DEDUCTION SYSTEMS

Answer Set Programming: Basics

Markus Krötzsch
Chair for Knowledge-Based Systems

Slides by Sebastian Rudolph, and based on a lecture by Martin Gebser and Torsten Schaub (CC-By 3.0)

TU Dresden, 18 June 2018
ASP Basics: Overview

1. ASP in a Nutshell
2. ASP Syntax
3. Semantics
4. Examples
5. Completion
6. Loops and Loop Formulas
Answer Set Programming
in a Nutshell

• ASP is an approach to declarative problem solving, combining
  – a rich yet simple modeling language
  – with high-performance solving capacities

• ASP has its roots in
  – (deductive) databases
  – logic programming (with negation)
  – (logic-based) knowledge representation and (nonmonotonic) reasoning
  – constraint solving (in particular, SATisfiability testing)

• ASP allows for solving all search problems in \( NP \) in a uniform way

• ASP is supported by several fast solvers, such as clasp, DLV, and smodels
Answer Set Programming
in a Nutshell

- ASP is an approach to **declarative problem solving**, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
Answer Set Programming
in a Nutshell

- ASP is an approach to **declarative problem solving**, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
Answer Set Programming
in a Nutshell

• ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

• ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)

• ASP allows for solving all search problems in \( NP \) (and \( NP^{NP} \)) in a uniform way
Answer Set Programming
in a Nutshell

• ASP is an approach to declarative problem solving, combining
  – a rich yet simple modeling language
  – with high-performance solving capacities

• ASP has its roots in
  – (deductive) databases
  – logic programming (with negation)
  – (logic-based) knowledge representation and (nonmonotonic) reasoning
  – constraint solving (in particular, SATisfiability testing)

• ASP allows for solving all search problems in \( NP \) (and \( NP^{NP} \))
in a uniform way

• ASP is supported by several fast solvers, such as clasp, DLV, and smodels
Outline

1. ASP in a Nutshell
2. ASP Syntax
3. Semantics
4. Examples
5. Completion
6. Loops and Loop Formulas
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 $0 \leq i \leq n$
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

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

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

A program $P$ is positive if $\text{body}(r)^- = \emptyset$ for all $r \in P$
Formal Definition
Stable models of positive programs
Formal Definition

Stable models of positive programs

- A set of atoms \( X \) is closed under a positive program \( P \) iff for any \( r \in P \), \( head(r) \in X \) whenever \( body(r)^+ \subseteq X \)
  - Then \( X \) (seen as an interpretation) corresponds to a model of \( P \) (seen as a propositional logic formula)
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 $\text{body}(r)^+ \subseteq X$
  - Then $X$ (seen as an interpretation) corresponds to a model of $P$ (seen as a propositional logic 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 $P$
A set of atoms \( X \) is closed under a positive program \( P \) iff for any \( r \in P \), \( \text{head}(r) \in X \) whenever \( \text{body}(r)^+ \subseteq X \)

- Then \( X \) (seen as an interpretation) corresponds to a model of \( P \) (seen as a propositional logic 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 \( P \)

The set \( Cn(P) \) of atoms is the stable model of a positive program \( P \)
Formal Definition
Stable models of normal programs

- The (Gelfond-Lifschitz) reduct $P^X$ of a program $P$ relative to a set $X$ of atoms is defined by

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

Stable models of normal programs

- The (Gelfond-Lifschitz) 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 $Cn(P^X) = X$
Formal Definition

Stable models of normal programs

- The (Gelfond-Lifschitz) 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 $\text{Cn}(P^X) = X$

- Note: $\text{Cn}(P^X)$ 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 $\sim a$ 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
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 $X$
A first example

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

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

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

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

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftrightarrow p )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftrightarrow p )</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>{q}</td>
<td>( p \leftrightarrow p )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p,q}</td>
<td>( p \leftrightarrow 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 )</td>
<td>{q} <strong>\xmark</strong></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>$\text{Cn}(P^X)$</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>${q}$</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 ) \qquad q \leftarrow )</td>
<td>( {q} \quad \times )</td>
</tr>
<tr>
<td>{p }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset \quad \times )</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 )</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, \ 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>
A second example

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

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

<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></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>$\emptyset$</td>
</tr>
</tbody>
</table>
A second example

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

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

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

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

\[ P = \{ p \leftarrow \sim p \} \]
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></td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
A third example

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

<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>{ }</td>
<td>( )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>{ p }</td>
<td>( )</td>
<td>( \times )</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}  X</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow )</td>
<td>( \emptyset )  X</td>
</tr>
</tbody>
</table>

