PROBLEM SOLVING AND SEARCH IN ARTIFICIAL INTELLIGENCE

Lecture 6 ASP II * slides adapted from Torsten Schaub [Gebser et al.(2012)]

Sarah Gaggl

Dresden, 13th and 27th May 2016
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 II

- Modeling
  - 1 Basic Modeling
  - 2 Methodology
- Language
  - 3 Motivation
  - 4 Core language
  - 5 Extended language
- Language Extensions
  - 6 Two kinds of negation
  - 7 Disjunctive logic programs
- Computational Aspects
  - 9 Complexity
Modeling: Overview

1. Basic Modeling
2. Methodology
Outline

1. Basic Modeling
2. Methodology
Modeling and Interpreting

- Problem
- Modeling
- Logic Program
- Solving
- Stable Models
- Interpreting
- Solution
Modeling

- For solving a problem class $C$ for a problem instance $I$, encode
  1. the problem instance $I$ as a set $P_I$ of facts and
  2. the problem class $C$ as a set $P_C$ of rules
  such that the solutions to $C$ for $I$ can be (polynomially) extracted from the stable models of $P_I \cup P_C$

- $P_I$ is (still) called problem instance
- $P_C$ is often called the problem encoding

- An encoding $P_C$ is uniform, if it can be used to solve all its problem instances
  That is, $P_C$ encodes the solutions to $C$ for any set $P_I$ of facts
Outline

1. Basic Modeling
2. Methodology
## Basic methodology

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generate and Test</strong></td>
<td>(or: Guess and Check)</td>
</tr>
<tr>
<td><strong>Generator</strong></td>
<td>Generate potential stable model candidates</td>
</tr>
<tr>
<td></td>
<td>(typically through non-deterministic constructs)</td>
</tr>
<tr>
<td><strong>Tester</strong></td>
<td>Eliminate invalid candidates</td>
</tr>
<tr>
<td></td>
<td>(typically through integrity constraints)</td>
</tr>
</tbody>
</table>
Basic methodology

**Methodology**

*Generate and Test* (or: *Guess and Check*)

- **Generator**: Generate potential stable model candidates (typically through non-deterministic constructs)
- **Tester**: Eliminate invalid candidates (typically through integrity constraints)

**Nutshell**

Logic program = Data + Generator + Tester (+ Optimizer)
Outline

1. Basic Modeling

2. Methodology
   - Satisfiability
   - Queens
   - Traveling Salesperson
• **Problem Instance:** A propositional formula $\phi$ in CNF
• **Problem Class:** Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true
Satisfiability testing

- **Problem Instance:** A propositional formula $\phi$ in CNF
- **Problem Class:** Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

- **Example:** Consider formula

  $$(a \lor \neg b) \land (\neg a \lor b)$$

- **Logic Program:**

  Generator:
  \[
  \{ a, b \} \leftarrow
  \]

  Tester:
  \[
  \leftarrow \text{not} \ a, b \\
  \leftarrow a, \text{not} \ b
  \]

  Stable models:
  \[
  X_1 = \{ a, b \} \\
  X_2 = \{ \}
  \]
Satisfiability testing

- **Problem Instance**: A propositional formula $\phi$ in CNF
- **Problem Class**: Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

- **Example**: Consider formula

$$(a \lor \neg b) \land (\neg a \lor b)$$

- **Logic Program**:

  Generator
  \[
  \{ a, b \} \leftarrow
  \]

  Tester
  \[
  \leftarrow \text{not } a, b
  \leftarrow a, \text{not } b
  \]

  Stable models
  \[
  X_1 = \{a, b\}
  X_2 = \{\}
  \]
Satisfiability testing

- **Problem Instance**: A propositional formula \( \phi \) in CNF
- **Problem Class**: Is there an assignment of propositional variables to true and false such that a given formula \( \phi \) is true

- **Example**: Consider formula

\[
(a \lor \neg b) \land (\neg a \lor b)
\]

- **Logic Program**:

\[
\begin{align*}
\textbf{Generator} & \quad \{a, b\} \quad \leftarrow \\
\textbf{Tester} & \quad \leftarrow \quad \text{not } a, b \\
& \quad \leftarrow \quad a, \text{not } b \\
\textbf{Stable models} & \quad X_1 = \{a, b\} \\
& \quad X_2 = \{\}
\end{align*}
\]
Satisfiability testing

- **Problem Instance**: A propositional formula $\phi$ in CNF
- **Problem Class**: Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

- **Example**: Consider formula

  $$(a \lor \neg b) \land (\neg a \lor b)$$

- **Logic Program**:

  Generator
  \[
  \{ a, b \} \leftarrow
  \]

  Tester
  
  \[
  \leftarrow \text{not } a, b \\
  \leftarrow a, \text{not } b
  \]

  Stable models
  \[
  X_1 = \{a, b\} \\
  X_2 = \{
  \} 
  \]
Outline

1. Basic Modeling

2. Methodology
   - Satisfiability
   - Queens
   - Traveling Salesperson
The n-Queens Problem

- Place $n$ queens on an $n \times n$ chess board
- Queens must not attack one another
Defining the Field

queens.lp

• Create file queens.lp
• Define the field
  – $n$ rows
  – $n$ columns
Running ...

$ gringo queens.lp --const n=5 | clasp
Answer: 1
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5)
SATISFIABLE

Models : 1
Time   : 0.000
  Prepare : 0.000
  Prepro. : 0.000
  Solving : 0.000
Guess a solution candidate
   by placing some queens on the board
Placing some Queens

Running ...

