Answer Set Solving in Practice

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

Potassco Slide Packages are licensed under a Creative Commons Attribution 3.0 Unported License.
Introduction: Overview

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

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

Problem

Modeling

Logic Program

Solving

Solution

Interpreting

Stable Models
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
  \[ a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]
  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

Notation

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

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
  \[ a_0 :\neg a_1, \ldots, \neg a_m, a_{m+1}, \ldots, a_n. \]
  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$.

- Notation
  \[
  h(r) = a_0 \\
  B(r) = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\} \\
  B(r)^+ = \{a_1, \ldots, a_m\} \\
  B(r)^- = \{a_{m+1}, \ldots, a_n\}
  \]

- 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

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

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

- Notation

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

- 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 $A$ of atoms is a finite set of rules.
- A (normal) rule, $r$, is of the form
  \[ a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]
  where $0 \leq m \leq n$ and each $a_i \in A$ is an atom for $0 \leq i \leq n$.
- Notation
  \[
  h(r) = a_0 \\
  B(r) = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \\
  B(r)^+ = \{a_1, \ldots, a_m\} \\
  B(r)^- = \{a_{m+1}, \ldots, a_n\} \\
  A(P) = \bigcup_{r \in P} (\{h(r)\} \cup B(r)^+ \cup B(r)^-) \\
  B(P) = \{B(r) \mid r \in P\} \\
  h(P) = \{h(r) \mid r \in P\} \]
Normal logic programs

- A logic program, $P$, over a set $A$ of atoms is a finite set of rules.
- A (normal) rule, $r$, is of the form
  
  $$a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n$$

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

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

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

Torsten Schaub (KRR@UP)
Normal logic programs

- A logic program, $P$, over a set $A$ of atoms is a finite set of rules
- A (normal) rule, $r$, is of the form
  \[ a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]
  where $0 \leq m \leq n$ and each $a_i \in A$ is an atom for $0 \leq i \leq n$
- Notation
  \[
  h(r) = a_0 \\
  B(r) = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \\
  B(r)^+ = \{a_1, \ldots, a_m\} \\
  B(r)^- = \{a_{m+1}, \ldots, a_n\}
  \]
- A literal is an atom or a negated atom
- A program $P$ is positive if $B(r)^- = \emptyset$ for all $r \in P$
### Syntax

#### Example rules

- $a \leftarrow b, \sim c$
- $a \leftarrow \sim c, b$
- $a \leftarrow$
- $a \leftarrow b$
- $a \leftarrow \sim c$
- $\text{bachelor}(joe) \leftarrow \text{male}(joe), \sim \text{married}(joe)$

#### Example literals

- $a, b, c, \text{bachelor}(joe), \text{male}(joe), \text{married}(joe)$
- $\sim c, \sim \text{married}(joe)$
Example rules

- $a \leftarrow b, \neg c$
- $a \leftarrow \neg c, b$
- $a \leftarrow$
- $a \leftarrow b$
- $a \leftarrow \neg c$
- $\text{bachelor}(\text{joe}) \leftarrow \text{male}(\text{joe}), \neg \text{married}(\text{joe})$

Example literals

- $a, b, c, \text{bachelor}(\text{joe}), \text{male}(\text{joe}), \text{married}(\text{joe})$
- $\neg c, \neg \text{married}(\text{joe})$
Examples

- Example rules
  - $a \leftarrow b, \neg c$
  - $a \leftarrow \neg c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \neg c$
  - $\text{bachelor}(joe) \leftarrow \text{male}(joe), \neg \text{married}(joe)$

- Example literals
  - $a, b, c, \text{bachelor}(joe), \text{male}(joe), \text{married}(joe)$
  - $\neg c, \neg \text{married}(joe)$
Example rules
- $a \leftarrow b, \neg c$
- $a \leftarrow \neg c, b$
- $a \leftarrow$
- $a \leftarrow b$
- $a \leftarrow \neg c$
- $\text{bachelor}(\text{joe}) \leftarrow \text{male}(\text{joe}), \neg \text{married}(\text{joe})$