TU Dresden, 18 June 2018

Deduction Systems
Some properties

- A logic program may have zero, one, or multiple stable models!
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 propositional logic formula with negation instead of $\sim$)
- If $X$ and $Y$ are stable models of a normal program $P$, then $X \not\subset Y$
Motivation

- Question: Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?
Motivation

• Question: Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

• Observation: Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom.
Motivation

• Question: Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

• Observation: Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom

• Idea: The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart
Let $P$ be a normal logic program

- The (Clark) completion $CF(P)$ of $P$ is defined as follows

$$CF(P) = \left\{ a \leftrightarrow \bigvee_{r \in P, head(r)=a} BF(body(r)) \mid a \in \text{atom}(P) \right\}$$

where

$$BF(body(r)) = \bigwedge_{a \in body(r)} a \land \bigwedge_{a \in body(r)} \neg a$$
An example

\[ P = \begin{cases} 
  a & \leftarrow \top \\
  b & \leftarrow \neg a \\
  c & \leftarrow a, \neg d \\
  d & \leftarrow \neg c, \neg e \\
  e & \leftarrow b, \neg f \\
  e & \leftarrow e 
\end{cases} \]
An example

\[ P = \begin{cases} 
  a \leftarrow & a \\
  b \leftarrow & \neg a \\
  c \leftarrow & a, \neg d \\
  d \leftarrow & \neg c, \neg e \\
  e \leftarrow & b, \neg f \\
  e \leftarrow & e 
\end{cases} \]

\[ CF(P) = \begin{cases} 
  a \leftrightarrow & \top \\
  b \leftrightarrow & \neg a \\
  c \leftrightarrow & a \land \neg d \\
  d \leftrightarrow & \neg c \land \neg e \\
  e \leftrightarrow & (b \land \neg f) \lor e \\
  f \leftrightarrow & \bot 
\end{cases} \]
A closer look

- $CF(P)$ is logically equivalent to $\hat{CF}(P) \cup \hat{F}(P)$, where

\[
\hat{CF}(P) = \left\{ a \leftarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in \text{atom}(P) \right\}
\]

\[
\hat{F}(P) = \left\{ a \rightarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in \text{atom}(P) \right\}
\]

$body_P(a) = \{body(r) \mid r \in P \text{ and head}(r) = a\}$
A closer look

- \( CF(P) \) is logically equivalent to \( \overline{CF}(P) \cup \overline{CF}(P) \), where

\[
\begin{align*}
\overline{CF}(P) &= \{ a \leftarrow \bigvee_{B \in body_p(a)} BF(B) \mid a \in \text{atom}(P) \} \\
\overline{CF}(P) &= \{ a \rightarrow \bigvee_{B \in body_p(a)} BF(B) \mid a \in \text{atom}(P) \} \\
body_p(a) &= \{ \text{body}(r) \mid r \in P \text{ and head}(r) = a \}
\end{align*}
\]

- \( \overline{CF}(P) \) characterizes the classical models of \( P \)
- \( \overline{CF}(P) \) completes \( P \) by adding necessary conditions for all atoms
A closer look

\[
P = \{ \begin{align*}
  a & \leftarrow \\
  b & \leftarrow \neg a \\
  c & \leftarrow a, \neg d \\
  d & \leftarrow \neg c, \neg e \\
  e & \leftarrow b, \neg f \\
  e & \leftarrow e
\end{align*} \}
\]
A closer look

\[
P = \begin{cases} 
  a &\leftarrow a \\
  b &\leftarrow \neg a \\
  c &\leftarrow a, \neg d \\
  d &\leftarrow \neg c, \neg e \\
  e &\leftarrow b, \neg f \\
  e &\leftarrow e 
\end{cases} \quad \overline{CF}(P) = \begin{cases} 
  a &\leftarrow \top \\
  b &\leftarrow \neg a \\
  c &\leftarrow a \land \neg d \\
  d &\leftarrow \neg c \land \neg e \\
  e &\leftarrow (b \land \neg f) \lor e \\
  f &\leftarrow \bot 
\end{cases}
\]
A closer look

\[
\hat{CF}(P) = \begin{cases} 
  a & \leftarrow \top \\
  b & \leftarrow \neg a \\
  c & \leftarrow a \land \neg d \\
  d & \leftarrow \neg c \land \neg e \\
  e & \leftarrow (b \land \neg f) \lor e \\
  f & \leftarrow \bot 
\end{cases}
\]
A closer look

