# Answer Set Solving in Practice

Martin Gebser and Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

# Rough Roadmap

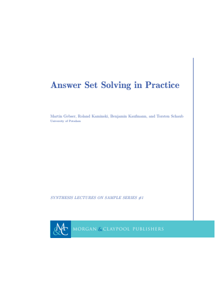

1 Introduction
2 Language
3 Modeling
4 Grounding
5 Foundations
6 Solving
7 Systems
8 Applications

# Resources

- Course material
  - `http://www.cs.uni-potsdam.de/wv/lehre`
  - `http://moodle.cs.uni-potsdam.de`
  - `http://potassco.sourceforge.net/teaching.html`
- Systems
  - clasp `http://potassco.sourceforge.net`
  - dlv `http://www.dlvsystem.com`
  - smodels `http://www.tcs.hut.fi/Software/smodels`
  - gringo `http://potassco.sourceforge.net`
  - lparse `http://www.tcs.hut.fi/Software/smodels`
  - clingo `http://potassco.sourceforge.net`
  - iclingo `http://potassco.sourceforge.net`
  - oclingo `http://potassco.sourceforge.net`
  - asparagus `http://asparagus.cs.uni-potsdam.de`

# The Potassco Book

1. Motivation
2. Introduction
3. Basic modeling
4. Grounding
5. Characterizations
6. Solving
7. Systems
8. Advanced modeling
9. Conclusions




## Resources

- http://potassco.sourceforge.net/book.html
- http://potassco.sourceforge.net/teaching.html

Potassco

## Literature

Books [4], [29], [53]

Surveys [50], [2], [39], [21], [11]

Articles [41], [42], [6], [61], [54], [49], [40], etc.

# Language: Overview

1. Motivation
2. Core language
   - Integrity constraint
   - Choice rule
   - Cardinality rule
   - Weight rule
3. Extended language
   - Conditional literal
   - Optimization statement
4. smodels format
5. ASP language standard

Potassco

# Outline


1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# 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
- This translation might also be used for implementing the language extension

Potassco

# 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
- 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
- This translation might also be used for implementing the language extension

# Outline


1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Outline

1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Integrity constraint

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

$$\leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n$$

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

$$x \leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n, \sim x$$

where $x$ is a new symbol, that is, $x \notin \mathcal{A}$.

- Another example $P = \{a \leftarrow \sim b,\ b \leftarrow \sim a\}$
versus $P' = P \cup \{\leftarrow a\}$ and $P'' = P \cup \{\leftarrow \sim a\}$

Potassco

# Integrity constraint

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

$$\leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n$$

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

$$x \leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n, \sim x$$

where $x$ is a new symbol, that is, $x \notin \mathcal{A}$.

- Another example $P = \{a \leftarrow \sim b,\ b \leftarrow \sim a\}$
versus $P' = P \cup \{\leftarrow a\}$ and $P'' = P \cup \{\leftarrow \sim a\}$

## Integrity constraint

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

$$\leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n$$

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

$$x \leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n, \sim x$$

where $x$ is a new symbol, that is, $x \notin \mathcal{A}$.

- Another example $P = \{a \leftarrow \sim b,\ b \leftarrow \sim a\}$
versus $P' = P \cup \{\leftarrow a\}$ and $P'' = P \cup \{\leftarrow \sim a\}$

# Outline


1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Choice rule

- Idea Choices over subsets
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o$$

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 $\{a_1, \dots, a_m\}$ 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 $\{a, b\}$

Potassco

# Choice rule

- Idea Choices over subsets
- Syntax A choice rule is of the form

  $$\{a_1,\dots,a_m\} \leftarrow a_{m+1},\dots,a_n,\sim a_{n+1},\dots,\sim a_o$$

  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 $\{a_1,\dots,a_m\}$ 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 $\{a,b\}$

# Choice rule

- Idea Choices over subsets
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o$$

  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 $\{a_1, \dots, a_m\}$ 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 $\{a, b\}$

Potassco

# Choice rule

- Idea Choices over subsets
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o$$

  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 $\{a_1, \dots, a_m\}$ 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

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, {\sim}a_{n+1}, \dots, {\sim}a_o$$

can be translated into $2m+1$ normal rules

$$\begin{array}{lcl} a' & \leftarrow & a_{m+1}, \dots, a_n, {\sim}a_{n+1}, \dots, {\sim}a_o \\ a_1 \leftarrow a', {\sim}\overline{a_1} & \dots & a_m \leftarrow a', {\sim}\overline{a_m} \\ \overline{a_1} \leftarrow {\sim}a_1 & \dots & \overline{a_m} \leftarrow {\sim}a_m \end{array}$$

by introducing new atoms $a', \overline{a_1}, \dots, \overline{a_m}$.

# Embedding in normal rules

- A choice rule of form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o$$

can be translated into $2m + 1$ normal rules

$$\begin{array}{rcl} a' & \leftarrow & a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o \\ a_1 & \leftarrow & a', \sim\overline{a_1} \qquad \dots \qquad a_m \leftarrow a', \sim\overline{a_m} \\ \overline{a_1} & \leftarrow & \sim a_1 \qquad\quad\; \dots \qquad \overline{a_m} \leftarrow \sim a_m \end{array}$$

by introducing new atoms $a', \overline{a_1}, \dots, \overline{a_m}$.

# Embedding in normal rules

- A choice rule of form

$$\{a_1,\dots,a_m\} \leftarrow a_{m+1},\dots,a_n,\sim a_{n+1},\dots,\sim a_o$$

can be translated into $2m+1$ normal rules

$$\begin{array}{lcl} a' & \leftarrow & a_{m+1},\dots,a_n,\sim a_{n+1},\dots,\sim a_o \\ a_1 & \leftarrow & a',\sim\overline{a_1} \qquad \dots \qquad a_m \leftarrow a',\sim\overline{a_m} \\ \overline{a_1} & \leftarrow & \sim a_1 \qquad\quad\ \dots \qquad \overline{a_m} \leftarrow \sim a_m \end{array}$$

by introducing new atoms $a',\overline{a_1},\dots,\overline{a_m}$.

