Answer Set Solving in Practice

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

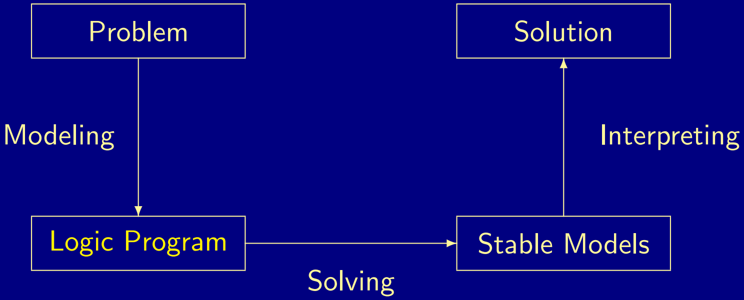
Potassco

Introduction: Overview

1. Syntax
2. Semantics
3. Examples
4. Reasoning
5. Language
6. Variables

Outline

Syntax


Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

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

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\} \\ B(r)^+ &= \{a_1, \dots, a_m\} \\ B(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

A literal is an atom or a negated atom

A program P is positive if $B(r)^- = \emptyset$ for all $r \in P$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

$$\mathtt{a}_0 \;\mathtt{:-}\; \mathtt{a}_1,\ldots,\mathtt{a}_m,\; \mathtt{not}\; \mathtt{a}_{m+1},\ldots,\; \mathtt{not}\; \mathtt{a}_n.$$

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

- Notation

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1,\ldots,a_m,{\sim}a_{m+1},\ldots,{\sim}a_n\} \\ B(r)^+ &= \{a_1,\ldots,a_m\} \\ B(r)^- &= \{a_{m+1},\ldots,a_n\} \end{aligned}$$

- A literal is an atom or a negated atom
- A program P is positive if $B(r)^- = \emptyset$ for all $r \in P$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

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

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

- Notation

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\} \\ B(r)^+ &= \{a_1, \dots, a_m\} \\ B(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

- A literal is an atom or a negated atom
- A program P is positive if $B(r)^- = \emptyset$ for all $r \in P$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

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

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- Notation

$$\begin{aligned}
h(r) &= a_0 \\
B(r) &= \{a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n\} \\
B(r)^+ &= \{a_1, \dots, a_m\} \\
B(r)^- &= \{a_{m+1}, \dots, a_n\} \\
A(P) &= \bigcup_{r \in P} \left(\{h(r)\} \cup B(r)^+ \cup B(r)^-\right) \\
B(P) &= \{B(r) \mid r \in P\} \\
h(P) &= \{h(r) \mid r \in P\}
\end{aligned}$$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

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

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

- Notation

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\} \\ B(r)^+ &= \{a_1, \dots, a_m\} \\ B(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

- A literal is an atom or a negated atom
- A program P is positive if $B(r)^- = \emptyset$ for all $r \in P$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

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

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- Notation

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\} \\ B(r)^+ &= \{a_1, \dots, a_m\} \\ B(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

- A literal is an atom or a negated atom
- A program P is positive if $B(r)^- = \emptyset$ for all $r \in P$

Examples

- Example rules
  - $a \leftarrow b, \sim c$
  - $a \leftarrow \sim c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \sim c$
  - $bachelor(joe) \leftarrow male(joe), \sim married(joe)$
- Example literals

  $a, b, c, bachelor(joe), male(joe), married(joe)$

  $\sim c, \sim married(joe)$

Examples

- Example rules
  - $a \leftarrow b, \sim c$
  - $a \leftarrow \sim c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \sim c$
  - $bachelor(joe) \leftarrow male(joe), \sim married(joe)$
- Example literals
  - $a, b, c, bachelor(joe), male(joe), married(joe)$
  - $\sim c, \sim married(joe)$

Examples

- Example rules
  - $a \leftarrow b, \sim c$
  - $a \leftarrow \sim c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \sim c$
  - $bachelor(joe) \leftarrow male(joe), \sim married(joe)$
- Example literals
  - $a, b, c, bachelor(joe), male(joe), married(joe)$
  - $\sim c, \sim married(joe)$