Example literals
- $a, b, c, \text{bachelor}(\text{joe}), \text{male}(\text{joe}), \text{married}(\text{joe})$
- $\neg c, \neg \text{married}(\text{joe})$
Syntax

Notational convention

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

<table>
<thead>
<tr>
<th></th>
<th>true, false</th>
<th>if</th>
<th>and</th>
<th>or</th>
<th>iff</th>
<th>default notation</th>
<th>classical notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>source code</td>
<td>:- , ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>not</td>
<td>-</td>
</tr>
<tr>
<td>logic program</td>
<td>← , ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~</td>
<td>¬</td>
</tr>
<tr>
<td>formula</td>
<td>⊥, ⊤</td>
<td></td>
<td>→∧</td>
<td>∨</td>
<td>↔</td>
<td>~</td>
<td>¬</td>
</tr>
</tbody>
</table>
Semantics

Problem

Modeling

Logic Program

Solving

Stable Models

Solution

Interpreting

Answer Set Solving in Practice
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 $P$
Semantics

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 $P$
Semantics

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:
    \[ a_0 \lor \neg a_1 \lor \cdots \lor \neg a_m \]

- 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:
    \[ a_0 \lor \neg a_1 \lor \cdots \lor \neg a_m \]
  - 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:
    \[ a_0 \lor \neg a_1 \lor \cdots \lor \neg a_m \]

- 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 \( P \)
Consider the logical formula $\Phi$ and its three (classical) models:

\[
\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}
\]

Formula $\Phi$ has one stable model, often called answer set:

\[
\{p, q\}
\]

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 $P$.
Consider the logical formula $\Phi$ and its three (classical) models:

\[
\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}
\]

Formula $\Phi$ has one stable model, often called answer set:

\[
\{p, q\}
\]

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if
- $X$ is a (classical) model of $P$ and
- all atoms in $X$ are justified by some rule in $P$
Consider the logical formula $\Phi$ and its three (classical) models:

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

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

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 $P$.
Consider the logical formula $\Phi$ and its three (classical) models:

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

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

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 $P$
Consider the logical formula $\Phi$ and its three (classical) models:

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

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

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 $P$
Consider the logical formula \( \Phi \) and its three (classical) models:

\[
\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}
\]

Formula \( \Phi \) has one stable model, often called answer set:

\[
\{p, q\}
\]

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 \( P \)
Consider the logical formula $\Phi$ and its three (classical) models:

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

Formula $\Phi$ has one stable model, often called answer set:

$\{p, q\}$

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 $P$
Consider the logical formula $\Phi$ and its three (classical) models:

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

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

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 $P$
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 $P$”
Semantics

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
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p ) &lt;br&gt;( q \leftarrow )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow p )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q}</td>
<td>( p \leftarrow p )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( p \leftarrow p )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>{q} \quad \times</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>{q} \quad \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>(p \leftarrow p) \qquad q \leftarrow \</td>
<td>{q} \quad \times</td>
</tr>
<tr>
<td>{p}</td>
<td>(p \leftarrow p) \qquad q \leftarrow \</td>
<td>\emptyset \qquad \quad</td>
</tr>
<tr>
<td>{q}</td>
<td>(p \leftarrow p) \qquad q \leftarrow \</td>
<td>{q} \quad \checkmark</td>
</tr>
<tr>
<td>{p, q}</td>
<td>(p \leftarrow p) \qquad q \leftarrow \</td>
<td>\emptyset \quad \quad</td>
</tr>
</tbody>
</table>

