Answer Set Programming: Basics

Sebastian Rudolph

Computational Logic Group
Technische Universität Dresden

December 15, 2013

Slides based on a lecture by Martin Gebser and Torsten Schaub.
Potassco Slide Packages are licensed under a Creative Commons Attribution 3.0 Unported License.
Answer Set Programming – Basics:
Overview

1. Motivation: ASP vs. Prolog and SAT
2. ASP Syntax
3. Semantics
4. Examples
5. Variables
6. Reasoning modes
Outline

1 Motivation: ASP vs. Prolog and SAT

2 ASP Syntax

3 Semantics

4 Examples

5 Variables

6 Reasoning modes
KR’s shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation
Motivation: ASP vs. Prolog and SAT

KR’s shift of paradigm

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation
LP-style playing with blocks

Prolog program

on(a,b). on(b,c).

above(X,Y) :- on(X,Y). above(X,Y) :- on(X,Z), above(Z,Y).

Prolog queries

?- above(a,c). true.  ?- above(c,a). no.
LP-style playing with blocks

Prolog program

\[
\text{on}(a,b). \text{on}(b,c).
\]

\[
\text{above}(X,Y) :- \text{on}(X,Y). \quad \text{above}(X,Y) :- \text{on}(X,Z), \text{above}(Z,Y).
\]

Prolog queries

\[
?- \text{above}(a,c). \text{ true.} \\
?- \text{above}(c,a). \text{ no.}
\]
LP-style playing with blocks

Prolog program

\[
\text{on}(a,b). \ \text{on}(b,c).
\]

\[
\text{above}(X,Y) :- \ \text{on}(X,Y). \ \text{above}(X,Y) :- \ \text{on}(X,Z), \ \text{above}(Z,Y).
\]

Prolog queries

?- \text{above}(a,c). \ \text{true}. \ \text{?- above}(c,a). \ \text{no}.
LP-style playing with blocks

Prolog program

on(a,b). on(b,c).

above(X,Y) :- on(X,Y). above(X,Y) :- on(X,Z), above(Z,Y).

Prolog queries (testing entailment)

?- above(a,c). true.  ?- above(c,a). no.
LP-style playing with blocks

Shuffled Prolog program

\[
on(a,b). \ on(b,c).
\]
\[
\text{above}(X,Y) \ :- \ \text{above}(X,Z), \ \text{on}(Z,Y). \ \text{above}(X,Y) \ :- \ \text{on}(X,Y).
\]

Prolog queries

?- \text{above}(a,c). \ \text{Fatal Error: local stack overflow.}
LP-style playing with blocks

Shuffled Prolog program

on(a,b). on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y). above(X,Y) :- on(X,Y).

Prolog queries

?- above(a,c). Fatal Error: local stack overflow.
LP-style playing with blocks

Shuffled Prolog program

\begin{verbatim}
on(a,b). on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y). above(X,Y) :- on(X,Y).
\end{verbatim}

Prolog queries (answered via fixed execution)

?- above(a,c). Fatal Error: local stack overflow.
KR’s shift of paradigm

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
KR’s shift of paradigm

**Theorem Proving based approach** (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a *derivation* of a query

**Model Generation based approach** (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a *model* of the representation
SAT-style playing with blocks

Formula

\[ \begin{align*}
on(a, b) & \\
\land on(b, c) & \\
\land (on(X, Y) \rightarrow above(X, Y)) & \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y)) & \\
\end{align*} \]

Herbrand model

\[ \{ \begin{array}{c}
on(a, b), \quad on(b, c), \quad on(a, c), \quad on(b, b), \\
above(a, b), \quad above(b, c), \quad above(a, c), \quad above(b, b), \quad above(c, b) \\
\end{array} \} \]
SAT-style playing with blocks

Formula

\[\begin{align*}
on(a, b) & \\
\land & on(b, c) \\
\land & (on(X, Y) \rightarrow above(X, Y)) \\
\land & (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\end{align*}\]

