Logic Programming

Queries

Michael Genesereth
Computer Science Department
Stanford University
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 or 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_buil,
“Mind your p’s & q’s!”
```

*Same as before.*
Types of Constants

**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

*Same as before.*
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): \( \text{person}(joe) \)

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

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

In defining vocabulary, we sometimes notate the arity of a constructor or predicate by annotating with a slash and the arity, e.g. \text{male}/1, \text{parent}/2, and \text{prefers}/3.

*Same as before.*
Variables

A **variable** is either a lone underscore or a string of letters, digits, underscores, and periods beginning with an upper case letter.

```
X   Y23   Somebody   _
```
Symbols
  art
  bob

Variables
  X
  Y23

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

Query terms are not necessarily ground!
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.
goal(a,b) :- p(a,b) & ~q(b)
Sample Queries with Variables

\[
\text{goal}(X,b) \ :- \ p(X,b) \ & \ \neg q(b)
\]
\[
\text{goal}(X,b) \ :- \ p(X,Y) \ & \ \neg q(Y)
\]
\[
\text{goal}(X,X) \ :- \ p(X,Y) \ & \ \neg q(Y)
\]
\[
\text{goal}(X,f(Y)) \ :- \ p(X,Y) \ & \ \neg q(Y)
\]
\[
\text{goal}(X,Y) \ :- \ p(X,f(Y)) \ & \ \neg q(Y)
\]
A **query** is a non-empty, finite set of query rules.

\[
\text{goal}(X,Y) :\!\!\!\!\!\!: \ p(X,Y) \ & \ q(X) \\
\text{goal}(X,Y) :\!\!\!\!\!\!: \ p(X,Y) \ & \ \neg q(Y)
\]

**NB:** The IDEs for most Logic Programming systems (including Sierra) do *not* support queries with multiple rules. Reasons discussed later.
Semantics
p(a,b)  
p(b,c)  
p(c,d)  
p(d,c)  

+  

goal(X,Y) :- p(X,Y) & p(Y,X)  

=  

goal(c,d)  
goal(d,c)
An instance of a query is a query in which all variables have been consistently replaced by ground terms.

Rule

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

Herbrand Universe

\{a, b\}

Instances

\[ \begin{align*}
\text{goal}(a,a) & :\text{=} \ p(a,a) \ & \ \sim q(a) \\
\text{goal}(a,b) & :\text{=} \ p(a,b) \ & \ \sim q(b) \\
\text{goal}(b,a) & :\text{=} \ p(b,a) \ & \ \sim q(a) \\
\text{goal}(b,b) & :\text{=} \ p(b,b) \ & \ \sim q(b)
\end{align*} \]
The **result of applying a query to a dataset** is defined to be the set of all $\psi$ such that

1. $\psi$ is the *head* of an instance of the rule,
2. every positive subgoal in the instance is in the dataset,
3. no negative subgoal is in the dataset.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a,b)</td>
<td>goal(c,d)</td>
</tr>
<tr>
<td>p(b,c)</td>
<td>goal(d,c)</td>
</tr>
<tr>
<td>p(c,d)</td>
<td></td>
</tr>
<tr>
<td>p(d,c)</td>
<td></td>
</tr>
</tbody>
</table>

**Rule**

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

**Positive instances (2)**

\[
\text{goal}(c,d) :- p(c,d) \land p(d,c)
\]

\[
\text{goal}(d,c) :- p(d,c) \land p(c,d)
\]

**Negative instances (14)**

\[
\text{goal}(a,b) :- p(a,b) \land p(b,a)
\]

\[
\text{goal}(b,c) :- p(b,c) \land p(b,a)
\]

...
### Example

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a,b)</td>
<td>goal(a,b)</td>
</tr>
<tr>
<td>p(b,c)</td>
<td>goal(b,c)</td>
</tr>
<tr>
<td>p(c,d)</td>
<td></td>
</tr>
<tr>
<td>p(d,c)</td>
<td></td>
</tr>
</tbody>
</table>

### Rule

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

### Positive instances (2)

\[
\begin{align*}
goal(a,b) & :- p(a,b) \land \neg p(b,a) \\
goal(b,c) & :- p(b,c) \land \neg p(c,b)
\end{align*}
\]

### Negative instances (14)

\[
\begin{align*}
goal(a,c) & :- p(a,c) \land \neg p(c,a) \\
goal(c,d) & :- p(c,d) \land \neg p(d,c) \\
& \ldots
\end{align*}
\]
Quiz

Dataset
   p(a,b)
   p(b,c)
   p(c,d)
   p(d,c)

Query
   goal(X) :- p(X,Y) & p(Y,X)

Result
   goal(c)
   goal(d)
Quiz

Dataset

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

Query

- goal(X,X) :- p(X,Y) & p(Y,X)

Result

- goal(c,c)
- goal(d,d)
Quiz

**Dataset**

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

**Query**

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

**Result**

\[ \text{goal}(c, b) \]
\[ \text{goal}(d, b) \]
Quiz

Dataset
  p(a,b)
  p(b,c)
  p(c,d)
  p(d,c)

Query
  goal(X,f(X)) :- p(X,Y) & p(Y,X)

Result
  goal(c,f(c))
  goal(d,f(d))
Non-Examples

Dataset

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

Rule

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

Not Results

\[ \text{goal}(c, d) \]
\[ \text{goal}(a, b) \]
\[ \text{goal}(b, c) \]
\[ \text{goal}(c, d) \]
\[ \text{goal}(d, c) \]

Too few. Too many.
