Answer Set Solving in Practice

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

Potassco

Language: Overview

1. Motivation
2. Core language
3. Extended language
4. Intermediate formats

Potassco

Outline

1 Motivation
2 Core language
3 Extended language
4 Intermediate formats

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

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

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

Potassco

Outline

1 Motivation
2 Core language
3 Extended language
4 Intermediate formats

Potassco

Outline

1 Motivation

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

3 Extended language

4 Intermediate formats

Potassco

Integrity constraint

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

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

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

- Example

```
:- edge(3,7), color(3,red), color(7,red).
```

- Example programs

$$\{\, a \leftarrow {\sim}b,\ b \leftarrow {\sim}a \,\}$$
$$\{\, a \leftarrow {\sim}b,\ b \leftarrow {\sim}a \,\} \cup \{\, \leftarrow a \,\}$$
$$\{\, a \leftarrow {\sim}b,\ b \leftarrow {\sim}a \,\} \cup \{\, \leftarrow {\sim}a \,\}$$

Potassco

Integrity constraint

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

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

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

- Example

```
:- edge(3,7), color(3,red), color(7,red).
```

- Example programs

$$\{\ a \leftarrow {\sim}b,\ b \leftarrow {\sim}a\ \}$$
$$\{\ a \leftarrow {\sim}b,\ b \leftarrow {\sim}a\ \} \cup \{\ \leftarrow a\ \}$$
$$\{\ a \leftarrow {\sim}b,\ b \leftarrow {\sim}a\ \} \cup \{\ \leftarrow {\sim}a\ \}$$

Potassco

Integrity constraint

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

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

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

- Example

```
:- edge(3,7), color(3,red), color(7,red).
```

- Example programs

$$\{\ a \leftarrow {\sim}b,\ \ b \leftarrow {\sim}a\ \}$$
$$\{\ a \leftarrow {\sim}b,\ \ b \leftarrow {\sim}a\ \} \cup \{\ \leftarrow a\ \}$$
$$\{\ a \leftarrow {\sim}b,\ \ b \leftarrow {\sim}a\ \} \cup \{\ \leftarrow {\sim}a\ \}$$

Potassco

Embedding in normal rules

- An integrity constraint of form

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

can be translated into the normal rule

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

where x is a new symbol

Potassco

Embedding in normal rules

- An integrity constraint of form

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

  can be translated into the normal rule

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

  where x is a new symbol

Outline

1 Motivation

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

3 Extended language

4 Intermediate formats

Potassco

Choice rule

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

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

  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 $\{a_1, \ldots, a_m\}$ can be included in the stable model
- Example

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

- Example program

$$\{\ \{a\} \leftarrow b,\ \ b \leftarrow\ \}$$

Potassco

Choice rule

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

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

  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 $\{a_1, \ldots, a_m\}$ can be included in the stable model
- Example

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

- Example program

$$\{\ \{a\} \leftarrow b,\ \ b \leftarrow\ \}$$

Potassco

Choice rule

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

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

  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 $\{a_1, \ldots, a_m\}$ can be included in the stable model
- Example

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

- Example program

$$\{\ \{a\} \leftarrow b,\ b \leftarrow\ \}$$

Potassco

Choice rule

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

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

  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 $\{a_1, \ldots, a_m\}$ can be included in the stable model
- Example

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

- Example program

$$\{\ \{a\} \leftarrow b,\ \ b \leftarrow\ \}$$

Potassco

Embedding in normal rules

- A choice rule of form

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

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

$$\begin{array}{rcl} b & \leftarrow & a_{m+1}, \ldots, a_n, {\sim}a_{n+1}, \ldots, {\sim}a_o \\ a_1 & \leftarrow & b, {\sim}a'_1 \quad \ldots \quad a_m \leftarrow b, {\sim}a'_m \\ a'_1 & \leftarrow & {\sim}a_1 \quad \ldots \quad a'_m \leftarrow {\sim}a_m \end{array}$$

by introducing new atoms $b, a'_1, \ldots, a'_m$

Embedding in normal rules

- A choice rule of form

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

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

$$\begin{array}{rcl} b & \leftarrow & a_{m+1}, \ldots, a_n, {\sim}a_{n+1}, \ldots, {\sim}a_o \\ a_1 & \leftarrow & b, {\sim}a'_1 \quad \ldots \quad a_m \leftarrow b, {\sim}a'_m \\ a'_1 & \leftarrow & {\sim}a_1 \quad \ldots \quad a'_m \leftarrow {\sim}a_m \end{array}$$

