(1, 2), c(2)\} \} \)
- \( X \) is a stable model of \( P \) because \( X \in \text{min} \subseteq (\text{ground}(P)^X) \)
An example with variables

\[
\text{ground}(P) = \begin{cases} 
a(1, 2) & \leftarrow 
b(1) ; c(1) & \leftarrow a(1, 1), \text{not } c(1) 
b(1) ; c(2) & \leftarrow a(1, 2), \text{not } c(2) 
b(2) ; c(1) & \leftarrow a(2, 1), \text{not } c(1) 
b(2) ; c(2) & \leftarrow a(2, 2), \text{not } c(2) 
\end{cases}
\]
An example with variables

ground(P) = \{
    a(1, 2) ← a(1, 1), \text{not} \ c(1),
    b(1) ; c(1) ← a(1, 2), \text{not} \ c(2),
    b(1) ; c(2) ← a(2, 1), \text{not} \ c(1),
    b(2) ; c(1) ← a(2, 2), \text{not} \ c(2),
    b(2) ; c(2) ← a(2, 2), \text{not} \ c(2),
\}

- Consider $X = \{a(1, 2), c(2)\}$
An example with variables

\[ \text{ground}(P)^X = \begin{cases} 
\{ a(1, 2) \} & \leftarrow \\
\{ b(1) ; c(1) \} & \leftarrow \ a(1, 1) \\
\{ b(2) ; c(1) \} & \leftarrow \ a(2, 1) \\
\end{cases} \]

- Consider \( X = \{ a(1, 2), c(2) \} \)
An example with variables

\[ \text{ground}(P)^X = \begin{cases} 
  a(1, 2) & \leftarrow a(1, 1) \\
  b(1) \land c(1) & \leftarrow a(1, 1) \\
  b(2) \land c(1) & \leftarrow a(2, 1) 
\end{cases} \]

- Consider \( X = \{a(1, 2), c(2)\} \)
- We get \( \text{min}_{\subseteq}(\text{ground}(P)^X) = \{\{a(1, 2)\}\} \)
An example with variables

\[ \text{ground}(P)^X = \left\{ \begin{array}{l}
    a(1, 2) \leftarrow \\
    b(1) ; c(1) \leftarrow a(1, 1) \\
    b(2) ; c(1) \leftarrow a(2, 1) \\
    c(1) \leftarrow a(1, 2) \\
    b(2) ; c(1) \leftarrow a(2, 1) \\
\end{array} \right. \]

- Consider \( X = \{a(1, 2), c(2)\} \)
- We get \( \text{min}_{\subseteq}(\text{ground}(P)^X) = \{ \{a(1, 2)\} \} \)
- \( X \) is no stable model of \( P \) because \( X \not\in \text{min}_{\subseteq}(\text{ground}(P)^X) \)
Consider disjunctive rules of the form

\[ a_1; \ldots; a_m; \text{not } a_{m+1}; \ldots; \text{not } a_n \leftarrow a_{n+1}, \ldots, a_o, \text{not } a_{o+1}, \ldots, \text{not } a_p \]

where \(0 \leq m \leq n \leq o \leq p\) and each \(a_i\) is an atom for \(0 \leq i \leq p\)
Default negation in rule heads

- Consider disjunctive rules of the form

\[ a_1 ; \ldots ; a_m ; \text{not } a_{m+1} ; \ldots ; \text{not } a_n \leftarrow a_{n+1}, \ldots, a_o, \text{not } a_{o+1}, \ldots, \text{not } a_p \]

where \( 0 \leq m \leq n \leq o \leq p \) and each \( a_i \) is an atom for \( 0 \leq i \leq p \)

- Given a program \( P \) over \( A \), consider the program

\[
\tilde{P} = \{ \text{head}(r)^+ \leftarrow \text{body}(r) \cup \{ \text{not } \tilde{a} \mid a \in \text{head}(r)^- \} \mid r \in P \} \\
\cup \{ \tilde{a} \leftarrow \text{not } a \mid r \in P \text{ and } a \in \text{head}(r)^- \}
\]

A set \( X \) of atoms is a stable model of a disjunctive program \( P \) (with default negation in rule heads) over \( A \), if \( X = Y \cap A \) for some stable model \( Y \) of \( \tilde{P} \) over \( A \cup \tilde{A} \).
Default negation in rule heads

