Answer Set Solving in Practice

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

Potassco

Language Extensions: Overview

1 Two kinds of negation
2 Disjunctive logic programs
3 Propositional theories
4 Aggregates
5 Gringo language

Outline

Motivation

- Classical versus default negation
  - Symbol $\neg$ and $\sim$
  - Idea
    - $\neg a \quad \approx \quad \neg a \in X$
    - $\sim a \quad \approx \quad a \notin X$
  - Example
    - $cross \leftarrow \neg train$
    - $cross \leftarrow \sim train$

Motivation

- Classical versus default negation
  - Symbol $\neg$ and $\sim$
  - Idea
    - $\neg a \;\approx\; \neg a \in X$
    - $\sim a \;\approx\; a \notin X$
  - Example
    - $cross \leftarrow \neg train$
    - $cross \leftarrow \sim train$

Potassco

Motivation

- Classical versus default negation
  - Symbol $\neg$ and $\sim$
  - Idea
    - $\neg a \;\approx\; \neg a \in X$
    - $\sim a \;\approx\; a \notin X$
  - Example
    - $cross \leftarrow \neg train$
    - $cross \leftarrow \sim train$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

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

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{\neg}$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

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

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{\neg}$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

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

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{\neg}$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

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

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{\neg}$

An example

- The program

$$P \ = \ \{a \leftarrow {\sim}b,\ b \leftarrow {\sim}a\} \cup \{c \leftarrow b,\ \neg c \leftarrow b\}$$

induces

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

- The stable models of P are given by the ones of $P \cup P^{\neg}$, viz $\{a\}$

An example

- The program

$$P \;=\; \{a \leftarrow {\sim}b,\; b \leftarrow {\sim}a\} \cup \{c \leftarrow b,\; \neg c \leftarrow b\}$$

induces

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

- The stable models of P are given by the ones of $P \cup P^{\neg}$, viz $\{a\}$

An example

- The program

$$P \;=\; \{a \leftarrow {\sim}b,\; b \leftarrow {\sim}a\} \cup \{c \leftarrow b,\; \neg c \leftarrow b\}$$

induces

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

- The stable models of P are given by the ones of $P \cup P^{\neg}$, viz $\{a\}$

Properties

- The only inconsistent stable "model" is $X = \mathcal{A} \cup \overline{\mathcal{A}}$

  Strictly speaking, an inconsistemt set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model

- For a logic program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All stable models of P are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only stable model of P

Potassco

Properties

- The only inconsistent stable "model" is $X = \mathcal{A} \cup \overline{\mathcal{A}}$

  Strictly speaking, an inconsistemt set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model

- For a logic program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All stable models of P are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only stable model of P

Potassco

Properties

- The only inconsistent stable "model" is $X = \mathcal{A} \cup \overline{\mathcal{A}}$

  Strictly speaking, an inconsistemt set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model

- For a logic program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All stable models of P are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only stable model of P

Potassco

Train spotting

- $P_1 = \{cross \leftarrow \sim train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train, \ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train, \ \neg train \leftarrow, \ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train, \ \neg train \leftarrow \sim train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train, \ \neg train \leftarrow \sim train, \ \neg cross \leftarrow\}$
  - no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  - no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train, \ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train, \ \neg train \leftarrow, \ \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train, \ \neg train \leftarrow {\sim}train\}$
  stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train, \ \neg train \leftarrow {\sim}train, \ \neg cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  - no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow \sim train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow \sim train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow \sim train,\ \neg cross \leftarrow\}$
  no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train, \ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train, \ \neg train \leftarrow, \ \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train, \ \neg train \leftarrow {\sim}train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train, \ \neg train \leftarrow {\sim}train, \ \neg cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow \sim train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train, \ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train, \ \neg train \leftarrow, \ \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train, \ \neg train \leftarrow \sim train\}$
  stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train, \ \neg train \leftarrow \sim train, \ \neg cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow {\sim}train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train\}$
  stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow {\sim}train,\ \neg cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow \sim train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$
- $P_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$
- $P_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow \sim train\}$
  - stable model: $\{cross, \neg train\}$
- $P_6 = \{cross \leftarrow \neg train,\ \neg train \leftarrow \sim train,\ \neg cross \leftarrow\}$
  - no stable model

Default negation in rule heads

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

$$\begin{array}{rcl}\widetilde{P} & = & \{r \in P \mid h(r) \neq {\sim}a\} \\ & & \cup\ \{\leftarrow B(r) \cup \{{\sim}\widetilde{a}\} \mid r \in P \text{ and } h(r) = {\sim}a\} \\ & & \cup\ \{\widetilde{a} \leftarrow {\sim}a \mid r \in P \text{ and } h(r) = {\sim}a\}\end{array}$$

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

Potassco

Default negation in rule heads

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

$$\begin{array}{rcl}\widetilde{P} & = & \{r \in P \mid h(r) \neq {\sim}a\} \\ & & \cup\ \{\leftarrow B(r) \cup \{{\sim}\widetilde{a}\} \mid r \in P \text{ and } h(r) = {\sim}a\} \\ & & \cup\ \{\widetilde{a} \leftarrow {\sim}a \mid r \in P \text{ and } h(r) = {\sim}a\}\end{array}$$

