Answer Set Programming

CMPT 411/721

(based on slides by Torsten Schaub)
Introduction:
Model-Based Problem Solving
Goal: Declarative problem solving

In declarative problem solving:

- Instead of asking: “How can the problem be solved?”
- Ask: “How can the problem be described?”

Then use a domain-independent solver to compute a solution
Goal: Declarative problem solving

In declarative problem solving:
- Instead of asking: "How can the problem be solved?"
- Ask: "How can the problem be described?"

Then use a domain-independent solver to compute a solution.

General KR Methodology:

```
<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling</td>
<td>Interpretation</td>
</tr>
<tr>
<td>Representation</td>
<td>Output</td>
</tr>
<tr>
<td>Computation</td>
<td></td>
</tr>
</tbody>
</table>
```
Answer set programming (ASP)

- Has its roots in
  - Knowledge representation and reasoning
    - In particular nonmonotonic reasoning
  - Deductive databases
  - Constraint solving (in particular, SAT solving)
  - Logic programming (with negation)

- Allows for solving all search problems within NP (and \(NP^{NP}\)) (over finite domains).

- Allows for using powerful off-the-shelf systems
  (nowadays capable of dealing with millions of variables)
## Example: 3–colourability of graphs

### C(I)
- vertex(1) ← edge(1,2)
- vertex(2) ← edge(2,3)
- vertex(3) ← edge(3,1)

### C(P)
- coloured(V,r) ← not coloured(V,b), not coloured(V,g), vertex(V)
- coloured(V,b) ← not coloured(V,r), not coloured(V,g), vertex(V)
- coloured(V,g) ← not coloured(V,r), not coloured(V,b), vertex(V)
  - ← edge(V,U), coloured(V,C), coloured(U,C), colour(C)

### Answer set
{ coloured(1,r), coloured(2,b), coloured(3,g), ... }

---

**Goal:** Find a *minimal* set of literals that *satisfies* the rules.

Such a set of literals is called an *answer set*.
Model-Based Problem Solving

Compare:

I Inference-based approach

1. Provide a specification of the problem.
2. A solution is given by a derivation of an appropriate query.
   - E.g. resolution in logic, top-down rule-based reasoning, Prolog
Model-Based Problem Solving

Compare:

I Inference-based approach

1. Provide a specification of the problem.
2. A solution is given by a derivation of an appropriate query.
   - E.g. resolution in logic, top-down rule-based reasoning, Prolog

II Model-based approach

1. Provide a specification of the problem.
2. A solution is given by a model of the specification.
   - E.g. ASP, also SAT
Model-Based Problem Solving

Compare:

I Inference-based approach

1. Provide a specification of the problem.
2. A solution is given by a *derivation* of an appropriate query.
   - E.g. resolution in logic, top-down rule-based reasoning, Prolog

II Model-based approach

1. Provide a specification of the problem.
2. A solution is given by a *model* of the specification.
   - E.g. ASP, also SAT

Key Idea: Rules represent *constraints* on the problem.
Applications of ASP

- Combinatorial search problems:
  - auctions, bio-informatics, computer-aided verification, configuration, constraint satisfaction, diagnosis, information integration, planning and scheduling, security analysis, semantic web, wire-routing, zoology and linguistics, ...

- ASP has also been used as a target language into which a high level language can be compiled.
  - E.g.: Action language $\Rightarrow$ ASP
Introduction to ASP
ASP: Idea

- A (normal) rule, \( r \), is of the form

\[
A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n,
\]

- \textit{not} can be read as negation as failure.
- Variables are treated as standing for all possible instances.
ASP: Idea

- A (normal) rule, $r$, is of the form
  
  \[ A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n, \]

- *not* can be read as negation as failure.
- Variables are treated as standing for all possible instances.

- Want to determine answer sets of a set of rules, or program.
- An answer set is a minimal set of atoms satisfying the rules.
  
  - i.e. for rule $r$ above, if $X$ is an answer set, then if $A_1, \ldots, A_m$ are in $X$ and no $A_{m+1}, \ldots, A_n$ is in $X$ then $A_0$ is in $X$. 

E.g. \{ $a \leftarrow b$, not c., $b$ \} has answer set \{ $a$, $b$ \}. 