$$\overleftarrow{CF}(P) = \begin{cases} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \land \neg d \\ d \leftarrow \neg c \land \neg e \\ e \leftarrow (b \land \neg f) \lor e \\ f \leftarrow \bot \end{cases} \quad \begin{cases} a \rightarrow \top \\ b \rightarrow \neg a \\ c \rightarrow a \land \neg d \\ d \rightarrow \neg c \land \neg e \\ e \rightarrow (b \land \neg f) \lor e \\ f \rightarrow \bot \end{cases} = \overrightarrow{CF}(P)$$
A closer look

\[ \overrightarrow{\text{CF}}(P) = \begin{cases} 
    a & \leftrightarrow \top \\
    b & \leftrightarrow \neg a \\
    c & \leftrightarrow a \land \neg d \\
    d & \leftrightarrow \neg c \land \neg e \\
    e & \leftrightarrow (b \land \neg f) \lor e \\
    f & \leftrightarrow \bot 
\end{cases} \]

\[ \overleftarrow{\text{CF}}(P) = \begin{cases} 
    a & \rightarrow \top \\
    b & \rightarrow \neg a \\
    c & \rightarrow a \land \neg d \\
    d & \rightarrow \neg c \land \neg e \\
    e & \rightarrow (b \land \neg f) \lor e \\
    f & \rightarrow \bot 
\end{cases} \]

\[ \text{CF}(P) = \begin{cases} 
    a & \leftrightarrow \top \\
    b & \leftrightarrow \neg a \\
    c & \leftrightarrow a \land \neg d \\
    d & \leftrightarrow \neg c \land \neg e \\
    e & \leftrightarrow (b \land \neg f) \lor e \\
    f & \leftrightarrow \bot 
\end{cases} \]
A closer look

$$\overrightarrow{CF}(P) = \begin{cases} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \land \neg d \\ d \leftarrow \neg c \land \neg e \\ e \leftarrow (b \land \neg f) \lor e \\ f \leftarrow \bot \end{cases} \quad \begin{cases} a \rightarrow \top \\ b \rightarrow \neg a \\ c \rightarrow a \land \neg d \\ d \rightarrow \neg c \land \neg e \\ e \rightarrow (b \land \neg f) \lor e \\ f \rightarrow \bot \end{cases} = \overrightarrow{CF}(P)$$

$$CF(P) = \begin{cases} a \leftrightarrow \top \\ b \leftrightarrow \neg a \\ c \leftrightarrow a \land \neg d \\ d \leftrightarrow \neg c \land \neg e \\ e \leftrightarrow (b \land \neg f) \lor e \\ f \leftrightarrow \bot \end{cases} \equiv \overrightarrow{CF}(P) \cup \overrightarrow{CF}(P)$$
Supported models

- Every stable model of $P$ is a model of $CF(P)$,
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa
- Models of $CF(P)$ are called the supported models of $P$
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa
- Models of $CF(P)$ are called the supported models of $P$
- In other words, every stable model of $P$ is a supported model of $P$
- By definition, every supported model of $P$ is also a model of $P$
An example

\[ P = \{ \begin{array}{ccc} a \leftarrow & c \leftarrow a, \neg d & e \leftarrow b, \neg f \\ b \leftarrow \neg a & d \leftarrow \neg c, \neg e & e \leftarrow e \end{array} \} \]
An example

\[ P = \{ a \leftarrow c \leftarrow a, \sim d \leftarrow b, \sim e \leftarrow e, e \leftarrow b, \sim f \} \]

- \( P \) has 21 models, including \( \{a, c\} \), \( \{a, d\} \), but also \( \{a, b, c, d, e, f\} \)
An example

\[ P = \left\{ \begin{array}{ccc}
    a & \leftarrow & c \leftarrow a, \sim d \\
    b & \leftarrow & \sim a \\
    d & \leftarrow & \sim c, \sim e \\
    e & \leftarrow & e \\
\end{array} \right\} \]

- \( P \) has 21 models, including \( \{a, c\}, \{a, d\} \), but also \( \{a, b, c, d, e, f\} \)
- \( P \) has 3 supported models, namely \( \{a, c\}, \{a, d\} \), and \( \{a, c, e\} \)
An example

\[ P = \left\{ \begin{array}{ccc} a & \leftarrow & c \leftarrow a, \sim d \\ b & \leftarrow & \sim a \\ c & \leftarrow & a, \sim d \\ d & \leftarrow & \sim c, \sim e \\ e & \leftarrow & b, \sim f \\ e & \leftarrow & e \end{array} \right\} \]