$ gringo queens.lp --const n=5 | clasp 3
Answer: 1
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5) queen(1,1)
Answer: 3
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5) queen(2,1)
SATISFIABLE

Models : 3+
...

TU Dresden, 13th and 27th May 2016 PSSAI
Placing some Queens: Answer 1

Answer 1

5 0 0 0 0
4 0 0 0 0
3 0 0 0 0
2 0 0 0 0
1 0 0 0 0

1 2 3 4 5
Placing some Queens: Answer 2

Answer 2

 TU Dresden, 13th and 27th May 2016   PSSAI
Placing some Queens: Answer 3

Answer 3
Placing \( n \) Queens

```lp
row(1..n).
col(1..n).
\{ queen(I,J) : row(I), col(J) \}.
:- not n \{ queen(I,J) \} n.
```

- Place exactly \( n \) queens on the board
Placing $n$ Queens

Running ...

$ gringo queens.lp --const n=5 | clasp 2

Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,1) queen(4,1) queen(3,1) \
queen(2,1) queen(1,1)

Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(1,2) queen(4,1) queen(3,1) \
queen(2,1) queen(1,1)

...
Placing $n$ Queens: Answer 1

Answer 1

TU Dresden, 13th and 27th May 2016

PSSAI

slide 28 of 200
Placing $n$ Queens: Answer 2

Answer 2

TU Dresden, 13th and 27th May 2016
Horizonal and Vertical Attack

queens.lp

row(1..n).
col(1..n).
\{ queen(I,J) : row(I), col(J) \}.
:- not n \{ queen(I,J) \} n.
:- queen(I,J), queen(I,J'), J != J'.

- Forbid horizontal attacks
Horizontal and Vertical Attack

**queens.lp**

```lp
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
```

- Forbid horizontal attacks
- Forbid vertical attacks
Running ...

$ gringo queens.lp --const n=5 | clasp
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,5) queen(4,4) queen(3,3) \
queen(2,2) queen(1,1)
...

Horiztonal and Vertical Attack
Horizontal and Vertical Attack: Answer 1
Diagonal Attack

queens.lp

row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- not n { queen(I,J) } n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.

• Forbid diagonal attacks
Running ...

$ gringo queens.lp --const n=5 | clasp
Answer: 1
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5) \nqueen(4,5) queen(1,4) queen(3,3) queen(5,2) queen(2,1)
SATISFIABLE

Models : 1+
Time    : 0.000
  Prepare : 0.000
  Prepro. : 0.000
  Solving : 0.000

TU Dresden, 13th and 27th May 2016
Diagonal Attack: Answer 1

Answer 1

TU Dresden, 13th and 27th May 2016
Optimizing

**queens-opt.lp**

1 \{ queen(I,1..n) \} 1 :- I = 1..n.
1 \{ queen(1..n,J) \} 1 :- J = 1..n.
    :- 2 \{ queen(D-J,J) \}, D = 2..2*n.
    :- 2 \{ queen(D+J,J) \}, D = 1-n..n-1.

- Encoding can be optimized
- Much faster to solve
And sometimes it rocks

$ clingo -c n=5000 queens-opt-diag.lp -config=jumpy -q -stats=3
clingo version 4.1.0
Solving...
SATISFIABLE

Models : 1+
Time : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time : 3758.320s

Choices : 288594554
Conflicts : 3442 (Analyzed: 3442)
Restarts : 17 (Average: 202.47 Last: 3442)
Model-Level : 7594728.0
Problems : 1 (Average Length: 0.00 Splits: 0)
Lemmas : 3442 (Deleted: 0)
  Binary : 0 (Ratio: 0.00%)
  Ternary : 0 (Ratio: 0.00%)
  Conflict : 3442 (Average Length: 229056.5 Ratio: 100.00%)
  Loop : 0 (Average Length: 0.0 Ratio: 0.00%)
  Other : 0 (Average Length: 0.0 Ratio: 0.00%)

Atoms : 75084857 (Original: 75069989 Auxiliary: 14868)
Bodies : 25090103
Equivalences : 125029999 (Atom=Atom: 50009999 Body=Body: 0 Other: 75020000)
Tight : Yes
Variables : 25024868 (Eliminated: 11781 Frozen: 25000000)
Constraints : 66664 (Binary: 35.6% Ternary: 0.0% Other: 64.4%)

Backjumps : 3442 (Average: 681.19 Max: 169512 Sum: 2344658)
  Executed : 3442 (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
  Bounded : 0 (Average: 0.00 Max: 0 Sum: 0 Ratio: 0.00%)

TU Dresden, 13th and 27th May 2016  PSSAI  slide 38 of 200
Outline

1. Basic Modeling

2. Methodology
   - Satisfiability
   - Queens
   - Traveling Salesperson
Traveling Salesperson
Traveling Salesperson

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)).
Traveling Salesperson

define

- 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)).

- cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
- cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
- cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
- cost(4,1,1).  cost(4,2,2).
- cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
- cost(6,2,4).  cost(6,3,3).  cost(6,5,1).
Traveling Salesperson

data(1..6).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).

eq(X,Y) :- cost(X,Y,1).
Traveling Salesperson

1 { cycle(X,Y) : edge(X,Y) } 1 :- node(X).
1 { cycle(X,Y) : edge(X,Y) } 1 :- node(Y).

reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).
:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
Traveling Salesperson

1 { cycle(X,Y) : edge(X,Y) } 1 :- node(X).
1 { cycle(X,Y) : edge(X,Y) } 1 :- node(Y).

reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).
Traveling Salesperson

1 \{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} 1 :- \text{node}(X).
1 \{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} 1 :- \text{node}(Y).