\{ $a \leftarrow \text{not } b$.., $b \leftarrow \text{not } a$. \} has answer sets \{ $a$ \} and \{ $b$ \}. 

ASP: Idea

- A (normal) rule, $r$, is of the form

  $$A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n,$$

- $\text{not}$ can be read as negation as failure.
- Variables are treated as standing for all possible instances.

- Want to determine answer sets of a set of rules, or program.

- An answer set is a minimal set of atoms satisfying the rules.

  - I.e. for rule $r$ above, if $X$ is an answer set, then if $A_1, \ldots, A_m$ are in $X$ and no $A_{m+1}, \ldots, A_n$ is in $X$ then $A_0$ is in $X$.

- E.g. $\{a \leftarrow b, \text{not } c, , b.\}$ has answer set $\{a, b\}$. 
ASP: Idea

- A (normal) rule, \( r \), is of the form
  \[
  A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n,
  \]
  - \text{not} can be read as negation as failure.
  - Variables are treated as standing for all possible instances.

- Want to determine answer sets of a set of rules, or program.

- An answer set is a minimal set of atoms satisfying the rules.
  - I.e. for rule \( r \) above, if \( X \) is an answer set, then if \( A_1, \ldots, A_m \) are in \( X \) and no \( A_{m+1}, \ldots, A_n \) is in \( X \) then \( A_0 \) is in \( X \).

- E.g. \{\( a \leftarrow b \), \text{not } c., \ b.\} \) has answer set \{\( a, b \).\}
  \{\( a \leftarrow \text{not } b., \ b \leftarrow \text{not } a.\)\} \) has answer sets \{\( a \)\} and \{\( b \)\}. 

Normal logic programs

- A (normal) rule, $r$, is of the form
  $$A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n,$$
  where $n, m \geq 0$, and each $A_i$ ($0 \leq i \leq n$) is an atom.
- A (normal) logic program is a finite set of rules.
- Notation
  - $\text{head}(r) = A_0$
  - $\text{body}(r) = \{A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n\}$
  - $\text{body}^+(r) = \{A_1, \ldots, A_m\}$
  - $\text{body}^-(r) = \{A_{m+1}, \ldots, A_n\}$
- A program is called positive if $\text{body}^-(r) = \emptyset$ for all its rules.
- $= \text{set of Horn clauses}$
(Rough) Notational Conventions

The following notation is used interchangeably in order to stress a particular view:

<table>
<thead>
<tr>
<th></th>
<th>if</th>
<th>and</th>
<th>or</th>
<th>negation as failure</th>
<th>classical negation</th>
</tr>
</thead>
<tbody>
<tr>
<td>logic program</td>
<td>←</td>
<td>,</td>
<td>;</td>
<td>not/∼</td>
<td>¬</td>
</tr>
<tr>
<td>formula</td>
<td>→</td>
<td>∧</td>
<td>∨</td>
<td></td>
<td>¬</td>
</tr>
<tr>
<td>source code</td>
<td>:-</td>
<td>,</td>
<td></td>
<td>not</td>
<td>−</td>
</tr>
</tbody>
</table>
Answer Set: Intuitions

• An **answer set** for a program $P$ is a **minimal** set of atoms $X$ such that, for every rule:

$$A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n,$$

if

$$\{A_1, \ldots, A_m\} \subseteq X$$

and

$$\{A_{m+1}, \ldots, A_n\} \cap X = \emptyset$$

then

$$A_0 \in X.$$ 

• This is a **nonconstructive** specification.

• Think of rules as specifying **constraints** on an answer set.
Answer Set: Formal Definition

Positive programs

- A set of atoms $X$ is **closed under** a positive program $\Pi$ iff for any $r \in \Pi$, $\text{head}(r) \in X$ whenever $\text{body}^+(r) \subseteq X$.
  - $X$ corresponds to a model of $\Pi$ (seen as a formula).
Answer Set: Formal Definition

Positive programs

- A set of atoms $X$ is **closed under** a positive program $\Pi$ iff for any $r \in \Pi$, $\text{head}(r) \in X$ whenever $\text{body}^+(r) \subseteq X$.
- $X$ corresponds to a model of $\Pi$ (seen as a formula).
- The **smallest** set of atoms which is closed under a positive program $\Pi$ is denoted by $Cn(\Pi)$.
  - $Cn(\Pi)$ corresponds to the $\subseteq$-smallest model of $\Pi$
  - This is just the set of consequences obtained by forward chaining.
Answer Set: Formal Definition