Torsten Schaub (KRR@UP)  
Answer Set Solving in Practice  
October 20, 2018 46 / 538
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ } { }</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>{q} \text{X}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>{q}</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>{q} \text{✓}</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{ q } X</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset ) X</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p )</td>
<td>{ q } ✓</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{ q } \times</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p )</td>
<td>\emptyset \times</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p )</td>
<td>{ q } \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>{ q } \xmark</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>( \emptyset ) \xmark</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p ) ( q \leftarrow )</td>
<td>{ q } \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset ) \xmark</td>
</tr>
</tbody>
</table>
### Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ } }</td>
<td>( p \leftarrow p )</td>
<td>( { q } ) ( \times )</td>
</tr>
<tr>
<td>{p }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset ) ( \times )</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p )</td>
<td>( { q } ) ( \checkmark )</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset ) ( \times )</td>
</tr>
</tbody>
</table>
### Example one

\[ P = \{ p \leftarrow p, \; q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p )</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>( { p, q } )</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>( { p } )</td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow )</td>
<td>( { q } )</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{ }</td>
</tr>
<tr>
<td>{ q }</td>
<td>q \leftarrow</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p }</td>
<td>p \leftarrow</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>q \leftarrow</td>
<td>{ }</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>{ p \leftarrow }</td>
<td>{ p, q }</td>
</tr>
<tr>
<td>{ q }</td>
<td>{ q \leftarrow }</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p }</td>
<td>{ p \leftarrow }</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>{ q \leftarrow }</td>
<td>{ }</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>(p \leftarrow )</td>
<td>{p, q} \xmark</td>
</tr>
<tr>
<td>{p}</td>
<td>(p \leftarrow )</td>
<td>{p} \checkmark</td>
</tr>
<tr>
<td>{q}</td>
<td>(q \leftarrow )</td>
<td>{q} \checkmark</td>
</tr>
<tr>
<td>{p, q}</td>
<td>(q \leftarrow )</td>
<td>} }</td>
</tr>
</tbody>
</table>

Torsten Schaub (KRR@UP)
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p, q}</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td>{p, q}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{q}</td>
<td>( q \leftarrow )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( q \leftarrow )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Example two

\( P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \)

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{    | p \space \leftarrow }</td>
<td>{ p, q }</td>
<td>x</td>
</tr>
<tr>
<td>{ p }</td>
<td>p \space \leftarrow</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ q }</td>
<td>q \space \leftarrow</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>q \space \leftarrow</td>
<td>{ q }</td>
</tr>
<tr>
<td>}</td>
<td></td>
<td>{ p }</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p, q} ( \times )</td>
</tr>
<tr>
<td>( p )</td>
<td>( p \leftarrow )</td>
<td>{p} ( \checkmark )</td>
</tr>
<tr>
<td>{q}</td>
<td>( q \leftarrow )</td>
<td>{q} ( \checkmark )</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( q \leftarrow )</td>
<td>( \emptyset ) ( \times )</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>( { p, q } )</td>
</tr>
<tr>
<td></td>
<td>( q \leftarrow )</td>
<td>( \times )</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>( { p } )</td>
</tr>
<tr>
<td></td>
<td>( \text{✓} )</td>
<td></td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow )</td>
<td>( { q } )</td>
</tr>
<tr>
<td></td>
<td>( \text{✓} )</td>
<td></td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td></td>
<td>( \times )</td>
<td></td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>(p \leftarrow)</td>
<td>{p, q}</td>
</tr>
<tr>
<td>{p}</td>
<td>(p \leftarrow)</td>
<td>{p}</td>
</tr>
<tr>
<td>{q}</td>
<td></td>
<td>{q}</td>
</tr>
<tr>
<td>{p, q}</td>
<td></td>
<td>{}</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ p }</td>
<td></td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { }, { } )</td>
<td>( p \leftarrow )</td>
<td>( { p } )</td>
</tr>
<tr>
<td>( { p } )</td>
<td></td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>(p\leftarrow)</td>
<td>{p} (\times)</td>
</tr>
<tr>
<td>{p}</td>
<td>()</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{ p } \xmark</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{p}</td>
<td></td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>{ }</th>
<th>{ p }</th>
<th>{ p }</th>
<th>\emptyset</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>( { p } )</td>
<td>( \emptyset )</td>
<td>X</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>\emptyset</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
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 $X \not\subset Y$. 
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 $X \not\subset Y$. \