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

Default negation in rule heads

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

$$\begin{aligned} \widetilde{P} \quad = \quad & \{r \in P \mid h(r) \neq {\sim}a\} \\ & \cup \{\leftarrow B(r) \cup \{{\sim}\widetilde{a}\} \mid r \in P \text{ and } h(r) = {\sim}a\} \\ & \cup \{\widetilde{a} \leftarrow {\sim}a \mid r \in P \text{ and } h(r) = {\sim}a\} \end{aligned}$$

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

Potassco

Default negation in rule heads

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

$$\begin{aligned}\widetilde{P} \quad = \quad & \{r \in P \mid h(r) \neq {\sim}a\}\\ & \cup \{\leftarrow B(r) \cup \{{\sim}\widetilde{a}\} \mid r \in P \text{ and } h(r) = {\sim}a\}\\ & \cup \{\widetilde{a} \leftarrow {\sim}a \mid r \in P \text{ and } h(r) = {\sim}a\}\end{aligned}$$

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

Outline

Disjunctive logic programs

- A disjunctive rule, r, is of the form

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

  where $0 \leq m \leq n \leq o$ and each a_i is an atom for $0 \leq i \leq o$
- A disjunctive logic program is a finite set of disjunctive rules
- Notation

$$\begin{array}{rcl} H(r) & = & \{a_1, \dots, a_m\} \\ B(r) & = & \{a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o\} \\ B(r)^+ & = & \{a_{m+1}, \dots, a_n\} \\ B(r)^- & = & \{a_{n+1}, \dots, a_o\} \\ A(P) & = & \bigcup_{r \in P} \left(H(r) \cup B(r)^+ \cup B(r)^-\right) \\ B(P) & = & \{B(r) \mid r \in P\} \end{array}$$

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

Disjunctive logic programs

- A disjunctive rule, r, 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 $0 \leq i \leq o$
- A disjunctive logic program is a finite set of disjunctive rules
- Notation

$$\begin{array}{rcl} H(r) & = & \{a_1, \dots, a_m\} \\ B(r) & = & \{a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o\} \\ B(r)^+ & = & \{a_{m+1}, \dots, a_n\} \\ B(r)^- & = & \{a_{n+1}, \dots, a_o\} \\ A(P) & = & \bigcup_{r \in P} \left(H(r) \cup B(r)^+ \cup B(r)^-\right) \\ B(P) & = & \{B(r) \mid r \in P\} \end{array}$$

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

Disjunctive logic programs

- A disjunctive rule, r, 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 $0 \leq i \leq o$
- A disjunctive logic program is a finite set of disjunctive rules
- Notation

$$\begin{array}{rcl} H(r) & = & \{a_1, \ldots, a_m\} \\ B(r) & = & \{a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o\} \\ B(r)^+ & = & \{a_{m+1}, \ldots, a_n\} \\ B(r)^- & = & \{a_{n+1}, \ldots, a_o\} \\ A(P) & = & \bigcup_{r \in P} \left(H(r) \cup B(r)^+ \cup B(r)^-\right) \\ B(P) & = & \{B(r) \mid r \in P\} \end{array}$$

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

Stable models

- Positive programs
  - A set X of atoms is closed under a positive program P iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
    - X corresponds to a model of P (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program P is denoted by $\min_\subseteq(P)$
    - $\min_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of P (ditto)
- Disjunctive programs
  - The reduct, P^X, of a disjunctive program P relative to a set X of atoms is defined by

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

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

Stable models

- Positive programs
  - A set X of atoms is closed under a positive program P iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
    - X corresponds to a model of P (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program P is denoted by $\min_\subseteq(P)$
    - $\min_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of P (ditto)
- Disjunctive programs
  - The reduct, P^X, of a disjunctive program P relative to a set X of atoms is defined by

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

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

Stable models

- Positive programs
  - A set X of atoms is closed under a positive program P iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
    - X corresponds to a model of P (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program P is denoted by $\min_\subseteq(P)$
    - $\min_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of P (ditto)
- Disjunctive programs
  - The reduct, P^X, of a disjunctive program P relative to a set X of atoms is defined by

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

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

Potassco

A "positive" example

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

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under P
- We have $\min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}$

A "positive" example

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

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under P
- We have $\min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}$

A “positive” example

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

- The sets $\{a,b\}$, $\{a,c\}$, and $\{a,b,c\}$ are closed under P
- We have $\min_{\subseteq}(P) = \{\{a,b\},\{a,c\}\}$

Graph coloring (reloaded)

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).


assign(X,r) ; assign(X,b) ; assign(X,g) :- node(X).


:- edge(X,Y), assign(X,C), assign(Y,C).
```

Graph coloring (reloaded)

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

color(r).   color(b).   color(g).

assign(X,C) : color(C) :- node(X).

:- edge(X,Y), assign(X,C), assign(Y,C).
```

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow {\sim}a, {\sim}c\ ,\ a\,;c \leftarrow {\sim}b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow {\sim}a, {\sim}c\ ,\ a\,;c \leftarrow {\sim}b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \sim a, \sim c\ ,\ a\,;c \leftarrow \sim b\}$
  stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow {\sim}a, {\sim}c\ ,\ a\,;c \leftarrow {\sim}b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \sim a, \sim c\ ,\ a\,;c \leftarrow \sim b\}$
  stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \sim a, \sim c\ ,\ a\,;c \leftarrow \sim b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \sim a, \sim c\ ,\ a\,;c \leftarrow \sim b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a \,;\, b \,;\, c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a \;,\; b \leftarrow c \;,\; c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a \,;\, b \leftarrow c \;,\; b \leftarrow \sim a, \sim c \;,\; a \,;\, c \leftarrow \sim b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \sim a, \sim c\ ,\ a\,;c \leftarrow \sim b\}$
  - stable models $\{a\}$ and $\{b\}$

