Logic Programming

Queries

Michael Genesereth
Computer Science Department
Stanford University
Queries

True or False questions:
  e.g. Is Art the parent of Bob?

Fill-in-the-blanks questions:
  e.g. Art is the parent of ____?
  e.g. ____ is the parent of Bob?
  e.g. ____ is the parent of ____?

Compound questions:
  e.g. Is Art the parent of Bob and the parent of Bud?
  e.g. ____ has sons and no daughters?
Syntax
A **constant** is a string of lower case letters, digits, underscores, and periods *or* a string of ascii characters within double quotes.

Examples:

```
joe, bill, cs151, 3.14159
person, worksfor, office
the_house_that_jack_built,
"Mind your p’s & q’s!"
```

Non-examples:

```
Art, p&q, the-house-that-jack-built
```
Symbols / object constants represent objects.
   joe, bill, harry, a23, 3.14159
   the_house_that_jack_built
   “Mind your p’s & q’s!”

Constructors / function constants represent functions.
   pair, triple, set

Predicates / relation constants represent relations.
   person, parent, prefers
The **arity** of a constructor or a predicate is the number of arguments that can be associated with the constructor or predicate in writing complex expressions in the language.

**Unary** predicate (1 argument): \texttt{person(joe)}

**Binary** predicate (2 arguments): \texttt{parent(art,bob)}

**Ternary** predicate (3 arguments): \texttt{prefers(art,bob,bea)}

In defining vocabulary, we sometimes notate the arity of a constructor or predicate by annotating with a slash and the arity, e.g. \texttt{male/1, parent/2, and prefers/3}.
A variable is either a lone underscore or a string of letters, digits, underscores beginning with an upper case letter.

\[
\begin{array}{ccc}
X & Y23 & \text{Somebody} & _
\end{array}
\]
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 negation is a *negative literal*. 
goal(a,b) :- p(a,b) & ~q(b)
Query Rules with Variables

\[
\text{goal}(X,Y) : \rightsquigarrow p(X,Y) \land \lnot q(Y)
\]

\text{head} \quad \text{subgoal} \quad \text{subgoal} \quad \text{body}
A **query** is a non-empty, finite set of query rules.

\[
\begin{align*}
\text{goal}(X,Y) & :\!\!\!\!\!:- \ p(X,Y) & \& & q(X) \\
\text{goal}(X,Y) & :\!\!\!\!\!:- \ p(X,Y) & \& & \neg q(Y)
\end{align*}
\]

NB: The IDEs for most Logic Programming systems (including Sierra) do not support queries with multiple rules.
Semantics
An **instance of a rule** is a rule in which all variables have been consistently replaced by ground terms.

**Rule**

\[
goal(X,Y) :\neg p(X,Y) \& \neg q(Y)
\]

**Herbrand Universe**

\[
\{a, b\}
\]

**Instances**

\[
goal(a,a) :\neg p(a,a) \& \neg q(a)
\]
\[
goal(a,b) :\neg p(a,b) \& \neg q(b)
\]
\[
goal(b,a) :\neg p(b,a) \& \neg q(a)
\]
\[
goal(b,b) :\neg p(b,b) \& \neg q(b)
\]
The result of applying a rule $r$ to a dataset $\Delta$ (written $\nu(r,\Delta)$) is the set of all $\psi$ such that (1) $\psi$ is the head of an arbitrary instance of $r$, (2) every positive subgoal in the instance is a member of $\Delta$, and (3) no negative subgoal in the instance is a member of $\Delta$. 
Example

Dataset

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

Rule

\[ \text{goal}(X, Y) :- p(X, Y) \land \neg p(Y, X) \]

Result

\[ \text{goal}(a, b) \]
\[ \text{goal}(b, c) \]
Non-Examples

Dataset

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

Rule

\[ \text{goal}(X,Y) :- \ p(X,Y) \ & \ \sim p(Y,X) \]

Not Results

\[ \text{goal}(a,b) \]
\[ \text{goal}(b,c) \]
\[ \text{goal}(c,d) \]
\[ \text{goal}(d,c) \]
The extension of a query (written $E(\Omega,\Delta)$), is the set of all facts that can be “deduced" form our dataset $\Delta$ on the basis of the rules in the program, i.e. it is the union of $v(r_i,\Delta)$ for each $r_i$ in our query $\Omega$.