Torsten Schaub (KRR@UP)
### Exemplars

<table>
<thead>
<tr>
<th>Logic program</th>
<th>Answer sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>{a}</td>
</tr>
<tr>
<td>a :- b.</td>
<td>{}</td>
</tr>
<tr>
<td>a :- b.</td>
<td>{a,b}</td>
</tr>
<tr>
<td>a :- b. b :- a.</td>
<td>{}</td>
</tr>
<tr>
<td>a :- not c.</td>
<td>{a}</td>
</tr>
<tr>
<td>a :- not c. c.</td>
<td>{c}</td>
</tr>
<tr>
<td>a :- not c. c :- not a.</td>
<td>{a}, {c}</td>
</tr>
<tr>
<td>a :- not a.</td>
<td></td>
</tr>
</tbody>
</table>
Outline

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

- Problem
  - Modeling
  - Logic Program
- Solution
  - Interpreting
  - Stable Models
- Solving
Reasoning modes

- Satisfiability
- Enumeration\(^\dagger\)
- Projection\(^\dagger\)
- Intersection\(^\ddagger\)
- Union\(^\ddagger\)
- Optimization
- and combinations of them

\(^\dagger\) without solution recording
\(^\ddagger\) without solution enumeration
Extended syntax

Problem
  |
  |
  |
  |
  |

Modeling

Logic Program

Solving

Solution

Interpreting

Stable Models
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
  \[ \sim q(X), p(X,C) [C] \]
  \[ #\text{minimize} \{ C : q(X), p(X,C) \} \]
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**
  \[ s(Y) :- r(Y), 2 \text{#sum}\{ X : p(X,Y), q(X) \} 7 \]

- **Optimization**
  \[ :\sim q(X), p(X,C) [C] \#
  \text{minimize} \{ C : q(X), p(X,C) \} \]
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**
  
  \[ \sim q(X), p(X,C) [C] \]
  
  \[ \#\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

\[
p(X) :- q(X) \\
p :- q(X) : r(X) \\
p(X) ; q(X) :- r(X) \\
:- q(X), p(X) \\
2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \\
s(Y) :- r(Y), 2 \sum\{ X : p(X,Y), q(X) \} 7 \\
:\sim q(X), p(X,C) [C] \\
#\text{minimize} \{ C : q(X), p(X,C) \}
\]
Language constructs

- Variables
  \[ p(X) :- q(X) \]
  \[ p :- q(X) : r(X) \]
  \[ p(X) ; q(X) :- r(X) \]
  \[ :- q(X), p(X) \]
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]
- Conditional literals
  \[ s(Y) :- r(Y), 2 \#\text{sum}\{ X : p(X,Y), q(X) \} 7 \]
- Disjunction
  \[ :- q(X), p(X) \]
- Integrity constraints
  \[ :- q(X), p(X) \]
- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]
- Aggregates
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]
- Optimization
  \[ :- q(X), p(X,C) [C] \]
  \[ \#\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- **Variables**
  
- **Conditional literals**
  
- **Disjunction**
  
- **Integrity constraints**
  
- **Choice**
  
- **Aggregates**

**Optimization**

- **Weak constraints**

---

\[ \text{Language constructs} \]

\[ p(X) :- q(X) \]

\[ p :- q(X) : r(X) \]

\[ p(X) ; q(X) :- r(X) \]

\[ :- q(X), p(X) \]

\[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

\[ s(Y) :- r(Y), 2 \#\text{sum}\{ X : p(X,Y), q(X) \} 7 \]

\[ \sim q(X), p(X,C) [C] \]

\#\text{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
  \[ s(Y) :- r(Y), 2 \#\text{sum}\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  - Weak constraints
    \[ :- q(X), p(X,C) [C] \]
  - Statements
    \[ \#\text{minimize} \{ C : q(X), p(X,C) \} \]
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
  \[ \sim q(X), p(X,C) [C] \]
  \[ \#minimize \{ C : q(X), p(X,C) \} \]
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**
    
    \[ #\text{minimize} \{ C : q(X), p(X,C) \} \]
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
  \[ s(Y) :- r(Y), 2 \#\text{sum}\{ X : p(X,Y), q(X) \} 7 \]