by introducing new atoms $b, a'_1, \ldots, a'_m$

Embedding in normal rules

- A choice rule of form

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

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

$$\begin{array}{rcl} b & \leftarrow & a_{m+1}, \ldots, a_n, {\sim}a_{n+1}, \ldots, {\sim}a_o \\ a_1 & \leftarrow & b, {\sim}a'_1 \quad \ldots \quad a_m \leftarrow b, {\sim}a'_m \\ a'_1 & \leftarrow & {\sim}a_1 \quad \ldots \quad a'_m \leftarrow {\sim}a_m \end{array}$$

by introducing new atoms $b, a'_1, \ldots, a'_m$

Outline

1 Motivation

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

3 Extended language

4 Intermediate formats

Potassco

Cardinality rule

- Idea Control (lower) cardinality of subsets of literals
- 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 (acting as a lower bound on the body)

- Informal meaning The head atom belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example

```
pass(c42) :- 2 { pass(a1); pass(a2); pass(a3) }.
```

- Example program

$$\{ \ a \leftarrow 1 \ \{b, c\}, \ \ b \leftarrow \ \}$$

Potassco

Cardinality rule

- Idea Control (lower) cardinality of subsets of literals
- 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 (acting as a lower bound on the body)

- Informal meaning The head atom belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example

```
pass(c42) :- 2 { pass(a1); pass(a2); pass(a3) }.
```

- Example program

$$\{ \ a \leftarrow 1 \ \{b, c\}, \ \ b \leftarrow \ \}$$

Potassco

Cardinality rule

- Idea Control (lower) cardinality of subsets of literals
- 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 (acting as a lower bound on the body)

- Informal meaning The head atom belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example

```
pass(c42) :- 2 { pass(a1); pass(a2); pass(a3) }.
```

- Example program

$$\{ \ a \leftarrow 1 \ \{b, c\}, \ b \leftarrow \ \}$$

Potassco

Cardinality rule

- Idea Control (lower) cardinality of subsets of literals
- 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 (acting as a lower bound on the body)

- Informal meaning The head atom belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example

```
pass(c42) :- 2 { pass(a1); pass(a2); pass(a3) }.
```

- Example program

$$\{ \ a \leftarrow 1 \ \{b, c\}, \ \ b \leftarrow \ \}$$

Potassco

Embedding in normal rules

- A cardinality rule of form

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

  is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and
- The 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

Potassco

Embedding in normal rules

- A cardinality rule of form

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

  is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and
- The atom $ctr(i, j)$ represents the fact that at least j of the literals having an equal or greater index than i, are in a stable model

Embedding in normal rules

- A cardinality rule of form

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

  is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and
- The atom $ctr(i, j)$ represents the fact that at least j of the literals having an equal or greater index than i, are in a stable model

Embedding in normal rules

- A cardinality rule of form

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

is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and for $0 \leq k \leq l$ the rules

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

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

Embedding in normal rules

- A cardinality rule of form

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

is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and for $0 \leq k \leq l$ the rules

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

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

Embedding in normal rules

- A cardinality rule of form

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

is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and for $0 \leq k \leq l$ the rules

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

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

Embedding in normal rules

- A cardinality rule of form

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

is translated into the normal rule $a_0 \leftarrow ctr(1, l)$ and for $0 \leq k \leq l$ the rules

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

- The 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

An example

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

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

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

An example

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

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

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

. . . and vice versa

- A normal rule

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

can be represented by the cardinality rule

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

Cardinality rules with upper bounds

- A rule of the form

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

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

stands for

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

where b and c are new symbols

- Note The expression in the body of the cardinality rule (1) is referred to as a cardinality constraint with lower and upper bound l and u

Cardinality rules with upper bounds

- A rule of the form

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

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

stands for

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

where b and c are new symbols

- Note The expression in the body of the cardinality rule (1) is referred to as a cardinality constraint with lower and upper bound l and u

Potassco

Cardinality rules with upper bounds

- A rule of the form

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

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

stands for

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

where b and c are new symbols

- Note The expression in the body of the cardinality rule (1) is referred to as a cardinality constraint with lower and upper bound l and 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

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

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

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

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

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

Potassco

Cardinality constraints as heads

- A rule of the form

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

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

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

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

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

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

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

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

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

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

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

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 each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$

stands for

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

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

where a, b_i, c_i are new symbols (and $\cdot^+$ is defined as on Slide **??**)

Potassco

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 each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

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

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

where a, b_i, c_i are new symbols (and $\cdot^+$ is defined as on Slide **??**)

Potassco

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 each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

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

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

where a, b_i, c_i are new symbols (and $\cdot^+$ is defined as on Slide **??**)

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 each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