Some properties

- A disjunctive logic program may have zero, one, or multiple stable models
- If X is a stable model of a disjunctive logic program P, then X is a model of P (seen as a formula)
- If X and Y are stable models of a disjunctive logic program P, then $X \not\subset Y$

- If $a \in X$ for some stable model X of a disjunctive logic program P, then there is a rule $r \in P$ such that $B(r)^+ \subseteq X$, $B(r)^- \cap X = \emptyset$, and $H(r) \cap X = \{a\}$

Some properties

- A disjunctive logic program may have zero, one, or multiple stable models
- If X is a stable model of a disjunctive logic program P, then X is a model of P (seen as a formula)
- If X and Y are stable models of a disjunctive logic program P, then $X \not\subset Y$
- If $a \in X$ for some stable model X of a disjunctive logic program P, then there is a rule $r \in P$ such that $B(r)^+ \subseteq X$, $B(r)^- \cap X = \emptyset$, and $H(r) \cap X = \{a\}$

An example with variables

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

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

For every stable model X of P, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$

An example with variables

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

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

For every stable model X of P, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$

An example with variables

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

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

For every stable model X of P, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$

Potassco

An example with variables

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

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

An example with variables

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

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

An example with variables

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

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

Potassco

An example with variables

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

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

An example with variables

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

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

An example with variables

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

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

An example with variables

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

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

An example with variables

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

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

An example with variables

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

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

Potassco

An example with variables

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

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

Default negation in rule heads

- Consider disjunctive rules of the form

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

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

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

$$\begin{aligned}\widetilde{P} \;=\; & \{H(r)^+ \leftarrow B(r) \cup \{\sim\widetilde{a} \mid a \in H(r)^-\} \mid r \in P\} \\ & \cup\ \{\widetilde{a} \leftarrow \sim a \mid r \in P \text{ and } a \in H(r)^-\}\end{aligned}$$

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

Default negation in rule heads

- Consider disjunctive rules of the form

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

  where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $0 \leq i \leq p$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{aligned} \widetilde{P} \;=\; & \{H(r)^+ \leftarrow B(r) \cup \{\sim\widetilde{a} \mid a \in H(r)^-\} \mid r \in P\} \\ & \cup \, \{\widetilde{a} \leftarrow \sim a \mid r \in P \text{ and } a \in H(r)^-\} \end{aligned}$$

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

Default negation in rule heads

- Consider disjunctive rules of the form

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

  where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $0 \leq i \leq p$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{aligned}\widetilde{P} \;=\; & \{H(r)^{+} \leftarrow B(r) \cup \{{\sim}\widetilde{a} \mid a \in H(r)^{-}\} \mid r \in P\} \\ & \cup\ \{\widetilde{a} \leftarrow {\sim}a \mid r \in P \text{ and } a \in H(r)^{-}\}\end{aligned}$$

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

An example

- The program

$$P = \{a\ ;\ {\sim}a \leftarrow\}$$

yields

$$\widetilde{P} = \{a \leftarrow {\sim}\widetilde{a}\} \cup \{\widetilde{a} \leftarrow {\sim}a\}$$

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

An example

- The program

$$P = \{a\,;\,\sim a \leftarrow\}$$

  yields

$$\widetilde{P} = \{a \leftarrow \sim\widetilde{a}\} \cup \{\widetilde{a} \leftarrow \sim a\}$$

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

An example

- The program

$$P = \{a \,;\, {\sim}a \leftarrow\}$$

yields

$$\widetilde{P} = \{a \leftarrow {\sim}\widetilde{a}\} \cup \{\widetilde{a} \leftarrow {\sim}a\}$$

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

An example

- The program

$$P = \{a \,;\, {\sim}a \leftarrow\}$$

yields

$$\widetilde{P} = \{a \leftarrow {\sim}\widetilde{a}\} \cup \{\widetilde{a} \leftarrow {\sim}a\}$$

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

Outline

Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$

  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)
- Notation

$$\begin{array}{rcl} \top & = & (\bot \rightarrow \bot) \\ {\sim}\phi & = & (\phi \rightarrow \bot) \end{array}$$

- A propositional theory is a finite set of formulas

Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$

  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)
- Notation

$$\begin{array}{rcl} \top & = & (\bot \rightarrow \bot) \\ \sim\phi & = & (\phi \rightarrow \bot) \end{array}$$

- A propositional theory is a finite set of formulas

Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$

  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)
- Notation

$$\begin{array}{rcl} \top & = & (\bot \rightarrow \bot) \\ \sim\!\phi & = & (\phi \rightarrow \bot) \end{array}$$

- A propositional theory is a finite set of formulas

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:

$$
\begin{array}{ll}
\phi^X = \bot & \text{if } X \not\models \phi \\
\phi^X = \phi & \text{if } \phi \in X \\
\phi^X = (\psi^X \circ \varphi^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \varphi) \text{ for } \circ \in \{\land, \lor, \rightarrow\}
\end{array}
$$

If $\phi = {\sim}\psi = (\psi \rightarrow \bot)$,
then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:

$\phi^X = \bot$	if $X \not\models \phi$
$\phi^X = \phi$	if $\phi \in X$
$\phi^X = (\psi^X \circ \varphi^X)$	if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$

  If $\phi = \ {\sim}\psi = (\psi \rightarrow \bot)$,
  then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

