TECHNISCHE UNIVERSITÄT DRESDEN

Artificial Intelligence, Computational Logic

# PROBLEM SOLVING AND SEARCH IN ARTIFICIAL INTELLIGENCE

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

**Sarah Gaggl**

Dresden

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

# 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
  5. Extended language
- Language Extensions
  6. Two kinds of negation
  7. Disjunctive logic programs
- Computational Aspects
  8. Complexity

# Language: Overview

1. Extended language

# Outline

1. Extended language

# Outline

# 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). 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). 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).  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). 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).   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) }.
```

# 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

$$minimize\ \{\ w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n\ \}.$$

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

  $$\mathit{minimize}\ \{\ w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n\ \}.$$

  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

  $$\textit{minimize} \; \{ \; w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n \; \}.$$

  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

$$maximize\ \{\ w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n\ \}$$

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

# Optimization statement

- A maximize statement of the form

$$maximize \; \{ \; w_1@p_1 : \ell_1, \ldots, w_n@p_n : \ell_n \; \}$$

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

- 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

# Language Extensions: Overview

2. Two kinds of negation
3. Disjunctive logic programs

# Outline

2 Two kinds of negation

3 Disjunctive logic programs

# Motivation

- Classical versus default negation
  - Symbol $\neg$ and *not*

# Motivation

- Classical versus default negation
  - Symbol $\neg$ and $not$
  - Idea
    - $\neg a \quad \approx \quad \neg a \in X$
    - $not\ a \quad \approx \quad a \notin X$

# Motivation

- Classical versus default negation
  - Symbol $\neg$ and $not$
  - Idea
    - $\neg a \quad \approx \quad \neg a \in X$
    - $not\ a \quad \approx \quad a \notin X$
  - Example
    - $cross \leftarrow \neg train$
    - $cross \leftarrow not\ train$

# 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 $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program $P$ over $\mathcal{A}$, classical negation is encoded by adding

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

- 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

$$P = \{a \leftarrow not\ b,\ b \leftarrow not\ a\} \cup \{c \leftarrow b,\ \neg c \leftarrow b\}$$

# An example

- The program

$$P = \{a \leftarrow not\ b,\ b \leftarrow not\ a\} \cup \{c \leftarrow b,\ \neg c \leftarrow b\}$$

induces

$$P^{\neg} = \left\{ \begin{array}{rclcrclcrcl} a & \leftarrow & a, \neg a & & a & \leftarrow & b, \neg b & & a & \leftarrow & c, \neg c \\ \neg a & \leftarrow & a, \neg a & & \neg a & \leftarrow & b, \neg b & & \neg a & \leftarrow & c, \neg c \\ b & \leftarrow & a, \neg a & & b & \leftarrow & b, \neg b & & b & \leftarrow & c, \neg c \\ \neg b & \leftarrow & a, \neg a & & \neg b & \leftarrow & b, \neg b & & \neg b & \leftarrow & c, \neg c \\ c & \leftarrow & a, \neg a & & c & \leftarrow & b, \neg b & & c & \leftarrow & c, \neg c \\ \neg c & \leftarrow & a, \neg a & & \neg c & \leftarrow & b, \neg b & & \neg c & \leftarrow & c, \neg c \end{array} \right\}$$

# An example

- The program

$$P = \{a \leftarrow not\ b,\ b \leftarrow not\ a\} \cup \{c \leftarrow b,\ \neg c \leftarrow b\}$$

induces

$$P^{\neg} = \left\{ \begin{array}{rcl rcl rcl} a & \leftarrow & a, \neg a & a & \leftarrow & b, \neg b & a & \leftarrow & c, \neg c \\ \neg a & \leftarrow & a, \neg a & \neg a & \leftarrow & b, \neg b & \neg a & \leftarrow & c, \neg c \\ b & \leftarrow & a, \neg a & b & \leftarrow & b, \neg b & b & \leftarrow & c, \neg c \\ \neg b & \leftarrow & a, \neg a & \neg b & \leftarrow & b, \neg b & \neg b & \leftarrow & c, \neg c \\ c & \leftarrow & a, \neg a & c & \leftarrow & b, \neg b & c & \leftarrow & c, \neg c \\ \neg c & \leftarrow & a, \neg a & \neg c & \leftarrow & b, \neg b & \neg c & \leftarrow & c, \neg c \end{array} \right\}$$

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

# Train spotting

- $P_1 = \{cross \leftarrow not\ train\}$
- $P_2 = \{cross \leftarrow \neg train\}$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

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

# Train spotting

- $P_1 = \{cross \leftarrow not\ train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$ inconsistent as $\mathcal{A} \cup \bar{\mathcal{A}}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - no stable model

# Default negation in rule heads

- 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 $\widetilde{\mathcal{A}} = \{\widetilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \widetilde{\mathcal{A}} = \emptyset$

# Default negation in rule heads

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

$$
\begin{aligned}
\widetilde{P} \quad = \quad & \{r \in P \mid head(r) \neq not\ a\} \\