# Outline

1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Cardinality rule

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

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n \; \}$$

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 `pass(c42) :- 2 { pass(a1), pass(a2), pass(a3) }.`
- Another Example $P = \{a \leftarrow 1\{b, c\}, \; b \leftarrow\}$ has stable model $\{a, b\}$

Potassco

# Cardinality rule

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

$$a_0 \leftarrow l \ \{ \ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \ \}$$

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 `pass(c42) :- 2 { pass(a1), pass(a2), pass(a3) }.`
- Another Example $P = \{a \leftarrow 1\{b, c\}, \ b \leftarrow\}$ has stable model $\{a, b\}$

Potassco

## Cardinality rule

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

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \; \}$$

  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** `pass(c42) :- 2 { pass(a1), pass(a2), pass(a3) }.`
- **Another Example** $P = \{a \leftarrow 1\{b, c\}, \; b \leftarrow\}$ has stable model $\{a, b\}$

Potassco

## Cardinality rule

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

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \; \}$$

  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 `pass(c42) :- 2 { pass(a1), pass(a2), pass(a3) }.`
- Another Example $P = \{a \leftarrow 1\{b, c\}, \; b \leftarrow\}$ has stable model $\{a, b\}$

Potassco

# Embedding in normal rules

- Replace each cardinality rule

$$a_0 \leftarrow l \ \{ \ a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \ \}$$

by $\quad a_0 \leftarrow ctr(1, l)$

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

$$\begin{array}{rcll} ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\ ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\ ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), {\sim}a_j & \\ ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\ ctr(n+1, 0) & \leftarrow & & \end{array}$$

# Embedding in normal rules

- Replace each cardinality rule

$$a_0 \leftarrow l \ \{ \ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \ \}$$

by $\quad a_0 \leftarrow ctr(1, l)$

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

$$\begin{array}{rcll}
ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\
ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\
ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), \sim a_j & \\
ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\
ctr(n+1, 0) & \leftarrow & &
\end{array}$$

# Embedding in normal rules

- Replace each cardinality rule

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \; \}$$

by $\quad a_0 \leftarrow ctr(1, l)$

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

$$\begin{array}{rcll} ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\ ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\ ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), \sim a_j & \\ ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\ ctr(n+1, 0) & \leftarrow & & \end{array}$$

# Embedding in normal rules

- Replace each cardinality rule

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \; \}$$

by $\quad a_0 \leftarrow ctr(1, l)$

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

$$\begin{array}{rcll} ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\ ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\ ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), {\sim}a_j & \\ ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\ ctr(n+1, 0) & \leftarrow & & \end{array}$$

Potassco

# Embedding in normal rules

- Replace each cardinality rule

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \; \}$$

by $\quad a_0 \leftarrow ctr(1, l)$

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

$$\begin{array}{rcll}
ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\
ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\
ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), \sim a_j & \\
ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\
ctr(n+1, 0) & \leftarrow & &
\end{array}$$

# Embedding in normal rules

- Replace each cardinality rule

$$a_0 \leftarrow l \ \{ \ a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \ \}$$

by $\quad a_0 \leftarrow ctr(1, l)$

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

$$\begin{array}{rcll} ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\ ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\ & & & \\ ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), \sim a_j & \\ ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\ & & & \\ ctr(n+1, 0) & \leftarrow & & \end{array}$$

## Embedding in normal rules

- Replace each cardinality rule

  $$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \; \}$$

  by $\quad a_0 \leftarrow ctr(1, l)$

  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

$$\begin{array}{rcll} ctr(i, k{+}1) & \leftarrow & ctr(i+1, k), a_i & \\ ctr(i, k) & \leftarrow & ctr(i+1, k) & \text{for } 1 \leq i \leq m \\ ctr(j, k{+}1) & \leftarrow & ctr(j+1, k), \sim a_j & \\ ctr(j, k) & \leftarrow & ctr(j+1, k) & \text{for } m+1 \leq j \leq n \\ ctr(n+1, 0) & \leftarrow & & \end{array}$$

# An example

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

$$
\begin{array}{rclrcl}
a & \leftarrow & \qquad & c & \leftarrow & ctr(1,1) \\
 & & & ctr(1,2) & \leftarrow & ctr(2,1), a \\
 & & & ctr(1,1) & \leftarrow & ctr(2,1) \\
 & & & ctr(2,2) & \leftarrow & ctr(3,1), b \\
 & & & ctr(2,1) & \leftarrow & ctr(3,1) \\
 & & & ctr(1,1) & \leftarrow & ctr(2,0), a \\
 & & & ctr(1,0) & \leftarrow & ctr(2,0) \\
 & & & ctr(2,1) & \leftarrow & ctr(3,0), b \\
 & & & ctr(2,0) & \leftarrow & ctr(3,0) \\
 & & & ctr(3,0) & \leftarrow &
\end{array}
$$

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

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

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

## . . . and vice versa

- A normal rule

$$a_0 \leftarrow a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n,$$

can be represented by the cardinality rule

$$a_0 \leftarrow n\ \{a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n\}$$

## Cardinality rules with upper bounds

- A rule of the form

$$a_0 \leftarrow l \ \{ \ a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \ \} \ u$$

  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

$$\begin{array}{rcl} a_0 & \leftarrow & b, \sim c \\ b & \leftarrow & l \ \{ \ a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \ \} \\ c & \leftarrow & u{+}1 \ \{ \ a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n \ \} \end{array}$$

  where $b$ and $c$ are new symbols

- The single constraint in the body of the above cardinality rule is referred to as a cardinality constraint

