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* ***or*** *the parent of Bud?*

e.g. ____ *has sons* ***and*** *no daughters?*

Syntax

Vocabulary

A **constant** is a string of lower case letters, digits, underscores, and periods *or* a string of ascii characters within double quotes.

```
a, b, c, 1, 2, 3, joe, cs151
f, g, pair, triple
p, q, r, person, parent, prefers
```

Same as before

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    _
```

Terms

Symbols

```
art
bob
```

Variables

```
X
Y23
```

Query terms are not necessarily ground!

Compound Terms

```
pair(art,bob)
pair(X,Y23)
pair(pair(art,bob),pair(X,Y23))
```

Atoms, Negations, and Literals

Atoms

```
p(a,b)
p(a,X)
p(Y,c)
```

Atoms are like factoids in datasets
except that
they may contain variables.

Negations

```
~p(a,b)
```

Literals (atoms or negations of atoms)

```
p(a,Y)
~p(a,Y)
```

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

Ground Query

$$\underbrace{\texttt{goal(a,b)}}_{\textit{head}}\ \texttt{:-}\ \underbrace{\overbrace{\texttt{p(a,b)}}^{\textit{subgoal}}\ \texttt{\&}\ \overbrace{\sim\texttt{q(b)}}^{\textit{subgoal}}}_{\textit{body}}$$

Intuitive meaning: `goal(a,b)` *is true if* `p(a,b)` *is true and* `q(b)` *is false*.

Queries May Contain Variables

```
goal(a,b) :- p(a,b) & ~q(b)

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

goal(X,b) :- p(X,b) & ~q(b)
goal(X,b) :- p(X,Y) & ~q(Y)
goal(X,f(Y)) :- p(X,Y) & ~q(Y)
goal(X,Y) :- p(X,f(Y)) & ~q(Y)
```

Semantics

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)
```

Instances

An **instance of a query** is a query in which all variables have been consistently replaced by ground terms.

Rule

```
goal(X,Y) :- p(X,Y) & ~q(Y)
```

Herbrand Universe

```
{a,b}
```

Instances

```
goal(a,a) :- p(a,a) & ~q(a)
goal(a,b) :- p(a,b) & ~q(b)
goal(b,a) :- p(b,a) & ~q(a)
goal(b,b) :- p(b,b) & ~q(b)
```

Query Result

The **result of applying a query to a dataset** is defined to be the set of all ψ such that

(1) ψ is the *head of an instance* of the rule,

(2) every *positive subgoal of the instance* is in the dataset,

(3) no *negative subgoal of the instance* is in the dataset.

Example

Dataset

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

Query

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

Result

```
goal(c,d)
goal(d,c)
```

Instances

```
  goal(a,a) :- p(a,a) & p(a,a)
  goal(a,b) :- p(a,b) & p(b,a)
  goal(a,c) :- p(a,c) & p(c,a)
  goal(a,d) :- p(a,d) & p(d,a)
  goal(b,a) :- p(b,a) & p(a,b)
  goal(b,b) :- p(b,b) & p(b,b)
  goal(b,c) :- p(b,c) & p(c,b)
  goal(b,d) :- p(b,d) & p(d,b)
  goal(c,a) :- p(c,a) & p(a,c)
  goal(c,b) :- p(c,b) & p(b,c)
  goal(c,c) :- p(c,c) & p(c,c)
→ goal(c,d) :- p(c,d) & p(d,c)
  goal(d,a) :- p(d,a) & p(a,d)
  goal(d,b) :- p(d,b) & p(b,d)
→ goal(d,c) :- p(d,c) & p(c,d)
  goal(d,d) :- p(d,d) & p(d,d)
```

Example

Dataset

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

Query

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

Result

```
goal(a,b)
goal(b,c)
```

Instances

```
  goal(a,a) :- p(a,a) & ~p(a,a)
→ goal(a,b) :- p(a,b) & ~p(b,a)
  goal(a,c) :- p(a,c) & ~p(c,a)
  goal(a,d) :- p(a,d) & ~p(d,a)
  goal(b,a) :- p(b,a) & ~p(a,b)
  goal(b,b) :- p(b,b) & ~p(b,b)
→ goal(b,c) :- p(b,c) & ~p(c,b)
  goal(b,d) :- p(b,d) & ~p(d,b)
  goal(c,a) :- p(c,a) & ~p(a,c)
  goal(c,b) :- p(c,b) & ~p(b,c)
  goal(c,c) :- p(c,c) & ~p(c,c)
  goal(c,d) :- p(c,d) & ~p(d,c)
  goal(d,a) :- p(d,a) & ~p(a,d)
  goal(d,b) :- p(d,b) & ~p(b,d)
  goal(d,c) :- p(d,c) & ~p(c,d)
  goal(d,d) :- p(d,d) & ~p(d,d)
```

