Answer Set Programming: Basics

Sebastian Rudolph

Computational Logic Group
Technische Universität Dresden

January 6, 2015

Slides based on a lecture by Martin Gebser and Torsten Schaub.

Answer Set Programming – Basics: Overview

Outline

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

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

```
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

```
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

```
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).

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

```
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

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

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

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

$$\begin{array}{ll} & on(a,b) \\ \wedge & on(b,c) \\ \wedge & (on(X,Y) \rightarrow above(X,Y)) \\ \wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y)) \end{array}$$

Herbrand model

$$\left\{ \begin{array}{lllll} on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\ above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b) \end{array} \right\}$$

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model

$$
\left\{
\begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array}
\right\}
$$

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a, b) \\
\wedge & on(b, c) \\
\wedge & (on(X, Y) \rightarrow above(X, Y)) \\
\wedge & (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y))
\end{array}
$$

Herbrand model (among 426!)

$$
\left\{ \begin{array}{ccccc}
on(a, b), & on(b, c), & on(a, c), & on(b, b), & \\
above(a, b), & above(b, c), & above(a, c), & above(b, b), & above(c, b)
\end{array} \right\}
$$

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model (among 426!)

$$
\left\{ \begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array} \right\}
$$

SAT-style playing with blocks

Formula

$$
\begin{array}{ll}
 & on(a,b) \\
\wedge & on(b,c) \\
\wedge & (on(X,Y) \rightarrow above(X,Y)) \\
\wedge & (on(X,Z) \wedge above(Z,Y) \rightarrow above(X,Y))
\end{array}
$$

Herbrand model (among 426!)

$$
\left\{ \begin{array}{ccccc}
on(a,b), & on(b,c), & on(a,c), & on(b,b), & \\
above(a,b), & above(b,c), & above(a,c), & above(b,b), & above(c,b)
\end{array} \right\}
$$

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).

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

Stable Herbrand model

$\{ \text{on}(a, b),\ \text{on}(b, c),\ \text{above}(b, c),\ \text{above}(a, b),\ \text{above}(a, c) \}$

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

$\{\ \mathtt{on}(\mathtt{a},\mathtt{b}),\ \mathtt{on}(\mathtt{b},\mathtt{c}),\ \mathtt{above}(\mathtt{b},\mathtt{c}),\ \mathtt{above}(\mathtt{a},\mathtt{b}),\ \mathtt{above}(\mathtt{a},\mathtt{c})\ \}$

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)

$\{\ \mathtt{on}(\mathtt{a},\mathtt{b}),\ \mathtt{on}(\mathtt{b},\mathtt{c}),\ \mathtt{above}(\mathtt{b},\mathtt{c}),\ \mathtt{above}(\mathtt{a},\mathtt{b}),\ \mathtt{above}(\mathtt{a},\mathtt{c})\ \}$

ASP-style playing with blocks

Logic program

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

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

Stable Herbrand model (and no others)

$$\{ \texttt{on}(\texttt{a},\texttt{b}),\ \texttt{on}(\texttt{b},\texttt{c}),\ \texttt{above}(\texttt{b},\texttt{c}),\ \texttt{above}(\texttt{a},\texttt{b}),\ \texttt{above}(\texttt{a},\texttt{c}) \}$$

ASP versus LP

ASP	Prolog
Model generation	Query orientation
Bottom-up	Top-down
Modeling language	Programming language
Rule-based format	
Instantiation Flat terms	Unification Nested terms
(Turing +) $NP(^{NP})$	Turing

ASP versus SAT

ASP	SAT
Model generation	
Bottom-up	
Constructive Logic	Classical Logic
Closed (and open) world reasoning	Open world reasoning
Modeling language	—
Complex reasoning modes	Satisfiability testing
Satisfiability	Satisfiability
Enumeration/Projection	—
Intersection/Union	—
Optimization	—
(Turing +) $NP^{(NP)}$	NP

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

Normal logic programs

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

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

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

- Notation

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

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

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

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

$$\begin{aligned}
head(r) &= a_0 \\
body(r) &= \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \\
body(r)^+ &= \{a_1, \ldots, a_m\} \\
body(r)^- &= \{a_{m+1}, \ldots, a_n\} \\
atom(P) &= \textstyle\bigcup_{r \in P} \left(\{head(r)\} \cup body(r)^+ \cup body(r)^-\right) \\
body(P) &= \{body(r) \mid r \in P\}
\end{aligned}$$

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

Rough notational convention

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

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

Outline

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 $body(r)^+ \subseteq X$
  - 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 $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
  - 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 $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
  - 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 $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
  - 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

Basic idea

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

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

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

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))

Basic idea

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

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

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

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

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

$$\{p,q\}$$

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))

Basic idea

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

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

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

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

Basic idea

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

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

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

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

$$\{p, q\}$$

P_Φ $\begin{array}{|lll|} q & \leftarrow & \\ p & \leftarrow & q, \neg r \end{array}$

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

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

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

Basic idea

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

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

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

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

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

$$\{p, q\}$$

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))

Basic idea

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

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

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

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

$$\{p, q\}$$

P_Φ

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

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

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

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

Basic idea

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

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

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

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

$$\{p, q\}$$

P_Φ

q	$\leftarrow$	
p	$\leftarrow$	$q, \ \sim r$

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))

Basic idea

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

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

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

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

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

$\{p, q\}$

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))

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

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

- A set X of atoms is a stable model of a 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

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

- A set X of atoms is a stable model of a 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

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

- A set X of atoms is a stable model of a 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 $\sim a$ 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 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

Outline

A first example

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

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

A first example

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

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

A first example

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

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

A first example

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

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

A first example

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

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

A first example

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

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

A first example

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

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

A second example

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

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

A second example

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

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

A second example

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

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

A second example

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

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

A second example

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

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

A second example

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

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

A third example

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

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

A third example

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

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

A third example

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

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

A third example

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

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

Some properties

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

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 $X \not\subset Y$

Outline

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 $\mathcal{T}$:

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

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

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

Programs with Variables

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 constructable from $\mathcal{T}$ (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 $\mathcal{T}$:

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

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

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

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 $\mathcal{T}$:

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

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

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

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 $\mathcal{T}$:

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

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

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

An example

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

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

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

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

- Intelligent Grounding aims at reducing the ground instantiation

An example

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

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

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

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

- Intelligent Grounding aims at reducing the ground instantiation

An example

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

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

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

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

- Intelligent Grounding aims at reducing the ground instantiation

Stable models of programs with Variables

Let P be a normal logic program with variables

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

Stable models of programs with Variables

Let P be a normal logic program with variables

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

Outline

Reasoning Modes

- Satisfiability
- Enumeration†
- Projection†
- Intersection‡
- Union‡
- Optimization

- and combinations of them

† without solution recording

‡ without solution enumeration