Examples

- Example rules
  - $a \leftarrow b, \sim c$
  - $a \leftarrow \sim c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \sim c$
  - $bachelor(joe) \leftarrow male(joe), \sim married(joe)$
- Example literals
  - $a, b, c, bachelor(joe), male(joe), married(joe)$
  - $\sim c, \sim married(joe)$

Notational convention

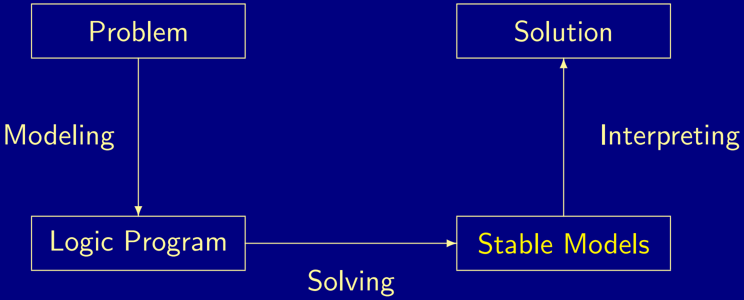
We sometimes use the following notation interchangeably
in order to stress the respective view:

	true, false	if	and	or	iff	default negation	classical negation
source code		`:-`	`,`	`;`		`not`	`-`
logic program		$\leftarrow$	$,$	$;$		$\sim$	$\neg$
formula	$\bot, \top$	$\rightarrow$	$\wedge$	$\vee$	$\leftrightarrow$	$\sim$	$\neg$

Outline

Potassco

Semantics


Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$
  - X corresponds to a model of P (seen as a formula)
- The smallest set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P (ditto)
- The set $Cn(P)$ of atoms is the stable model of a *positive* program P

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$
  - X corresponds to a model of P (seen as a formula)
- The smallest set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P (ditto)
- The set $Cn(P)$ of atoms is the stable model of a *positive* program P

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$
  - X corresponds to a model of P (seen as a formula)
- The smallest set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P (ditto)
- The set $Cn(P)$ of atoms is the stable model of a *positive* program P

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$
  - X corresponds to a model of P (seen as a formula)
- The smallest set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P (ditto)
- The set $Cn(P)$ of atoms is the stable model of a *positive* program P

Some "logical" remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

    $$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model
- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Some “logical” remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

    $$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model
- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Some "logical" remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

    $$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model
- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \wedge (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ $\boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}}$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \ \wedge \ (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ $\boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}}$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

$$\Phi \quad \boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$$

p	$\mapsto$	1
q	$\mapsto$	1
r	$\mapsto$	0

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

$$P_\Phi \quad \begin{array}{|lll|} q & \leftarrow & \\ p & \leftarrow & q, \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ
$$\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \; \sim r \end{array}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Φ $\boxed{q \ \wedge \ (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

P_Φ $\boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}}$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p,q\}, \{q,r\}, \text{ and } \{p,q,r\}$$

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

Formula Φ has one stable model, often called answer set:

$$\{p,q\}$$

P_Φ
$$\boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q,\ \sim r \end{array}}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

$$P_\Phi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Basic idea

Consider the logical formula Φ and its three (classical) models:

Φ $\boxed{q \;\wedge\; (q \wedge \neg r \rightarrow p)}$

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula Φ has one stable model, often called answer set:

$$\{p, q\}$$

$$P_\Phi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ \sim r \end{array}}$$

Informally, a set X of atoms is a stable model of a logic program P

- if X is a (classical) model of P and
- if all atoms in X are justified by some rule in P

Potassco

Formal definition

Stable models of normal programs

- The reduct, P^X, of a 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 program P, if $Cn(P^X) = X$

- Remarks

  $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of P^X

  Each atom in X is justified by an *"applying rule from P"*
  Set X is stable under *"applying rules from P"*

Formal definition

Stable models of normal programs

- The **reduct**, P^X, of a 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 program P, if $Cn(P^X) = X$

- Remarks
  - $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of P^X
  - Each atom in X is justified by an *"applying rule from P"*
  - Set X is stable under *"applying rules from P"*

Formal definition

Stable models of normal programs

- The reduct, P^X, of a 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 program P, if $Cn(P^X) = X$