  The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Potassco

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \sim\psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \sim\psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = {\sim}\psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \sim\psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \sim\psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$

- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$

- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$

- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$

- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$

- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$
    For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$

- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$
    For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✗
    For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$

- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$
    For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$

- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Potassco

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$
    For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
    For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Potassco

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\perp \vee (\perp \rightarrow \perp)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\perp\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\perp \rightarrow \perp)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\perp \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$ ✔

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee ({\sim}p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$ ✔

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\sim\phi] = \sim\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms, X is a stable model of P iff X is a stable model of $\tau[P]$

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\sim\phi] = \ \sim\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms, X is a stable model of P iff X is a stable model of $\tau[P]$

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\sim\phi] = \ \sim\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms, X is a stable model of P iff X is a stable model of $\tau[P]$

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\sim\phi] = \sim\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms,
  X is a stable model of P iff X is a stable model of $\tau[P]$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$ corresponds to $\tau[P] = \{{\sim}q \rightarrow p,\ {\sim}p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\,;q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow {\sim}{\sim}p\}$ corresponds to $\tau[P] = \{{\sim}{\sim}p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$ corresponds to $\tau[P] = \{{\sim}q \rightarrow p,\ {\sim}p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\,;q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow {\sim}{\sim}p\}$ corresponds to $\tau[P] = \{{\sim}{\sim}p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$ corresponds to $\tau[P] = \{{\sim}q \rightarrow p,\ {\sim}p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p \,;\, q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow {\sim}{\sim}p\}$ corresponds to $\tau[P] = \{{\sim}{\sim}p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$ corresponds to $\tau[P] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\, ;q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \sim\sim p\}$ corresponds to $\tau[P] = \{\sim\sim p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$ corresponds to $\tau[P] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p \,;\, q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \sim\sim p\}$ corresponds to $\tau[P] = \{\sim\sim p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$ corresponds to $\tau[P] = \{{\sim}q \rightarrow p,\ {\sim}p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p \,;\, q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow {\sim}{\sim}p\}$ corresponds to $\tau[P] = \{{\sim}{\sim}p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Outline

Motivation

- Aggregates provide a general way to obtain a single value from a collection of input values
- Popular aggregate (functions)
  - average
  - count
  - maximum
  - minimum
  - sum
- Cardinality and weight constraints rely on count and sum aggregates

Potassco

Syntax

- An aggregate has the form:

$$\alpha\ \{w_1 : a_1, \dots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \dots, w_n : \sim a_n\} \prec k$$

  where for $1 \leq i \leq n$
  - α stands for a function mapping multisets over $\mathbb{Z}$ to $\mathbb{Z} \cup \{+\infty, -\infty\}$
  - $\prec$ stands for a relation between $\mathbb{Z} \cup \{+\infty, -\infty\}$ and $\mathbb{Z}$
  - $k \in \mathbb{Z}$
  - a_i are atoms and
  - w_i are integers
- Example $sum\ \{30 : hd(a), \dots, 50 : hd(m)\} \leq 300$

Semantics

- A (positive) aggregate $\alpha \{w_1 : a_1, \dots, w_n : a_n\} \prec k$ can be represented by the formula:

$$\bigwedge_{I \subseteq \{1,\dots,n\}, \alpha\{w_i \mid i \in I\} \not\prec k} \left(\bigwedge_{i \in I} a_i \rightarrow \bigvee_{i \in \overline{I}} a_i \right)$$

  where $\overline{I} = \{1, \dots, n\} \setminus I$ and $\not\prec$ is the complement of $\prec$
- Then, $\alpha \{w_1 : a_1, \dots, w_n : a_n\} \prec k$ is true in X iff the above formula is true in X

Example

- Consider $sum\{1 : p, 1 : q\} \neq 1$
  That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\} = 1$
$\emptyset$	$\{\}$	0	*false*
$\{1\}$	$\{1\}$	1	*true*
$\{2\}$	$\{1\}$	1	*true*
$\{1, 2\}$	$\{1, 1\}$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\{1 : p, 1 : q\} = 1$

Potassco

Example

- Consider $sum\{1 : p, 1 : q\} \neq 1$
  That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\} = 1$
$\emptyset$	$\{\}$	0	*false*
$\{1\}$	$\{1\}$	1	*true*
$\{2\}$	$\{1\}$	1	*true*
$\{1,2\}$	$\{1,1\}$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\{1 : p, 1 : q\} = 1$

Example

- Consider $sum\{1 : p, 1 : q\} \neq 1$
  That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\} = 1$
$\emptyset$	$\{\}$	0	*false*
$\{1\}$	$\{1\}$	1	*true*
$\{2\}$	$\{1\}$	1	*true*
$\{1, 2\}$	$\{1, 1\}$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\{1 : p, 1 : q\} = 1$

Monotonicity

- Monotone aggregates
  - For instance,
    - $B(r)^+$
    - $sum\{1:p,1:q\} > 1$ amounts to $p \wedge q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i|i\in I\}\not\prec k} \bigvee_{i\in\bar{I}} a_i$
- Anti-monotone aggregates
  - For instance,
    - $B(r)^-$
    - $sum\{1:p,1:q\} < 1$ amounts to $\neg p \wedge \neg q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i|i\in I\}\not\prec k} \neg \bigwedge_{i\in I} a_i$
- Non-monotone aggregates
  - For instance, $sum\{1:p,1:q\} \neq 1$ is non-monotone.