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)
```

Instances

```
goal(a) :- p(a,a) & p(a,a)
goal(a) :- p(a,b) & p(b,a)
goal(a) :- p(a,c) & p(c,a)
goal(a) :- p(a,d) & p(d,a)
goal(b) :- p(b,a) & p(a,b)
goal(b) :- p(b,b) & p(b,b)
goal(b) :- p(b,c) & p(c,b)
goal(b) :- p(b,d) & p(d,b)
goal(c) :- p(c,a) & p(a,c)
goal(c) :- p(c,b) & p(b,c)
goal(c) :- p(c,c) & p(c,c)
goal(c) :- p(c,d) & p(d,c)
goal(d) :- p(d,a) & p(a,d)
goal(d) :- p(d,b) & p(b,d)
goal(d) :- p(d,c) & p(c,d)
goal(d) :- p(d,d) & p(d,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)
```

Instances

```
  goal(a,a) :- p(a,a) & p(a,a)
  goal(a,a) :- p(a,b) & p(b,a)
  goal(a,a) :- p(a,c) & p(c,a)
  goal(a,a) :- p(a,d) & p(d,a)
  goal(b,b) :- p(b,a) & p(a,b)
  goal(b,b) :- p(b,b) & p(b,b)
  goal(b,b) :- p(b,c) & p(c,b)
  goal(b,b) :- p(b,d) & p(d,b)
  goal(c,c) :- p(c,a) & p(a,c)
  goal(c,c) :- p(c,b) & p(b,c)
  goal(c,c) :- p(c,c) & p(c,c)
→ goal(c,c) :- p(c,d) & p(d,c)
  goal(d,d) :- p(d,a) & p(a,d)
  goal(d,d) :- p(d,b) & p(b,d)
→ goal(d,d) :- p(d,c) & p(c,d)
  goal(d,d) :- p(d,d) & p(d,d)
```

Quiz

Dataset

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

Query

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

Result

```
goal(c,b)
goal(d,b)
```

Instances

```
  goal(a,b) :- p(a,a) & p(a,a)
  goal(a,b) :- p(a,b) & p(b,a)
  goal(a,b) :- p(a,c) & p(c,a)
  goal(a,b) :- p(a,d) & p(d,a)
  goal(b,b) :- p(b,a) & p(a,b)
  goal(b,b) :- p(b,b) & p(b,b)
  goal(b,b) :- p(b,c) & p(c,b)
  goal(b,b) :- p(b,d) & p(d,b)
  goal(c,b) :- p(c,a) & p(a,c)
  goal(c,b) :- p(c,b) & p(b,c)
  goal(c,b) :- p(c,c) & p(c,c)
→ goal(c,b) :- p(c,d) & p(d,c)
  goal(d,b) :- p(d,a) & p(a,d)
  goal(d,b) :- p(d,b) & p(b,d)
→ goal(d,b) :- p(d,c) & p(c,d)
  goal(d,b) :- p(d,d) & p(d,d)
```

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))
```

Instances

```
  goal(a,f(a)) :- p(a,a) & p(a,a)
  goal(a,f(a)) :- p(a,b) & p(b,a)
  goal(a,f(a)) :- p(a,c) & p(c,a)
  goal(a,f(a)) :- p(a,d) & p(d,a)
  goal(b,f(b)) :- p(b,a) & p(a,b)
  goal(b,f(b)) :- p(b,b) & p(b,b)
  goal(b,f(b)) :- p(b,c) & p(c,b)
  goal(b,f(b)) :- p(b,d) & p(d,b)
  goal(c,f(c)) :- p(c,a) & p(a,c)
  goal(c,f(c)) :- p(c,b) & p(b,c)
  goal(c,f(c)) :- p(c,c) & p(c,c)
→ goal(c,f(c)) :- p(c,d) & p(d,c)
  goal(d,f(d)) :- p(d,a) & p(a,d)
  goal(d,f(d)) :- p(d,b) & p(b,d)
→ goal(d,f(d)) :- p(d,c) & p(c,d)
  goal(d,f(d)) :- p(d,d) & p(d,d)
```

Non-Examples

Dataset

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

Query

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

Not Results

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

Too few.	Too many.