- Remarks
  - $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of P^X
  - Each atom in X is justified by an *"applying rule from P"*
  - Set X is stable under *"applying rules from P"*

Formal definition

Stable models of normal programs

- The reduct, P^X, of a 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 program P, if $Cn(P^X) = X$

- Remarks
  - $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of P^X
  - Each atom in X is justified by an *"applying rule from P"*
  - Set X is stable under *"applying rules from P"*

A closer look at P^X

- Alternatively, given a set X of atoms from P,

  P^X is obtained from P by deleting

  1. each rule having $\sim a$ in its body with $a \in X$
     and then
  2. all negative atoms of the form $\sim a$
     in the bodies of the remaining rules

- Note Only negative body literals are evaluated

A closer look at P^X

- Alternatively, given a set X of atoms from P,

  P^X is obtained from P by deleting

  1. each rule having $\sim a$ in its body with $a \in X$ and then
  2. all negative atoms of the form $\sim a$ in the bodies of the remaining rules

- Note Only negative body literals are evaluated

Outline

Example one

$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$

X	P^X	$Cn(P^X)$
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p\}$	$p \leftarrow p$	$\emptyset$
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p, q\}$	$p \leftarrow p$	$\emptyset$

Example one

$P = \{p \leftarrow p, \;\; q \leftarrow \sim p\}$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\quad\}$	$p \leftarrow p$	$\emptyset$	
$\{\quad q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Example one

$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✓
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Example one

$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Example one

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Example one

$P = \{p \leftarrow p,\ q \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✘
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

Potassco

Example one

$P = \{p \leftarrow p,\ q \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Example one

$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Potassco

Example one

$P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✔
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✔

Example two

$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$
$\{p\ \}$	$p \leftarrow$	$\{p\}$
$\{\ q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		$\emptyset$

Potassco

Example two

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✗
$\{p\quad\}$	$p \leftarrow$	$\{p\}$	✓
$\{\quad q\}$	$q \leftarrow$	$\{q\}$	✓
$\{p, q\}$		$\emptyset$	

Example two

$$P = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$
$\{p\ \}$	$p \leftarrow$	$\{p\}$
$\{\ q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		$\emptyset$

Example two

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$	✔
$\{\ q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	

Example two

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\quad\}$	$p \leftarrow$	$\{p\}$	✔
$\{\quad q\}$	$q \leftarrow$	$\{q\}$	
$\{p, q\}$		$\emptyset$	

Potassco

Example two

$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$	✔
$\{\ q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	

Potassco

Example two

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$	✔
$\{\ q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

Example two

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\quad\}$	$p \leftarrow$	$\{p\}$	✔
$\{\quad q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

Example two

$$P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$	✔
$\{\ q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✔

Example three

$$P = \{p \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$
$\{\ \}$	$p \leftarrow$	$\{p\}$
$\{p\}$		$\emptyset$

Example three

$P = \{p \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✗
$\{p\}$		$\emptyset$	

Example three

$P = \{p \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$
$\{\ \}$	$p \leftarrow$	$\{p\}$
$\{p\}$		$\emptyset$

Example three

$P = \{p \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✗
$\{p\}$		$\emptyset$	

Example three

$P = \{p \leftarrow {\sim}p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	✘

Example three

$P = \{p \leftarrow \sim p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	✘

Example three

$P = \{p \leftarrow \neg p\}$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	✔

Some properties

- A logic program may have zero, one, or multiple stable models
- If X is a stable model of a logic program P,
  then $X \subseteq h(P)$
- If X is a stable model of a logic program P,
  then X is a (classical) model of P
- If X and Y are stable models of a *normal* program P,
  then $X \not\subset Y$

Potassco

Some properties

- A logic program may have zero, one, or multiple stable models
- If X is a stable model of a logic program P, then $X \subseteq h(P)$
- If X is a stable model of a logic program P, then X is a (classical) model of P
- If X and Y are stable models of a *normal* program P, then $X \not\subset Y$

Some properties

- A logic program may have zero, one, or multiple stable models
- If X is a stable model of a logic program P, then $X \subseteq h(P)$
- If X is a stable model of a logic program P, then X is a (classical) model of P
- If X and Y are stable models of a *normal* program P, then $X \not\subset Y$