Positive programs

- A set of atoms $X$ is **closed under** a positive program $\Pi$ iff for any $r \in \Pi$, $\text{head}(r) \in X$ whenever $\text{body}^+(r) \subseteq X$.
  - $X$ corresponds to a model of $\Pi$ (seen as a formula).
- The **smallest** set of atoms which is closed under a positive program $\Pi$ is denoted by $Cn(\Pi)$.
  - $Cn(\Pi)$ corresponds to the $\subseteq$-smallest model of $\Pi$
  - This is just the set of consequences obtained by forward chaining.
- The set $Cn(\Pi)$ is an **answer set** of a **positive** program $\Pi$.

**Example**

\[
\{p \leftarrow, \quad q \leftarrow p, \quad r \leftarrow p, q, \quad t \leftarrow s\}
\]

has answer set $\{p, q, r\}$. 

Some “logical” remarks

Recall:

- Positive rules are also called **definite clauses**.
  - Definite clauses are disjunctions with **exactly one** positive atom:

\[ A_0 \lor \neg A_1 \lor \cdots \lor \neg A_m \]

- A set of definite clauses has a (unique) smallest model (where “smallest” is in terms of atoms true in the model).
Some “logical” remarks

Recall:

• Positive rules are also called definite clauses.
  • Definite clauses are disjunctions with exactly one positive atom:
    \[ A_0 \lor \neg A_1 \lor \cdots \lor \neg A_m \]
  
    • A set of definite clauses has a (unique) smallest model (where “smallest” is in terms of atoms true in the model).

• Horn clauses are clauses with at most one positive atom.
  • Every definite clause is a Horn clause but not vice versa.
  • A set of Horn clauses has a smallest model or none.
Another “logical” remark

Answer sets versus (minimal) models

- Program \( \{ a \leftarrow \text{not } b \} \) has answer set \( \{ a \} \).
- Clause \( a \lor b \) (which is equivalent in classical logic to \( a \leftarrow \neg b \))
  - has models \( \{ a \} \), \( \{ b \} \), and \( \{ a, b \} \),
  - among which \( \{ a \} \) and \( \{ b \} \) are minimal.

The negation-as-failure operator \textit{not} makes a difference!
Consider the logical formula $\Phi$ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}.$$
Answer sets: Basic idea

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

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

The corresponding logic program has one answer set:

$$\{p, q\}$$
Answer sets: Basic idea

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

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

The corresponding logic program has one answer set:

$$\{p, q\}$$

Roughly, a set of atoms $X$ is an answer set of a logic program $\Pi$ if

- $X$ is a (classical) model of $\Pi$ and
- all atoms in $X$ are justified by some rule in $\Pi$
Answer set: Formal Definition

Normal programs

The reduct, \( \Pi_X \), of a program \( \Pi \) relative to a set \( X \) of atoms is defined by

\[
\Pi_X = \{ \text{head}(r) \leftarrow \text{body} + (r) \mid r \in \Pi \text{ and } \text{body}(r) \cap X = \emptyset \}.
\]

Think of \( X \) as being a "guess" of an answer set.

The reduct "compiles out" negation as failure, given \( X \).

A set \( X \) of atoms is an answer set of a program \( \Pi \) if \( \text{Cn}(\Pi_X) = X \).

Recall: \( \text{Cn}(\Pi_X) \) is the ⊆–smallest (classical) model of \( \Pi_X \).

Intuition: Every atom in \( X \) is justified by an "applying rule" from \( \Pi \).
The reduct, $\Pi^X$, of a program $\Pi$ relative to a set $X$ of atoms is defined by

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

- Think of $X$ as being a “guess” of an answer set.
- The reduct “compiles out” negation as failure, given $X$. 

Recall: $C_n(\Pi^X)$ is the $\subseteq$–smallest (classical) model of $\Pi^X$. Intuition: Every atom in $X$ is justified by an “applying rule” from $\Pi$. 