\text{reached}(Y) :- \text{cycle}(1,Y).
\text{reached}(Y) :- \text{cycle}(X,Y), \text{reached}(X).

:- \text{node}(Y), \text{not reached}(Y).
Traveling Salesperson

\[
\begin{align*}
&1 \{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} 1 :- \text{node}(X). \\
&1 \{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} 1 :- \text{node}(Y). \\
\text{reached}(Y) :- \text{cycle}(1,Y). \\
\text{reached}(Y) :- \text{cycle}(X,Y), \text{reached}(X). \\
&:- \text{node}(Y), \text{not} \ \text{reached}(Y). \\
\text{#minimize} \ { C,X,Y : \text{cycle}(X,Y), \ \text{cost}(X,Y,C) } .
\end{align*}
\]
Language: Overview

3 Motivation
4 Core language
5 Extended language
Outline

3 Motivation

4 Core language

5 Extended language
Basic language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.

- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
Basic language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, e.g. classical negation.
The expressiveness of a language can be enhanced by introducing new constructs.

To this end, we must address the following issues:
- What is the syntax of the new language construct?
- What is the semantics of the new language construct?
- How to implement the new language construct?

A way of providing semantics is to furnish a translation removing the new constructs, e.g., classical negation.

This translation might also be used for implementing the language extension.
Outline

3 Motivation

4 Core language

5 Extended language
Outline

3 Motivation

4 Core language
   - Integrity constraint
   - Choice rule
   - Cardinality rule
   - Weight rule

5 Extended language
   - Conditional literal
   - Optimization statement
Integrity constraint

- **Idea** Eliminate unwanted solution candidates
- **Syntax** An integrity constraint is of the form

\[ ← a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\)
- **Example**

\[ :- \text{edge(3,7)}, \text{color(3,red)}, \text{color(7,red)}. \]
Integrity constraint

- **Idea** Eliminate unwanted solution candidates
- **Syntax** An **integrity constraint** is of the form

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

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \)

- **Example** \(-\) edge(3,7), color(3,red), color(7,red).

- **Embedding** The above integrity constraint can be turned into the normal rule

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

where \( x \) is a new symbol, that is, \( x \not\in A \).
Integrity constraint

- **Idea**: Eliminate unwanted solution candidates
- **Syntax**: An integrity constraint is of the form

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

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\)
- **Example**: \(\text{ :- edge}(3,7), \text{ color}(3,\text{red}), \text{ color}(7,\text{red}).\)
- **Embedding**: The above integrity constraint can be turned into the normal rule

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

where \(x\) is a new symbol, that is, \(x \notin A\).
- **Another example**: \(P = \{a \leftarrow \text{not } b, b \leftarrow \text{not } a\}\)
  
  versus \(P' = P \cup \{\leftarrow a\}\) and \(P'' = P \cup \{\leftarrow \text{not } a\}\)
Outline

3 Motivation

4 Core language
   - Integrity constraint
   - Choice rule
   - Cardinality rule
   - Weight rule

5 Extended language
   - Conditional literal
   - Optimization statement
Choice rule

- **Idea** Choices over subsets
- **Syntax** A choice rule 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 \(1 \leq i \leq o\)
Choice rule

- **Idea** Choices over subsets
- **Syntax** A choice rule 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 \(1 \leq i \leq o\)

- **Informal meaning** If the body is satisfied by the stable model at hand, then any subset of \(\{a_1, \ldots, a_m\}\) can be included in the stable model.
Choice rule

- **Idea** Choices over subsets
- **Syntax** A choice rule 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 \(1 \leq i \leq o\)

- **Informal meaning** If the body is satisfied by the stable model at hand, then any subset of \(\{a_1, \ldots, a_m\}\) can be included in the stable model

- **Example**

\[ \{ \text{buy(pizza)}; \text{buy(wine)}; \text{buy(corn)} \} \leftarrow \text{at(grocery)}. \]
Choice rule

- **Idea** Choices over subsets
- **Syntax** A choice rule 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 \(1 \leq i \leq o\)

- **Informal meaning** If the body is satisfied by the stable model at hand, then any subset of \(\{a_1, \ldots, a_m\}\) can be included in the stable model

- **Example**

  \{ buy(pizza); buy(wine); buy(corn) \} :- at(grocery).

- **Another Example**

  \[ P = \{\{a\} \leftarrow b, b \leftarrow\} \] has two stable models: \{b\} and \{a, b\}
Embedding in normal rules

- A choice rule of form

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

can be translated into \(2m + 1\) normal rules

\[
\begin{align*}
b & \leftarrow a_{m+1}, \ldots, a_n, \text{not } a_{n+1}, \ldots, \text{not } a_o \\
a_1 & \leftarrow b, \text{not } a'_1 \quad \ldots \quad a_m & \leftarrow b, \text{not } a'_m \\
a'_1 & \leftarrow \text{not } a_1 \quad \ldots \quad a'_m & \leftarrow \text{not } a_m
\end{align*}
\]

by introducing new atoms \(b, a'_1, \ldots, a'_m\).
Embedding in normal rules

- A choice rule of form

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

can be translated into \(2m + 1\) normal rules

\[
\begin{align*}
 b & \leftarrow a_{m+1}, \ldots, a_n, \text{not } a_{n+1}, \ldots, \text{not } a_o \\
 a_1 & \leftarrow b, \text{not } a_1' \ldots a_m \leftarrow b, \text{not } a_m' \\
a_1' & \leftarrow \text{not } a_1 \ldots a_m' \leftarrow \text{not } a_m
\end{align*}
\]