Herbrand model

\[
\{ \begin{array}{c}
on(a, b), \quad on(b, c), \quad on(a, c), \quad on(b, b), \\
above(a, b), \quad above(b, c), \quad above(a, c), \quad above(b, b), \quad above(c, b) \end{array} \}
\]
SAT-style playing with blocks

Formula

\[
\begin{align*}
on(a, b) \\
\land \on(b, c) \\
\land (\on(X, Y) \rightarrow \above(X, Y)) \\
\land (\on(X, Z) \land \above(Z, Y) \rightarrow \above(X, Y))
\end{align*}
\]

Herbrand model (among 426!)

\[
\{ \on(a, b), \on(b, c), \on(a, c), \on(b, b), \\
\above(a, b), \above(b, c), \above(a, c), \above(b, b), \above(c, b) \}
\]
SAT-style playing with blocks

Formula

\[
\begin{align*}
on(a, b) \\
\land on(b, c) \\
\land (on(X, Y) \rightarrow above(X, Y)) \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\end{align*}
\]

Herbrand model (among 426!)

\[
\{\begin{array}{l}
on(a, b), \quad on(b, c), \quad on(a, c), \quad on(b, b), \\
above(a, b), \quad above(b, c), \quad above(a, c), \quad above(b, b), \quad above(c, b)
\end{array}\}
\]
SAT-style playing with blocks

Formula

\[\begin{align*}
on(a, b) \\
\land on(b, c) \\
\land (on(X, Y) \rightarrow above(X, Y)) \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\end{align*}\]

Herbrand model (among 426!)

\[\{ \text{on}(a, b), \text{on}(b, c), \text{on}(a, c), \text{on}(b, b), \text{above}(a, b), \text{above}(b, c), \text{above}(a, c), \text{above}(b, b), \text{above}(c, b) \} \]
Motivation: ASP vs. Prolog and SAT

KR’s shift of paradigm

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
KR’s shift of paradigm

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation

Answer Set Programming (ASP)
ASP-style playing with blocks

Logic program

\[
on(a,b). \ on(b,c).
on(X,Y) \ :- \ on(X,Y). \ above(X,Y) \ :- \ on(X,Z), \ above(Z,Y).
\]

Stable Herbrand model

\[
\{ \ on(a,b), \ on(b,c), \ above(b,c), \ above(a,b), \ above(a,c) \}
\]
ASP-style playing with blocks

Logic program

\[
on(a, b). \ on(b, c).
\]

\[
\text{above}(X, Y) :- \ on(X, Y). \ \text{above}(X, Y) :- \ on(X, Z), \ \text{above}(Z, Y).
\]

Stable Herbrand model

\[
\{ \ on(a, b), \ on(b, c), \ \text{above}(b, c), \ \text{above}(a, b), \ \text{above}(a, c) \ \}
\]
Motivation: ASP vs. Prolog and SAT

ASP-style playing with blocks

Logic program

\[
on(a, b). \ on(b, c).

above(X,Y) : - on(X,Y). \ above(X,Y) : - on(X,Z), above(Z,Y).
\]

Stable Herbrand model (and no others)

\[
\{ \ on(a, b), \ on(b, c), \ above(b, c), \ above(a, b), \ above(a, c) \}
\]
ASP-style playing with blocks

Logic program

\[
on(a,b). \ on(b,c).
\]

\[
\text{above}(X,Y) :- \text{above}(Z,Y), \ on(X,Z). \ \text{above}(X,Y) :- \ on(X,Y).
\]

Stable Herbrand model (and no others)

\[
\{ \ on(a,b), \ on(b,c), \ \text{above}(b,c), \ \text{above}(a,b), \ \text{above}(a,c) \ \}
\]
## ASP versus LP

<table>
<thead>
<tr>
<th>ASP</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model generation</td>
<td>Query orientation</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>Top-down</td>
</tr>
<tr>
<td>Modeling language</td>
<td>Programming language</td>
</tr>
</tbody>
</table>