# Cardinality rules with upper bounds

- A rule of the form

$$a_0 \leftarrow l \ \{ \ a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \ \} \ u$$

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

$$\begin{array}{rcl} a_0 & \leftarrow & b, {\sim}c \\ b & \leftarrow & l \ \{ \ a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \ \} \\ c & \leftarrow & u{+}1 \ \{ \ a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \ \} \end{array}$$

where $b$ and $c$ are new symbols

- The single constraint in the body of the above cardinality rule is referred to as a cardinality constraint

# Cardinality rules with upper bounds

- A rule of the form

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \; \} \; u$$

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

$$\begin{array}{rcl} a_0 & \leftarrow & b, {\sim}c \\ b & \leftarrow & l \; \{ \; a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \; \} \\ c & \leftarrow & u{+}1 \; \{ \; a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \; \} \end{array}$$

where $b$ and $c$ are new symbols

- The single constraint in the body of the above cardinality rule is referred to as a cardinality constraint

Potassco

## Cardinality constraints

- Syntax A cardinality constraint is of the form

$$l \; \{ \; a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n \; \} \; u$$

  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

- Syntax A cardinality constraint is of the form

$$l \ \{ \ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \ \} \ u$$

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

- Syntax A cardinality constraint is of the form

$$l \ \{ \ a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n \ \} \ u$$

  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

$$l \leq |\,(\{a_1, \dots, a_m\} \cap X) \cup (\{a_{m+1}, \dots, a_n\} \setminus X)\,| \leq u$$

# Cardinality constraints as heads

- A rule of the form

$$l\ \{a_1,\dots,a_m,\sim a_{m+1},\dots,\sim a_n\}\ u \leftarrow a_{n+1},\dots,a_o,\sim a_{o+1},\dots,\sim a_p$$

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

$$\begin{array}{rcl} b & \leftarrow & a_{n+1},\dots,a_o,\sim a_{o+1},\dots,\sim a_p \\ \{a_1,\dots,a_m\} & \leftarrow & b \\ c & \leftarrow & l\ \{a_1,\dots,a_m,,\sim a_{m+1},\dots,\sim a_n\}\ u \\ & \leftarrow & b,\sim c \end{array}$$

where $b$ and $c$ are new symbols

- Example `1 { color(v42,red),color(v42,green),color(v42,blue) } 1.`

Potassco

# Cardinality constraints as heads

- A rule of the form

$$l \; \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\} \; u \leftarrow a_{n+1}, \dots, a_o, \sim a_{o+1}, \dots, \sim a_p$$

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

$$\begin{array}{rcl} b & \leftarrow & a_{n+1}, \dots, a_o, \sim a_{o+1}, \dots, \sim a_p \\ \{a_1, \dots, a_m\} & \leftarrow & b \\ c & \leftarrow & l \; \{a_1, \dots, a_m, , \sim a_{m+1}, \dots, \sim a_n\} \; u \\ & \leftarrow & b, \sim c \end{array}$$

where $b$ and $c$ are new symbols

- Example `1 { color(v42,red),color(v42,green),color(v42,blue) } 1.`

# Cardinality constraints as heads

- A rule of the form

$$l\ \{a_1,\dots,a_m,\sim a_{m+1},\dots,\sim a_n\}\ u \leftarrow a_{n+1},\dots,a_o,\sim a_{o+1},\dots,\sim a_p$$

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

$$\begin{array}{rcl} b & \leftarrow & a_{n+1},\dots,a_o,\sim a_{o+1},\dots,\sim a_p \\ \{a_1,\dots,a_m\} & \leftarrow & b \\ c & \leftarrow & l\ \{a_1,\dots,a_m,,\sim a_{m+1},\dots,\sim a_n\}\ u \\ & \leftarrow & b,\sim c \end{array}$$

where $b$ and $c$ are new symbols

- Example `1 { color(v42,red),color(v42,green),color(v42,blue) } 1.`

# Full-fledged cardinality rules

- A rule of the form

$$l_0 \; S_0 \; u_0 \leftarrow l_1 \; S_1 \; u_1, \ldots, l_n \; S_n \; u_n$$

where for $0 \leq i \leq n$ each $l_i \; S_i \; u_i$

stands for $0 \leq i \leq n$

$$a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n$$

$$\begin{array}{lll} {S_0}^+ & \leftarrow & a \\ & \leftarrow & a, \sim b_0 \\ & \leftarrow & a, c_0 \end{array} \qquad \begin{array}{lll} b_i & \leftarrow & l_i \; S_i \\ c_i & \leftarrow & u_i{+}1 \; S_i \end{array}$$

where $a, b_i, c_i$ are new symbols

# Full-fledged cardinality rules

- A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \dots, l_n\ S_n\ u_n$$

where for $0 \leq i \leq n$ each $l_i\ S_i\ u_i$

stands for $0 \leq i \leq n$

$$a \leftarrow b_1, \dots, b_n, \sim c_1, \dots, \sim c_n$$

$$\begin{array}{lcllcl} {S_0}^+ & \leftarrow & a & & & \\ & \leftarrow & a, \sim b_0 & b_i & \leftarrow & l_i\ S_i \\ & \leftarrow & a, c_0 & c_i & \leftarrow & u_i{+}1\ S_i \end{array}$$

where $a, b_i, c_i$ are new symbols

# Full-fledged cardinality rules

- A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \dots, l_n\ S_n\ u_n$$

where for $0 \leq i \leq n$ each $l_i\ S_i\ u_i$

stands for $0 \leq i \leq n$

$$a \leftarrow b_1, \dots, b_n, \sim c_1, \dots, \sim c_n$$

$$\begin{array}{lcllcl} {S_0}^+ & \leftarrow & a & & & \\ & \leftarrow & a, \sim b_0 & b_i & \leftarrow & l_i\ S_i \\ & \leftarrow & a, c_0 & c_i & \leftarrow & u_i{+}1\ S_i \end{array}$$

