PROBLEM SOLVING AND SEARCH IN ARTIFICIAL INTELLIGENCE

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

Sarah Gaggl

Dresden
Agenda

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

- Modeling
  1. Basic Modeling
  2. Methodology
- Language
  3. Motivation
  4. Core language
Modeling: Overview

1. Basic Modeling
2. Methodology
Outline

1. Basic Modeling
2. Methodology
Modeling and Interpreting

Problem

Logic Program

Solution

Stable Models

Modeling

Solving

Interpreting
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

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

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

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Stable models</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, b} \leftarrow$</td>
<td>$\leftarrow not a, b$</td>
<td>$X_1 = {a, b}$</td>
</tr>
<tr>
<td>$\leftarrow a, not b$</td>
<td></td>
<td>$X_2 = {}$</td>
</tr>
</tbody>
</table>
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
  $\leftarrow$ not $a, b$
  $\leftarrow$ $a, \neg b$

  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*}
  \text{Generator} & \quad \{a, b\} \quad \leftarrow \\
  \text{Tester} & \quad \leftarrow \quad \text{not } a, b \\
  & \quad \leftarrow \quad a, \text{not } b \\
  \text{Stable models} & \quad X_1 \quad = \quad \{a, b\} \\
  & \quad X_2 \quad = \quad \{\} 
  \end{align*}
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

- Create file `queens.lp`
- Define the field
  - $n$ rows
  - $n$ columns
$ 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
Placing some Queens

**queens.lp**

```prolog
row(1..n).
col(1..n).
\{ queen(I,J) : row(I), col(J) \}.
```

- 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+ ...

Placing some Queens: Answer 1

Answer 1

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
\hline
1 & \times & \times & \times & \\
2 & \times & \times & \times & \\
3 & \times & \times & \times & \\
4 & \times & \times & \times & \\
5 & \times & \times & \times & \\
\end{array}
\]
Placing some Queens: Answer 2

```
5
Z0Z0Z
4
0Z0Z0
3
Z0Z0Z
2
0Z0Z0
1
L0Z0Z
```

Answer 2
Placing some Queens: Answer 3

Answer 3

TU Dresden PSSAI slide 25 of 101
Placing $n$ Queens

- Place exactly $n$ queens on the board

```prolog
queens.lp

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

\[
\begin{array}{ccccc}
& 1 & 2 & 3 & 4 & 5 \\
1 & & & & & \\
2 & & & & & \\
3 & & & & & \\
4 & & & & & \\
5 & & & & & \\
\end{array}
\]
Placing $n$ Queens: Answer 2

Answer 2
Horizontal 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

\[
\begin{align*}
\text{row}(1..n). \\
\text{col}(1..n). \\
\{ \text{queen}(I,J) : \text{row}(I), \text{col}(J) \}. \\
:- \text{not } n \{ \text{queen}(I,J) \} \text{ } n. \\
:- \text{queen}(I,J), \text{queen}(I,J'), J' = J'. \\
:- \text{queen}(I,J), \text{queen}(I',J), I' = I'.
\end{align*}
\]

- 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) \ncol(1) col(2) col(3) col(4) col(5) \nqueen(5,5) queen(4,4) queen(3,3) \nqueen(2,2) queen(1,1)
...
Horizontal and Vertical Attack: Answer 1

Answer 1

5
4
3
2
1

1 2 3 4 5
Diagonal Attack

queens.lp

\begin{verbatim}
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'.
\end{verbatim}

- Forbid diagonal attacks
Diagonal Attack

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
Diagonal Attack: Answer 1

Answer 1
Optimizing

**queens-opt.lp**

```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%)
Outline

1. Basic Modeling

2. Methodology
   - Satisfiability
   - Queens
   - 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)).
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

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).
define
  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

\[
\text{node}(1..6).
\]

\[
\text{cost}(1,2,2). \quad \text{cost}(1,3,3). \quad \text{cost}(1,4,1).
\]
\[
\text{cost}(2,4,2). \quad \text{cost}(2,5,2). \quad \text{cost}(2,6,4).
\]
\[
\text{cost}(3,1,3). \quad \text{cost}(3,4,2). \quad \text{cost}(3,5,2).
\]
\[
\text{cost}(4,1,1). \quad \text{cost}(4,2,2).
\]
\[
\text{cost}(5,3,2). \quad \text{cost}(5,4,2). \quad \text{cost}(5,6,1).
\]
\[
\text{cost}(6,2,4). \quad \text{cost}(6,3,3). \quad \text{cost}(6,5,1).
\]

\[
\text{edge}(X,Y) :- \text{cost}(X,Y,\_).
\]
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) .
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).
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).
Traveling Salesperson

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

Motivation

Core language
Outline

3  Motivation

4  Core 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?

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.
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, eg. 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, eg. classical negation.

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

3 Motivation

4 Core language
Outline

3 Motivation

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

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

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

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\)
- **Example**  
  \[\neg \text{edge}(3,7), \text{color}(3,\text{red}), \text{color}(7,\text{red}).\]
Integrity constraint

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

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

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

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

where \(x\) is a new symbol, that is, \(x \not\in A\).
3 Motivation

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

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

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

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

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

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

by introducing new atoms \(b, a'_1, \ldots, a'_m\).
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 \(b, a'_1, \ldots, a'_m\).
Outline

4 Core language
  - Integrity constraint
  - Choice rule
  - Cardinality rule
  - Weight rule
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
\[ \text{pass(c42)} \leftarrow 2 \{ \text{pass(a1)}; \text{pass(a2)}; \text{pass(a3)} \}. \]

Another Example
\[ P = \{ a \leftarrow 1 \{ b, c \} \}, b \leftarrow \} \text{has stable model} \{ a, b \}. \]
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
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

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

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
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

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

for \( m + 1 \leq j \leq n \)
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 \( a_0 \leftarrow ctr(1, l) \)

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

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

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

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

- Replace each cardinality rule

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\leftarrow$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(1, 2)$</td>
<td>$ctr(1, 1)$</td>
</tr>
<tr>
<td></td>
<td>$ctr(1, 2)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(1, 1)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(2, 2)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(2, 1)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(1, 1)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(1, 0)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(2, 1)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(2, 0)$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td></td>
<td>$ctr(3, 0)$</td>
<td>$\leftarrow$</td>
</tr>
</tbody>
</table>

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

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

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

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

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

having stable model \{a, \text{ctr}(3, 0), \text{ctr}(2, 0), \text{ctr}(1, 0), \text{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

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

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 \(\{a, \ ctr(3, 0), \ ctr(2, 0), \ ctr(1, 0), \ ctr(1, 1), \ c\}\)
... 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

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

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

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

(1)

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

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

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

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

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 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

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

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
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

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

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

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

3 Motivation

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

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

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

\[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \not\in X} w_i \right) \leq u \]
Weight constraints

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

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

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

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

\[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \not\in X} w_i \right) \leq u \]

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

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

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

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**

\[
10 \{ \begin{array}{l}
4: \text{course(db)}; 6: \text{course(ai)}; 8: \text{course(project)}; 3: \text{course(xml)} \\
\end{array} \} 20
\]
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