& \cup \{\leftarrow body(r) \cup \{not\ \widetilde{a}\} \mid r \in P \text{ and } head(r) = not\ a\} \\
& \cup \{\widetilde{a} \leftarrow not\ a \mid r \in P \text{ and } head(r) = not\ a\}
\end{aligned}
$$

# Default negation in rule heads

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

$$
\begin{array}{rcl}
\widetilde{P} & = & \{r \in P \mid head(r) \neq not\ a\} \\
& & \cup\ \{\leftarrow body(r) \cup \{not\ \widetilde{a}\} \mid r \in P \text{ and } head(r) = not\ a\} \\
& & \cup\ \{\widetilde{a} \leftarrow not\ a \mid r \in P \text{ and } head(r) = not\ a\}
\end{array}
$$

- 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 $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

# Outline

# 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

# Disjunctive logic programs

- A disjunctive rule, $r$, is of the form

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

  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

$$\begin{array}{rcl}
\mathit{head}(r) & = & \{a_1, \ldots, a_m\} \\
\mathit{body}(r) & = & \{a_{m+1}, \ldots, a_n, \mathit{not}\ a_{n+1}, \ldots, \mathit{not}\ a_o\} \\
\mathit{body}(r)^+ & = & \{a_{m+1}, \ldots, a_n\} \\
\mathit{body}(r)^- & = & \{a_{n+1}, \ldots, a_o\} \\
\mathit{atom}(P) & = & \bigcup_{r \in P} \left(\mathit{head}(r) \cup \mathit{body}(r)^+ \cup \mathit{body}(r)^-\right) \\
\mathit{body}(P) & = & \{\mathit{body}(r) \mid r \in P\}
\end{array}$$

# 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

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

- A program is called positive if $body(r)^- = \emptyset$ for all its rules

# Stable models

- Positive programs
  - A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $head(r) \cap X \neq \emptyset$ whenever $body(r)^+ \subseteq X$
    - $X$ 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)

# Stable models

- Positive programs
  - A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $head(r) \cap X \neq \emptyset$ whenever $body(r)^+ \subseteq X$
    - $X$ 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

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

# Stable models

- Positive programs
  - A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $head(r) \cap X \neq \emptyset$ whenever $body(r)^+ \subseteq X$
    - $X$ 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

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

  - A set $X$ of atoms is a stable model of a disjunctive program $P$, if $X \in \min_\subseteq(P^X)$

# A “positive” example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b \,;\, c & \leftarrow & a \end{array} \right\}$$

# A "positive" example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b\,;c & \leftarrow & a \end{array} \right\}$$

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under $P$

# A "positive" example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b \,;\, c & \leftarrow & a \end{array} \right\}$$

- 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 $\{a\}$, $\{b\}$, and $\{c\}$

# More Examples

- $P_2 = \{a \,; b \,; c \leftarrow \,, \leftarrow a\}$

# 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

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

# 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 $\{b, c\}$
- $P_4 = \{a \,;b \leftarrow c \,, \; b \leftarrow \mathit{not}\ a, \mathit{not}\ c \,, \; a \,;c \leftarrow \mathit{not}\ b\}$
  - 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 $X \not\subset Y$

# 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 $X \not\subset Y$
- 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
  $body(r)^+ \subseteq X$, $body(r)^- \cap X = \emptyset$, and $head(r) \cap X = \{A\}$

# An example with variables

$$P \quad = \quad \left\{ \begin{array}{lll} a(1,2) & \leftarrow & \\ b(X)\ ;c(Y) & \leftarrow & a(X,Y), not\ c(Y) \end{array} \right\}$$

# An example with variables

$$P = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X)\ ;c(Y) & \leftarrow & a(X,Y), \mathit{not}\ c(Y) \end{array} \right\}$$

$$\mathit{ground}(P) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ;c(1) & \leftarrow & a(1,1), \mathit{not}\ c(1) \\ b(1)\ ;c(2) & \leftarrow & a(1,2), \mathit{not}\ c(2) \\ b(2)\ ;c(1) & \leftarrow & a(2,1), \mathit{not}\ c(1) \\ b(2)\ ;c(2) & \leftarrow & a(2,2), \mathit{not}\ c(2) \end{array} \right\}$$

# An example with variables

$$P = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X) \; ; c(Y) & \leftarrow & a(X,Y), not\ c(Y) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) \; ; c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1) \; ; c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2) \; ; c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2) \; ; c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

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

$$ground(P) \quad = \quad \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ; c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\ ; c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\ ; c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\ ; c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

# An example with variables

$$ground(P) \quad = \quad \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ; c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\ ; c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\ ; c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\ ; c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), b(1)\}$

