Answer Set Solving in Practice

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

Rough Roadmap

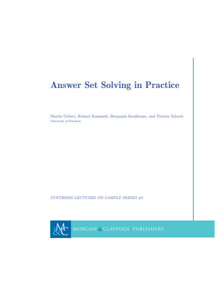
1. Introduction
2. Language
3. Modeling
4. Grounding
5. Foundations
6. Solving
7. Systems
8. Applications

Resources

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

The Potassco Book

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



Resources

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

Literature

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

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

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

Language Extensions: Overview

1. Two kinds of negation
2. Disjunctive logic programs
3. Propositional theories

Outline

Motivation

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

Motivation

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

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 $P \cup P^{\neg}$

Potassco

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 $P \cup P^{\neg}$

Potassco

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 $P \cup P^{\neg}$

Potassco

An example

- The program

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

induces

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

- 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 $\{a\}$

Potassco

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 $X = \mathcal{A} \cup \overline{\mathcal{A}}$
- Note 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 $X = \mathcal{A} \cup \overline{\mathcal{A}}$
- Note 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 $X = \mathcal{A} \cup \overline{\mathcal{A}}$
- Note 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

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

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

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

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

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

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

- 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

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

- 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 $\mathcal{A} \cup \widetilde{\mathcal{A}}$

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

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

- 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

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

- 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

Outline

Potassco

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

$$\begin{aligned} head(r) &= \{a_1, \ldots, a_m\} \\ body(r) &= \{a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o\} \\ body(r)^+ &= \{a_{m+1}, \ldots, a_n\} \\ body(r)^- &= \{a_{n+1}, \ldots, a_o\} \\ atom(P) &= \bigcup_{r \in P} \left(head(r) \cup body(r)^+ \cup body(r)^-\right) \\ body(P) &= \{body(r) \mid r \in P\} \end{aligned}$$

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

Potassco

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

$$\begin{aligned}
head(r) &= \{a_1,\ldots,a_m\} \\
body(r) &= \{a_{m+1},\ldots,a_n,\sim a_{n+1},\ldots,\sim a_o\} \\
body(r)^+ &= \{a_{m+1},\ldots,a_n\} \\
body(r)^- &= \{a_{n+1},\ldots,a_o\} \\
atom(P) &= \textstyle\bigcup_{r\in P}\left(head(r)\cup body(r)^+\cup body(r)^-\right) \\
body(P) &= \{body(r) \mid r \in P\}
\end{aligned}$$

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

Potassco

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

$$\begin{aligned}
head(r) &= \{a_1,\ldots,a_m\}\\
body(r) &= \{a_{m+1},\ldots,a_n,\sim a_{n+1},\ldots,\sim a_o\}\\
body(r)^+ &= \{a_{m+1},\ldots,a_n\}\\
body(r)^- &= \{a_{n+1},\ldots,a_o\}\\
atom(P) &= \bigcup_{r\in P}\left(head(r)\cup body(r)^+\cup body(r)^-\right)\\
body(P) &= \{body(r) \mid r \in P\}
\end{aligned}$$

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

Stable models

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

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

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

Potassco

Stable models

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

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

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

Potassco

Stable models

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

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

  - 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 $\min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}$

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 $\min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}$

Potassco

Graph coloring (reloaded)

```
node(1..6).

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


color(X,r) | color(X,b) | color(X,g) :- node(X).


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

Graph coloring (reloaded)

```
node(1..6).

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

col(r).   col(b).   col(g).

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

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

Potassco

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 $\{b\}$

Potassco

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 $body(r)^+ \subseteq X$, $body(r)^- \cap X = \emptyset$, and $head(r) \cap X = \{A\}$

Potassco

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 $body(r)^+ \subseteq X$, $body(r)^- \cap X = \emptyset$, and $head(r) \cap X = \{A\}$

Potassco

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

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

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

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

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

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

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 $X \in \min_{\subseteq}(ground(P)^X)$

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 $X \in \min_{\subseteq}(ground(P)^X)$

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 $X \in \min_{\subseteq}(ground(P)^X)$

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 $X \in \min_{\subseteq}(ground(P)^X)$

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 $X \in \min_\subseteq(ground(P)^X)$

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 $X \notin \min_\subseteq(ground(P)^X)$

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

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

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

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

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

- 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 $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Potassco

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

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

- 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

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

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

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

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

- 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

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

- 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

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:

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

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 $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

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:

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

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 $\phi \in X$
  - $\phi^X = (\psi^X \circ H^X)$ if $X \models \phi$ and $\phi = (\psi \circ H)$ 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 $\phi \in X$
  - $\phi^X = (\psi^X \circ H^X)$ if $X \models \phi$ and $\phi = (\psi \circ H)$ 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 $\phi \in X$
  - $\phi^X = (\psi^X \circ H^X)$ if $X \models \phi$ and $\phi = (\psi \circ H)$ 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 $\phi \in X$
  - $\phi^X = (\psi^X \circ H^X)$ if $X \models \phi$ and $\phi = (\psi \circ H)$ 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 $\phi \in X$
  - $\phi^X = (\psi^X \circ H^X)$ if $X \models \phi$ and $\phi = (\psi \circ H)$ 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

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 $X \in \min_{\subseteq}(\Phi)$!

Potassco

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 $X \in \min_{\subseteq}(\Phi)$!

Potassco

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 $X \in \min_{\subseteq}(\Phi)$!

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 $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

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 $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$ ✔

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 $\tau[P]$

Potassco

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 $\tau[P]$

Potassco

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 $\{p\}$

Potassco

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 $\{p\}$

Potassco

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 $\{p\}$

Potassco

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 $\{p\}$

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

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

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

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

Potassco

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

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

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

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

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

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

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

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

[12] K. Clark.
Negation as failure.

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

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

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

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

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

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

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

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

Potassco

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.

Potasscco

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Potassco

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

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

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

[40] M. Gelfond and N. Leone.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Master's thesis, Simon Fraser University, 2004.

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

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

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

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

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