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

where a, b_i, c_i are new symbols (and $\cdot^+$ is defined as on Slide **??**)

Potassco

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 each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

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

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

where a, b_i, c_i are new symbols (and $\cdot^+$ is defined as on Slide **??**)

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 each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

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

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

where a, b_i, c_i are new symbols (and $\cdot^+$ is defined as on Slide **??**)

Outline

1 Motivation

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

3 Extended language

4 Intermediate formats

Potassco

Weight rule

- Idea Bound (lower) sum of subsets of literal weights
- Syntax A weighted literal $w : k$ associates the weight w with literal k
- Syntax A weight rule is the form

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

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

- Informal meaning The head atom belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least l
- Note A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$

Potassco

Weight rule

- Idea Bound (lower) sum of subsets of literal weights
- Syntax A weighted literal $w : k$ associates the weight w with literal k
- Syntax A weight rule is the form

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

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

- Informal meaning The head atom belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least l
- Note A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$

Potassco

Weight rule

- Idea Bound (lower) sum of subsets of literal weights
- Syntax A weighted literal $w : k$ associates the weight w with literal k
- Syntax A weight rule is the form

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

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

- Informal meaning The head atom belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least l
- Note A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$

Potassco

Weight constraints

- Syntax A weight constraint is of the form

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

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

- Meaning A weight constraint is satisfied by a stable model X, if

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

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

- Example

```
5 { 4:course(db); 6:course(ai); 3:course(xml) } 10
```

Potassco

Weight constraints

- Syntax A weight constraint is of the form

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

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

- Meaning A weight constraint is satisfied by a stable model X, if

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

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

- Example

```
5 { 4:course(db); 6:course(ai); 3:course(xml) } 10
```

Potassco

Weight constraints

- Syntax A weight constraint is of the form

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

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

- Meaning A weight constraint is satisfied by a stable model X, if

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

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

- Example

```
5 { 4:course(db); 6:course(ai); 3:course(xml) } 10
```

Potassco

Weight constraints

- Syntax A weight constraint is of the form

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

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

- Meaning A weight constraint is satisfied by a stable model X, if

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

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

- Example

```
5 { 4:course(db); 6:course(ai); 3:course(xml) } 10
```

Potassco

Outline

1 Motivation
2 Core language
3 Extended language
4 Intermediate formats

Potassco

Outline

1 Motivation

2 Core language

3 Extended language
- Conditional literal
- Optimization statement

4 Intermediate formats

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  ```
  r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
  ```

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  ```
  r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
  ```

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  `r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.`

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  ```
  r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
  ```

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  ```
  r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
  ```

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  ```
  r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
  ```

Potassco

Conditional literals

- Syntax A conditional literal is of the form

  $$l : l_1, \ldots, l_n$$

  where l and l_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 $\{l \mid l_1, \ldots, l_n\}$
- Note The expansion of conditional literals is context dependent
- Example Given ‘`p(1..3). q(2).`’

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

  is instantiated to

  ```
  r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
  ```

Potassco

Outline

1 Motivation

2 Core language

3 Extended language
- Conditional literal
- Optimization statement

4 Intermediate formats

Potassco

Optimization statement

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

$$\mathit{minimize}\ \{\ w_1@p_1 : l_{1_1}, \ldots, l_{m_1}; \ldots; w_n@p_n : l_{1_n}, \ldots, l_{m_n}\ \}.$$

where each l_{j_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 (multiple) cost functions subject to minimization and/or maximization
- Syntax A minimize statement is of the form

$$\mathit{minimize}\ \{\ w_1@p_1 : l_{1_1}, \ldots, l_{m_1}; \ldots; w_n@p_n : l_{1_n}, \ldots, l_{m_n}\ \}.$$

  where each l_{j_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 (multiple) cost functions subject to minimization and/or maximization
- Syntax A minimize statement is of the form

$$\mathit{minimize}\ \{\ w_1@p_1 : l_{1_1}, \ldots, l_{m_1}; \ldots; w_n@p_n : l_{1_n}, \ldots, l_{m_n}\ \}.$$

  where each l_{j_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

- A maximize statement of the form

  $$\mathit{maximize}\ \{\ w_1@p_1 : l_1, \ldots, w_n@p_n : l_n\ \}$$

  stands for $\mathit{minimize}\ \{\ -w_1@p_1 : l_1, \ldots, -w_n@p_n : l_n\ \}$

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

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

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

Potassco

Optimization statement

- A maximize statement of the form

  $$\mathit{maximize}\ \{\ w_1@p_1 : l_1, \ldots, w_n@p_n : l_n\ \}$$

  stands for $\mathit{minimize}\ \{\ -w_1@p_1 : l_1, \ldots, -w_n@p_n : l_n\ \}$

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

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

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

Optimization statement

- A maximize statement of the form

  $$\mathit{maximize}\ \{\ w_1@p_1 : l_1, \ldots, w_n@p_n : l_n\ \}$$

  stands for $\mathit{minimize}\ \{\ -w_1@p_1 : l_1, \ldots, -w_n@p_n : l_n\ \}$

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

