Some Extensions to ASP
Language extensions

• Motivation
• Extending the formalism
  • Classical negation
  • Disjunction
• Other language extensions
  • Intervals and conditions
  • Choice rules
  • Cardinality constraints
  • Cardinality rules
  • (Weight constraints and optimization statements)
• Relation with Default Logic
Language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- Issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- One way of providing semantics is to give a translation into a normal program
  - eg. classical negation.
- Such a translation might also be used for implementing the language extension.
- Here we’ll focus on syntax and informal semantics.
Classical Negation: Syntax

Normal logic programs

- In logic programs *not* (or ∼) denotes default negation.
Classical Negation: Syntax

Normal logic programs

- In logic programs not (or ∼) denotes default negation.

Generalization

- We allow classical negation for atoms (only!).
  - Logic programs in “negation normal form.”
- Given an alphabet \( A \) of atoms, let
  \[ \overline{A} = \{ \neg A \mid A \in A \} \]
  - So \( A \cap \overline{A} = \emptyset \).
- The atoms \( A \) and \( \neg A \) are complementary.
  - \( \neg A \) is the classical negation of \( A \), and vice versa.
• Given a set $X$, the difference between *not* $a$ and $\neg a$ amounts to:

$$a \not\in X \quad \text{versus} \quad \neg a \in X$$
Syntax (ctd)

- Given a set $X$, the difference between $\text{not } a$ and $\neg a$ amounts to:
  \[ a \notin X \quad \text{versus} \quad \neg a \in X \]

- Example:
  
  \[
  \text{cross} \leftarrow \text{not train} \quad \quad \text{cross} \leftarrow \neg \text{train}
  \]
• Given a set $X$, the difference between *not* $a$ and $\neg a$ amounts to:

\[
a \not\in X \text{ versus } \neg a \in X
\]

• Example:

\[
\begin{align*}
\text{cross} & \leftarrow \text{not train} \\
X & = \{\text{cross}\}
\end{align*}
\]

\[
\begin{align*}
\text{cross} & \leftarrow \neg \text{train} \\
X & = \emptyset
\end{align*}
\]
A set \( X \) of atoms is an answer set of a logic program \( \Pi \) over \( \mathcal{A} \cup \overline{\mathcal{A}} \) if \( X \) is an answer set of \( \Pi \cup \Pi' \) where

\[
\Pi' = \{ \leftarrow A, \neg A \mid A \in \mathcal{A} \}
\]

The text has a more general definition, which we won’t bother with:

“A set \( X \) of atoms is an answer set of a logic program \( \Pi \) over \( \mathcal{A} \cup \overline{\mathcal{A}} \) if \( X \) is an answer set of \( \Pi \cup \Pi' \) where

\[
\Pi' = \{ B \leftarrow A, \neg A \mid A \in \mathcal{A} \text{ and } B \in (\mathcal{A} \cup \overline{\mathcal{A}}) \}
\]

Thus if \( A \) and \( \neg A \) are in an answer set, then so is every other literal.
To cross or not to cross...?

• $\Pi_1 = \{\text{cross} \leftarrow \text{not train}\}$

• $\Pi_2 = \{\text{cross} \leftarrow \neg \text{train}\}$

• $\Pi_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$

• $\Pi_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
To cross or not to cross...?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
- $\Pi_3 = \{cross \leftarrow \neg train, \neg train \leftarrow\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \neg train \leftarrow, \neg cross \leftarrow\}$
To cross or not to cross...?

- $\Pi_1 = \{ cross \leftarrow not\ train \}$
  - Answer set: $\{ cross \}$
- $\Pi_2 = \{ cross \leftarrow \neg train \}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{ cross \leftarrow \neg train, \neg train \leftarrow \}$
- $\Pi_4 = \{ cross \leftarrow \neg train, \neg train \leftarrow, \neg cross \leftarrow \}$
To cross or not to cross...?

- $\Pi_1 = \{\text{cross} \leftarrow \text{not train}\}$
  - Answer set: $\{\text{cross}\}$
- $\Pi_2 = \{\text{cross} \leftarrow \neg \text{train}\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$
  - Answer set: $\{\text{cross}, \neg \text{train}\}$
- $\Pi_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
To cross or not to cross...?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train, \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train, \neg train \leftarrow, \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
A disjunctive rule, \( r \), is of the form

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

where \( o \geq n \geq m \geq 0 \), and each \( A_i \) (\( 0 \leq i \leq o \)) is an atom.