by introducing new atoms \(b, a_1', \ldots, a_m'\).
Embedding in normal rules

- A choice rule of form

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

can be translated into \(2m + 1\) normal rules

\[
\begin{align*}
b & \leftarrow a_{m+1}, \ldots, a_n, \text{not } a_{n+1}, \ldots, \text{not } a_o \\
a_1 & \leftarrow b, \text{not } a_1' \ldots a_m \\
a_1' & \leftarrow \text{not } a_1 \ldots a_m' \\
a_m' & \leftarrow \text{not } a_m
\end{align*}
\]

by introducing new atoms \(b, a_1', \ldots, a_m'\).
Outline

3 Motivation

4 Core language
   - Integrity constraint
   - Choice rule
   - **Cardinality rule**
   - Weight rule

5 Extended language
   - Conditional literal
   - Optimization statement
Cardinality rule

- **Idea** Control (lower) cardinality of subsets
- **Syntax** A cardinality rule is the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, not a_{m+1}, \ldots, not a_n \} \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \
\( l \) is a non-negative integer.
Cardinality rule

- **Idea** Control (lower) cardinality of subsets
- **Syntax** A cardinality rule is the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) is a non-negative integer.

- **Informal meaning** The head atom belongs to the stable model, if at least \( l \) elements of the body are included in the stable model.

- **Note** \( l \) acts as a lower bound on the body.
Cardinality rule

- **Idea** Control (lower) cardinality of subsets
- **Syntax** A cardinality rule is the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\); \(l\) is a non-negative integer.

- **Informal meaning** The head atom belongs to the stable model, if at least \(l\) elements of the body are included in the stable model.

- **Note** \(l\) acts as a lower bound on the body.

- **Example**
  \[ \text{pass(c42)} : - 2 \{ \text{pass(a1)}; \text{pass(a2)}; \text{pass(a3)} \}. \]
Cardinality rule

- **Idea** Control (lower) cardinality of subsets
- **Syntax** A cardinality rule is the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) is a non-negative integer.

- **Informal meaning** The head atom belongs to the stable model, if at least \( l \) elements of the body are included in the stable model
- **Note** \( l \) acts as a lower bound on the body

- **Example**
  \[
  \text{pass(c42)} : - 2 \{ \text{pass(a1)}; \text{pass(a2)}; \text{pass(a3)} \}.
  \]
- **Another Example** \( P = \{ a \leftarrow 1\{b,c\}, \ b \leftarrow \} \) has stable model \( \{a, b\} \)
Embedding in normal rules

- Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, not a_{m+1}, \ldots, not a_n \} \]

by \[ a_0 \leftarrow ctr(1, l) \]

where atom \( ctr(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model
Embedding in normal rules

- Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

by \[ a_0 \leftarrow \text{ctr}(1, l) \]

where atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.

- The definition of \( \text{ctr}/2 \) is given for \( 0 \leq k \leq l \) by the rules

\[
\begin{align*}
\text{ctr}(i, k + 1) & \leftarrow \text{ctr}(i + 1, k), a_i \\
\text{ctr}(i, k) & \leftarrow \text{ctr}(i + 1, k) & \text{for } 1 \leq i \leq m \\
\text{ctr}(j, k + 1) & \leftarrow \text{ctr}(j + 1, k), \text{not } a_j \\
\text{ctr}(j, k) & \leftarrow \text{ctr}(j + 1, k) & \text{for } m + 1 \leq j \leq n \\
\text{ctr}(n + 1, 0) & \leftarrow \\
\end{align*}
\]
Embedding in normal rules

• Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

by \[ a_0 \leftarrow \text{ctr}(1, l) \]

where atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model

• The definition of \( \text{ctr}/2 \) is given for \( 0 \leq k \leq l \) by the rules

\[
\begin{align*}
\text{ctr}(i, k+1) & \leftarrow \text{ctr}(i + 1, k), a_i \\
\text{ctr}(i, k) & \leftarrow \text{ctr}(i + 1, k) & \text{for } 1 \leq i \leq m \\
\text{ctr}(j, k+1) & \leftarrow \text{ctr}(j + 1, k), \text{not } a_j \\
\text{ctr}(j, k) & \leftarrow \text{ctr}(j + 1, k) & \text{for } m + 1 \leq j \leq n \\
\text{ctr}(n + 1, 0) & \leftarrow \\
\end{align*}
\]
Embedding in normal rules

- Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, not a_{m+1}, \ldots, not a_n \} \]

by \[ a_0 \leftarrow ctr(1, l) \]

where atom \( ctr(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model

- The definition of \( ctr/2 \) is given for \( 0 \leq k \leq l \) by the rules

\[
\begin{align*}
ctr(i, k+1) & \leftarrow ctr(i + 1, k), a_i \\
ctr(i, k) & \leftarrow ctr(i + 1, k) & \text{for } 1 \leq i \leq m \\
ctr(j, k+1) & \leftarrow ctr(j + 1, k), not a_j \\
ctr(j, k) & \leftarrow ctr(j + 1, k) & \text{for } m+1 \leq j \leq n \\
ctr(n + 1, 0) & \leftarrow 
\end{align*}
\]
Embedding in normal rules

- Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

by \[ a_0 \leftarrow \text{ctr}(1, l) \]

where atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.