where $a, b_i, c_i$ are new symbols

# Full-fledged cardinality rules

- A rule of the form

$$l_0 \; S_0 \; u_0 \leftarrow l_1 \; S_1 \; u_1, \ldots, l_n \; S_n \; u_n$$

where for $0 \leq i \leq n$ each $l_i \; S_i \; u_i$

stands for $0 \leq i \leq n$

$$a \quad \leftarrow \quad b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n$$

$$\begin{array}{lcl} {S_0}^+ & \leftarrow & a \\ & \leftarrow & a, \sim b_0 \\ & \leftarrow & a, c_0 \end{array} \qquad \begin{array}{lcl} b_i & \leftarrow & l_i \; S_i \\ c_i & \leftarrow & u_i{+}1 \; S_i \end{array}$$

where $a, b_i, c_i$ are new symbols

# Full-fledged cardinality rules

- A rule of the form

$$l_0 \; S_0 \; u_0 \leftarrow l_1 \; S_1 \; u_1, \ldots, l_n \; S_n \; u_n$$

where for $0 \leq i \leq n$ each $l_i \; S_i \; u_i$

stands for $0 \leq i \leq n$

$$\begin{array}{rcl} a & \leftarrow & b_1, \ldots, b_n, {\sim}c_1, \ldots, {\sim}c_n \\ {S_0}^+ & \leftarrow & a \\ & \leftarrow & a, {\sim}b_0 \\ & \leftarrow & a, c_0 \end{array} \qquad \begin{array}{rcl} b_i & \leftarrow & l_i \; S_i \\ c_i & \leftarrow & u_i{+}1 \; S_i \end{array}$$

where $a, b_i, c_i$ are new symbols

Potassco

# Full-fledged cardinality rules

- A rule of the form

$$l_0 \; S_0 \; u_0 \leftarrow l_1 \; S_1 \; u_1, \dots, l_n \; S_n \; u_n$$

where for $0 \leq i \leq n$ each $l_i \; S_i \; u_i$

stands for $0 \leq i \leq n$

$$a \leftarrow b_1, \dots, b_n, \sim c_1, \dots, \sim c_n$$

$$\begin{array}{rcl} {S_0}^+ & \leftarrow & a \\ & \leftarrow & a, \sim b_0 \\ & \leftarrow & a, c_0 \end{array} \qquad \begin{array}{rcl} b_i & \leftarrow & l_i \; S_i \\ c_i & \leftarrow & u_i{+}1 \; S_i \end{array}$$

where $a, b_i, c_i$ are new symbols

# Outline

1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Weight rule

- Syntax A weight rule is the form

$$a_0 \leftarrow l \; \{ \; a_1 = w_1, \ldots, a_m = w_m, \sim a_{m+1} = w_{m+1}, \ldots, \sim a_n = w_n \; \}$$

  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, $\ell_i = w_i$, 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 rule

- Syntax A weight rule is the form

$$a_0 \leftarrow l \{ a_1 = w_1, \ldots, a_m = w_m, \sim a_{m+1} = w_{m+1}, \ldots, \sim a_n = w_n \}$$

  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, $\ell_i = w_i$, 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

$$l \; \{ \; a_1 = w_1, \ldots, a_m = w_m, {\sim}a_{m+1} = w_{m+1}, \ldots, {\sim}a_n = w_n \; \} \; u$$

  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 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20`

# Weight constraints

- Syntax A weight constraint is of the form

$$l \; \{ \; a_1 = w_1, \dots, a_m = w_m, \sim a_{m+1} = w_{m+1}, \dots, \sim a_n = w_n \; \} \; u$$

  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 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20`

# Weight constraints

- Syntax A weight constraint is of the form

$$l \; \{ \; a_1 = w_1, \ldots, a_m = w_m, {\sim}a_{m+1} = w_{m+1}, \ldots, {\sim}a_n = w_n \; \} \; u$$

  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 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20`

Potassco

## Weight constraints

- Syntax A weight constraint is of the form

$$l \; \{ \; a_1 = w_1, \dots, a_m = w_m, {\sim}a_{m+1} = w_{m+1}, \dots, {\sim}a_n = w_n \; \} \; u$$

  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 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20`

# Outline


1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard


Potassco

# Outline

1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Conditional literals (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

$$\ell : \ell_1 : \cdots : \ell_n$$

  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 $\{\ell \mid \ell_1, \dots, \ell_n\}$

- Note The expansion of conditional literals is context dependent

- Example Given '`p(1). p(2). p(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)}.
```

Potassco

# Conditional literals (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

  $$\ell : \ell_1 : \dots : \ell_n$$

  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 $\{\ell \mid \ell_1, \dots, \ell_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given '`p(1). p(2). p(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)}.
  ```

Potassco

# Conditional literals (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

$$\ell : \ell_1 : \cdots : \ell_n$$

  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). p(2). p(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)}.
```

Potassco

# Conditional literals (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

  $$\ell : \ell_1 : \cdots : \ell_n$$

  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 $\{\ell \mid \ell_1, \dots, \ell_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given '`p(1). p(2). p(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 (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

  $$\ell : \ell_1 : \cdots : \ell_n$$

  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). p(2). p(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)}.
```

Potassco

# Conditional literals (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

  $$\ell : \ell_1 : \cdots : \ell_n$$

  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). p(2). p(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 (in *lparse* & *gringo* 3)