The result of applying a set of queries to a dataset is the union of the results of applying the queries to the dataset.

Dataset: \{p(a,b), p(b,c)\}

\[
\begin{align*}
goal(X) & :- p(X,Y) & \{goal(a), goal(b)\} \\
goal(Y) & :- p(X,Y) & \{goal(b), goal(c)\}
\end{align*}
\]

Extension: \{goal(a), goal(b), goal(c)\}

 NB: A query set is effectively a disjunction.

 NB: Most logic programming systems (including Sierra) do not support query sets directly. They are handled indirectly, as discussed later.
Safety
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:
\[
\text{goal}(X,Z) :- \ p(X,Y) \ & \ q(Y,Z) \ & \ \neg r(X,Y)
\]

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

Unsafe Rule:
\[
\text{goal}(X,Y) :- \ p(X,Y) \ & \ \neg q(Y,Z)
\]
Rule

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

Herbrand Universe \{a, b\}

Dataset \{p(a,a)\}

Instances

\[
\begin{align*}
goal(a,a,a) :- & \ p(a,a) \\
goal(a,b,a) :- & \ p(a,b) \\
goal(b,a,a) :- & \ p(b,a) \\
goal(b,b,a) :- & \ p(b,b) \\
goal(a,a,b) :- & \ p(a,a) \\
goal(a,b,b) :- & \ p(a,b) \\
goal(b,a,b) :- & \ p(b,a) \\
goal(b,b,b) :- & \ p(b,b)
\end{align*}
\]

Results

\[
\begin{align*}
goal(a,a,a) \\
goal(a,a,b) \\
goal(b,a,a) \\
goal(b,b,a) \\
goal(a,a,b) \\
goal(a,b,b) \\
goal(b,a,b) \\
goal(b,b,b)
\end{align*}
\]
### Rule

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

### Herbrand Universe

\{a, b, f(a), f(b), f(f(a)), \ldots\}

### Dataset

\{p(a, a)\}

<table>
<thead>
<tr>
<th>Instances</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal(a, a, a) :- p(a, a)</td>
<td>goal(a, a, a)</td>
</tr>
<tr>
<td>goal(a, a, b) :- p(a, a)</td>
<td>goal(a, a, b)</td>
</tr>
<tr>
<td>goal(a, a, f(a)) :- p(a, a)</td>
<td>goal(a, a, f(a))</td>
</tr>
<tr>
<td>goal(a, a, f(b)) :- p(a, a)</td>
<td>goal(a, a, f(b))</td>
</tr>
<tr>
<td>goal(a, a, f(f(a))) :- p(a, a)</td>
<td>goal(a, a, f(f(a)))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Unsafe Rule
\[ \text{goal}(X,Y) \leftarrow p(X,Y) \land \neg q(Y,Z) \]

Herbrand Universe \{a, b, c, d\}

Dataset \{p(a,b), p(a,c), q(c,d)\}

Possible Meanings
Find all \( X \) and \( Y \) such that \( p(X,Y) \) is true and there is no \( Z \) for which \( q(Y,Z) \) is true.
\[ \{\text{goal}(a,b)\} \]

Find all \( X \) and \( Y \) such that \( p(X,Y) \) is true and there is some \( Z \) for which \( q(Y,Z) \) is false.
\[ \{\text{goal}(a,b), \text{goal}(a,c)\} \]
Unsafe Rule

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

Herbrand Universe \{a, b, c, d\}

Dataset \{p(a,b), p(a,c), q(c,d)\}

Instances

\[
\begin{align*}
\text{goal}(a,b) & :- p(a,b) \& \neg q(b,a) & \text{goal}(a,b) \\
\text{goal}(a,c) & :- p(a,c) \& \neg q(c,a) & \text{goal}(a,c)
\end{align*}
\]

Results

\[
\begin{align*}
\text{goal}(a,b) & \\
\text{goal}(a,c) & \\
\text{goal}(a,c) & :- p(a,c) \& \neg q(c,d)
\end{align*}
\]
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
**Evaluable Terms**

**Evaluable term** - constant, variable, \( f(t_1, \ldots, t_n) \)
- \( f \) is a predefined function or user-defined function (*later*)
- \( t_1, \ldots, t_n \) are evaluable terms

**Examples**

- \( \text{plus}(2,3) \) \[\rightarrow 5\]
- \( \text{stringappend}("abc","def") \) \[\rightarrow "abcdef"\]
- \( \text{stringify}(\text{vinay}) \) \[\rightarrow "\text{vinay}"\]
- \( \text{symbolize}("\text{vinay}"") \) \[\rightarrow \text{vinay}\]
- \( \text{min}(\text{plus}(2,3),\text{times}(2,3)) \) \[\rightarrow 5\]

NB: Many predefined functions are variadic, e.g. \( \text{plus} \).