- \( P \) has 21 models, including \{a, c\}, \{a, d\}, but also \{a, b, c, d, e, f\}
- \( P \) has 3 supported models, namely \{a, c\}, \{a, d\}, and \{a, c, e\}
- \( P \) has 2 stable models, namely \{a, c\} and \{a, d\}
Outline

1. ASP in a Nutshell
2. ASP Syntax
3. Semantics
4. Examples
5. Completion
6. Loops and Loop Formulas
Motivation

- Question: Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?
Motivation

- Question: Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- Observation: Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.
Motivation

- **Question:** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation:** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

- **Idea:** Add formulas prohibiting circular support of sets of atoms.
Motivation

• Question: Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

• Observation: Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

• Idea: Add formulas prohibiting circular support of sets of atoms.

• Note: Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program's positive atom dependency graph.
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$. A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$. That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$. We denote the set of all loops of $P$ by $\text{loop}(P)$. Note: A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 
Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$.

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$. 

TU Dresden, 18 June 2018 Deduction Systems
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$.

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$.

That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$. 
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$
  - if it induces a non-trivial strongly connected subgraph of $G(P)$
  - That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$

- We denote the set of all loops of $P$ by $\text{loop}(P)$
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$
  - That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$

- We denote the set of all loops of $P$ by $\text{loop}(P)$

- Note: A program $P$ is tight iff $\text{loop}(P) = \emptyset$
Example

\[ P = \{ a \leftarrow c, \sim d, e \leftarrow b, \sim f \} \]
Example

- \( P = \{ \begin{align*}
  a & \leftarrow c \\
  b & \leftarrow \neg a \\
  c & \leftarrow a, \neg d \\
  d & \leftarrow \neg c, \neg e \\
  e & \leftarrow b, \neg f
\end{align*} \} \)

- \( \text{loop}(P) = \{ e \} \)
Another example

\[ P = \{ a \leftarrow \neg b, \quad c \leftarrow a, b, \quad d \leftarrow a, \quad e \leftarrow \neg a, \neg b \\
   b \leftarrow \neg a, \quad c \leftarrow d, \quad d \leftarrow b, c \} \]
Another example

\[ P = \{ a \leftarrow \neg b \quad c \leftarrow a, b \quad d \leftarrow a \quad e \leftarrow \neg a, \neg b \quad b \leftarrow \neg a \quad c \leftarrow d \quad d \leftarrow b, c \} \]

\[ \text{loop}(P) = \{ \{c, d\} \} \]
Yet another example

\[ P = \left\{ \begin{array}{l} a \leftarrow \sim b \quad c \leftarrow a \quad d \leftarrow b, c \quad e \leftarrow b, \sim a \\ b \leftarrow \sim a \quad c \leftarrow b, d \quad d \leftarrow e \quad e \leftarrow c, d \end{array} \right\} \]
Yet another example

\[ P = \{ a \leftarrow \sim b, \quad c \leftarrow a, \quad d \leftarrow b, c, \quad e \leftarrow b, \sim a, \quad b \leftarrow \sim a, \quad c \leftarrow b, d, \quad d \leftarrow e, \quad e \leftarrow c, d \} \]
Yet another example

- $P = \{ a \leftarrow \neg b \quad c \leftarrow a \quad d \leftarrow b, c \quad e \leftarrow b, \neg a \\
\quad b \leftarrow \neg a \quad c \leftarrow b, d \quad d \leftarrow e \quad e \leftarrow c, d \}$

- $\text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \}$
Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as

$$\text{ES}_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}$$
Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as

$$ES_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = \text{body}(ES_P(L))$
Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as
  \[ ES_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \} \]

- Define the external bodies of $L$ in $P$ as $EB_P(L) = \text{body}(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is
  \[
  LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))
  \equiv (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)
  \]
Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as

  $$ES_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = \text{body}(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is

  $$LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))$$
  $$\equiv (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)$$

- Note: The loop formula of $L$ enforces all atoms in $L$ to be \textit{false} whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) \mid L \in \text{loop}(P) \}$
Example

- \( P = \{ a \leftarrow c \leftarrow a, \sim d \leftarrow c, \sim e \rightarrow e, b \rightarrow a, \sim f \rightarrow b \} \)

- \( \text{loop}(P) = \{ \{ e \} \} \)
Example

- $P = \{ a \leftarrow c \leftarrow a, \sim d \leftarrow \sim e \leftarrow \sim a, \sim c, \sim e \leftarrow e \leftarrow e \}$