```
#maximize { P@1:hd(I,P,C) }.
#minimize { C@2:hd(I,P,C) }.
```

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

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\quad :\sim l_1, \ldots, l_n \; [w@l]$

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

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

Potassco

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\quad :\sim l_1, \ldots, l_n \ [w@l]$

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

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

Potassco

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\quad :\sim l_1, \ldots, l_n \ [w@l]$

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

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

Potassco

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\quad :\sim l_1, \ldots, l_n\ [w@l]$

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

```
:~ hd(I,P,C). [C@2]
```

Potassco

Outline

1 Motivation
2 Core language
3 Extended language
4 Intermediate formats

Potassco

Outline

1 Motivation
2 Core language
3 Extended language
4 Intermediate formats
- smodels format
- aspif format

Potassco

smodels format

- The *smodels* format consists of
  - normal rules
  - choice rules
  - cardinality rules
  - weight rules
  - minimization statements
- Block-oriented format
- Note Minimization statements are not part of the logic program

smodels format

- The *smodels* format consists of
  - normal rules
  - choice rules
  - cardinality rules
  - weight rules
  - minimization statements
- Block-oriented format
- Note Minimization statements are not part of the logic program

smodels format in detail

Type/Format
Normal rule Slide ?? $1 \sqcup \iota(a_0) \sqcup n \sqcup n - m \sqcup \iota(a_{m+1}) \sqcup \ldots \sqcup \iota(a_n) \sqcup \iota(a_1) \sqcup \ldots \sqcup \iota(a_m)$
Cardinality rule Slide 23 $2 \sqcup \iota(a_0) \sqcup n \sqcup n - m \sqcup l \sqcup \iota(a_{m+1}) \sqcup \ldots \sqcup \iota(a_n) \sqcup \iota(a_1) \sqcup \ldots \sqcup \iota(a_m)$
Choice rule Slide 15 $3 \sqcup m \sqcup \iota(a_1) \sqcup \ldots \sqcup \iota(a_m) \sqcup o - m \sqcup o - n \sqcup \iota(a_{n+1}) \sqcup \ldots \sqcup \iota(a_o) \sqcup \iota(a_{m+1}) \sqcup \ldots \sqcup \iota(a_n)$
Weight rule Slide 53 $5 \sqcup \iota(a_0) \sqcup l \sqcup n \sqcup n - m \sqcup \iota(a_{m+1}) \sqcup \ldots \sqcup \iota(a_n) \sqcup \iota(a_1) \sqcup \ldots \sqcup \iota(a_m) \sqcup w_{m+1} \sqcup \ldots \sqcup w_n \sqcup w_1 \sqcup \ldots \sqcup w_m$
Minimize statement[2] Slide 70 $6 \sqcup 0 \sqcup n \sqcup n - m \sqcup \iota(a_{m+1}) \sqcup \ldots \sqcup \iota(a_n) \sqcup \iota(a_1) \sqcup \ldots \sqcup \iota(a_m) \sqcup w_{m+1} \sqcup \ldots \sqcup w_n \sqcup w_1 \sqcup \ldots \sqcup w_m$
Disjunctive rule Slide ?? $8 \sqcup m \sqcup \iota(a_1) \sqcup \ldots \sqcup \iota(a_m) \sqcup o - m \sqcup o - n \sqcup \iota(a_{n+1}) \sqcup \ldots \sqcup \iota(a_o) \sqcup \iota(a_{m+1}) \sqcup \ldots \sqcup \iota(a_n)$

- The function ι represents a mapping of atoms to numbers

Potassco

Outline

1 Motivation
2 Core language
3 Extended language
4 Intermediate formats
- smodels format
- aspif format

Potassco

aspif format

- The *aspif* format consists of
  - rule statements
  - minimize statements
  - projection statements
  - output statements
  - external statements
  - assumption statements
  - heuristic statements
  - edge statements
  - theory terms and atoms
  - comments
- Line-oriented format

Rule statements

Rule statements have the form $1 \sqcup H \sqcup B$

- Head H has form $h \sqcup m \sqcup a_1 \sqcup \ldots \sqcup a_m$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal

  Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules.
- Body B has one of two forms
  - normal bodies have form $0 \sqcup n \sqcup l_1 \sqcup \ldots \sqcup l_n$
    - $n \geq 0$ is the length of the rule body, and
    - each l_i is a literal.
  - weight bodies have form $1 \sqcup l \sqcup n \sqcup l_1 \sqcup w_1 \sqcup \ldots \sqcup l_n \sqcup w_n$
    - l is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each l_i and w_i are a literal and a positive integer

Potassco

Rule statements

Rule statements have the form $1 \sqcup H \sqcup B$

- Head H has form $h \sqcup m \sqcup a_1 \sqcup \ldots \sqcup a_m$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal

  Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules.
- Body B has one of two forms
  - normal bodies have form $0 \sqcup n \sqcup l_1 \sqcup \ldots \sqcup l_n$
    - $n \geq 0$ is the length of the rule body, and
    - each l_i is a literal.
  - weight bodies have form $1 \sqcup l \sqcup n \sqcup l_1 \sqcup w_1 \sqcup \ldots \sqcup l_n \sqcup w_n$
    - l is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each l_i and w_i are a literal and a positive integer

Potassco

Rule statements

Rule statements have the form $1 \sqcup H \sqcup B$

- Head H has form $h \sqcup m \sqcup a_1 \sqcup \ldots \sqcup a_m$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal

  Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules.
- Body B has one of two forms
  - normal bodies have form $0 \sqcup n \sqcup l_1 \sqcup \ldots \sqcup l_n$
    - $n \geq 0$ is the length of the rule body, and
    - each l_i is a literal.
  - weight bodies have form $1 \sqcup l \sqcup n \sqcup l_1 \sqcup w_1 \sqcup \ldots \sqcup l_n \sqcup w_n$
    - l is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each l_i and w_i are a literal and a positive integer

Potassco

Example