Dataset \{h(a,2), w(a,3), h(b,4), w(b,2)\}

Possible Rule
\[
\text{goal}(X, \text{times}(H,W)) :\text{=} h(X,H) \& w(X,W)
\]

Results
\[
\text{goal}(a, \text{times}(2,3)) \\
\text{goal}(b, \text{times}(4,2))
\]
evaluate(x,v)
   x is a term
   v is the value of x

Examples
   goal :- evaluate(times(2,3),6)

   goal :- evaluate(plus(times(2,3),4),10)

   goal(X,A) :-
       h(X,H) & w(X,W) & evaluate(times(H,W),A)

Safety: unbound variables allowed in second argument only.
Example

\[
\text{goal}(Z) :- \\
\quad \text{evaluate}\left(\text{min}(\text{plus}(2,3), \text{times}(2,3)), Z\right)
\]

Result

\[
\text{goal}(5)
\]
**Aggregate Terms**

**Aggregate operators** are used to create sets of answers as terms and then count, add, average those sets.

**Predefined Aggregates**
- setofall
- countofall
**Aggregates**

**Dataset** \{p(a,b), p(a,c), p(b,d)\}

**Example**

\[
\text{goal}(X,L) :- \\
p(X,Y) & \\
\text{evaluate(countofall}(Z,p(a,Z)),L)
\]

**Result** \{goal(a,2), goal(b,1)\}

**Example**

\[
\text{goal}(X,L) :- \\
p(X,Y) & \\
\text{evaluate(setofall}(Z,p(a,Z)),L)
\]

**Result** \{goal(a,[b,c]), goal(b,[d])\}
Identity

same(t₁,t₂) is true iff t₁ and t₂ are identical

Difference

distinct(t₁,t₂) is true iff t₁ and t₂ are different
mutex(t₁,...,tn) is true iff t₁,...,t₂ are all different

Examples

same(a,a) is true
same(a,b) is false

distinct(a,a) is false
distinct(a,b) is true
mutex(a,b,c) is true

Safety: No unbound variables allowed!!!
NB: This is not ordinary equality (e.g. $2+2 = 4$)

\[
\begin{align*}
\text{same}(\text{plus}(2,2),4) & \quad \text{is false} \\
\text{distinct}(\text{plus}(2,2),4) & \quad \text{is true}
\end{align*}
\]

NB: Use evaluate to get values

\[
\begin{align*}
\text{evaluate}(\text{plus}(2,2),V) \ & \& \text{same}(V,4) & \quad \text{is true} \\
\text{evaluate}(\text{plus}(2,2),4) & \quad \text{is true}
\end{align*}
\]
Epilog provides a means to define new evaluable functions in terms of existing functions.

**Example**

\[
f(X) := \text{plus}(\text{pow}(X,2), \text{times}(2,X), 1)
\]

\[
goal(Z) := \text{evaluate}(f(3), Z)
\]

User-defined functions are quite useful in practice because they make some rules more readable and they can be evaluated very efficiently.

_**NB:** We won't be talking more about user-defined functions._
\begin{align*}
p(a, b) \\
p(b, c) \\
p(c, d)
\end{align*}

Pattern: \( \text{goal}(X, Z) \)

Query: \( p(X, Y) \land p(Y, Z) \)
Lambda

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

Pattern \( \text{goal}(X,Z) \)
Query \( p(X,Y) \land p(Y,Z) \)

goal(a,c)
goal(b,d)
Lambda

p(a,b)
p(b,c)
p(c,d)

Query

Pattern: goal(X,Z)
Query: p(X,Y) & p(Y,Z)

goal(a,c)
goal(b,d)
Lambda

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

Query

Pattern: \( \text{goal}(X, Z) \)
Query: \( p(X, Y) \land p(Y, Z) \)

100 result(s)
Autorefresh
Lambda

Pattern: goal(X, Z)
Query: p(X,Y) & p(Y,Z)

goal(a,c)
goal(b,d)
Lambda

\begin{align*}
p(a,b) \\
p(b,c) \\
p(c,d) \\
p(d,e)
\end{align*}

\begin{align*}
\text{Pattern} & : \text{goal}(X,Z) \\
\text{Query} & : p(X,Y) \land p(Y,Z)
\end{align*}

\begin{align*}
\text{goal}(a,c) \\
\text{goal}(b,d) \\
\text{goal}(c,e)
\end{align*}