Some properties

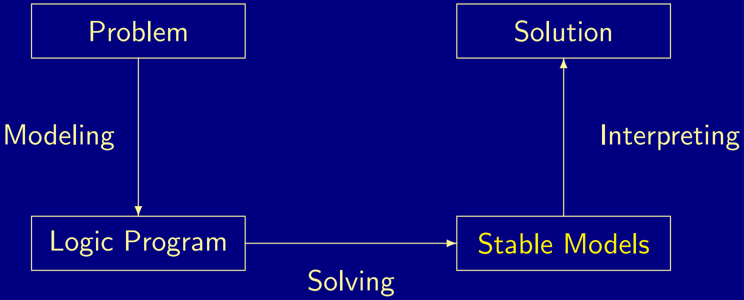
- A logic program may have zero, one, or multiple stable models
- If X is a stable model of a logic program P,
  then $X \subseteq h(P)$
- If X is a stable model of a logic program P,
  then X is a (classical) model of P
- If X and Y are stable models of a *normal* program P,
  then $X \not\subset Y$

Exemplars

Logic program	Answer sets
`a.`	{a}
`a :- b.`	{}
`a :- b.    b.`	{a,b}
`a :- b.    b :- a.`	{}
`a :- not c.`	{a}
`a :- not c.    c.`	{c}
`a :- not c.    c :- not a.`	{a}, {c}
`a :- not a.`	

Outline

Reasoning modes



Reasoning modes

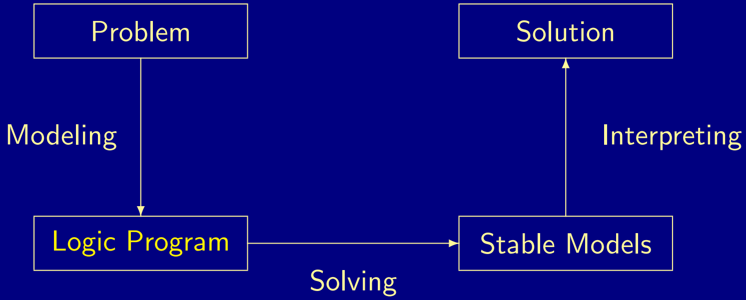
- Satisfiability
- Enumeration†
- Projection†
- Intersection‡
- Union‡
- Optimization
- and combinations of them

† without solution recording

‡ without solution enumeration

Outline

Extended syntax



Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

`:~ q(X), p(X,C) [C]`

`#minimize { C : q(X), p(X,C) }`

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Multi-objective optimization
  - Weak constraints `:~ q(X), p(X,C) [C@42]`
  - Statements `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Outline

Example

$$d(a)$$
$$d(c)$$
$$d(d)$$
$$p(a, b)$$
$$p(b, c)$$
$$p(c, d)$$
$$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$$
$$q(a)$$
$$q(b)$$
$$q(X) \leftarrow \sim r(X), d(X)$$
$$r(X) \leftarrow \sim q(X), d(X)$$
$$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$$

Example

$$
\begin{array}{l}
d(a) \\
d(c) \\
d(d) \\
p(a,b) \\
p(b,c) \\
p(c,d) \\
p(X,Z) \leftarrow p(X,Y), p(Y,Z) \\
q(a) \\
q(b) \\
q(X) \leftarrow \sim r(X), d(X) \\
r(X) \leftarrow \sim q(X), d(X) \\
s(X) \leftarrow \sim r(X), p(X,Y), q(Y)
\end{array}
$$

Grounding instantiation

Let P be a logic program

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r; θ is a (ground) substitution

- Ground instantiation of P: $ground(P) = \bigcup_{r \in P} ground(r)$

Potassco

Grounding instantiation

Let P be a logic program

- Let $\mathcal{T}$ be a set of variable-free terms (also called Herbrand universe)
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$ (also called alphabet or Herbrand base)
- A variable-free atom is also called ground
- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r; θ is a (ground) substitution
- Ground instantiation of P: $ground(P) = \bigcup_{r \in P} ground(r)$

Grounding instantiation

Let P be a logic program

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r; θ is a (ground) substitution