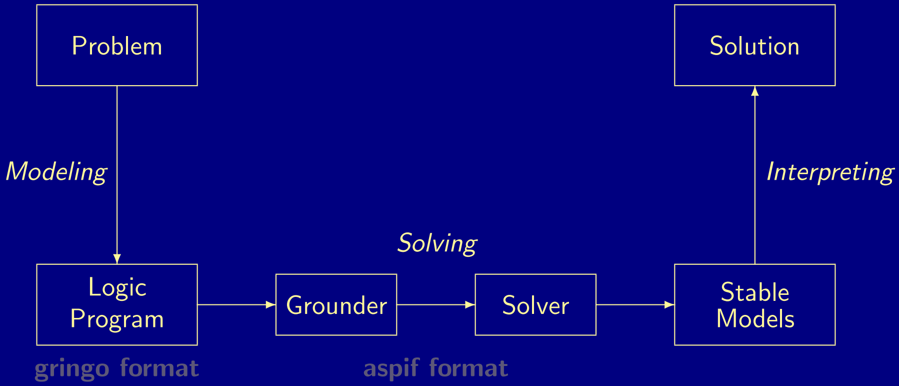
Monotonicity

- Monotone aggregates
  - For instance,
    - $B(r)^+$
    - $sum\{1:p,1:q\} > 1$ amounts to $p \wedge q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i\mid i\in I\}\not\prec k} \bigvee_{i\in\bar{I}} a_i$
- Anti-monotone aggregates
  - For instance,
    - $B(r)^-$
    - $sum\{1:p,1:q\} < 1$ amounts to $\neg p \wedge \neg q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i\mid i\in I\}\not\prec k} \neg \bigwedge_{i\in I} a_i$
- Non-monotone aggregates
  - For instance, $sum\{1:p,1:q\} \neq 1$ is non-monotone.

Outline

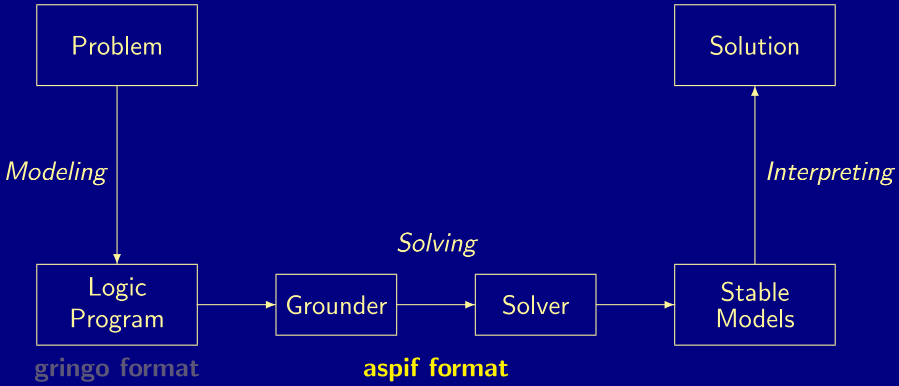
Potassco

Gringo language



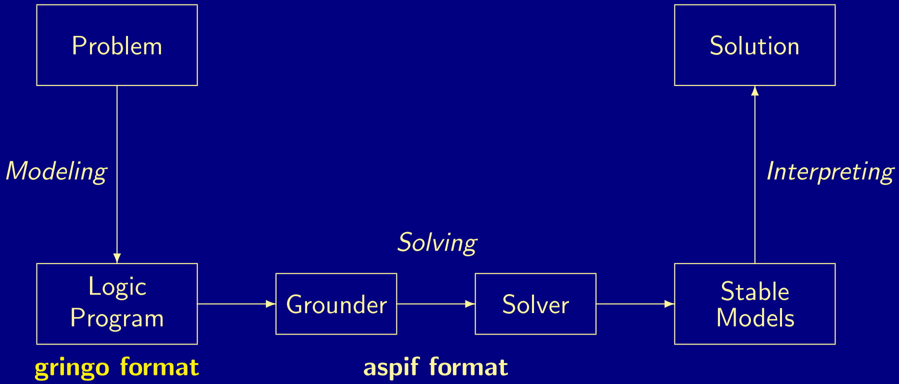
- aspif format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2*

Gringo language



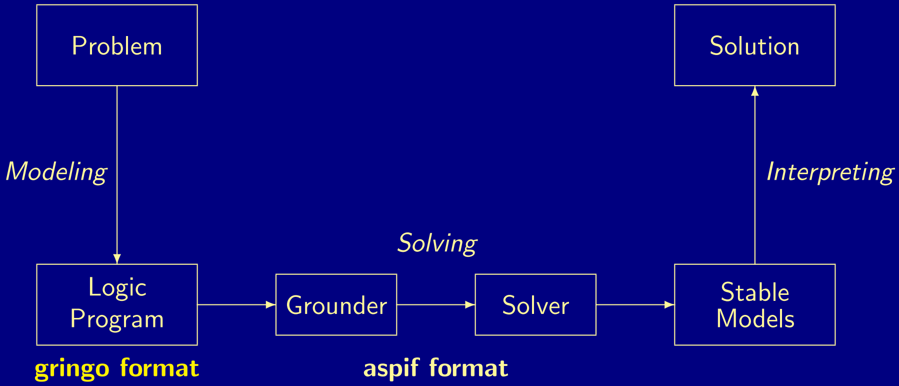
- aspif format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2*