- The definition of \( \text{ctr}/2 \) is given for \( 0 \leq k \leq l \) by the rules

\[
\begin{align*}
\text{ctr}(i, k+1) & \leftarrow \text{ctr}(i+1, k), a_i \\
\text{ctr}(i, k) & \leftarrow \text{ctr}(i+1, k) \quad \text{for } 1 \leq i \leq m \\
\text{ctr}(j, k+1) & \leftarrow \text{ctr}(j+1, k), \text{not } a_j \\
\text{ctr}(j, k) & \leftarrow \text{ctr}(j+1, k) \quad \text{for } m+1 \leq j \leq n \\
\text{ctr}(n+1, 0) & \leftarrow
\end{align*}
\]
Embedding in normal rules

• Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

by \[ a_0 \leftarrow \text{ctr}(1, l) \]

where atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model

• The definition of \( \text{ctr}/2 \) is given for \( 0 \leq k \leq l \) by the rules

\[
\begin{align*}
\text{ctr}(i, k+1) & \leftarrow \text{ctr}(i+1, k), a_i \\
\text{ctr}(i, k) & \leftarrow \text{ctr}(i+1, k) \\
& \quad \text{for } 1 \leq i \leq m \\
\text{ctr}(j, k+1) & \leftarrow \text{ctr}(j+1, k), \text{not } a_j \\
\text{ctr}(j, k) & \leftarrow \text{ctr}(j+1, k) \\
& \quad \text{for } m+1 \leq j \leq n
\end{align*}
\]

\[ \text{ctr}(n + 1, 0) \leftarrow \]
Embedding in normal rules

• Replace each cardinality rule

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, not a_{m+1}, \ldots, not a_n \} \]

by \[ a_0 \leftarrow ctr(1, l) \]

where atom \( ctr(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model

• The definition of \( ctr/2 \) is given for \( 0 \leq k \leq l \) by the rules

\[
\begin{align*}
ctr(i, k+1) & \leftarrow ctr(i+1, k), a_i & \text{for } 1 \leq i \leq m \\
ctr(i, k) & \leftarrow ctr(i+1, k) & \\
ctr(j, k+1) & \leftarrow ctr(j+1, k), not a_j & \text{for } m+1 \leq j \leq n \\
ctr(j, k) & \leftarrow ctr(j+1, k) & \\
ctr(n+1, 0) & \leftarrow \\
\end{align*}
\]
An example

- Program \(\{a \leftarrow, \ c \leftarrow 1 \ \{a, b}\}\) has the stable model \(\{a, c\}\)
An example

- Program \{a \leftarrow, c \leftarrow 1 \{a, b\}\} has the stable model \{a, c\}
- Translating the cardinality rule yields the rules

\[
\begin{align*}
    a & \leftarrow \\
    ctr(1, 2) & \leftarrow ctr(2, 1), a \\
    ctr(1, 1) & \leftarrow ctr(2, 1) \\
    ctr(2, 2) & \leftarrow ctr(3, 1), b \\
    ctr(2, 1) & \leftarrow ctr(3, 1) \\
    ctr(1, 1) & \leftarrow ctr(2, 0), a \\
    ctr(1, 0) & \leftarrow ctr(2, 0) \\
    ctr(2, 1) & \leftarrow ctr(3, 0), b \\
    ctr(2, 0) & \leftarrow ctr(3, 0) \\
    ctr(3, 0) & \leftarrow
\end{align*}
\]

having stable model \{a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c\}
An example

- Program \( \{a \leftarrow, \ c \leftarrow 1 \ {a, b}\} \) has the stable model \( \{a, c\} \)
- Translating the cardinality rule yields the rules

\[
\begin{align*}
a & \leftarrow \\
c & \leftarrow \ ctr(1, 1) \\
ctr(1, 2) & \leftarrow \ ctr(2, 1), a \\
ctr(1, 1) & \leftarrow \ ctr(2, 1) \\
ctr(2, 2) & \leftarrow \ ctr(3, 1), b \\
ctr(2, 1) & \leftarrow \ ctr(3, 1) \\
ctr(1, 1) & \leftarrow \ ctr(2, 0), a \\
ctr(1, 0) & \leftarrow \ ctr(2, 0) \\
ctr(2, 1) & \leftarrow \ ctr(3, 0), b \\
ctr(2, 0) & \leftarrow \ ctr(3, 0) \\
ctr(3, 0) & \leftarrow
\end{align*}
\]

having stable model \( \{a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c\} \)
An example

- Program \{a ←, c ← 1 \{a, b\}\} has the stable model \{a, c\}
- Translating the cardinality rule yields the rules

\[
\begin{align*}
a & \leftarrow \quad c & \leftarrow & \text{ctr}(1, 1) \\
ctr(1, 2) & \leftarrow & \text{ctr}(2, 1), a \\
ctr(1, 1) & \leftarrow & \text{ctr}(2, 1) \\
ctr(2, 2) & \leftarrow & \text{ctr}(3, 1), b \\
ctr(2, 1) & \leftarrow & \text{ctr}(3, 1) \\
ctr(1, 1) & \leftarrow & \text{ctr}(2, 0), a \\
ctr(1, 0) & \leftarrow & \text{ctr}(2, 0) \\
ctr(2, 1) & \leftarrow & \text{ctr}(3, 0), b \\
ctr(2, 0) & \leftarrow & \text{ctr}(3, 0) \\
ctr(3, 0) & \leftarrow &
\end{align*}
\]

having stable model \{a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c\}
An example