- Syntax A conditional literal is of the form

  $$\ell : \ell_1 : \cdots : \ell_n$$

  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 $\{\ell \mid \ell_1, \dots, \ell_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given '`p(1). p(2). p(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)}.
```

Potassco

# Outline

1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# Optimization statement

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

$$minimize\{\ \ell_1 = w_1@p_1, \dots, \ell_n = w_n@p_n\ \}.$$

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

Potassco

# Optimization statement

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

$$minimize\{\ \ell_1 = w_1@p_1, \dots, \ell_n = w_n@p_n\ \}.$$

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

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

  $$minimize\{\ \ell_1 = w_1@p_1, \dots, \ell_n = w_n@p_n\ \}.$$

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

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

# Optimization statement

- A maximize statement of the form

$$maximize\{\ \ell_1 = w_1@p_1, \dots, \ell_n = w_n@p_n\ \}$$

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

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

```
#maximize[ hd(1)=250@1, hd(2)=500@1, hd(3)=750@1, hd(4)=1000@1 ].
#minimize[ hd(1)=30@2, hd(2)=40@2, hd(3)=60@2, hd(4)=80@2 ].
```

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

## Optimization statement

- A maximize statement of the form

$$maximize\{\ \ell_1 = w_1@p_1, \dots, \ell_n = w_n@p_n\ \}$$

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

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

```
#maximize[ hd(1)=250@1, hd(2)=500@1, hd(3)=750@1, hd(4)=1000@1 ].
#minimize[ hd(1)=30@2, hd(2)=40@2, hd(3)=60@2, hd(4)=80@2 ].
```

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

# Outline


1 Motivation

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

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

5 ASP language standard

# *smodels* format

- Logic programs in *smodels* format consist of
  - normal rules
  - choice rules
  - cardinality rules
  - weight rules
  - optimization statements
- Such a format is obtained by grounders *lparse* and *gringo*

# Outline


1 Motivation

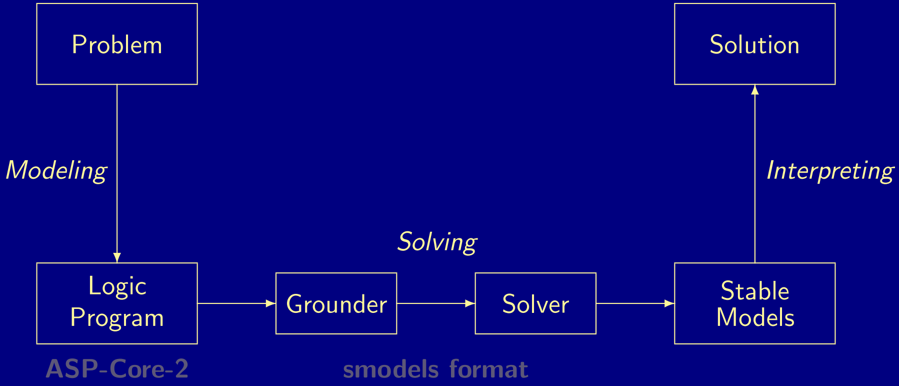
2 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

3 Extended language
- Conditional literal
- Optimization statement

4 smodels format

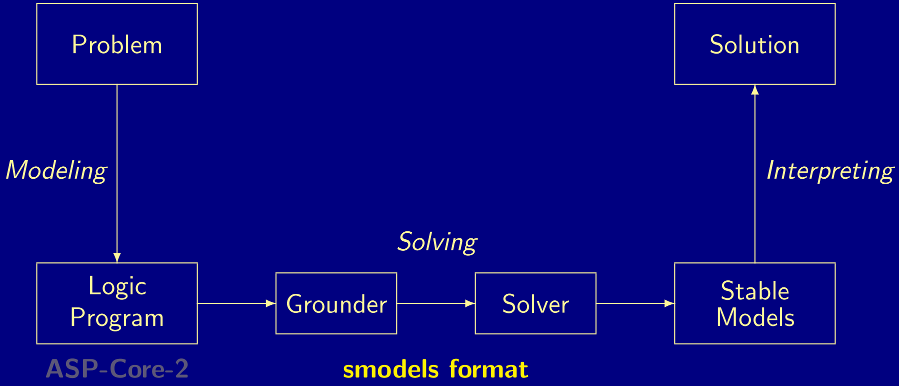
5 ASP language standard

# ASP-Core-2



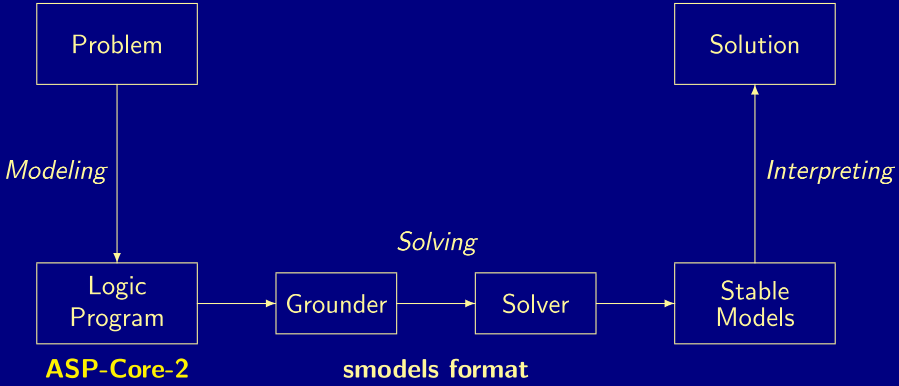

- smodels format is a machine-oriented standard for ground programs
- ASP-Core-2 is a user-oriented standard for (non-ground) programs, extending the input languages of *dlv* and *gringo* series 3

# ASP-Core-2



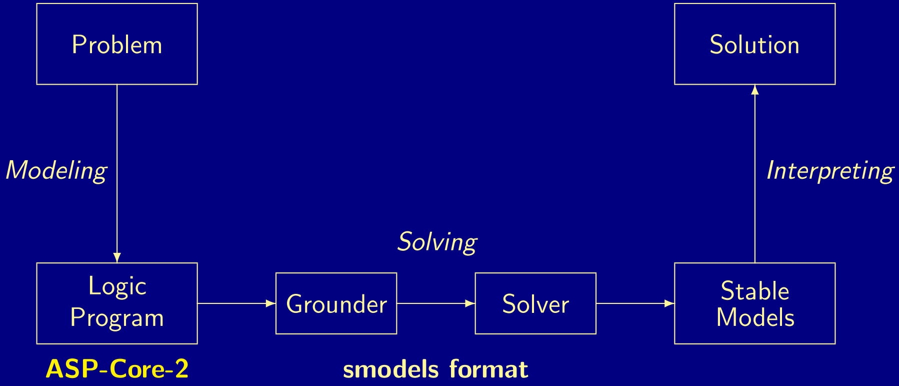

- smodels format is a machine-oriented standard for ground programs
- ASP-Core-2 is a user-oriented standard for (non-ground) programs, extending the input languages of *dlv* and *gringo* series 3

# ASP-Core-2




- smodels format is a machine-oriented standard for ground programs
- ASP-Core-2 is a user-oriented standard for (non-ground) programs, extending the input languages of *dlv* and *gringo* series 3

# ASP-Core-2




- smodels format is a machine-oriented standard for ground programs
- ASP-Core-2 is a user-oriented standard for (non-ground) programs, extending the input languages of *dlv* and *gringo* series 3

# Aggregates

- Syntax ASP-Core-2 aggregates are of the form

$$t_1 \prec_1 \#\mathtt{A}\{t_{1_1},\dots,t_{m_1} : \ell_{1_1},\dots,\ell_{n_1}\} \prec_2 t_2$$

  where
  - $\#\mathtt{A} \in \{\#\mathtt{count}, \#\mathtt{sum}, \#\mathtt{max}, \#\mathtt{min}\}$
  - $\prec_1, \prec_2 \in \{<, \leq, =, \neq, >, \geq\}$
  - $t_{1_1},\dots,t_{m_1}$ and $t_1, t_2$ are terms
  - $\ell_{1_1},\dots,\ell_{n_1}$ are literals
- Example Weight constraint