NB: For our language, there one and only one extension for any dataset and ruleset, i.e. the extension is unique.
Safety
Ruleset

\[ s(X, Y, Z) : - p(X, Y) \]

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

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

Dataset

\[
\begin{align*}
p(a,b) & \\
p(a,c) & \\
q(c,d) & \\
\end{align*}
\]

Output

\[
\begin{align*}
t(a,b) & \\
t(a,c) & \\
\end{align*}
\]

Alternative Rules

\[
\begin{align*}
t(X,Y) :\neg p(X,Y) & \neg qqq(Y) \\
qqq(Y) :\neg q(Y,Z) & \\
\end{align*}
\]
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:

\[
\text{goal}(X,Z) :- p(X,Y) \land q(Y,Z) \land \neg r(X,Y)
\]

Unsafe Rule:

\[
\text{goal}(X,Z) :- p(X,Y) \land q(Y,X)
\]

Unsafe Rule:

\[
\text{goal}(X,Y) :- p(X,Y) \land \neg q(Y,Z)
\]
Predefined Concepts
Predefined Concepts

Functions
  Arithmetic Functions (e.g. plus, times, min, max, etc.)
  String functions (e.g. concatenate, string matching, etc.)
  Other (e.g. converting between formulas and strings, etc.)
  Aggregates (e.g. sets of objects with given properties)

Relations
  Equality and Inequality
same and distinct

Identity

\text{same}(t_1, t_2) \text{ is true iff } t_1 \text{ and } t_2 \text{ are identical}

Difference

\text{distinct}(t_1, t_2) \text{ is true iff } t_1 \text{ and } t_2 \text{ are different}

NB: This is the same as ordinary equality (e.g. 2+2 = 2*2)

\neg \text{same} (\text{plus}(2,2), \text{times}(2,2))
\text{distinct}(\text{plus}(2,2), \text{times}(2,2))
Evaluable term - constant, variable, \( f(t_1, \ldots, t_n) \)

- \( f \) is a predefined function or user-defined function
- \( t_1, \ldots, t_n \) are evaluable terms

Examples

- \( \text{plus}(2, 3) \)
- \( \text{stringappend}(\text{"abc"}, \text{"def"}) \)
- \( \text{stringify}(\text{vinay}) \)
- \( \text{symbolize}(\text{"vinay"}) \)

\[ \text{min}(\text{plus}(2, 3), \text{times}(2, 3)) \]

NB: Many predefined functions are variadic, e.g. \( \text{plus} \).
evaluate(x,v)
    x is a term with
    v is the standard value of x

Example
    evaluate(plus(2,2),4)

childless(X) :-
    evaluate(length(setofall(Y,parent(X,Y))),0)

numchildren(X,N) :-
    evaluate(length(setofall(Y,parent(X,Y))),N)

NB: unbound variables allowed in second argument only
Predefined Aggregates

setofall
countofall

Example

goal(X,N) :-
  person(X) &
  evaluate(countofall(Y,person(X,Y)),N)
Kinship
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bud,coe)
parent(bud,cory)
**Query**

\[
\text{goal}(X,Z) \quad :\quad \text{parent}(X,Y) \quad \& \quad \text{parent}(Y,Z)
\]

**Dataset:**

- parent(art,bob)
- parent(art,bea)
- parent(bob,cal)
- parent(bob,cam)
- parent(bud,coe)
- parent(bud,cory)

**Result:**

- goal(art,cal)
- goal(art,cam)
- goal(art,coe)
- goal(art,cory)
Query:

\[
\text{goal}(X) :- \text{parent}(X,Y) \\
\text{goal}(X) :- \text{parent}(Y,X)
\]

Dataset:

- parent(art,bob)
- parent(art,bea)
- parent(bob,cal)
- parent(bob,cam)
- parent(bea,coe)
- parent(bea,cory)

Result:

- goal(art)
- goal(bob)
- goal(bea)
- goal(cal)
- goal(coe)
- goal(cory)
**Query**

\[ \text{goal}(Y,Z) : \neg \text{parent}(X,Y) \land \text{parent}(X,Z) \land \text{distinct}(Y,Z) \]

**Dataset:**

<table>
<thead>
<tr>
<th>parent(art,bob)</th>
<th>parent(art,bea)</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent(bob,cal)</td>
<td>parent(bob,cam)</td>
</tr>
<tr>
<td>parent(bea,coe)</td>
<td>parent(bea,cory)</td>
</tr>
</tbody>
</table>

**Result:**

<table>
<thead>
<tr>
<th>goal(bob,bea)</th>
<th>goal(beat,bob)</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal(cal,cam)</td>
<td>goal(cam,cal)</td>
</tr>
<tr>
<td>goal(coe,cory)</td>
<td>goal(cory,coe)</td>
</tr>
</tbody>
</table>
Map Coloring
hue(red)
hue(green)
hue(blue)
hue(purple)
goal(C1, C2, C3, C4, C5, C6) :-
  hue(C1) & hue(C2) & hue(C3) & hue(C4) & hue(C5) & hue(C6) &
  distinct(C1, C2) & distinct(C1, C3) & distinct(C1, C5) &
  distinct(C1, C6) & distinct(C2, C3) & distinct(C2, C4) &
  distinct(C2, C5) & distinct(C2, C6) & distinct(C3, C4) &
  distinct(C3, C6) & distinct(C5, C6)