Logic Programming

View Definitions

Michael Genesereth
Computer Science Department
Stanford University
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
\( \text{grandparent}(X,Z) :- \text{parent}(X,Y) \& \text{parent}(Y,Z) \)
Deduction

parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)

+ 

grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)

= 

grandparent(art,cal)
grandparent(art,am)
grandparent(art,cat)
grandparent(art,coo)
Basic Logic Programming

Represent information as combination of data and rules. Encode information about “base” relations as **facts** in a dataset and write **rules** to define “view” relations in terms of the base relations.

**Benefits:**
- Economy - fewer facts need to be stored
- Less chance of getting out of sync
- Rules work for any number of objects
Syntax
Constants are strings of lower case letters, digits, underscores, and periods or strings of arbitrary ascii characters within double quotes. As before, we have symbols and constructors and predicates.

\begin{verbatim}
art  cs151  3.14159
  pair   triple
  person hasoffice
\end{verbatim}

Variables are strings of letters, digits, underscores beginning with an upper case letter.

\begin{verbatim}
  X  Y23    Somebody
\end{verbatim}
Symbols
  art
  bob

Variables
  X
  Y23

Compound Terms
  pair(art,bob)
  pair(X,Y23)
  pair(pair(art,bob),pair(X,Y23))
Atoms

- \( p(a,b) \)
- \( p(a,x) \)
- \( p(Y,c) \)

Negations

- \( \neg p(a,b) \)

Literals (atoms or negations of atoms)

- \( p(a,Y) \)
- \( \neg p(a,Y) \)

An atom is a *positive literal*.
A negations is a *negative literal*.
Rules

\[ r(X, Y) :\text{subgoal} \quad p(X, Y) \& \neg q(Y) \quad \text{subgoal} \]

\[ \text{head} \quad \text{body} \]
\[
\begin{align*}
  r(X,Y) &:= p(X,Y) \& \lnot q(Y) \\
p(X,Y) &:= f(X,Y) \\
p(X,Y) &:= m(X,Y)
\end{align*}
\]
Basic Logic Program

Dataset

\begin{align*}
    p(a, b) \\
    p(a, c) \\
    q(b, c) \\
    r(a) \\
    r(c)
\end{align*}

Ruleset

\begin{align*}
    s(X, Y) & :- \ p(X,Y) \\
    s(X, Y) & :- \ q(X,Y) \\
    t(X,Y) & :- s(X,Y) \ & \sim r(Y)
\end{align*}
A ruleset is **compatible** with a dataset if and only if

(1) all symbols shared between the dataset and the ruleset are of the same type (symbol, constructor, predicate)

(2) all constructors and predicates have the same arity

(3) none of the predicates in the dataset appear in the heads of any rules in the logic program.

The **vocabulary** of a basic logic program is the union of the vocabularies of the dataset and the logic program.
Semantics
The **Herbrand universe** for a basic logic program is the set of all *ground terms* that can be formed from the symbols and constructors in the vocabulary.

The **Herbrand base** for a basic logic program is the set of all *ground atoms* that can be formed from the vocabulary.

An **interpretation** of a basic logic program is an arbitrary *subset* of the program’s Herbrand base.

A **model** of a basic logic program is an interpretation that *satisfies* the program (definition to follow).
An **instance of a rule** is a rule in which all variables have been consistently replaced by ground terms.

**Rule**

\[ r(X,Y) := p(X,Y) \land \neg q(Y) \]

**Herbrand Universe**

\{a, b\}

**Instances**

\[
\begin{align*}
    r(a,a) & := p(a,a) \land \neg q(a) \\
    r(a,b) & := p(a,b) \land \neg q(b) \\
    r(b,a) & := p(b,a) \land \neg q(a) \\
    r(b,b) & := p(b,b) \land \neg q(b)
\end{align*}
\]
An interpretation $D$ satisfies a rule instance $p : - p_1 \& \ldots \& p_n$ if and only if $p \in D$ whenever $D$ satisfies $p_1, \ldots, p_n$. $D$ satisfies a ground atom $p$ if and only if $p \in D$. $D$ satisfies a ground negation $\neg p$ if and only if $p \notin D$.