```
10 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20
```

  is written as an ASP-Core-2 aggregate as

```
10 ≤ #sum{6,db:course(db); 6,ai:course(ai);
             8,project:course(project); 3,xml:course(xml)} ≤ 20
```

# Aggregates

- Syntax ASP-Core-2 aggregates are of the form

$$t_1 \prec_1 \#\mathtt{A}\{t_{1_1},\dots,t_{m_1} : \ell_{1_1},\dots,\ell_{n_1} ;\dots; t_{1_k},\dots,t_{m_k} : \ell_{1_k},\dots,\ell_{n_k}\} \prec_2 t_2$$

where

  - $\#\mathtt{A} \in \{\#\mathtt{count}, \#\mathtt{sum}, \#\mathtt{max}, \#\mathtt{min}\}$
  - $\prec_1, \prec_2 \in \{<, \leq, =, \neq, >, \geq\}$
  - $t_{1_1},\dots,t_{m_1},\dots,t_{1_k},\dots,t_{m_k}$, and $t_1, t_2$ are terms
  - $\ell_{1_1},\dots,\ell_{n_1},\dots,\ell_{1_k},\dots,\ell_{n_k}$ are literals

- Example Weight constraint

```
10 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20
```

is written as an ASP-Core-2 aggregate as

$$10 \leq \#\mathtt{sum}\{\mathtt{6,db:course(db);\ 6,ai:course(ai);}\ \mathtt{8,project:course(project);\ 3,xml:course(xml)}\} \leq 20$$

# Aggregates

- Syntax ASP-Core-2 aggregates are of the form

$$t_1 \prec_1 \#\mathtt{A}\{t_{1_1},\dots,t_{m_1} : \ell_{1_1},\dots,\ell_{n_1} \;;\dots;\; t_{1_k},\dots,t_{m_k} : \ell_{1_k},\dots,\ell_{n_k}\} \prec_2 t_2$$

  where
  - $\#\mathtt{A} \in \{\#\mathtt{count}, \#\mathtt{sum}, \#\mathtt{max}, \#\mathtt{min}\}$
  - $\prec_1, \prec_2 \in \{<, \leq, =, \neq, >, \geq\}$
  - $t_{1_1},\dots,t_{m_1},\dots,t_{1_k},\dots,t_{m_k}$, and $t_1, t_2$ are terms
  - $\ell_{1_1},\dots,\ell_{n_1},\dots,\ell_{1_k},\dots,\ell_{n_k}$ are literals
- Example Weight constraint

```
10 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20
```

  is written as an ASP-Core-2 aggregate as

$$10 \leq \#\mathtt{sum}\{\mathtt{6,db:course(db);\ 6,ai:course(ai);}$$
$$\mathtt{8,project:course(project);\ 3,xml:course(xml)}\} \leq 20$$

Potassco

# Aggregates

- Syntax ASP-Core-2 aggregates are of the form

$$t_1 \prec_1 \#\mathtt{A}\{t_{1_1},\dots,t_{m_1} : \ell_{1_1},\dots,\ell_{n_1} ;\dots;\ t_{1_k},\dots,t_{m_k} : \ell_{1_k},\dots,\ell_{n_k}\} \prec_2 t_2$$

  where
  - $\#\mathtt{A} \in \{\#\mathtt{count}, \#\mathtt{sum}, \#\mathtt{max}, \#\mathtt{min}\}$
  - $\prec_1, \prec_2 \in \{<, \leq, =, \neq, >, \geq\}$
  - $t_{1_1},\dots,t_{m_1},\dots,t_{1_k},\dots,t_{m_k}$, and $t_1, t_2$ are terms
  - $\ell_{1_1},\dots,\ell_{n_1},\dots,\ell_{1_k},\dots,\ell_{n_k}$ are literals
- Example Weight constraint