Gringo language



- aspif format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2*

Gringo language



- aspif format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2*

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t are formed from
  - constant symbols, eg `c`, `d`, ...
  - function symbols, eg `f`, `g`, ...
  - numeric symbols, eg 1, 2, ...
  - variable symbols, eg `X`, `Y`, ..., `_`
  - parentheses (,)
  - tuple delimiters $\langle, \rangle$ (omitted whenever possible)
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t are formed from
  - constants, eg `c`, `d`, ...
  - functions, eg `f`, `g`, ...
  - numerics, eg 1, 2, ...
  - variables, eg `X`, `Y`, ..., `_`
  - parentheses (,)
  - tuple delimiters $\langle$, $\rangle$
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Potassco

Terms and literals

- Terms t are formed from
  - constants, eg `c`, `d`, ...
  - functions, eg `f`, `g`, ...
  - numerics, eg `1`, `2`, ...
  - variables, eg `X`, `Y`, ..., `_`
  - parentheses (,)
  - tuple delimiters $\langle, \rangle$

  eg `f(3,c,Z)`, `g(42,_,_)`, or `f((3,c),X)`
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$ of terms
- Atoms a, $\neg a$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- (Negated) Atoms a, $\neg a$ are formed from
  - predicate symbols, eg p, q, ...
  - parentheses (,)
  - tuples of terms
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$ are formed from
  - predicates, eg p, q, ...
  - parentheses (,)
  - tuples of terms
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$ are formed from
  - predicates, eg p, q, ...
  - parentheses (,)
  - tuples of terms

  eg `-p(f(3,c,Z),g(42,_,_))` or `q()` written as `q`
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$, $\bot$, $\top$
  viz `#false` and `#true`
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
  eg `p(a,X)`, '`not p(a,X)`', '`not not p(a,X)`'
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$ where
  - t_1 and t_2 are terms
  - $\prec$ is a comparison symbol
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$ where
  - t_1 and t_2 are terms
  - $\prec$ is a comparison symbol

  eg `3<1` or `f(42)=X`
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$ of literals
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$ where
  - l is a symbolic or arithmetic literal
  - $\boldsymbol{L}$ is a tuple of symbol or arithmetic literals
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$ where
  - l is a symbolic or arithmetic literal
  - $\boldsymbol{L}$ is a tuple of symbol or arithmetic literals
  - $l : \boldsymbol{L}$ is written as l whenever $\boldsymbol{L}$ is empty
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$ where
  - l is a symbolic or arithmetic literal
  - $\boldsymbol{L}$ is a tuple of symbol or arithmetic literals

  eg '`p(X,Y):q(X),r(Y)`' or `p(42)` or '`#false:q`'
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\mathbf{t}$, $\mathbf{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \mathbf{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots ; \mathbf{t}_n : \mathbf{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\mathbf{t}_1 : \mathbf{L}_1, \dots, \mathbf{t}_n : \mathbf{L}_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t}_n : \boldsymbol{L}_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms
  - one (or both) of '$s_1 \prec_1$' and '$\prec_2 s_2$' can be omitted
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t_n} : \boldsymbol{L_n}$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms
  - omitting $\prec_1$ or $\prec_2$ defaults to $\leq$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t_n} : \boldsymbol{L_n}$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms

  eg `10 <= #sum {6,C:course(C); 3,S:seminar(S)} <= 20`
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\mathbf{t}$, $\mathbf{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \mathbf{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots; \mathbf{t}_n : \mathbf{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\mathbf{t}_1 : \mathbf{L}_1, \dots, \mathbf{t}_n : \mathbf{L}_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms

  eg `10 #sum {6,C:course(C); 3,S:seminar(S)} 20`
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$ where
  - a is an aggregate atom
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$ where
  - a is an aggregate atom

  eg `not 10 #sum {6,C:course(C); 3,S:seminar(S)} 20`
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $\neg a$, $\bot$, $\top$
- Symbolic literals a, $\sim a$, $\sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\sim a$, $\sim\sim a$
- Literals are conditional or aggregate literals

- For a detailed account please consult the user's guide!

Rules

- Rules are of the form

$$l_1 ; \dots ; l_m \leftarrow l_{m+1}, \dots, l_n \tag{2}$$

  where
  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m + 1 \leq i \leq n$

- Note Semicolons ';' must be used in (2) instead of commas ','
  whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$

- Example `a(X) :- b(X) : c(X), d(X); e(x).`

Potassco

Rules

- Rules are of the form

$$l_1 \,;\, \dots \,;\, l_m \leftarrow l_{m+1}, \dots, l_n \tag{2}$$

  where
  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m+1 \leq i \leq n$

- Note Semicolons ';' must be used in (2) instead of commas ',' whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$

- Example `a(X) :- b(X) : c(X), d(X); e(x).`

Rules

- Rules are of the form

$$l_1 \, ; \ldots \, ; l_m \leftarrow l_{m+1}, \ldots, l_n \tag{2}$$

  where
  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m+1 \leq i \leq n$

- Note Semicolons ';' must be used in (2) instead of commas ',' whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$
- Example `a(X) :- b(X) : c(X), d(X); e(x).`