Answer set: Formal Definition 

Normal programs
The reduct, $\Pi^X$, of a program $\Pi$ relative to a set $X$ of atoms is defined by

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

- Think of $X$ as being a “guess” of an answer set.
- The reduct “compiles out” negation as failure, given $X$.

A set $X$ of atoms is an answer set of a program $\Pi$ if $\text{Cn}(\Pi^X) = X$. 

Recall: $\text{Cn}(\Pi^X)$ is the ⊆–smallest (classical) model of $\Pi^X$. 

Intuition: Every atom in $X$ is justified by an “applying rule” from $\Pi$. 

Answer set: Formal Definition

Normal programs
Answer set: Formal Definition

Normal programs

- The reduct, \( \Pi^X \), of a program \( \Pi \) relative to a set \( X \) of atoms is defined by

\[
\Pi^X = \{ \text{head}(r) \leftarrow \text{body}^+(r) \mid r \in \Pi \text{ and } \text{body}^-(r) \cap X = \emptyset \}.
\]

- Think of \( X \) as being a “guess” of an answer set.
- The reduct “compiles out” negation as failure, given \( X \).

- A set \( X \) of atoms is an answer set of a program \( \Pi \) if \( Cn(\Pi^X) = X \).

Recall: \( Cn(\Pi^X) \) is the \( \subseteq \)–smallest (classical) model of \( \Pi^X \).
Answer set: Formal Definition

Normal programs

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

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

  - Think of $X$ as being a “guess” of an answer set.
  - The reduct “compiles out” negation as failure, given $X$.

- A set $X$ of atoms is an answer set of a program $\Pi$ if $Cn(\Pi^X) = X$.

Recall: $Cn(\Pi^X)$ is the $\subseteq$-smallest (classical) model of $\Pi^X$.

Intuition: Every atom in $X$ is justified by an “applying rule” from $\Pi$. 
A Closer Look at $\Pi^X$

Given a set of atoms $X$ from $\Pi$, $\Pi^X$ is obtained from $\Pi$ by deleting

1. each rule having a $not \ A$ in its body with $A \in X$
   and then

2. all negative atoms of the form $not \ A$ in the bodies of the remaining rules.

• Thus $\Pi^X$ is $\Pi$, but where negative atoms are taken into account.

• Then $X$ is an answer set of $\Pi$ just if $\Pi^X$ “generates” $X$, i.e. $Cn(\Pi^X) = X$. 
A first example

\[ \Pi = \{ p \leftarrow p, \quad q \leftarrow \text{not } p \} \]
A first example

\[ \Pi = \{ \ p \leftarrow p, \ q \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(\Pi^X)</th>
<th>(Cn(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</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 first example

\[ \Pi = \{ \ p \leftarrow p, \ q \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(\Pi^X)</th>
<th>(Cn(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(p \leftarrow p)</td>
<td>({q})</td>
</tr>
<tr>
<td></td>
<td>(q \leftarrow)</td>
<td>![red 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></td>
<td>(q \leftarrow)</td>
<td></td>
</tr>
</tbody>
</table>
A first example

\[ \Pi = \{ p \leftarrow p, \quad q \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>(\Pi^X)</th>
<th>(Cn(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</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

\[ \Pi = \{ p \leftarrow p, \quad q \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>( \mathcal{X} )</th>
<th>( \Pi^\mathcal{X} )</th>
<th>( C_n(\Pi^\mathcal{X}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</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

\[ \Pi = \{ \ p \leftarrow p, \ q \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( \Pi^X )</th>
<th>( Cn(\Pi^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</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 second example

\[ \Pi = \{ \ p \leftarrow \text{not } q, \ q \leftarrow \text{not } p \ \} \]
A second example

\[ \Pi = \{ p \leftarrow \text{not } q, \quad q \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( \Pi^X )</th>
<th>( Cn(\Pi^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</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 second example

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

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

\[ \Pi = \{ p \leftarrow \neg q, \quad q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>$X$</th>
<th>$\Pi^X$</th>
<th>$\text{Cn}(\Pi^X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$p \leftarrow$</td>
<td>${p, q}$</td>
</tr>
<tr>
<td></td>
<td>$q \leftarrow$</td>
<td>$\times$</td>
</tr>
<tr>
<td>${p}$</td>
<td>$p \leftarrow$</td>
<td>${p}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>${q}$</td>
<td></td>
<td>${q}$</td>
</tr>
<tr>
<td>${p, q}$</td>
<td>$q \leftarrow$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
A second example

\[ \Pi = \{ \ p \leftarrow \text{not } q, \ q \leftarrow \text{not } p \} \]