```
10 [course(db)=6,course(ai)=6,course(project)=8,course(xml)=3] 20
```

  is written as an ASP-Core-2 aggregate as

$$10 \leq \#\mathtt{sum}\{\mathtt{6,db:course(db);\ 6,ai:course(ai);}\ \ \mathtt{8,project:course(project);\ 3,xml:course(xml)}\} \leq 20$$

# Weak constraints

- Syntax A weak constraint is of the form

$$:\sim a_1,\dots,a_m,\sim a_{m+1},\dots,\sim a_n.\,[w@p,t_1,\dots,t_m]$$

  where
  - $a_1,\dots,a_n$ are atoms
  - $t_1,\dots,t_m$, $w$, and $p$ are terms
- $a_1,\dots,a_n$ may contain ASP-Core-2 aggregates
- $w$ and $p$ stand for a weight and priority level ($p=0$ if '@$p$' is omitted)
- Example Minimize statement

```
#minimize[ hd(1)=30@2, hd(2)=40@2, hd(3)=60@2, hd(4)=80@2 ].
```

  can be written in terms of weak constraints as

```
:~ hd(1). [30@2,1]        :~ hd(3). [60@2,3]
:~ hd(2). [40@2,2]        :~ hd(4). [80@2,4]
```

Potassco

# Weak constraints

- Syntax A weak constraint is of the form

$$:\sim a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n. \, [w@p, t_1, \dots, t_m]$$

  where
  - $a_1, \dots, a_n$ are atoms
  - $t_1, \dots, t_m$, $w$, and $p$ are terms
- $a_1, \dots, a_n$ may contain ASP-Core-2 aggregates
- $w$ and $p$ stand for a weight and priority level ($p = 0$ if '@$p$' is omitted)
- Example Minimize statement

```
#minimize[ hd(1)=30@2, hd(2)=40@2, hd(3)=60@2, hd(4)=80@2 ].
```

  can be written in terms of weak constraints as

```
:~ hd(1). [30@2,1]      :~ hd(3). [60@2,3]
:~ hd(2). [40@2,2]      :~ hd(4). [80@2,4]
```

# Weak constraints

- Syntax A weak constraint is of the form

$$:\sim a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n. \, [w@p, t_1, \dots, t_m]$$

  where
  - $a_1, \dots, a_n$ are atoms
  - $t_1, \dots, t_m$, $w$, and $p$ are terms
- $a_1, \dots, a_n$ may contain ASP-Core-2 aggregates
- $w$ and $p$ stand for a weight and priority level ($p = 0$ if '@$p$' is omitted)
- Example Minimize statement

```
#minimize[ hd(1)=30@2, hd(2)=40@2, hd(3)=60@2, hd(4)=80@2 ].
```

  can be written in terms of weak constraints as

```
:~ hd(1). [30@2,1]      :~ hd(3). [60@2,3]
:~ hd(2). [40@2,2]      :~ hd(4). [80@2,4]
```

## Weak constraints

- Syntax A weak constraint is of the form

$$:\sim a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n. \, [w@p, t_1, \dots, t_m]$$

  where
  - $a_1, \dots, a_n$ are atoms
  - $t_1, \dots, t_m$, $w$, and $p$ are terms
- $a_1, \dots, a_n$ may contain ASP-Core-2 aggregates
- $w$ and $p$ stand for a weight and priority level ($p = 0$ if '@$p$' is omitted)
- Example Minimize statement

```
#minimize[ hd(1)=30@2, hd(2)=40@2, hd(3)=60@2, hd(4)=80@2 ].
```

  can be written in terms of weak constraints as

```
:~ hd(1). [30@2,1]      :~ hd(3). [60@2,3]
:~ hd(2). [40@2,2]      :~ hd(4). [80@2,4]
```

# *gringo* 4

- The input language of *gringo* series 4 comprises
  - ASP-Core-2
  - concepts from *gringo* 3 (conditional literals, `#show` directives, ...)
- Example The *gringo* 3 rule

```
r(X):p(X):not q(X) :- r(X):p(X):not q(X), 1 {r(X):p(X):not q(X)}.
```

can be written as follows in the language of *gringo* 4:

```
r(X):p(X),not q(X) :- r(X):p(X),not q(X);
                      1 <= #count{X:r(X),p(X),not q(X)}.
```

Term-based `#show` directives as in

```
#show. #show hello. #show X : p(X).  1{p(earth);p(mars);p(venus)}1.
```

The languages of *gringo* 3 and 4 are not fully compatible

Many example programs given in this tutorial are written for *gringo* 3

Potassco

## *gringo* 4

- The input language of *gringo* series 4 comprises
  - ASP-Core-2
  - concepts from *gringo* 3 (conditional literals, #show directives, ...)
- Example The *gringo* 3 rule

```
r(X):p(X):not q(X) :- r(X):p(X):not q(X), 1 {r(X):p(X):not q(X)}.
```

can be written as follows in the language of *gringo* 4:

```
r(X):p(X),not q(X) :- r(X):p(X),not q(X);
                      1 <= #count{X:r(X),p(X),not q(X)}.
```

Term-based #show directives as in

```
#show. #show hello. #show X : p(X).  1{p(earth);p(mars);p(venus)}1.
```

The languages of *gringo* 3 and 4 are not fully compatible

Many example programs given in this tutorial are written for *gringo* 3

## *gringo* 4

- The input language of *gringo* series 4 comprises
  - ASP-Core-2
  - concepts from *gringo* 3 (conditional literals, `#show` directives, ...)
- Example The *gringo* 3 rule

```
r(X):p(X):not q(X) :- r(X):p(X):not q(X), 1 {r(X):p(X):not q(X)}.
```

can be written as follows in the language of *gringo* 4:

```
r(X):p(X),not q(X) :- r(X):p(X),not q(X);
                      1 <= #count{X:r(X),p(X),not q(X)}.
```

- New Term-based `#show` directives as in

```
#show. #show hello. #show X : p(X).  1{p(earth);p(mars);p(venus)}1.
```

- Attention The languages of *gringo* 3 and 4 are not fully compatible
  - Many example programs given in this tutorial are written for *gringo* 3

Potassco

## *gringo* 4

- The input language of *gringo* series 4 comprises
  - ASP-Core-2
  - concepts from *gringo* 3 (conditional literals, `#show` directives, ...)
- Example The *gringo* 3 rule

```
r(X):p(X):not q(X) :- r(X):p(X):not q(X), 1 {r(X):p(X):not q(X)}.
```

  can be written as follows in the language of *gringo* 4:

```
r(X):p(X),not q(X) :- r(X):p(X),not q(X);
                      1 <= #count{X:r(X),p(X),not q(X)}.
```

- New Term-based `#show` directives as in

```
#show. #show hello. #show X : p(X).  1{p(earth);p(mars);p(venus)}1.
```

- Attention The languages of *gringo* 3 and 4 are not fully compatible
  - Many example programs given in this tutorial are written for *gringo* 3

Potassco

## *gringo* 4

- The input language of *gringo* series 4 comprises
  - ASP-Core-2
  - concepts from *gringo* 3 (conditional literals, `#show` directives, ...)
- Example The *gringo* 3 rule

```
r(X):p(X):not q(X) :- r(X):p(X):not q(X), 1 {r(X):p(X):not q(X)}.
```

  can be written as follows in the language of *gringo* 4:

```
r(X):p(X),not q(X) :- r(X):p(X),not q(X);
                      1 <= #count{X:r(X),p(X),not q(X)}.
```

- New Term-based `#show` directives as in
  `#show. #show hello. #show X : p(X).  1{p(earth);p(mars);p(venus)}1.`
- Attention The languages of *gringo* 3 and 4 are not fully compatible
  - Many example programs given in this tutorial are written for *gringo* 3

Potassco

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub.
The `nomore++` approach to answer set solving.
In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.

[2] C. Anger, K. Konczak, T. Linke, and T. Schaub.
A glimpse of answer set programming.
*Künstliche Intelligenz*, 19(1):12–17, 2005.

[3] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[4] C. Baral.
*Knowledge Representation, Reasoning and Declarative Problem Solving*.
Cambridge University Press, 2003.

[5] C. Baral, G. Brewka, and J. Schlipf, editors.
*Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[6] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
*Journal of Logic Programming*, 12:1–80, 1994.

[7] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.
In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[8] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.

In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[9] A. Biere.
PicoSAT essentials.
*Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
*Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*.
IOS Press, 2009.

[11] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
*Communications of the ACM*, 54(12):92–103, 2011.

[12] K. Clark.
Negation as failure.

In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[13] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
*Handbook of Tableau Methods*.
Kluwer Academic Publishers, 1999.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[15] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
*Communications of the ACM*, 5:394–397, 1962.

[16] M. Davis and H. Putnam.
A computing procedure for quantification theory.
*Journal of the ACM*, 7:201–215, 1960.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.
Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.
In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability*

*Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
*Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.
*Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[23] P. Ferraris.

Answer sets for propositional theories.
In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
*Journal of Logic Programming*, 2(4):295–312, 1985.

[26] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[28] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [44], pages 250–264.

[29] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
*Answer Set Solving in Practice*.
Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[30] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.

clasp: A conflict-driven answer set solver.
In Baral et al. [5], pages 260–265.

[31] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [5], pages 136–148.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [68], pages 386–392.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[34] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.

In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[35] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[36] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [44], pages 235–249.

[37] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.

In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[38] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[39] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[40] M. Gelfond and N. Leone.

Logic programming and knowledge representation — the A-Prolog perspective.
*Artificial Intelligence*, 138(1-2):3–38, 2002.

[41] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.
In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[42] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[43] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.
*Journal of Automated Reasoning*, 36(4):345–377, 2006.

[44] P. Hill and D. Warren, editors.
*Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[45] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [68], pages 2318–2323.

[46] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
*Theory and Practice of Logic Programming*, 6(1-2):61–106, 2006.

[47] J. Lee.
A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[48] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
*ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[49] V. Lifschitz.
Answer set programming and plan generation.
*Artificial Intelligence*, 138(1-2):39–54, 2002.

[50] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[51] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?
*ACM Transactions on Computational Logic*, 7(2):261–268, 2006.

[52] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
*Artificial Intelligence*, 157(1-2):115–137, 2004.

[53] V. Marek and M. Truszczyński.
*Nonmonotonic logic: context-dependent reasoning*.
Artifical Intelligence. Springer-Verlag, 1993.

[54] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[55] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [10], chapter 4, pages 131–153.

[56] J. Marques-Silva and K. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
*IEEE Transactions on Computers*, 48(5):506–521, 1999.

[57] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.

In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[58] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
*Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.

[59] D. Mitchell.
A SAT solver primer.
*Bulletin of the European Association for Theoretical Computer Science*, 85:112–133, 2005.

[60] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[61] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
*Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[62] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
*Journal of the ACM*, 53(6):937–977, 2006.

[63] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[64] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.

Master's thesis, Simon Fraser University, 2004.

[65] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
*Artificial Intelligence*, 138(1-2):181–234, 2002.

[66] T. Syrjänen.
Lparse 1.0 user's manual.

[67] A. Van Gelder, K. Ross, and J. Schlipf.
The well-founded semantics for general logic programs.
*Journal of the ACM*, 38(3):620–650, 1991.

[68] M. Veloso, editor.
*Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*. AAAI/MIT Press, 2007.

[69] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.
In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.