An interpretation $D$ satisfies a basic logic program if and only if (1) all of the elements of the constituent dataset are included in $D$ and (2) $D$ satisfies all instances of all of the rules in the constituent ruleset.
Example

Dataset
\[ p(a, b) \]
\[ p(b, c) \]
\[ p(c, d) \]
\[ p(d, c) \]

Ruleset
\[ r(X, Y) :- p(X, Y) \land \neg p(Y, X) \]

Model
\[ p(a, b) \]
\[ p(b, c) \]
\[ p(c, b) \]
\[ p(d, c) \]
\[ r(a, b) \]
\[ r(b, c) \]
Non-Examples

Dataset

\[ p(a,b) \]
\[ p(b,c) \]
\[ p(c,d) \]
\[ p(d,c) \]

Ruleset

\[ r(X,Y) :- p(X,Y) \land \neg p(Y,X) \]

Not Models

\begin{align*}
\text{p}(a,b) & \quad \text{r}(a,b) & \quad \text{p}(a,b) \\
p(b,c) & \quad \text{r}(b,c) & \quad \text{p}(b,c) \\
p(c,d) & \quad \text{p}(c,d) \\
p(d,c) & \quad \text{r}(a,b) & \quad \text{r}(a,b) \\
\end{align*}
# Multiple Models

## Dataset

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p(a,b)</code></td>
<td><code>p(a,b)</code></td>
<td><code>p(a,b)</code></td>
<td><code>p(b,c)</code></td>
</tr>
<tr>
<td><code>p(b,c)</code></td>
<td><code>p(b,c)</code></td>
<td><code>p(c,d)</code></td>
<td><code>p(d,c)</code></td>
</tr>
<tr>
<td><code>p(c,d)</code></td>
<td><code>p(c,d)</code></td>
<td><code>p(d,c)</code></td>
<td><code>p(d,c)</code></td>
</tr>
<tr>
<td><code>p(d,c)</code></td>
<td><code>p(d,c)</code></td>
<td><code>p(d,c)</code></td>
<td><code>p(d,c)</code></td>
</tr>
</tbody>
</table>

## Ruleset

\[
 r(X, Y) :- \ p(X, Y) \ \land \ \neg p(Y, X) 
\]

## Models

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p(a,b)</code></td>
<td><code>p(a,b)</code></td>
<td><code>p(a,b)</code></td>
<td><code>r(b,c)</code></td>
</tr>
<tr>
<td><code>p(b,c)</code></td>
<td><code>p(b,c)</code></td>
<td><code>p(b,c)</code></td>
<td><code>r(c,d)</code></td>
</tr>
<tr>
<td><code>p(c,d)</code></td>
<td><code>p(c,d)</code></td>
<td><code>p(c,d)</code></td>
<td><code>r(d,c)</code></td>
</tr>
<tr>
<td><code>p(d,c)</code></td>
<td><code>p(d,c)</code></td>
<td><code>p(d,c)</code></td>
<td><code>r(d,c)</code></td>
</tr>
<tr>
<td><code>r(a,b)</code></td>
<td><code>r(a,b)</code></td>
<td><code>r(a,b)</code></td>
<td><code>r(b,c)</code></td>
</tr>
<tr>
<td><code>r(b,c)</code></td>
<td><code>r(b,c)</code></td>
<td><code>r(b,c)</code></td>
<td><code>r(c,d)</code></td>
</tr>
<tr>
<td><code>r(c,d)</code></td>
<td><code>r(d,c)</code></td>
<td><code>r(d,c)</code></td>
<td><code>r(d,c)</code></td>
</tr>
</tbody>
</table>
So What?

We want our definitions to be *if and only if*. We want to include among our conclusions only those facts that *must* be true.

All factoids in dataset *must* be true. All factoids required by rules *must* be true. Arbitrary factoids should be excluded.

What we want is logical entailment. A factoid is **logically entailed** by a basic logic program if and only if it is true in *every* model of the program, i.e. the set of conclusions is the intersection of all models of the program.
A model $D$ of a logic program $P$ is **minimal** if and only if no proper subset of $D$ is a model of $P$.

### Models

<table>
<thead>
<tr>
<th>$p(a,b)$</th>
<th>$p(a,b)$</th>
<th>$p(a,b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(b,c)$</td>
<td>$p(b,c)$</td>
<td>$p(b,c)$</td>
</tr>
<tr>
<td>$p(c,d)$</td>
<td>$p(c,d)$</td>
<td>$p(c,d)$</td>
</tr>
<tr>
<td>$p(d,c)$</td>
<td>$p(d,c)$</td>
<td>$p(d,c)$</td>
</tr>
<tr>
<td>$r(a,b)$</td>
<td>$r(a,b)$</td>
<td>$r(a,b)$</td>
</tr>
<tr>
<td>$r(b,c)$</td>
<td>$r(b,c)$</td>
<td>$r(b,c)$</td>
</tr>
<tr>
<td>$r(c,d)$</td>
<td>$r(c,d)$</td>
<td>$r(d,c)$</td>
</tr>
</tbody>
</table>
Minimal Models

If a program had just one minimal model, then every factoid true in that model would be trivially true in every model of the program.