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

$$\Pi = \{ \ p \leftarrow \text{not } q, \ q \leftarrow \text{not } p \ \}$$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$\Pi^X$</th>
<th>$\text{Cn}(\Pi^X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>${q}$</td>
<td></td>
<td>${q}$ ✓</td>
</tr>
<tr>
<td></td>
<td>$q \leftarrow$</td>
<td></td>
</tr>
<tr>
<td>${p, q}$</td>
<td></td>
<td>$\emptyset$ ✗</td>
</tr>
</tbody>
</table>
A third example

\[ \Pi = \{ p \leftarrow \text{not} \ p \} \]
A third example

$$\Pi = \{ \, p \leftarrow \text{not } p \, \}$$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$\Pi^X$</th>
<th>$Cn(\Pi^X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$p \leftarrow$</td>
<td>${p}$</td>
</tr>
<tr>
<td>${p}$</td>
<td></td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
A third example

\[ \Pi = \{ p \leftarrow \text{not } p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( \Pi^X )</th>
<th>( Cn(\Pi^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( p \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{p}</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
A third example

\[ \Pi = \{ \ p \leftarrow \text{not } p \ \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(\Pi^X)</th>
<th>(Cn(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(p \leftarrow)</td>
<td>{p} \times</td>
</tr>
<tr>
<td>{p}</td>
<td></td>
<td>(\emptyset) \times</td>
</tr>
</tbody>
</table>
A final example

\[ \Pi = \{ a \leftarrow, \ c \leftarrow \text{not } b, \text{not } d, \ d \leftarrow a, \text{not } c, \} \]
A final example

$$\Pi = \{ \ a \leftarrow, \ c \leftarrow \mathit{not \ b}, \mathit{not \ d}, \ d \leftarrow a, \mathit{not \ c}, \ \}$$

This program has two answer sets, \( \{a, c\} \) and \( \{a, d\} \).
A final example

\[ \Pi = \{ \ a \leftarrow, \ c \leftarrow \text{not } b, \text{not } d, \ d \leftarrow a, \text{not } c, \ \} \]

This program has two answer sets, \{a, c\} and \{a, d\}. Here are 3 possibilities for \(X\):

<table>
<thead>
<tr>
<th>(X)</th>
<th>(\Pi^X)</th>
<th>(Cn(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(a \leftarrow)</td>
<td>{a, c, d}</td>
</tr>
<tr>
<td></td>
<td>(c \leftarrow)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(d \leftarrow a)</td>
<td></td>
</tr>
<tr>
<td>{a, c}</td>
<td>(a \leftarrow)</td>
<td>{a, c}</td>
</tr>
<tr>
<td></td>
<td>(c \leftarrow)</td>
<td></td>
</tr>
<tr>
<td>{a, b, c, d}</td>
<td>(a \leftarrow)</td>
<td>{a}</td>
</tr>
</tbody>
</table>
A final example

$$\Pi = \{ \ a \leftarrow, \ c \leftarrow \text{not } b, \text{not } d, \ d \leftarrow a, \text{not } c, \ \}$$

This program has two answer sets, \( \{a, c\} \) and \( \{a, d\} \).

Here are 3 possibilities for \( X \):

<table>
<thead>
<tr>
<th>( X )</th>
<th>( \Pi^X )</th>
<th>( \text{Cn}(\Pi^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( a \leftarrow )</td>
<td>( {a, c, d} \times )</td>
</tr>
<tr>
<td></td>
<td>( c \leftarrow )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( d \leftarrow a )</td>
<td></td>
</tr>
<tr>
<td>( {a, c} )</td>
<td>( a \leftarrow )</td>
<td>( {a, c} )</td>
</tr>
<tr>
<td></td>
<td>( c \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>( {a, b, c, d} )</td>
<td>( a \leftarrow )</td>
<td>( {a} )</td>
</tr>
</tbody>
</table>
A final example

$$\Pi = \{ a \leftarrow, \ c \leftarrow \text{not } b, \text{not } d, \ d \leftarrow a, \text{not } c, \}$$