- $\text{loop}(P) = \{ \{ e \} \}$

- $\text{LF}(P) = \{ e \rightarrow b \land \sim f \}$
Another example

- $P = \{ a \leftarrow \neg b, c \leftarrow a, b, d \leftarrow a, e \leftarrow \neg a, \neg b \}
\quad b \leftarrow \neg a, c \leftarrow d, d \leftarrow b, c \}$

- $\text{loop}(P) = \{\{c, d\}\}$
Another example

\[ P = \left\{ \begin{array}{l}
   a \leftarrow \neg b \\
   c \leftarrow a, b \\
   d \leftarrow a \\
   e \leftarrow \neg a, \neg b \\
   b \leftarrow \neg a \\
   c \leftarrow d \\
   d \leftarrow b, c
\end{array} \right\} \]

• \(\text{loop}(P) = \{c, d\}\)
• \(\text{LF}(P) = \{c \lor d \rightarrow (a \land b) \lor a\}\)
Yet another example

- $P = \left\{ \begin{array}{l} a \leftarrow \sim b \\
             c \leftarrow a \\
             d \leftarrow b, c \\
             e \leftarrow b, \sim a \\
             b \leftarrow \sim a \\
             c \leftarrow b, d \\
             d \leftarrow e \\
             e \leftarrow c, d \end{array} \right\}$

- $\text{loop}(P) = \left\{ \{c, d\}, \{d, e\}, \{c, d, e\} \right\}$
Yet another example

• $P = \{ \begin{aligned} a &\leftarrow \neg b \\
c &\leftarrow a \\
d &\leftarrow b, c \\
e &\leftarrow b, \neg a \\
b &\leftarrow \neg a \\
c &\leftarrow b, d \\
d &\leftarrow e \\
e &\leftarrow c, d \end{aligned} \}$

• $\text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \}$

• $\text{LF}(P) = \{ \begin{aligned} c \lor d &\rightarrow a \lor e \\
d \lor e &\rightarrow (b \land c) \lor (b \land \neg a) \\
c \lor d \lor e &\rightarrow a \lor (b \land \neg a) \end{aligned} \}$
Yet another example

- \( P = \{ \)
  - \( a \leftarrow \sim b \)
  - \( c \leftarrow a \)
  - \( d \leftarrow b, c \)
  - \( e \leftarrow b, \sim a \)
  - \( b \leftarrow \sim a \)
  - \( c \leftarrow b, d \)
  - \( d \leftarrow e \)
  - \( e \leftarrow c, d \)
\( \}

- \( \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \)

- \( L_\mathcal{F}(P) = \{ \)
  - \( c \land d \rightarrow a \lor e \)
  - \( d \land e \rightarrow (b \land c) \lor (b \land \sim a) \)
  - \( c \land d \lor e \rightarrow a \lor (b \land \sim a) \)
\( \} \)
Yet another example

- $P = \{ a \leftarrow \sim b \quad c \leftarrow a \quad d \leftarrow b, c \quad e \leftarrow b, \sim a \quad b \leftarrow \sim a \quad c \leftarrow b, d \quad d \leftarrow e \quad e \leftarrow c, d \}$

- $\text{loop}(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$

- $\text{LF}(P) = \{\begin{align*}
c \lor d & \rightarrow a \lor e \\
d \lor e & \rightarrow (b \land c) \lor (b \land \sim a) \\
c \lor d \lor e & \rightarrow a \lor (b \land \sim a)
\end{align*}\}$
Yet another example

- $P = \{ a \leftarrow \sim b \quad c \leftarrow a \quad d \leftarrow b, c \quad e \leftarrow b, \sim a \\
 b \leftarrow \sim a \quad c \leftarrow b, d \quad d \leftarrow e \quad e \leftarrow c, d \}$

- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$

- $LF(P) = \{c \lor d \rightarrow a \lor e \\
d \lor e \rightarrow (b \land c) \lor (b \land \sim a) \\
c \lor d \lor e \rightarrow a \lor (b \land \sim a)\}$
The following result is due to Fangzhen Lin and Yuting Zhao [2004], who used it to implement ASP using SAT solvers:

**Theorem**

Let $P$ be a normal logic program and $X \subseteq \text{atom}(P)$

Then, $X$ is a stable model of $P$ iff $X \models CF(P) \cup LF(P)$

Note: There can be exponentially many loops in the worst case, so the reduction may incur a substantial blow-up. However, practical problems often include only a rather small number of loops.
Summary

Answer Set Programming is non-monotonic logic programming with a stable-model semantics.

Main reasoning task: computing (all, zero or more) stable models (a.k.a. answer sets).

Reduction to SAT is possible by
- Clark completion (supported models) +
- Loop formulae (answer sets)