A disjunctive logic program is a finite set of disjunctive rules.

(Generalized) Notation

- \( \text{head}(r) = \{A_1, \ldots, A_m\} \)
- \( \text{body}(r) = \{A_{m+1}, \ldots, A_n, \text{not } A_{n+1}, \ldots, \text{not } A_o\} \)
- \( \text{body}^+(r) = \{A_{m+1}, \ldots, A_n\} \)
- \( \text{body}^-(r) = \{A_{n+1}, \ldots, A_o\} \)

A program is called positive if \( \text{body}^-(r) = \emptyset \) for all its rules.
Answer sets

- Disjunctive positive programs:
  - A set \( X \) of atoms is **closed under** a positive program \( \Pi \) iff for any \( r \in \Pi \), if \( \text{body}^+(r) \subseteq X \) then \( \text{head}(r) \cap X \neq \emptyset \).
  - \( X \) is a model of \( \Pi \) (seen as a formula).
  - The set of all \( \subseteq \)-minimal sets of atoms closed under a positive program \( \Pi \) is denoted by \( \text{min}_{\subseteq}(\Pi) \).
  - \( \text{min}_{\subseteq}(\Pi) \) corresponds to the \( \subseteq \)-minimal models of \( \Pi \).
Answer sets

- Disjunctive positive programs:
  - A set $X$ of atoms is closed under a positive program $\Pi$ iff for any $r \in \Pi$, if $\text{body}^+(r) \subseteq X$ then $\text{head}(r) \cap X \neq \emptyset$.
  - $X$ is a model of $\Pi$ (seen as a formula).
  - The set of all $\subseteq$-minimal sets of atoms closed under a positive program $\Pi$ is denoted by $\text{min}_{\subseteq}(\Pi)$.
  - $\text{min}_{\subseteq}(\Pi)$ corresponds to the $\subseteq$-minimal models of $\Pi$.

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

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

- **Disjunctive positive programs:**
  - A set $X$ of atoms is **closed under** a positive program $\Pi$ iff for any $r \in \Pi$, if $body^+(r) \subseteq X$ then $head(r) \cap X \neq \emptyset$.
    - $X$ is a model of $\Pi$ (seen as a formula).
  - The set of all $\subseteq$-minimal sets of atoms closed under a positive program $\Pi$ is denoted by $\text{min}_{\subseteq}(\Pi)$.
    - $\text{min}_{\subseteq}(\Pi)$ corresponds to the $\subseteq$-minimal models of $\Pi$

- **Disjunctive programs:**
  - The **reduct**, $\Pi^X$, of a disjunctive program $\Pi$ relative to a set $X$ of atoms is defined by
    \[
    \Pi^X = \{ head(r) \leftarrow body^+(r) \mid r \in \Pi \text{ and } body^-(r) \cap X = \emptyset \}.\]
  - A set $X$ of atoms is an **answer set** of a disjunctive program $\Pi$ if $X \in \text{min}_{\subseteq}(\Pi^X)$. 
Example

\[ \Pi = \left\{ \begin{array}{c}
  a \\
  b; c \leftarrow a
\end{array} \right\} \]

The sets \( \{a, b\} \), \( \{a, c\} \), and \( \{a, b, c\} \) are closed under \( \Pi \).

We have \( \min \subseteq (\Pi) = \{\{a, b\}, \{a, c\}\} \), so these are the answer sets.
Π = \{ \begin{align*} &a \quad \leftarrow \quad b ; \\ &c \quad \leftarrow \quad a \end{align*} \} \\

- The sets \{a, b\}, \{a, c\}, and \{a, b, c\} are closed under Π.
Example

\[ \Pi = \left\{ \begin{array}{l} a \\ b ; c \leftarrow a \end{array} \right\} \]

- The sets \{a, b\}, \{a, c\}, and \{a, b, c\} are closed under \( \Pi \).
- We have
  \[ \text{min}_{\subseteq}(\Pi) = \{ \{a, b\}, \{a, c\} \} , \]
  so these are the answer sets.
### 3-colorability revisited