# An example with variables

$$
ground(P)^X \quad = \quad \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ; c(1) & \leftarrow & a(1,1) \\ b(1)\ ; c(2) & \leftarrow & a(1,2) \\ b(2)\ ; c(1) & \leftarrow & a(2,1) \\ b(2)\ ; c(2) & \leftarrow & a(2,2) \end{array} \right\}
$$

- Consider $X = \{a(1,2), b(1)\}$

# An example with variables

$$ground(P)^X \quad = \quad \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ;c(1) & \leftarrow & a(1,1) \\ b(1)\ ;c(2) & \leftarrow & a(1,2) \\ b(2)\ ;c(1) & \leftarrow & a(2,1) \\ b(2)\ ;c(2) & \leftarrow & a(2,2) \end{array} \right\}$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$

# An example with variables

$$
ground(P)^X \quad = \quad \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1) \\ b(1)\,;c(2) & \leftarrow & a(1,2) \\ b(2)\,;c(1) & \leftarrow & a(2,1) \\ b(2)\,;c(2) & \leftarrow & a(2,2) \end{array} \right\}
$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$
- $X$ is a stable model of $P$ because $X \in \min_{\subseteq}(ground(P)^X)$

# An example with variables

$$ground(P) = \left\{ \begin{array}{lll} a(1,2) & \leftarrow & \\ b(1) \; ; c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1) \; ; c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2) \; ; c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2) \; ; c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

# An example with variables

$$ground(P) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ; c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\ ; c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\ ; c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\ ; c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$

# An example with variables

$$
ground(P)^X \quad = \quad \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1) \\ & & \\ b(2)\,;c(1) & \leftarrow & a(2,1) \\ & & \end{array} \right\}
$$

- Consider $X = \{a(1,2), c(2)\}$

# An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\ ; c(1) & \leftarrow & a(1,1) \\ & & \\ b(2)\ ; c(1) & \leftarrow & a(2,1) \\ & & \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2)\}\ \}$

# An example with variables

$$
ground(P)^X = \left\{ \begin{array}{lll} a(1,2) & \leftarrow & \\ b(1)\ ; c(1) & \leftarrow & a(1,1) \\ & & \\ b(2)\ ; c(1) & \leftarrow & a(2,1) \\ & & \end{array} \right\}
$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2)\}\ \}$
- $X$ is no stable model of $P$ because $X \notin \min_{\subseteq}(ground(P)^X)$

# Default negation in rule heads

- Consider disjunctive rules of the form

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

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

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

  where $0 \le m \le n \le o \le p$ and each $a_i$ is an atom for $0 \le i \le p$

- Given a program $P$ over $\mathcal{A}$, consider the program

$$\begin{aligned} \widetilde{P} \quad = \quad & \{head(r)^+ \leftarrow body(r) \cup \{not\ \widetilde{a} \mid a \in head(r)^-\} \mid r \in P\} \\ & \cup \{\widetilde{a} \leftarrow not\ a \mid r \in P \text{ and } a \in head(r)^-\} \end{aligned}$$

# Default negation in rule heads

- Consider disjunctive rules of the form

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

  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

$$\begin{aligned} \widetilde{P} \quad = \quad & \{head(r)^+ \leftarrow body(r) \cup \{not\ \widetilde{a} \mid a \in head(r)^-\} \mid r \in P\} \\ & \cup \{\widetilde{a} \leftarrow not\ a \mid r \in P \text{ and } a \in head(r)^-\} \end{aligned}$$

- A set $X$ of atoms is a 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 $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

# An example

- The program

$$P = \{a \; ; \; not \; a \leftarrow\}$$

# An example

- The program

$$P = \{a \,;\, not\ a \leftarrow\}$$

yields

$$\widetilde{P} = \{a \leftarrow not\ \widetilde{a}\} \cup \{\widetilde{a} \leftarrow not\ a\}$$

# An example

- The program
$$P = \{a\ ;\ not\ a \leftarrow\}$$
yields
$$\widetilde{P} = \{a \leftarrow not\ \widetilde{a}\} \cup \{\widetilde{a} \leftarrow not\ a\}$$
- $\widetilde{P}$ has two stable models, $\{a\}$ and $\{\widetilde{a}\}$

# An example

- The program

$$P = \{a \,;\, not\ a \leftarrow\}$$

yields

$$\widetilde{P} = \{a \leftarrow not\ \widetilde{a}\} \cup \{\widetilde{a} \leftarrow not\ a\}$$

- $\widetilde{P}$ has two stable models, $\{a\}$ and $\{\widetilde{a}\}$
- This induces the stable models $\{a\}$ and $\emptyset$ of $P$

# Computational Aspects: Overview

4 Complexity

# Outline

# Complexity

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

# 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

# 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

# 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
- 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 $\Delta_2^P$-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 $\Delta_3^P$-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 $\Delta_3^P$-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.
Morgan and Claypool Publishers, 2012.
doi=10.2200/S00457ED1V01Y201211AIM019.

- See also: `http://potassco.sourceforge.net`