Shortcuts

- A rule of the form

$$s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k : \boldsymbol{L}_k\} \prec_2 s_2 \;\leftarrow\; l_{m+1}, \dots , l_n$$

where

  - α, $\prec_i$, s_i, $\boldsymbol{t}_j$ are as given above for $i = 1, 2$ and $1 \leq j \leq k$
  - $l_j : \boldsymbol{L}_j$ is a conditional literal for $1 \leq j \leq k$
  - l_i is a literal for $m + 1 \leq i \leq n$ (as in (2))

is a shorthand for the following $k + 1$ rules

$$\{l_j\} \leftarrow l_{m+1}, \dots , l_n, \boldsymbol{L}_j \qquad \text{for } 1 \leq j \leq k$$

$$\leftarrow l_{m+1}, \dots , l_n, \sim s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1, \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k, \boldsymbol{L}_k\} \prec_2 s_2$$

- Example `10 < #sum { C,X,Y : edge(X,Y) : cost(X,Y,C) }.`

Potassco

Shortcuts

- A rule of the form

$$s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k : \boldsymbol{L}_k\} \prec_2 s_2 \;\leftarrow\; l_{m+1}, \dots , l_n$$

  where
  - α, $\prec_i$, s_i, $\boldsymbol{t}_j$ are as given above for $i = 1, 2$ and $1 \leq j \leq k$
  - $l_j : \boldsymbol{L}_j$ is a conditional literal for $1 \leq j \leq k$
  - l_i is a literal for $m + 1 \leq i \leq n$ (as in (2))

  is a shorthand for the following $k + 1$ rules

$$\{l_j\} \leftarrow l_{m+1}, \dots , l_n, \boldsymbol{L}_j \qquad \text{for } 1 \leq j \leq k$$

$$\leftarrow l_{m+1}, \dots , l_n, \sim s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1, \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k, \boldsymbol{L}_k\} \prec_2 s_2$$

- Example `10 < #sum { C,X,Y : edge(X,Y) : cost(X,Y,C) }.`

Shortcuts

- A rule of the form

$$s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k : \boldsymbol{L}_k\} \prec_2 s_2 \ \leftarrow \ l_{m+1}, \dots, l_n$$

  where

  - α, $\prec_i$, s_i, $\boldsymbol{t}_j$ are as given above for $i = 1,2$ and $1 \leq j \leq k$
  - $l_j : \boldsymbol{L}_j$ is a conditional literal for $1 \leq j \leq k$
  - l_i is a literal for $m+1 \leq i \leq n$ (as in (2))

  is a shorthand for the following $k+1$ rules

$$\begin{array}{ll} \{l_j\} \leftarrow l_{m+1}, \dots, l_n, \boldsymbol{L}_j & \text{for } 1 \leq j \leq k \\ \quad\leftarrow l_{m+1}, \dots, l_n, \sim s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1, \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k, \boldsymbol{L}_k\} \prec_2 s_2 & \end{array}$$

- Example `10 < #sum { C,X,Y : edge(X,Y) : cost(X,Y,C) }.`

Potassco

Shortcuts

- The expression

  $$s_1 \ \{l_1 : \boldsymbol{L}_1; \ldots; l_k : \boldsymbol{L}_k\} \ s_2$$

  is a shortcut for

  - $s_1 \leq count\{t_1 : l_1 : \boldsymbol{L}_1; \ldots; t_k : l_k : \boldsymbol{L}_k\} \leq s_2$ if it appears in the head of a rule and
  - $s_1 \leq count\{t_1 : l_1, \boldsymbol{L}_1; \ldots; t_k : l_k, \boldsymbol{L}_k\} \leq s_2$ if it appears in the body of a rule

  where $t_i \neq t_j$ whenever $L_i \neq L_j$ for $i \neq j$ and $1 \leq i, j \leq k$

- Note one (or both) of s_1 and s_2 can be omitted

Shortcuts

- The expression

  $$s_1 \ \{l_1 : \boldsymbol{L}_1; \dots ; l_k : \boldsymbol{L}_k\} \ s_2$$

  is a shortcut for
  - $s_1 \leq count\{t_1 : l_1 : \boldsymbol{L}_1; \dots ; t_k : l_k : \boldsymbol{L}_k\} \leq s_2$
    if it appears in the head of a rule and
  - $s_1 \leq count\{t_1 : l_1, \boldsymbol{L}_1; \dots ; t_k : l_k, \boldsymbol{L}_k\} \leq s_2$
    if it appears in the body of a rule

  where $t_i \neq t_j$ whenever $L_i \neq L_j$ for $i \neq j$ and $1 \leq i, j \leq k$
- Note one (or both) of s_1 and s_2 can be omitted

Examples

- {a; b}

```
$ gringo --text <(echo "{a;b}.")
#count{1,0,a:a;1,0,b:b}.
```

gringo generates two distinct term tuples `1,0,a` and `1,0,b`

```
1 = { q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) }
```

Examples

- `{a; b}`

  ```
  $ gringo --text <(echo "{a;b}.")
  #count{1,0,a:a;1,0,b:b}.
  ```

  gringo generates two distinct term tuples `1,0,a` and `1,0,b`

  `1 = { q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) }`

Examples

- {a; b}

```
$ gringo --text <(echo "{a;b}.")
#count{1,0,a:a;1,0,b:b}.
```

gringo generates two distinct term tuples `1,0,a` and `1,0,b`

- `1 = { q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) }`

Examples

- `{a; b}`

  ```
  $ gringo --text <(echo "{a;b}.")
  #count{1,0,a:a;1,0,b:b}.
  ```

  gringo generates two distinct term tuples `1,0,a` and `1,0,b`

- `1 = { q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) }`

Potassco

Examples

- `{a; b}`

  ```
  $ gringo --text <(echo "{a;b}.")
  #count{1,0,a:a;1,0,b:b}.
  ```

  gringo generates two distinct term tuples `1,0,a` and `1,0,b`

- `1 = { q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) }`

Weak constraints

- Syntax A weak constraint is of the form

$$:\sim l_1, \ldots, l_n. [w@p, t_1, \ldots, t_m]$$

  where
  - $l_1, \ldots, l_n$ are literals
  - $t_1, \ldots, t_m$, w, and p are terms
- w and p stand for a weight and priority level ($p = 0$ if '@p' is omitted)
- Example The weak constraint