<table>
<thead>
<tr>
<th>C(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex(1) ← edge(1,2) ←</td>
</tr>
<tr>
<td>vertex(2) ← edge(2,3) ←</td>
</tr>
<tr>
<td>vertex(3) ← edge(3,1) ←</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>colored(V,r); colored(V,b); colored(V,g) ← vertex(V)</td>
</tr>
<tr>
<td>← edge(V,U), colored(V,C), colored(U,C)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Answer set</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ colored(1,r), colored(2,b), colored(3,g), othercolor(1,g),..., vertex(1),..., edge(1,2), ... }</td>
</tr>
</tbody>
</table>
More Examples

- $\Pi_1 = \{ a ; b ; c \leftarrow \}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$. 
More Examples

- $\Pi_1 = \{ a ; b ; c \leftarrow \}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{ a ; b ; c \leftarrow , \leftarrow a \}$ has answer sets $\{b\}$ and $\{c\}$.
More Examples

• $\Pi_1 = \{ a ; b ; c \leftarrow \} \text{ has answer sets } \{a\}, \{b\}, \text{ and } \{c\}.$
• $\Pi_2 = \{ a ; b ; c \leftarrow , \leftarrow a \} \text{ has answer sets } \{b\} \text{ and } \{c\}.$
• $\Pi_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \} \text{ has answer set } \{b, c\}.$
More Examples

- $\Pi_1 = \{ a ; b ; c \leftarrow \}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{ a ; b ; c \leftarrow , \leftarrow a \}$ has answer sets $\{b\}$ and $\{c\}$.
- $\Pi_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \}$ has answer set $\{b, c\}$.
- $\Pi_4 = \{ a ; b \leftarrow c , b \leftarrow not a , not c , a ; c \leftarrow not b \}$ has answer sets $\{a\}$ and $\{b\}$.
Other language extensions

- Intervals and Conditions
- Choice rules
- Cardinality constraints
- Cardinality rules
- Weight constraints and more
Intervals and Conditions

Intervals
For facts, an expression like $num(1..5)$ stands for its instances in the obvious way:
\[ num(1), num(2), num(3), num(4), num(5). \]

Conditions
- For a rule like:
  \[ meet \leftarrow available(X) : person(X) \]
  $available(X)$ is replaced by a conjunction, where $X$ is replaced by those values of $X$ that satisfy $person(X)$.
- If $john$ and $sue$ are the only $persons$, then one gets:
  \[ meet \leftarrow available(john) \land available(sue) \]

Both intervals and conditions can be used more generally; see the user’s manual or text for more.
Choice rules [Simons et al., 2002]

Idea:
Choices over subsets.

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

Informal meaning:
If the body is satisfied in an answer set, then any subset of \(\{A_1, \ldots, A_m\}\) can be included in the answer set.

Example:
The program \(\Pi = \{\{a\} \leftarrow b, b \leftarrow\}\) has two answer sets: \(\{b\}\) and \(\{a, b\}\).
Cardinality constraints

Syntax
A (positive) cardinality constraint is of the form $l \{A_1, \ldots, A_m\} u$

Informal meaning
A cardinality constraint is satisfied in an answer set $X$, if the number of atoms from \{A_1, \ldots, A_m\} satisfied in $X$ is between $l$ and $u$ (inclusive).

Conditions
\[ l \{A_1 : B_1, \ldots, A_m : B_m\} u \]
where $B_1, \ldots, B_m$ are used for restricting instantiations of variables occurring in $A_1, \ldots, A_m$.

Example
2 \{hd(a), \ldots, hd(m)\} 4
$n$-colorability revisited with $n \equiv 3$

<table>
<thead>
<tr>
<th>C(I)</th>
<th>vertex(1) ← edge(1,2) ← vertex(2) ← edge(2,3) ← vertex(3) ← edge(3,1) ←</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(P)</td>
<td>color(r) ← color(b) ← color(g) ← 1 {colored(V,C) : color(C)} 1 ← vertex(V) ← edge(V,U),color(C), colored(V,C),colored(U,C)</td>
</tr>
<tr>
<td>Answer set</td>
<td>{ colored(1,r), colored(2,b), colored(3,g), ... }</td>
</tr>
</tbody>
</table>
Another Example: Graph Clique

A clique in a graph is a set of vertices with edges between all vertices in the set.
Another Example: Graph Clique

A clique in a graph is a set of vertices with edges between all vertices in the set.

```
vertex(1..99).  % 1, ..., 99 are vertices
edge(3, 7).     % 3 is adjacent to 7
...
edge(X, Y) ← edge(Y, X), vertex(X), vertex(Y).
```
Another Example: Graph Clique

A clique in a graph is a set of vertices with edges between all vertices in the set.