- Multi-objective optimization
  - Weak constraints
    \[ \sim q(X), p(X,C) [C@42] \]
  - Statements
    \[ \#\text{minimize} \{ C@42 : q(X), p(X,C) \} \]
1 Syntax
2 Semantics
3 Examples
4 Reasoning
5 Language
6 Variables
Example

d(a)
d(c)
d(d)
p(a, b)
p(b, c)
p(c, d)
p(X, Z) ← p(X, Y), p(Y, Z)
q(a)
q(b)
q(X) ← ∼r(X), d(X)
r(X) ← ∼q(X), d(X)
s(X) ← ∼r(X), p(X, Y), q(Y)
Example

d(a)
d(c)
d(d)
p(a, b)
p(b, c)
p(c, d)
p(X, Z) ← p(X, Y), p(Y, Z)
q(a)
q(b)
q(X) ← ¬r(X), d(X)
r(X) ← ¬q(X), d(X)
s(X) ← ¬r(X), p(X, Y), q(Y)
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 $\mathcal{T}$:

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

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

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

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

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ 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 $T$ be a set of (variable-free) terms
- Let $A$ be a set of (variable-free) atoms constructible from $T$
- 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 $T$:

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

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

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

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

  where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

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

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

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

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

An example

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

\[ T = \{ a, b, c \} \]

\[ A = \{ r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \]

\[ \quad t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \} \]

\[ \text{ground}(P) = \left\{ \begin{array}{l}
  r(a, b) \leftarrow, \\
  r(b, c) \leftarrow, \\
  t(a, a) \leftarrow r(a, a), \ t(b, a) \leftarrow r(b, a), \ t(c, a) \leftarrow r(c, a), \\
  t(a, b) \leftarrow r(a, b), \ t(b, b) \leftarrow r(b, b), \ t(c, b) \leftarrow r(c, b), \\
  t(a, c) \leftarrow r(a, c), \ t(b, c) \leftarrow r(b, c), \ t(c, c) \leftarrow r(c, c) \end{array} \right\} \]

Grounding aims at reducing the ground instantiation
An example

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

\[ T = \{ a, b, c \} \]

\[ A = \{ r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \]

\[ t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \} \]

\[
ground(P) = \begin{array}{l}
  r(a, b) \leftarrow , \\
  r(b, c) \leftarrow , \\
  t(a, a) \leftarrow r(a, a), \ t(b, a) \leftarrow r(b, a), \ t(c, a) \leftarrow r(c, a), \\
  t(a, b) \leftarrow r(a, b), \ t(b, b) \leftarrow r(b, b), \ t(c, b) \leftarrow r(c, b), \\
  t(a, c) \leftarrow r(a, c), \ t(b, c) \leftarrow r(b, c), \ t(c, c) \leftarrow r(c, c) \\
\end{array} \]

Grounding aims at reducing the ground instantiation
An example

\[ P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \} \]
\[ T = \{a, b, c\} \]
\[ A = \{r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c)\} \]

\[ \text{ground}(P) = \begin{cases}  
  r(a, b) \leftarrow, \\
  r(b, c) \leftarrow, \\
  t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), \\
  t(a, b) \leftarrow r(a, b), t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), \\
  t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c) 
\end{cases} \]

- Grounding aims at reducing the ground instantiation
An example

\[ P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \} \]
\[ T = \{a, b, c\} \]
\[ A = \{ r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \\
  t(a), t(a, b), t(a, c), t(b), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \} \]

\[ \text{ground}(P) = \left\{ \begin{array}{l}
  r(a, b) \leftarrow , \\
  r(b, c) \leftarrow , \\
  t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), \\
  t(a, b) \leftarrow , t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), \\
  t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c) \\
\end{array} \right\} \]

- Grounding aims at reducing the ground instantiation
An example

\[ P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \} \]
\[ T = \{ a, b, c \} \]
\[ A = \{ r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \} \]
\[ \text{ground}(P) = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), t(a, b) \leftarrow, t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c) \} \]