**Rule-based format**

- Instantiation
- Flat terms

- (Turing +) \( NP^{NP} \)

- Unification
- Nested terms

- Turing
### ASP versus SAT

<table>
<thead>
<tr>
<th></th>
<th>ASP</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model generation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottom-up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constructive Logic</td>
<td>Classical Logic</td>
<td></td>
</tr>
<tr>
<td>Closed (and open) world reasoning</td>
<td>Open world reasoning</td>
<td></td>
</tr>
<tr>
<td>Modeling language</td>
<td>_____________</td>
<td></td>
</tr>
<tr>
<td>Complex reasoning modes</td>
<td>Satisfiability testing</td>
<td></td>
</tr>
<tr>
<td>Satisfiability</td>
<td></td>
<td>Satisfiability</td>
</tr>
<tr>
<td>Enumeration/Projection</td>
<td>_____________</td>
<td></td>
</tr>
<tr>
<td>Intersection/Union</td>
<td>_____________</td>
<td></td>
</tr>
<tr>
<td>Optimization</td>
<td>_____________</td>
<td></td>
</tr>
<tr>
<td>((\text{Turing} , +) \ NP^{(NP)})</td>
<td></td>
<td>(NP)</td>
</tr>
</tbody>
</table>
What is ASP good for?

- Combinatorial search problems in the realm of $P$, $NP$, and $NP^{NP}$ (some with substantial amount of data), like
  - Automated Planning
  - Code Optimization
  - Composition of Renaissance Music
  - Database Integration
  - Decision Support for NASA shuttle controllers
  - Model Checking
  - Product Configuration
  - Robotics
  - Systems Biology
  - System Synthesis
  - (industrial) Team-building
  - and many many more
What is ASP good for?

- Combinatorial search problems in the realm of $P$, $NP$, and $NP^{NP}$ (some with substantial amount of data), like
  - Automated Planning
  - Code Optimization
  - Composition of Renaissance Music
  - Database Integration
  - Decision Support for NASA shuttle controllers
  - Model Checking
  - Product Configuration
  - Robotics
  - Systems Biology
  - System Synthesis
  - (industrial) Team-building
  - and many many more
Outline

1 Motivation: ASP vs. Prolog and SAT
2 ASP Syntax
3 Semantics
4 Examples
5 Variables
6 Reasoning modes
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

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

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

- Notation:

  - $\text{head}(r) = a_0$
  - $\text{body}(r) = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\}$
  - $\text{body}(r)^+ = \{a_1, \ldots, a_m\}$
  - $\text{body}(r)^- = \{a_{m+1}, \ldots, a_n\}$
  - $\text{atom}(P) = \bigcup_{r \in P} (\{\text{head}(r)\} \cup \text{body}(r)^+ \cup \text{body}(r)^-)$
  - $\text{body}(P) = \{\text{body}(r) \mid r \in P\}$

- A program $P$ is positive if $\text{body}(r)^- = \emptyset$ 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

$$\text{head}(r) = a_0$$
$$\text{body}(r) = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$$
$$\text{body}(r)^+ = \{a_1, \ldots, a_m\}$$
$$\text{body}(r)^- = \{a_{m+1}, \ldots, a_n\}$$
$$\text{atom}(P) = \bigcup_{r \in P} (\{\text{head}(r)\} \cup \text{body}(r)^+ \cup \text{body}(r)^-)$$
$$\text{body}(P) = \{\text{body}(r) \mid r \in P\}$$

- A program $P$ is positive if $\text{body}(r)^- = \emptyset$ 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

\[
\begin{align*}
\text{head}(r) &= a_0 \\
\text{body}(r) &= \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \\
\text{body}(r)^+ &= \{ a_1, \ldots, a_m \} \\
\text{body}(r)^- &= \{ a_{m+1}, \ldots, a_n \} \\
\text{atom}(P) &= \bigcup_{r \in P} (\{ \text{head}(r) \} \cup \text{body}(r)^+ \cup \text{body}(r)^-) \\
\text{body}(P) &= \{ \text{body}(r) \mid r \in P \}
\end{align*}
\]