- Ground instantiation of P: $ground(P) = \bigcup_{r \in P} ground(r)$

Potassco

Grounding instantiation

Let P be a logic program

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

where $var(r)$ stands for the set of all variables occurring in r;
θ is a (ground) substitution

- Ground instantiation of P: $ground(P) = \bigcup_{r \in P} ground(r)$

Grounding instantiation

Let P be a logic program

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r;
  θ is a (ground) substitution

- Ground instantiation of P: $ground(P) = \bigcup_{r \in P} ground(r)$

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lllllllll} r(a,b) & \leftarrow & , & & & & & & \\ r(b,c) & \leftarrow & , & & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow & r(c,c) \end{array} \right\}$$

Grounding aims at reducing the ground instantiation

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllllll} r(a,b) & \leftarrow & , & & & & & \\ r(b,c) & \leftarrow & , & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) \leftarrow & r(b,a), & t(c,a) & \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) \leftarrow & r(b,b), & t(c,b) & \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) \leftarrow & r(b,c), & t(c,c) & \leftarrow & r(c,c) \end{array} \right\}$$

- Grounding aims at reducing the ground instantiation

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lllllllll} r(a,b) & \leftarrow & , & & & & & & \\ r(b,c) & \leftarrow & , & & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow & r(c,c) \end{array} \right\}$$

- Grounding aims at reducing the ground instantiation

Potassco

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllllll} r(a,b) & \leftarrow & , & & & & & \\ r(b,c) & \leftarrow & , & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & , & t(b,b) & \leftarrow & r(b,b), & t(c,b) \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) \leftarrow & r(c,c) \end{array} \right\}$$

- Grounding aims at reducing the ground instantiation

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllll} r(a,b) \leftarrow & , & & & & \\ r(b,c) \leftarrow & , & & & & \\ t(a,a) \leftarrow & r(a,a), & t(b,a) \leftarrow & r(b,a), & t(c,a) \leftarrow & r(c,a), \\ t(a,b) \leftarrow & , & t(b,b) \leftarrow & r(b,b), & t(c,b) \leftarrow & r(c,b), \\ t(a,c) \leftarrow & r(a,c), & t(b,c) \leftarrow & r(b,c), & t(c,c) \leftarrow & r(c,c) \end{array} \right\}$$

- Grounding aims at reducing the ground instantiation

Safety

- A normal rule is safe, if each of its variables also occurs in some positive body literal
- A normal program is safe, if all of its rules are safe

Example

$d(a)$
$d(c)$
$d(d)$

$p(a,b)$
$p(b,c)$
$p(c,d)$
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$

$q(a)$
$q(b)$
$q(X) \leftarrow {\sim}r(X)$

$r(X) \leftarrow {\sim}q(X), d(X)$
$s(X) \leftarrow {\sim}r(X), p(X,Y), q(Y)$

Example

Safe ?

$d(a)$
$d(c)$
$d(d)$

$p(a, b)$
$p(b, c)$
$p(c, d)$
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$

$q(a)$
$q(b)$
$q(X) \leftarrow \sim r(X)$

$r(X) \leftarrow \sim q(X), d(X)$
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Example

Rule	**Safe ?**
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X,Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X,Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X)$	✘
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X,Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow {\sim}r(X), d(X)$	✔
$r(X) \leftarrow {\sim}q(X), d(X)$	
$s(X) \leftarrow {\sim}r(X), p(X,Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow {\sim}r(X), d(X)$	✔
$r(X) \leftarrow {\sim}q(X), d(X)$	
$s(X) \leftarrow {\sim}r(X), p(X,Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	✔
$r(X) \leftarrow \sim q(X), d(X)$	✔
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	✔
$r(X) \leftarrow \sim q(X), d(X)$	✔
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	✔
$r(X) \leftarrow \sim q(X), d(X)$	✔
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	✔

Stable models of programs with Variables

Let P be a normal logic program with variables

- A set X of (ground) atoms is a stable model of P, if $Cn(ground(P)^X) = X$

Stable models of programs with Variables

Let P be a normal logic program with variables

- A set X of (ground) atoms is a stable model of P,
  if $Cn(ground(P)^X) = X$

[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.

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*

(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.