- **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) ← p(X, Y), p(Y, Z)
q(a)
q(b)
q(X) ← ∼r(X)

r(X) ← ∼q(X), d(X)
s(X) ← ∼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) ← p(X, Y), p(Y, Z)
q(a)
q(b)
q(X) ← ¬r(X)
r(X) ← ¬q(X), d(X)
s(X) ← ¬r(X), p(X, Y), q(Y)
Example

Safe ?

\[d(a)\] ✓
\[d(c)\] ✓
\[d(d)\] ✓
\[p(a, b)\] ✓
\[p(b, c)\] ✓
\[p(c, d)\] ✓
\[p(X, Z) \leftarrow p(X, Y), p(Y, Z)\] ✓
\[q(a)\] ✓
\[q(b)\] ✓
\[q(X) \leftarrow \neg r(X)\] ✓
\[r(X) \leftarrow \neg q(X), d(X)\] ✓
\[s(X) \leftarrow \neg r(X), p(X, Y), q(Y)\]
Example

Variables

\[ d(a) \]
\[ d(c) \]
\[ d(d) \]
\[ p(a, b) \]
\[ p(b, c) \]
\[ p(c, d) \]
\[ p(X, Z) \leftarrow p(X, Y), p(Y, Z) \]
\[ q(a) \]
\[ q(b) \]
\[ q(X) \leftarrow \neg r(X) \]
\[ r(X) \leftarrow \neg q(X), d(X) \]
\[ s(X) \leftarrow \neg r(X), p(X, Y), q(Y) \]
Example

Safe ?

\[
\begin{align*}
\text{d}(a) & \quad \checkmark \\
\text{d}(c) & \quad \checkmark \\
\text{d}(d) & \quad \checkmark \\
p(a, b) & \quad \checkmark \\
p(b, c) & \quad \checkmark \\
p(c, d) & \quad \checkmark \\
p(X, Z) & \leftarrow p(X, Y), p(Y, Z) & \quad \checkmark \\
q(a) & \quad \checkmark \\
q(b) & \quad \checkmark \\
q(X) & \leftarrow \sim r(X) & \quad \checkmark \\
r(X) & \leftarrow \sim q(X), d(X) \\
s(X) & \leftarrow \sim r(X), p(X, Y), q(Y)
\end{align*}
\]
Example

Safe?

d(a)

✓

d(c)

✓

d(d)

✓
p(a, b)

✓
p(b, c)

✓
p(c, d)

✓
p(X, Z) ← p(X, Y), p(Y, Z)

✓
q(a)

✓
q(b)

✓
q(X) ← ¬r(X)

✗
r(X) ← ¬q(X), d(X)

s(X) ← ¬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

\begin{align*}
  d(a) & \quad \checkmark \\
  d(c) & \quad \checkmark \\
  d(d) & \quad \checkmark \\
  p(a, b) & \quad \checkmark \\
  p(b, c) & \quad \checkmark \\
  p(c, d) & \quad \checkmark \\
  p(X, Z) & \leftarrow p(X, Y), p(Y, Z) \quad \checkmark \\
  q(a) & \quad \checkmark \\
  q(b) & \quad \checkmark \\
  q(X) & \leftarrow \sim r(X), d(X) \quad \checkmark \\
  r(X) & \leftarrow \sim q(X), d(X) \\
  s(X) & \leftarrow \sim r(X), p(X, Y), q(Y)
\end{align*}
Variables

Example

Safe ?

\[
\begin{align*}
  d(a) & \quad \checkmark \\
  d(c) & \quad \checkmark \\
  d(d) & \quad \checkmark \\
  p(a, b) & \quad \checkmark \\
  p(b, c) & \quad \checkmark \\
  p(c, d) & \quad \checkmark \\
  p(X, Z) & \leftarrow p(X, Y), p(Y, Z) \quad \checkmark \\
  q(a) & \quad \checkmark \\
  q(b) & \quad \checkmark \\
  q(X) & \leftarrow \neg r(X), d(X) \quad \checkmark \\
  r(X) & \leftarrow \neg q(X), d(X) \quad \checkmark \\
  s(X) & \leftarrow \neg r(X), p(X, Y), q(Y) \quad \checkmark 
\end{align*}
\]
Variables

Example

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

Safe ?