This program has two answer sets, \{a, c\} and \{a, d\}. Here are 3 possibilities for \(X\):

<table>
<thead>
<tr>
<th>(X)</th>
<th>(\Pi^X)</th>
<th>(Cn(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(a \leftarrow)</td>
<td>({a, c, d})</td>
</tr>
<tr>
<td></td>
<td>(c \leftarrow)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(d \leftarrow a)</td>
<td></td>
</tr>
<tr>
<td>({a, c})</td>
<td>(a \leftarrow)</td>
<td>({a, c})</td>
</tr>
<tr>
<td></td>
<td>(c \leftarrow)</td>
<td></td>
</tr>
<tr>
<td>({a, b, c, d})</td>
<td>(a \leftarrow)</td>
<td>({a})</td>
</tr>
</tbody>
</table>
A final example

\[ \Pi = \{ \ a \leftarrow, \ c \leftarrow \text{not } b, \text{not } d, \ d \leftarrow a, \text{not } c, \} \]

This program has two answer sets, \{a, c\} and \{a, d\}. Here are 3 possibilities for \(X\):

<table>
<thead>
<tr>
<th>(X)</th>
<th>(\Pi^X)</th>
<th>(\text{Cn}(\Pi^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(a \leftarrow)</td>
<td>({a, c, d})</td>
</tr>
<tr>
<td>({a, c})</td>
<td>(a \leftarrow)</td>
<td>({a, c})</td>
</tr>
<tr>
<td>({a, b, c, d})</td>
<td>(a \leftarrow)</td>
<td>({a})</td>
</tr>
</tbody>
</table>
Answer sets: Some properties

- A program may have zero, one, or multiple answer sets.
- If $X$ is an answer set of a logic program $\Pi$, then $X$ is a model of $\Pi$ (seen as formulas of classical logic).
- If $X$ and $Y$ are answer sets of a logic program $\Pi$, then $X \not\subset Y$. 
Let $\Pi$ be a logic program.

- The *Herbrand Universe* $U^\Pi$ is the set of constants in $\Pi$.
- The *Herbrand Base* $B^\Pi$ is the set of (variable-free) atoms constructible from $U^\Pi$.

We usually denote this as $A$, and call it the *alphabet*. 
Programs with Variables

- **Ground instances** of $r \in \Pi$:

  Set of variable-free rules obtained by replacing all variables in $r$ by elements from $U^\Pi$:

  $$\text{ground}(r) = \{ r\theta \mid \theta : \text{var}(r) \rightarrow U^\Pi \}$$

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

- **Ground instances** of \( r \in \Pi \): 

  Set of variable-free rules obtained by replacing all variables in \( r \) by elements from \( U^\Pi \):

  \[
ground(r) = \{ r\theta \mid \theta : \text{var}(r) \rightarrow U^\Pi \}\]

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

- **Ground instantiation** of \( \Pi \):

  \[
ground(\Pi) = \{ \ground(r) \mid r \in \Pi \}\]
An Example

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

\[ \Pi = \{ \ r(a, b) \leftarrow, \ r(b, c) \leftarrow, \ t(X, Y) \leftarrow r(X, Y) \ \} \]
\[ U^\Pi = \{ a, b, c \} \]
An Example

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

\[ U^\Pi = \{ a, b, c \} \]

\[ B^\Pi = \left\{ \begin{array}{c}
  r(a, a), \; r(a, b), \; r(a, c), \\
  r(b, a), \; r(b, b), \; r(b, c), \\
  r(c, a), \; r(c, b), \; r(c, c), \\
  t(a, a), \; t(a, b), \; t(a, c), \\
  t(b, a), \; t(b, b), \; t(b, c), \\
  t(c, a), \; t(c, b), \; t(c, c)
\end{array} \right\} \]
\( \Pi = \{ \ r(a, b) \leftarrow, \ r(b, c) \leftarrow, \ t(X, Y) \leftarrow r(X, Y) \ \} \)

\( U^\Pi = \{ a, b, c \} \)