- A program \( P \) is positive if \( \text{body}(r)^- = \emptyset \) for all \( r \in P \)
Rough 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</th>
<th>classical</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>
Outline

1 Motivation: ASP vs. Prolog and SAT
2 ASP Syntax
3 Semantics
4 Examples
5 Variables
6 Reasoning modes
Formal Definition

Stable models of positive programs

- A set of atoms $X$ is closed under a positive program $P$ iff for any $r \in P$, $\text{head}(r) \in X$ whenever $\text{body}(r)^+ \subseteq X$
- $X$ 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 $P$
A set of atoms $X$ is closed under a positive program $P$ iff for any $r \in P$, $\text{head}(r) \in X$ whenever $\text{body}(r)^+ \subseteq X$.

- $X$ 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 $r \in P$, $\text{head}(r) \in X$ whenever $\text{body}(r)^+ \subseteq X$
  - $X$ 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 $r \in P$, $\text{head}(r) \in X$ whenever $\text{body}(r)^+ \subseteq X$
  - $X$ 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 $P$
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 $P$ (rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932)).
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 $P$ (rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932)).
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 \( P \)

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))
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 $P$

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))
Semantics

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\}

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

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))
Consider the logical formula $\Phi$ and its three (classical) models:

$\{p, q\}$, $\{q, r\}$, 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 $P$

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))
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 $P$

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))
Consider the logical formula $\Phi$ and its three (classical) models:

$\{p, q\}$, $\{q, r\}$, 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 $P$

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))
Formal Definition

Stable model of normal programs

The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

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

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

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

Note Every atom in $X$ is justified by an “applying rule from $P$”
Semantics

Formal Definition

Stable model of normal programs

- The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

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

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

- Note $Cn(P^X)$ is the $\subseteq$-smallest (classical) model of $P^X$
- Note Every atom in $X$ is justified by an “applying rule from $P$”
Formal Definition

Stable model of normal programs

- The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

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

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

- Note $Cn(P^X)$ is the $\subseteq$-smallest (classical) model of $P^X$
- Note Every atom in $X$ is justified by an “applying rule from $P$”
A closer look at $P^X$

In other words, given a set $X$ of atoms from $P$, $P^X$ is obtained from $P$ by deleting

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

Note Only negative body literals are evaluated wrt $X$
Semantics

A closer look at $P^X$

- In other words, 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 wrt $X$
Motivation: ASP vs. Prolog and SAT

ASP Syntax

Semantics

Examples

Variables

Reasoning modes
A first example

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

<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 )</td>
<td>( { q } )</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td>( X )</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>
### A first example

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

<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) X</td>
</tr>
</tbody>
</table>
A first example

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

<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 )</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>
A first example

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

<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></td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td></td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>{q}</td>
<td>( p \leftarrow p )</td>
<td>{q}</td>
</tr>
<tr>
<td></td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>{p, q}</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
### A first example

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

<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 } \xmark</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p )</td>
<td>{ } \xmark</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>{ } \xmark</td>
</tr>
</tbody>
</table>
### A first example

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

<table>
<thead>
<tr>
<th>( \chi )</th>
<th>( P^{\chi} )</th>
<th>( \text{Cn}(P^{\chi}) )</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>
### A first example

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

<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) \quad 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) \quad q \leftarrow \</td>
<td>({q})</td>
</tr>
<tr>
<td>({p, q})</td>
<td>(p \leftarrow p)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Examples

A second example

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

<table>
<thead>
<tr>
<th>X</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>{ p }</td>
<td>{ p, q } X</td>
</tr>
<tr>
<td>{ p }</td>
<td>{ p }</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ q }</td>
<td>{ q }</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>{ p, q }</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
A second example

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

<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></td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>{ p }  \checkmark</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow )</td>
<td>{ q }  \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td></td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
A second example