✓
✓
✓
✓
✓
✓
✓
✓
✓
✓
Example

Safe?

\[
\begin{align*}
\text{d}(a) & \quad \checkmark \\
\text{d}(c) & \quad \checkmark \\
\text{d}(d) & \quad \checkmark \\
\text{p}(a, b) & \quad \checkmark \\
\text{p}(b, c) & \quad \checkmark \\
\text{p}(c, d) & \quad \checkmark \\
\text{p}(X, Z) & \leftarrow \text{p}(X, Y), \text{p}(Y, Z) \quad \checkmark \\
\text{q}(a) & \quad \checkmark \\
\text{q}(b) & \quad \checkmark \\
\text{q}(X) & \leftarrow \neg \text{r}(X), \text{d}(X) \quad \checkmark \\
\text{r}(X) & \leftarrow \neg \text{q}(X), \text{d}(X) \quad \checkmark \\
\text{s}(X) & \leftarrow \neg \text{r}(X), \text{p}(X, Y), \text{q}(Y) \quad \checkmark 
\end{align*}
\]
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 $\text{Cn}(\text{ground}(P)^X) = X$
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 $\text{Cn}(\text{ground}(P)^X) = X$
Variables

Computing answer sets using program completion.

Knowledge Representation, Reasoning and Declarative Problem Solving.


Logic programming and knowledge representation.

Towards an integration of answer set and constraint solving.

Adaptive restart strategies for conflict driven SAT solvers.

PicoSAT essentials.

Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications.
Answer set programming at a glance.

Answer set optimization.
In G. Gottlob and T. Walsh, editors, Proceedings of the Eighteenth
International Joint Conference on Artificial Intelligence (IJCAI'03),

Negation as failure.
In H. Gallaire and J. Minker, editors, Logic and Data Bases, pages

Handbook of Tableau Methods.


Conflict-driven disjunctive answer set solving.


On the computational cost of disjunctive logic programming: Propositional case.

*Answer Set Programming: A Primer.*

[22] F. Fages.
Consistency of Clark’s completion and the existence of stable models.

Answer sets for propositional theories.


*Potassco User Guide.*

*A user’s guide to gringo, clasp, clingo, and iclingo.*

Engineering an incremental ASP solver.

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


In Baral et al. [3], pages 260–265.

In Baral et al. [3], pages 136–148.

In Veloso [74], pages 386–392.


Variables


[41] M. Gelfond.

**Answer sets.**


**Logic programming and knowledge representation — the A-Prolog perspective.**


**The stable model semantics for logic programming.**
Variables


*Logic programs with classical negation.*


*Answer set programming based on propositional satisfiability.*


*Zum intuitionistischen Aussagenkalkül.*


Die formalen Regeln der intuitionistischen Logik.


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

*Graphs and colorings for answer set programming*.  

A model-theoretic counterpart of loop formulas.

The DLV system for knowledge representation and reasoning. 

[54] V. Lifschitz. 
Answer set programming and plan generation. 

Introduction to answer set programming. 

[56] V. Lifschitz and A. Razborov. 
Why are there so many loop formulas?
[57] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.

[58] V. Marek and M. Truszczyński.
*Nonmonotonic logic: context-dependent reasoning.*

Stable models and an alternative logic programming paradigm.

Conflict-driven clause learning SAT solvers.
In Biere et al. [8], chapter 4, pages 131–153.

GRASP: A search algorithm for propositional satisfiability.

Integrating answer set reasoning with constraint solving techniques.

Integrating answer set programming and constraint logic programming.

A SAT solver primer.


Variables


Efficient algorithms for clause-learning SAT solvers.

Extending and implementing the stable model semantics.

[71] T. Son and E. Pontelli.
Planning with preferences using logic programming.

Lparse 1.0 user’s manual, 2001.

The well-founded semantics for general logic programs.  

[74] M. Veloso, editor.  

Efficient conflict driven learning in a Boolean satisfiability solver.  