- Consider disjunctive rules of the form
  \[ a_1; \ldots; a_m; \text{not } a_{m+1}; \ldots; \text{not } a_n \leftarrow a_{n+1}, \ldots, a_o, \text{not } a_{o+1}, \ldots, \text{not } a_p \]
  where \( 0 \leq m \leq n \leq o \leq p \) and each \( a_i \) is an atom for \( 0 \leq i \leq p \)

- Given a program \( P \) over \( \mathcal{A} \), consider the program
  \[
  \tilde{P} = \{ \text{head}(r)^+ \leftarrow \text{body}(r) \cup \{ \text{not } \tilde{a} \mid a \in \text{head}(r)^- \} \mid r \in P \} \\
  \cup \{ \tilde{a} \leftarrow \text{not } a \mid r \in P \text{ and } a \in \text{head}(r)^- \}
  \]

- A set \( X \) of atoms is a \text{stable model} of a disjunctive program \( P \) (with default negation in rule heads) over \( \mathcal{A} \), if \( X = Y \cap \mathcal{A} \) for some stable model \( Y \) of \( \tilde{P} \) over \( \mathcal{A} \cup \tilde{\mathcal{A}} \)
An example

- The program

\[ P = \{ a ; \text{not } a \leftarrow \} \]

\[ \sim P \] has two stable models, \( \{a\} \) and \( \{\sim a\} \)

This induces the stable models \( \{a\} \) and \( \emptyset \) of \( P \).
An example

- The program
  \[ P = \{ a \; ; \; not \; a \leftarrow \} \]

  yields
  \[ \tilde{P} = \{ a \leftarrow not \; \tilde{a} \} \cup \{ \tilde{a} \leftarrow not \; a \} \]
An example

- The program
  \[ P = \{ a ; \text{not } a \leftarrow \} \]

  yields
  \[ \tilde{P} = \{ a \leftarrow \text{not } \tilde{a} \} \cup \{ \tilde{a} \leftarrow \text{not } a \} \]

- \( \tilde{P} \) has two stable models, \( \{a\} \) and \( \{\tilde{a}\} \)
An example

- The program

\[ P = \{ a ; \text{not } a \leftarrow \} \]

yields

\[ \tilde{P} = \{ a \leftarrow \text{not } \tilde{a} \} \cup \{ \tilde{a} \leftarrow \text{not } a \} \]

- \( \tilde{P} \) has two stable models, \{a\} and \{\( \tilde{a} \)\}

- This induces the stable models \{a\} and \emptyset of \( P \)
Computational Aspects: Overview

4 Complexity
Outline

4 Complexity
Let $a$ be an atom and $X$ be a set of atoms.
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is P-complete
  - Deciding whether $a$ is in the stable model of $P$ is P-complete
Complexity

Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is P-complete
  - Deciding whether $a$ is in the stable model of $P$ is P-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is P-complete
  - Deciding whether $a$ is in a stable model of $P$ is NP-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is co-NP-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^P_2$-complete
Complexity

Let $a$ be an atom and $X$ be a set of atoms

- For a positive disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is co-NP-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP^{NP}$-complete

- For a disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is co-NP-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP^{NP}$-complete

- For a disjunctive logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is co-$NP^{NP}$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^P_3$-complete
Complexity

Let $a$ be an atom and $X$ be a set of atoms

- For a positive disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is co-NP-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP^{NP}$-complete

- For a disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is co-NP-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP^{NP}$-complete

- For a disjunctive logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is co-$NP^{NP}$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^P_3$-complete

- For a propositional theory $\Phi$:
  - Deciding whether $X$ is a stable model of $\Phi$ is co-NP-complete
  - Deciding whether $a$ is in a stable model of $\Phi$ is $NP^{NP}$-complete
References

Martin Gebser, Benjamin Kaufmann Roland Kaminski, and Torsten Schaub.
*Answer Set Solving in Practice.*
Synthesis Lectures on Artificial Intelligence and Machine Learning.
doi=10.2200/S00457ED1V01Y201211AIM019.

- See also: [http://potassco.sourceforge.net](http://potassco.sourceforge.net)