```
{a}.
b :- a.
c :- not a.
```

```
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
```

Potassco

Example

```
{a}.
b :- a.
c :- not a.
```

```
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
```

Potassco

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

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

[3] 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.

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

[5] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving

Potassco

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.

[6] 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.

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

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

Potassco

IOS Press, 2009.

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

[10] G. Brewka, I. Niemelä, and M. Truszczyński.
Answer set optimization.
In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 867–872. Morgan Kaufmann Publishers, 2003.

[11] K. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

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

Potassco

[13] 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.

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

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

[16] E. Di Rosa, E. Giunchiglia, and M. Maratea.
Solving satisfiability problems with preferences.
Constraints, 15(4):485–515, 2010.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.

Potassco

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.

Potassco

[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.

Potassco

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, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub.
Abstract Gringo.
Theory and Practice of Logic Programming, 15(4-5):449–463, 2015.
Available at `http://arxiv.org/abs/1507.06576`.

Potassco

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.
University of Potsdam, second edition edition, 2015.

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

[29] 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.

[30] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.

Potassco

On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [49], pages 250–264.

[31] 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.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [3], pages 260–265.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [3], pages 136–148.

[34] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [74], pages 386–392.

Potassco

[35] 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.

[36] 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.

[37] 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*

Potassco

(CPAIOR'09), volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[38] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [49], pages 235–249.

[39] 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.

[40] 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.

Potassco

[41] 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.

[42] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[43] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[44] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.

Potassco

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.

[45] 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.

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

[47] K. Gödel.
Zum intuitionistischen Aussagenkalkül.
Anzeiger der Akademie der Wissenschaften in Wien, page 65–66, 1932.

[48] A. Heyting.

Potassco

Die formalen Regeln der intuitionistischen Logik.
In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. Deutsche Akademie der Wissenschaften zu Berlin, 1930. Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

[49] 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.

[50] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [74], pages 2318–2323.

[51] 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.

[52] J. Lee.

Potassco

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.

[53] 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.

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

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

[56] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?

Potassco

ACM Transactions on Computational Logic, 7(2):261–268, 2006.

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

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

[59] 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.

[60] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [8], chapter 4, pages 131–153.

[61] J. Marques-Silva and K. Sakallah.

Potassco

GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[62] 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.

[63] 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.

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

Potassco

[65] 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.

[66] 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.

[67] 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.

[68] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.

Potassco

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.

[69] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

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

[71] T. Son and E. Pontelli.
Planning with preferences using logic programming.
Theory and Practice of Logic Programming, 6(5):559–608, 2006.

[72] T. Syrjänen.
Lparse 1.0 user's manual, 2001.

[73] A. Van Gelder, K. Ross, and J. Schlipf.

Potassco

The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

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

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

Potassco