A logic program that does not contain any negations has a unique minimal model.

A logic program with negations can have more than one minimal model (in addition to multiple non-minimal models).

If a program is stratified (as defined below), then once again there is only one minimal model.
Stratified Negation
Multiple *Minimal* Models

**Dataset**

\[
p(a, b) \\
p(b, a)
\]

**Ruleset**

\[
r(X) :- p(X, Y) \& \neg r(Y)
\]

**Interpretations**

\[
\begin{array}{cccc}
p(a, b) & p(a, b) & p(a, b) & p(a, b) \\
p(b, a) & p(b, a) & p(b, a) & p(b, a) \\
r(a) & r(b) & r(a) & r(b)
\end{array}
\]

Is \( r(a) \) true or not? What about \( r(b) \)?

*NB: The intersection of all models is not a model!*
The *dependency graph* for a set of rules is a directed graph in which (1) the nodes are the relations mentioned in the head and bodies of the rules and (2) there is an arc from a node $p$ to a node $q$ whenever $p$ occurs with the body of a rule in which $q$ is in the head.

\[
\begin{align*}
r(X,Y) & : - p(X,Y) \ & \land \ q(X,Y) \\
s(X,Y) & : - r(X,Y) \\
s(X,Z) & : - r(X,Y) \ & \land \ t(Y,Z) \\
t(X,Z) & : - s(X,Y) \ & \land \ s(Y,X)
\end{align*}
\]

A set of rules is *recursive* if it contains a cycle. Otherwise, it is *non-recursive*. 
The negation in a set of rules is said to be *stratified* if and only if there is no recursive cycle in the dependency graph involving a negation.

**Stratified Negation:**

\[
\begin{align*}
  r(X, Z) & : \neg p(X, Y) \\
  r(X, Z) & : r(X, Y) \land r(Y, Z)
\end{align*}
\]

**Negation that is not stratified:**

\[
\begin{align*}
  r(X, Z) & : \neg p(X, Y) \\
  r(X, Z) & : p(X, Y) \land \neg r(Y, Z)
\end{align*}
\]

*All negations must be stratified.*
We say that a set of view definitions is *stratified* if and only if its rules can be partitioned into *strata* in such a way that

(1) every stratum contains at least one rule

(2) the rules defining relations that appear in positive goals of a rule appear in the same stratum as that rule or in some lower stratum

(3) the rules defining relations that appear in negative subgoals of a rule occur in some *lower* stratum (not the same stratum)
Example

\[ r(X,Y) :- q(X,Y) \]
\[ r(X,Z) :- q(X,Y) \land r(Y,Z) \]
\[ s(X,Y) :- p(X) \land p(Y) \land \neg r(X,Y) \]

<table>
<thead>
<tr>
<th>Stratum</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ s(X,Y) :- p(X) \land p(Y) \land \neg r(X,Y) ]</td>
</tr>
</tbody>
</table>
| 1       | \[ r(X,Y) :- q(X,Y) \]
|         | \[ r(X,Z) :- q(X,Y) \land r(Y,Z) \] |

Non-example

\[ r(X,Y) :- p(X) \land p(Y) \land q(X,Y) \]
\[ s(X,Y) :- r(X,Y) \land \neg s(Y,X) \]
Safety
Ruleset

\[ s(X,Y,Z) \leftarrow p(X,Y) \]

Dataset

\[ p(a,b) \]

Output for \( s \)

\[ s(a,b,a) \]
\[ s(a,b,b) \]
\[ s(a,b,c) \]
\[ s(a,b,d) \]
\[ s(a,b,e) \]

\[ \ldots \]
Unbound Variables in Negation

Ruleset

\[ t(X, Y) :- p(X, Y) \& \sim q(Y, Z) \]

Dataset

- \( p(a, b) \)
- \( p(a, c) \)
- \( q(c, d) \)

Output

- \( t(a, b) \)
- \( t(a, c) \)

Alternative Rules

\[ t(X, Y) :- p(X, Y) \& \sim qqq(Y) \]
\[ qqq(Y) :- q(Y, Z) \]
A rule is *safe* if and only if every variable in the head appears in some positive subgoal in the body and every variable in a negative subgoal appears in a prior positive subgoal.

**Safe Rule:**
\[ r(X,Z) :- p(X,Y) \land q(Y,Z) \land \neg r(X,Y) \]

**Unsafe Rule:**
\[ r(X,Z) :- p(X,Y) \land q(Y,X) \]

**Unsafe Rule:**
\[ r(X,Y) :- p(X,Y) \land \neg q(Y,Z) \]