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

  amounts to the minimize statement

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

Weak constraints

- Syntax A weak constraint is of the form

$$:\sim l_1, \dots, l_n. [w@p, t_1, \dots, t_m]$$

  where
  - $l_1, \dots, l_n$ are literals
  - $t_1, \dots, t_m$, w, and p are terms
- w and p stand for a weight and priority level ($p = 0$ if '@p' is omitted)
- Example The weak constraint

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

  amounts to the minimize statement

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

Potassco

Weak constraints

- Syntax A weak constraint is of the form

$$:\sim l_1, \dots, l_n. [w@p, t_1, \dots, t_m]$$

  where
  - $l_1, \dots, l_n$ are literals
  - $t_1, \dots, t_m$, w, and p are terms
- w and p stand for a weight and priority level ($p = 0$ if '@p' is omitted)
- Example The weak constraint

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

  amounts to the minimize statement

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

Weak constraints

- Syntax A weak constraint is of the form

  $$:\sim l_1, \dots, l_n. [w@p, t_1, \dots, t_m]$$

  where
  - $l_1, \dots, l_n$ are literals
  - $t_1, \dots, t_m$, w, and p are terms
- w and p stand for a weight and priority level ($p = 0$ if '$@p$' is omitted)
- Example The weak constraint

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

  amounts to the minimize statement

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

Some more directives

- Output

  $\#\texttt{show}.\qquad \#\texttt{show}\ p/n.\qquad \#\texttt{show}\ t\ :\ l_1,\dots,l_n.$

- Projection

  $\#\texttt{project}\ p/n.\qquad \#\texttt{project}\ a\ :\ l_1,\dots,l_n.$

- Heuristics

  $\#\texttt{heuristic}\ a\ :\ l_1,\dots,l_n.\ [k@p,m]$

- Acyclicity

  $\#\text{edge}\ (u,v)\ :\ l_1,\dots,l_n.$

Some more directives

- Output

$$\#\texttt{show}. \qquad \#\texttt{show}\ p/n. \qquad \#\texttt{show}\ t : l_1, \dots, l_n.$$

- Projection

$$\#\texttt{project}\ p/n. \qquad \#\texttt{project}\ a : l_1, \dots, l_n.$$

- Heuristics

$$\#\texttt{heuristic}\ a : l_1, \dots, l_n. \ [k@p, m]$$

- Acyclicity

$$\#\mathrm{edge}\ (u, v) : l_1, \dots, l_n.$$

Some more directives

- Output

  $\#\texttt{show.}$ $\quad \#\texttt{show}\ p/n.$ $\quad \#\texttt{show}\ t : l_1, \dots, l_n.$

- Projection

  $\#\texttt{project}\ p/n.$ $\quad \#\texttt{project}\ a : l_1, \dots, l_n.$

- Heuristics

  $\#\texttt{heuristic}\ a : l_1, \dots, l_n.\ [k@p, m]$

- Acyclicity

  $\#\mathrm{edge}\ (u, v) : l_1, \dots, l_n.$

Some more directives

- Output

  $\#\texttt{show}.\qquad \#\texttt{show}\ p/n.\qquad \#\texttt{show}\ t\ :\ l_1,\dots,l_n.$

- Projection

  $\#\texttt{project}\ p/n.\qquad \#\texttt{project}\ a\ :\ l_1,\dots,l_n.$

- Heuristics

  $\#\texttt{heuristic}\ a\ :\ l_1,\dots,l_n.\ [k@p,m]$

- Acyclicity

  $\#\texttt{edge}\ (u,v)\ :\ l_1,\dots,l_n.$

gringo 3 versus 4/5

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- 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* 4/5:
  - ```
    r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
    ```
- Note Directives `#compute`, `#domain`, and `#hide` are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3

gringo 3 versus 4/5

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- 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* 4/5:
  - ```
    r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
    ```
- Note Directives `#compute`, `#domain`, and `#hide` are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3

Potassco

gringo 3 versus 4/5

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- 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* 4/5:
  - ```
    r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
           1 <= #count { 1,r(X) : r(X), p(X), not q(X) }.
    ```
- Note Directives `#compute`, `#domain`, and `#hide` are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3

gringo 3 versus 4/5

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- 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* 4/5:
  - ```
    r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
          1 { r(X) : p(X), not q(X) }.
    ```
- Note Directives `#compute`, `#domain`, and `#hide` are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3

Potassco

gringo 3 versus 4/5

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- 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* 4/5:
  - ```
    r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
           1 { r(X) : p(X), not q(X) }.
    ```
- Note Directives `#compute`, `#domain`, and `#hide` are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3

gringo 3 versus 4/5

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- 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* 4/5:
  - ```
    r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
           1 { r(X) : p(X), not q(X) }.
    ```
- Note Directives `#compute`, `#domain`, and `#hide` are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3

[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.

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*.

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.

[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.

[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, 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`.

[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.

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.

[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.

[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.

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.

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.

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?

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.

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.

[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.

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.

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.