- Program \( \{ a \leftarrow, c \leftarrow 1 \{ a, b \} \} \) has the stable model \( \{ a, c \} \)
- Translating the cardinality rule yields the rules

\[
a \leftarrow \\
c \leftarrow ctr(1, 1)\\nctr(1, 2) \leftarrow ctr(2, 1), a\\nctr(1, 1) \leftarrow ctr(2, 1)\\nctr(2, 2) \leftarrow ctr(3, 1), b\\nctr(2, 1) \leftarrow ctr(3, 1)\\nctr(1, 1) \leftarrow ctr(2, 0), a\\nctr(1, 0) \leftarrow ctr(2, 0)\\nctr(2, 1) \leftarrow ctr(3, 0), b\\nctr(2, 0) \leftarrow ctr(3, 0)\\nctr(3, 0) \leftarrow
\]

having stable model \( \{ a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c \} \)
An example

- Program $\{a \leftarrow, c \leftarrow 1 \{a, b\}\}$ has the stable model $\{a, c\}$
- Translating the cardinality rule yields the rules

$$
\begin{align*}
  a & \leftarrow \\
  c & \leftarrow ctr(1, 1) \\
  ctr(1, 2) & \leftarrow ctr(2, 1), a \\
  ctr(1, 1) & \leftarrow ctr(2, 1) \\
  ctr(2, 2) & \leftarrow ctr(3, 1), b \\
  ctr(2, 1) & \leftarrow ctr(3, 1) \\
  ctr(1, 1) & \leftarrow ctr(2, 0), a \\
  ctr(1, 0) & \leftarrow ctr(2, 0) \\
  ctr(2, 1) & \leftarrow ctr(3, 0), b \\
  ctr(2, 0) & \leftarrow ctr(3, 0) \\
  ctr(3, 0) & \leftarrow
\end{align*}
$$

having stable model $\{a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c\}$
An example

- Program \( \{ a \leftarrow, c \leftarrow 1 \{a, b\} \} \) has the stable model \( \{a, c\} \)
- Translating the cardinality rule yields the rules

\[
\begin{align*}
  a & \leftarrow \\
  ctr(1, 2) & \leftarrow ctr(2, 1), a \\
  ctr(1, 1) & \leftarrow ctr(2, 1) \\
  ctr(2, 2) & \leftarrow ctr(3, 1), b \\
  ctr(2, 1) & \leftarrow ctr(3, 1) \\
  ctr(1, 1) & \leftarrow ctr(2, 0), a \\
  ctr(1, 0) & \leftarrow ctr(2, 0) \\
  ctr(2, 1) & \leftarrow ctr(3, 0), b \\
  ctr(2, 0) & \leftarrow ctr(3, 0) \\
  ctr(3, 0) & \leftarrow
\end{align*}
\]

having stable model \( \{a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c\} \)
... and vice versa

- A normal rule

\[ a_0 \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \]

can be represented by the cardinality rule

\[ a_0 \leftarrow n \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]
Cardinality rules with upper bounds

- A rule of the form

\[
a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} u
\]  

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\); \(l\) and \(u\) are non-negative integers.
Cardinality rules with upper bounds

- A rule of the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} u \]  

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\);
\(l\) and \(u\) are non-negative integers

stands for

\[ a_0 \leftarrow b, \text{not } c \\
 b \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \\
c \leftarrow u+1 \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

where \(b\) and \(c\) are new symbols
Cardinality rules with upper bounds

• A rule of the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} u \]  

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \);
\( l \) and \( u \) are non-negative integers

stands for

\[ a_0 \leftarrow b, \text{not } c \]
\[ b \leftarrow l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]
\[ c \leftarrow u+1 \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \]

where \( b \) and \( c \) are new symbols

• Note The single constraint in the body of the cardinality rule (1) is referred to as a cardinality constraint
Cardinality constraints

- **Syntax** A cardinality constraint is of the form

\[
l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} \leq u
\]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\);
\(l\) and \(u\) are non-negative integers.
Cardinality constraints

• Syntax A cardinality constraint is of the form

\[ l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} u \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) and \( u \) are non-negative integers

• Informal meaning A cardinality constraint is satisfied by a stable model \( X \), if the number of its contained literals satisfied by \( X \) is between \( l \) and \( u \) (inclusive)
Cardinality constraints

- **Syntax** A cardinality constraint is of the form

\[
  l \{ a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \} u
\]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\); \(l\) and \(u\) are non-negative integers.

- **Informal meaning** A cardinality constraint is satisfied by a stable model \(X\), if the number of its contained literals satisfied by \(X\) is between \(l\) and \(u\) (inclusive).

- In other words, if

\[
l \leq |(\{a_1, \ldots, a_m\} \cap X) \cup (\{a_{m+1}, \ldots, a_n\} \setminus X)| \leq u
\]
Cardinality constraints as heads

- A rule of the form

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

where \(0 \leq m \leq n \leq o \leq p\) and each \(a_i\) is an atom for \(1 \leq i \leq p\); \(l\) and \(u\) are non-negative integers.
Cardinality constraints as heads

