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 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
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
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): \( \text{person}(\text{joe}) \)
**Binary** predicate (2 arguments): \( \text{parent}(\text{art}, \text{bob}) \)
**Ternary** predicate (3 arguments): \( \text{prefers}(\text{art}, \text{bob}, \text{bea}) \)

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.
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))
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 \textit{positive literal}.
A negations is a \textit{negative literal}.
goal(a,b) :- p(a,b) & ~q(b)

Query Rules

subgoal subgoal

head body
Query Rules with Variables

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

\[
\text{goal}(x,b) :- p(x,b) \land \neg q(b)
\]

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

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

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

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

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

**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**

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

**Herbrand Universe**

\[
\{a, b\}
\]

**Instances**

\[
goal(a, a) :- p(a, a) \land \neg q(a)
\]

\[
goal(a, b) :- p(a, b) \land \neg q(b)
\]

\[
goal(b, a) :- p(b, a) \land \neg q(a)
\]

\[
goal(b, b) :- p(b, b) \land \neg q(b)
\]
The result of applying a rule \( r \) to a dataset \( \Delta \) (written \( v(r,\Delta) \)) 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 \( \Delta \).
<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**

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

**Instances**

\[
goal(a,a) :- p(a,a) \land \neg p(a,a)
\]
\[
goal(a,b) :- p(a,b) \land \neg p(b,a)
\]
\[
goal(a,c) :- p(a,c) \land \neg p(c,a)
\]
\[
goal(a,d) :- p(a,d) \land \neg p(d,a)
\]
\[
\ldots
\]
\[
goal(b,c) :- p(b,c) \land \neg p(c,b)
\]
\[
\ldots
\]
\[
goal(c,d) :- p(c,d) \land \neg p(d,c)
\]
\[
\ldots
\]
Dataset
\[ p(a,b) \]
\[ p(b,c) \]
\[ p(c,d) \]
\[ p(d,c) \]

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

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" from a dataset $\Delta$ on the basis of all of the rules in the query, i.e. it is the union of $\nu(r_i, \Delta)$ for each $r_i$ in our query $\Omega$.

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

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

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

**NB:** *For our language, there is one and only one extension for any dataset and ruleset, i.e. the extension is unique.*
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:**
\[
\text{goal}(X, Y) :- \ p(X, Y) \ & \ \neg q(Y, Z)
\]
Unbound Variables in Head

Ruleset

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

Dataset

p(a,f(b))

Result

goal(a,f(b),a)
goal(a,f(b),b)
goal(a,f(b),f(a))
goal(a,f(b),f(b))
goal(a,f(b),f(f(a)))
goal(a,f(b),f(f(b)))

...
Unsafe Rule

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

Dataset

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

Result

- \( \text{goal}(a,b) \)
- \( \text{goal}(a,c) \)

English

Find all \( x \) and \( y \) such that \( p(x,y) \) is true and there is some \( z \) for which \( q(y,z) \) is false.

versus

Find all \( x \) and \( y \) such that \( p(x,y) \) is true and there is no \( z \) for which \( q(y,z) \) is true.
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
Identity

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

Difference

\[ \text{distinct}(t_1, t_2) \text{ is true iff } t_1 \text{ and } t_2 \text{ are different} \]
\[ \text{mutex}(t_1, \ldots, t_n) \text{ is true iff } t_1, \ldots, t_2 \text{ are all different} \]

NB: This is not ordinary equality (e.g. \(2+2 = 4\))

\[ \text{same}(\text{plus}(2,2),4) \text{ is false} \]
\[ \text{distinct}(\text{plus}(2,2),4) \text{ is true} \]

\[ \text{evaluate}(\text{plus}(2,2),V) \& \text{same}(V,4) \text{ is true} \]
\[ \text{evaluate}(\text{plus}(2,2),4) \text{ is true} \]
Evaluable term - constant, variable, f(t1,…tn)
  f is a predefined function or user-defined function
  t1,…,tn are evaluable terms

Examples
  plus(2,3)
  stringappend(“abc”,“def”)
  stringify(vinay)
  symbolize(“vinay”)

  min(plus(2,3),times(2,3))

NB: Many predefined functions are variadic, e.g. plus.
evaluate(x,v)
  x is a term
  v is the value of x

Examples
  evaluate(times(2,3),6)

  evaluate(plus(times(2,3),4),10)

goal(X,Z) :-
  height(X,H) & width(X,W) &
  evaluate(times(H,W),Z)

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

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

Query

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

result(s): 100
Autorefresh

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

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

100 result(s)

goal(a,c)
goal(b,d)
Pattern: \text{goal}(X, Z)\\
Query: \text{p}(X, Y) \land \text{p}(Y, Z)\\
goal(a, c)\\
goal(b, d)
p(a,b)  
p(b,c)  
p(c,d)  
p(d,e)  

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

Goal:  
goal(a,c)  
goal(b,d)  
goal(c,e)