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

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow ) ( \mathbf{p, q} )</td>
<td>( \times )</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td>( \mathbf{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>
A second example

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

<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>{ ( 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>{ } )</td>
</tr>
</tbody>
</table>
A second example

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

\[
\begin{array}{c|c|c}
   \ X & P^X & Cn(P^X) \\
\hline
   \{ \} & p \leftarrow & \{ p, q \} \quad \times \\
   & q \leftarrow & \\
\hline
   \{ p \} & p \leftarrow & \{ p \} \quad \checkmark \\
\hline
   \{ q \} & & \{ q \} \quad \checkmark \\
   & q \leftarrow & \\
\hline
   \{ p, q \} & & \emptyset \\
\end{array}
\]
A second example

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

<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>{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>\emptypset</td>
</tr>
</tbody>
</table>
A third example

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

<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}</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
A third example

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

\[
\begin{array}{c|c|c}
X & P^X & Cn(P^X) \\
\{\} & p \leftarrow & \{p\} \quad \times \\
\{p\} & & \emptyset
\end{array}
\]
### A third example

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

<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 } )</td>
</tr>
<tr>
<td>( { p } )</td>
<td>( p \leftarrow )</td>
<td>( { p } )</td>
</tr>
</tbody>
</table>
A third example

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

<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 }</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>( \emptyset )</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 $X$ is a model of $P$ (seen as a formula)
- If $X$ and $Y$ are stable models of a normal program $P$, then $X \not\subseteq Y$
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 $X$ is a model of $P$ (seen as a formula)
- If $X$ and $Y$ are stable models of a normal program $P$, then $X \not\subset Y$
Variables

Outline

1 Motivation: ASP vs. Prolog and SAT
2 ASP Syntax
3 Semantics
4 Examples
5 Variables
6 Reasoning modes
Programs with Variables

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 constructable from $\mathcal{T}$

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 $r$; $\theta$ is a (ground) substitution

Ground Instantiation of $P$: $ground(P) = \bigcup_{r \in P} ground(r)$
Programs with Variables

Let $P$ be a logic program

- Let $T$ be a set of variable-free terms (also called Herbrand universe)
- Let $A$ be a set of (variable-free) atoms constructable from $T$ (also called alphabet or Herbrand base)

- 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 $P$: $\text{ground}(P) = \bigcup_{r \in P} \text{ground}(r)$
Programs with Variables

Let $P$ be a logic program

- Let $T$ be a set of (variable-free) terms
- Let $A$ be a set of (variable-free) atoms constructable from $T$

**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 $P$:**

$$\text{ground}(P) = \bigcup_{r \in P} \text{ground}(r)$$
Variables

Programs with Variables

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 constructable from $\mathcal{T}$

- 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 $r$; $\theta$ is a (ground) substitution

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

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

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

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

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

Intelligent Grounding aims at reducing the ground instantiation.
An example

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

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

Intelligent Grounding aims at reducing the ground instantiation
An example

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

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

\[ \mathcal{A} = \begin{cases} r(a, a), \; r(a, b), \; r(a, c), \; r(b, a), \; r(b, b), \; r(b, c), \; r(c, a), \; r(c, b), \; r(c, c), \\ t(a, a), \; t(a, b), \; t(a, c), \; t(b, a), \; t(b, b), \; t(b, c), \; t(c, a), \; t(c, b), \; t(c, c) \end{cases} \]

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

- Intelligent Grounding aims at reducing the ground instantiation
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 $Cn(\text{ground}(P)^X) = X$
Reasoning modes

Outline

1 Motivation: ASP vs. Prolog and SAT
2 ASP Syntax
3 Semantics
4 Examples
5 Variables
6 Reasoning modes
Reasoning Modes

- Satisfiability
- Enumeration\(^\dagger\)
- Projection\(^\dagger\)
- Intersection\(^\dagger\)
- Union\(^\dagger\)
- Optimization

and combinations of them

\(^\dagger\) without solution recording
\(^\dagger\) without solution enumeration