\( B^\Pi = \{ \begin{array}{c}
  r(a, a), \ r(a, b), \ r(a, c), \\
  r(b, a), \ r(b, b), \ r(b, c), \\
  r(c, a), \ r(c, b), \ r(c, c), \\
  t(a, a), \ t(a, b), \ t(a, c), \\
  t(b, a), \ t(b, b), \ t(b, c), \\
  t(c, a), \ t(c, b), \ t(c, c)
\end{array} \} \)

\( \text{ground}(\Pi) = \{ \begin{array}{c}
  r(a, b) \leftarrow, \\
  r(b, c) \leftarrow, \\
  t(a, a) \leftarrow r(a, a), \ t(b, a) \leftarrow r(b, a), \ t(c, a) \leftarrow r(c, a), \\
  t(a, b) \leftarrow r(a, b), \ t(b, b) \leftarrow r(b, b), \ t(c, b) \leftarrow r(c, b), \\
  t(a, c) \leftarrow r(a, c), \ t(b, c) \leftarrow r(b, c), \ t(c, c) \leftarrow r(c, c)
\end{array} \} \)
An Example

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

\[ U^\Pi = \{ a, b, c \} \]

\[ B^\Pi = \begin{cases} 
    \{ \ r(a, a), \ r(a, b), \ r(a, c), \\
    \ r(b, a), \ r(b, b), \ r(b, c), \\
    \ r(c, a), \ r(c, b), \ r(c, c), \\
    \ t(a, a), \ t(a, b), \ t(a, c), \\
    \ t(b, a), \ t(b, b), \ t(b, c), \\
    \ t(c, a), \ t(c, b), \ t(c, c) \} 
\end{cases} \]

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

• Intelligent Grounding aims to reduce the ground instantiation.
Answer Sets of Programs with Variables

Let $\Pi$ be a normal logic program with variables.

We define a set $X$ of \textit{(ground)} atoms as an \textit{answer set} of $\Pi$ if $Cn(\text{ground}(\Pi)^X) = X$. 
Programs with Integrity Constraints

**Purpose:** Integrity constraints eliminate unwanted candidate solutions
Programs with Integrity Constraints

Purpose: Integrity constraints eliminate unwanted candidate solutions

Syntax: An integrity constraint is of the form

\[ \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n, \]

where \( n \geq m \geq 1 \), and each \( A_i \) (\( 1 \leq i \leq n \)) is a atom.

Example

\[ \leftarrow Edge(X, Y), Col(X, C), Col(Y, C) \]
Programs with Integrity Constraints

**Purpose:** Integrity constraints eliminate unwanted candidate solutions

**Syntax:** An integrity constraint is of the form

\[ \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n, \]

where \( n \geq m \geq 1 \), and each \( A_i \) (\( 1 \leq i \leq n \)) is a atom.

**Example**

\[ \leftarrow \text{Edge}(X, Y), \text{Col}(X, C), \text{Col}(Y, C) \]

**Implementation:** For a new symbol \( x \),

map: \[ \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n \]

to: \[ x \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n, \text{not } x \]
Computation: Standard Scheme

Global parameters: Logic program $\Pi$ and its set of atoms $\mathcal{A}$.

- $X$ is a set of atoms known to be true;
- $Y$ is a set of atoms known to be false.

$$\text{answer}\text{set}_\Pi(X, Y) :$$

1. $(X, Y) \leftarrow \text{propagation}_\Pi(X, Y)$
2. if $(X \cap Y) \neq \emptyset$ then fail
3. if $(X \cup Y) = \mathcal{A}$ then return($X$)
4. select $A \in \mathcal{A} \setminus (X \cup Y)$
5. $\text{answer}\text{set}_\Pi(X \cup \{A\}, Y)$
6. $\text{answer}\text{set}_\Pi(X, Y \cup \{A\})$
Computation: Standard Scheme

Comments:

• \((X, Y)\) is supposed to be a 3-valued model such that \(X \subseteq Z\) and \(Y \cap Z = \emptyset\) for any answer set \(Z\) of \(\Pi\).

• Key operations:
  • \(\text{propagation}_{\Pi}(X, Y)\) and
  • “\(\text{select } A \in \mathcal{A} \setminus (X \cup Y)\)”

• Worst case complexity: \(\mathcal{O}(2^{\lvert \mathcal{A} \rvert})\)

More later...