- A rule of the form

\[
\begin{align*}
l \{a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n\} & \quad u \leftarrow a_{n+1}, \ldots, a_o, \text{not } a_{o+1}, \ldots, \text{not } a_p \\
\end{align*}
\]

where \(0 \leq m \leq n \leq o \leq p\) and each \(a_i\) is an atom for \(1 \leq i \leq p\); \(l\) and \(u\) are non-negative integers

stands for

\[
\begin{align*}
b & \leftarrow a_{n+1}, \ldots, a_o, \text{not } a_{o+1}, \ldots, \text{not } a_p \\
\{a_1, \ldots, a_m\} & \leftarrow b \\
c & \leftarrow l \{a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n\} \quad u \\
& \leftarrow b, \text{not } c
\end{align*}
\]

where \(b\) and \(c\) are new symbols
Cardinality constraints as heads

- A rule of the form

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

where \( 0 \leq m \leq n \leq o \leq p \) and each \( a_i \) is an atom for \( 1 \leq i \leq p \); \( l \) and \( u \) are non-negative integers

stands for

\[
\begin{align*}
  b & \leftarrow a_{n+1}, \ldots, a_o, \text{not } a_{o+1}, \ldots, \text{not } a_p \\
  \{a_1, \ldots, a_m\} & \leftarrow b \\
  c & \leftarrow l \{a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n\} \ u \\
  & \leftarrow b, \text{not } c
\end{align*}
\]

where \( b \) and \( c \) are new symbols

- **Example** \( 1\{ \text{color(v42,red); color(v42,green); color(v42,blue) } \}1. \)
Outline

3 Motivation

4 Core language
   - Integrity constraint
   - Choice rule
   - Cardinality rule
   - Weight rule

5 Extended language
   - Conditional literal
   - Optimization statement
### Weight rule

- **Syntax** A weight rule is the form

\[
a_0 \leftarrow l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : not a_{m+1}, \ldots, w_n : not a_n \}
\]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom; \(l\) and \(w_i\) are integers for \(1 \leq i \leq n\)

- A weighted literal \(w_i : \ell_i\) associates each literal \(\ell_i\) with a weight \(w_i\)
Weight rule

- **Syntax** A weight rule is the form
  
  \[ a_0 \leftarrow l \{ \; w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : not a_{m+1}, \ldots, w_n : not a_n \; \} \]
  
  where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
  \( l \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- A weighted literal \( w_i : \ell_i \) associates each literal \( \ell_i \) with a weight \( w_i \)

- **Note** A cardinality rule is a weight rule where \( w_i = 1 \) for \( 0 \leq i \leq n \)
Weight constraints

- **Syntax** A *weight constraint* is of the form

\[ l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \text{not } a_{m+1}, \ldots, w_n : \text{not } a_n \} u \]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom;
\(l, u\) and \(w_i\) are integers for \(1 \leq i \leq n\)
Weight constraints

- **Syntax** A weight constraint is of the form

  \[ l \{ \ w_1 : a_1, \ldots, \ w_m : a_m, \ w_{m+1} : not \ a_{m+1}, \ldots, \ w_n : not \ a_n \ \} \ u \]

  where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
  \( l, u \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- **Meaning** A weight constraint is satisfied by a stable model \( X \), if

  \[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m \leq i \leq n, a_i \notin X} w_i \right) \leq u \]
Weight constraints

- **Syntax** A weight constraint is of the form

  \[ l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : not a_{m+1}, \ldots, w_n : not a_n \} u \]

  where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
  \( l, u \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- **Meaning** A weight constraint is satisfied by a stable model \( X \), if

  \[ l \leq \left( \sum_{1 \leq i \leq m,a_i \in X} w_i + \sum_{m < i \leq n,a_i \notin X} w_i \right) \leq u \]

- **Note** (Cardinality and) weight constraints amount to constraints on (count and) sum aggregate functions
Weight constraints

- **Syntax** A weight constraint is of the form

\[ l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : not a_{m+1}, \ldots, w_n : not a_n \} u \]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom; \(l, u\) and \(w_i\) are integers for \(1 \leq i \leq n\)

- **Meaning** A weight constraint is satisfied by a stable model \(X\), if

\[
l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \notin X} w_i \right) \leq u
\]

- **Note** (Cardinality and) weight constraints amount to constraints on (count and) sum aggregate functions

- **Example**

\[
10 \{ 4:课程(db); 6:课程(ai); 8:课程(project); 3:课程(xml) \} 20
\]
Outline

3 Motivation

4 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

5 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 ‘\( p(1..3). \ q(2). \)’

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

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 ‘\( p(1..3). \quad q(2). \)’

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

is instantiated to

\[
\begin{align*}
  r(1); \quad r(3) & : - r(1), \quad r(3); \quad 1 \{ r(1), \quad r(3) \}.
\end{align*}
\]
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 ‘\( p(1..3). \quad q(2). \)’

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

  is instantiated to

  \[
  r(1); \quad r(3) \quad :- \quad r(1), \quad r(3), \quad 1 \quad \{ \quad r(1), \quad r(3) \}.
  \]
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 ‘\( p(1..3). \; q(2). \)’

\[
\begin{align*}
  r(X) : p(X), \; \text{not} \; q(X) & : - \; r(X) : p(X), \; \text{not} \; q(X) ; \; 1 \{ \; r(X) : p(X), \; \text{not} \; q(X) \} .
\end{align*}
\]

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 \( \{ \ell \mid \ell_1, \ldots, \ell_n \} \)

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

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

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

is instantiated to

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

3 Motivation

4 Core language
   - Integrity constraint
   - Choice rule
   - Cardinality rule
   - Weight rule

5 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 minimize \{ \ -w_1 @ p_1 : \ell_1, \ldots, -w_n @ p_n : \ell_n \ \}
Optimization statement

- A maximize statement of the form

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

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

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

\[
\#\text{maximize } \{ 250\cdot1:hd(1), 500\cdot1:hd(2), 750\cdot1:hd(3), 1000\cdot1:hd(4) \ \}.
\#\text{minimize } \{ 30\cdot2:hd(1), 40\cdot2:hd(2), 60\cdot2:hd(3), 80\cdot2:hd(4) \ \}.
\]