\[
\text{vertex}(1..99). \quad \% \quad 1, \ldots, 99 \text{ are vertices}
\]

\[
\text{edge}(3, 7). \quad \% \quad 3 \text{ is adjacent to } 7
\]

\[
\ldots
\]

\[
\text{edge}(X, Y) \leftarrow \text{edge}(Y, X), \text{vertex}(X), \text{vertex}(Y).
\]

\[
5\{\text{in}(X) : \text{vertex}(X)\}.
\]
Another Example: Graph Clique

A clique in a graph is a set of vertices with edges between all vertices in the set.

\[
\text{vertex}(1..99). \quad \text{edge}(3, 7). \quad \ldots \quad \text{edge}(X, Y) \leftarrow \text{edge}(Y, X), \text{vertex}(X), \text{vertex}(Y).
\]

\[
5\{\text{in}(X) : \text{vertex}(X)\}. \leftarrow \text{in}(X), \text{in}(Y) \quad X \neq Y, \text{not edge}(X, Y).
\]
Cardinality Rules [Simons et al., 2002]

Idea
Control cardinality of subsets.

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

Informal meaning
If at least \( l \) elements and no more than \( u \) of the “body” are true in an answer set, then add \( A_0 \) to the answer set.

\( l \) is a lower bound on the “body” and \( u \) is an upper bound

Example
Program \( \Pi = \{ \ a \leftarrow 1\{b, c\}, \ b \leftarrow \} \) has one answer set: \( \{a, b\} \).
Example: Vertex Cover

A vertex cover of size $k$ in a graph $G = (V, E)$ is a set of vertices $V' \subseteq V$ where $V' \leq k$ and for each edge $(v, u) \in E$, at least one of $u, v$ is in $V'$. 
Example: Vertex Cover

A vertex cover of size $k$ in a graph $G = (V, E)$ is a set of vertices $V' \subseteq V$ where $V' \leq k$ and for each edge $(v, u) \in E$, at least one of $u, v$ is in $V'$.

vertex(1..99).

edge(4, 8).

\ldots
Example: Vertex Cover

A vertex cover of size $k$ in a graph $G = (V, E)$ is a set of vertices $V' \subseteq V$ where $|V'| \leq k$ and for each edge $(v, u) \in E$, at least one of $u, v$ is in $V'$.

vertex(1..99).
edge(4, 8).
...

1{\text{incover}(X), \text{incover}(Y)} \leftarrow \text{edge}(X, Y).
Example: Vertex Cover

A vertex cover of size \( k \) in a graph \( G = (V, E) \) is a set of vertices \( V' \subseteq V \) where \( V' \leq k \) and for each edge \( (v, u) \in E \), at least one of \( u, v \) is in \( V' \).

\[
\text{vertex}(1..99).
\]
\[
\text{edge}(4, 8).
\]
\[
\ldots
\]
\[
1\{\text{incover}(X), \text{incover}(Y)\} \leftarrow \text{edge}(X, Y).
\]
\[
\leftarrow k + 1\{\text{incover}(X) : \text{vertex}(X)\}.
\]
Extensions: Summary

- Intervals and conditions are clearly a convenience, and make programs more compact.
- Classical negation, choice rules, cardinality constraints and cardinality rules are redundant:
  - each can be translated into an equivalent (wrt answer sets) program with “regular” normal rules only.
  - See the ASP text for details.
- Disjunctive rules represent a genuine increase in expressivity.
  - Allows for solving all search problems within $NP^{NP}$

And more:
- Weight constraints
- Optimization statements
  (See text)
Extensions: Summary

- Intervals and conditions are clearly a convenience, and make programs more compact.
- Classical negation, choice rules, cardinality constraints and cardinality rules are redundant:
  - each can be translated into an equivalent (wrt answer sets) program with “regular” normal rules only.
  - See the ASP text for details.
- Disjunctive rules represent a genuine increase in expressivity.
  - Allows for solving all search problems within $NP^{NP}$

And more:

- Weight constraints
- Optimization statements

(See text)
C. Baral.  
*Knowledge Representation, Reasoning and Declarative Problem Solving.*  

A user’s guide to gringo, clasp, clingo, and iclingo.  
Available at http://potassco.sourceforge.net.

M. Gelfond and V. Lifschitz.  
Logic programs with classical negation.  

Extending and implementing the stable model semantics.  

T. Syrjänen.  
Lparse 1.0 user’s manual.