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

6. Two kinds of negation
7. Disjunctive logic programs
Outline

6 Two kinds of negation

7 Disjunctive logic programs
Motivation

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

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

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

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

- 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 \( P \cup P^{-} \)
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

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

• The program

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

induces

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

• The stable models of \( P \) are given by the ones of \( P \cup P^n \), 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

* $P_1 = \{ \text{cross } \leftarrow \text{not train} \}$

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

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

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

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

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

- $P_3 = \{cross \leftarrow \neg train, \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
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 \( A \cup \bar{A} \)
Train spotting

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train}\}$
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

\[ P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \; \neg \text{train} \leftarrow \text{not train}, \; \neg \text{cross} \leftarrow \} \]
Train spotting

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

- \( P_1 = \{\text{cross } \leftarrow \text{not train}\} \)
  - stable model: \{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: \{cross, \neg\text{train}\}

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

- \( P_5 = \{\text{cross } \leftarrow \neg\text{train}, \neg\text{train }\leftarrow \text{not train}\} \)
  - stable model: \{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
Default negation in rule heads

- We consider logic programs with default negation in rule heads

Given an alphabet $A$ of atoms, let $\overline{A} = \{\overline{a} | a \in A\}$ such that $A \cap \overline{A} = \emptyset$.

Given a program $P$ over $A$, consider the program $\overline{P} = \{r \in P | head(r) \neq \neg a\} \cup \{\neg \overline{a} | r \in P \text{ and } head(r) = \neg a\} \cup \{\overline{a} \leftarrow \neg a | r \in P \text{ and } head(r) = \neg a\}$.

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

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

• 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

6 Two kinds of negation

7 Disjunctive logic programs
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
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

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

\[
a_1 ; \ldots ; a_m \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o
\]

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

- \( \text{head}(r) = \{a_1, \ldots, a_m\} \)
- \( \text{body}(r) = \{a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o\} \)
- \( \text{body}(r)^+ = \{a_{m+1}, \ldots, a_n\} \)
- \( \text{body}(r)^- = \{a_{n+1}, \ldots, a_o\} \)
- \( \text{atom}(P) = \bigcup_{r \in P} \left( \text{head}(r) \cup \text{body}(r)^+ \cup \text{body}(r)^- \right) \)
- \( \text{body}(P) = \{\text{body}(r) \mid r \in P\} \)

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 $\min_{\subseteq}(P)$
  - $\min_{\subseteq}(P)$ 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 $r \in P$, $\text{head}(r) \cap X \neq \emptyset$ 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 $\text{min}_{\subseteq}(P)$
    - $\text{min}_{\subseteq}(P)$ 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

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

  - A set $X$ of atoms is a **stable model** of a disjunctive program $P$, if $X \in \text{min}_{\subseteq}(P^X)$
A “positive” example

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

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

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

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

- 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

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

• $P_4 = \{a \ ; b \gets c \ , \ b \gets \not a, \not c \ , \ a \ ; c \gets \not b\}$
• $P_4 = \{a\; ;\; b \leftarrow c\; ,\; b \leftarrow \text{not a, not c} ,\; a\; ;\; c \leftarrow \text{not b}\}$
  – 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 $X \not\subset Y$
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 \( \text{body}(r)^+ \subseteq X, \text{body}(r)^- \cap X = \emptyset \), and \( \text{head}(r) \cap X = \{A\} \).
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

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

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

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

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

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

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

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

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

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

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

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

- 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

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

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

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

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

- 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

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

- Consider \( X = \{a(1, 2), c(2)\} \)
- We get \( \min_\subseteq(\text{ground}(P)^X) = \{ \{a(1, 2)\} \} \)
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 \( X = \{ a(1, 2), c(2) \} \)
- We get \( \text{min}_{\subseteq} (\text{ground}(P)^X) = \{ \{ a(1, 2) \} \} \)
- \( X \) is no stable model of \( P \) because \( X \notin \text{min}_{\subseteq} (\text{ground}(P)^X) \)
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 \( 0 \leq i \leq p \)
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 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 \( \mathcal{A} \cup \tilde{\mathcal{A}} \).
Default negation in rule heads

• Consider disjunctive rules of the form

\[ a_1; \ldots; a_m; \neg a_{m+1}; \ldots; \neg a_n \leftarrow a_{n+1}, \ldots, a_o, \neg a_{o+1}, \ldots, \neg a_p \]

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

\[
\tilde{P} = \{ \text{head}(r)^+ \leftarrow \text{body}(r) \cup \{ \neg \tilde{a} \mid a \in \text{head}(r)^- \} \mid r \in P \} \\
\cup \{ \tilde{a} \leftarrow \neg a \mid r \in P \text{ and } a \in \text{head}(r)^- \}
\]

• A set \(X\) of atoms is a stable model of a disjunctive program \(P\) (with default negation in rule heads) over \(\mathcal{A}\), 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

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

yields

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

- The program

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

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

\[ P = \{ a \; ; \; not \; a \leftrightarrow \} \]

yields

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

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

• This induces the stable models \( \{ a \} \) and \( \emptyset \) of \( P \)
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

- 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
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 \( \Delta^P_2 \)-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$^P$-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$^P$-complete

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

TU Dresden, 13th and 27th May 2016
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\(^\text{NP} \)-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\(^\text{NP} \)-complete

- For a disjunctive logic program \( P \) with optimization statements:
  - Deciding whether \( X \) is an optimal stable model of \( P \) is co-NP\(^\text{NP} \)-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\(^\text{NP} \)-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)