Query Sets

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)}
```

Queries

```
goal(X) :- p(X,Y)  ⟶  {goal(a), goal(b)}
goal(Y) :- p(X,Y)  ⟶  {goal(b), goal(c)}
```

Result

```
{goal(a),goal(b),goal(c)}
```

NB: A query set is effectively a disjunction.

Safety

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:

```
goal(X,Z) :- p(X,Y) & q(Y,Z) & ~r(X,Y)
```

Unsafe Rule:

```
goal(X,Z) :- p(X,Y) & q(Y,X)
```

Unsafe Rule:

```
goal(X,Y) :- p(X,Y) & ~q(Y,Z)
```

Unbound Variables in Head

Rule

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

Herbrand Universe {a, b}

Dataset {p(a,a)}

Instances	Results
goal(a,a) :- p(a,a)	goal(a,a)
goal(a,a) :- p(a,b)	
goal(b,a) :- p(b,a)	
goal(b,a) :- p(b,b)	
goal(a,b) :- p(a,a)	goal(a,b)
goal(a,b) :- p(a,b)	
goal(b,b) :- p(b,a)	
goal(b,b) :- p(b,b)	

Unbound Variables in Head

Rule

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

Herbrand Universe {`a`, `b`, `f(a)`, `f(b)`, `f(f(a))`, ...}

Dataset {`p(a,a)`}

Instances	**Results**
`goal(a,a) :- p(a,a)`	`goal(a,a)`
`goal(a,b) :- p(a,a)`	`goal(a,b)`
`goal(a,f(a)) :- p(a,a)`	`goal(a,f(a))`
`goal(a,f(b)) :- p(a,a)`	`goal(a,f(b))`
`goal(a,f(f(a))) :- p(a,a)`	`goal(a,f(f(a)))`
...	...

Unbound Variables in Negation

Query

```
goal(X) :- p(X,Y) & ~p(Y,Z)
```

Herbrand Universe {a, b, c}

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

What is the result?

Possible Meanings

Query

```
goal(X) :- p(X,Y) & ~p(Y,Z)
```

Possible Meanings

Find all `X` such that `p(X,Y)` is true and there is *no* `Z` for which `p(Y,Z)` is *true*.

Find all `X` such that `p(X,Y)` is true and there is *some* `Z` for which `p(Y,Z)` is *false*.

Possible Meanings

Query

```
goal(X) :- p(X,Y) & ~p(Y,Z)
```

Herbrand Universe `{a, b, c}`

Dataset `{p(a,b),p(b,c)}`

Results

Find all `X` such that `p(X,Y)` is true and there is *no* `Z` for which `p(Y,Z)` is *true*.

`{goal(b)}`

Find all `X` such that `p(X,Y)` is true and there is *some* `Z` for which `p(Y,Z)` is *false*.

`{goal(a),goal(b)}`

Unbound Variables in Negation

Unsafe Rule

```
goal(X) :- p(X,Y) & ~p(Y,Z)
```

Herbrand Universe {a, b, c}

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

Instances	Results
...	
goal(a) :- p(a,b) & ~p(b,a)	goal(a)
goal(a) :- p(a,b) & ~p(b,b)	goal(a)
goal(a) :- p(a,b) & ~p(b,c)	
...	
goal(b) :- p(b,c) & ~p(c,a)	goal(b)
goal(b) :- p(b,c) & ~p(c,b)	goal(b)
goal(b) :- p(b,c) & ~p(c,c)	goal(b)
...	

Predefined Concepts

Predefined Concepts

Evaluable 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)

Evaluable Relations

`evaluate`

`same`, `distinct`, `mutex`

Evaluable Terms

Evaluable term - constant, variable, $f(t_1, \ldots, t_n)$

f is a predefined function or user-defined function (*later*)

i.e. ***not*** *a constructor / function constant*

$t_1, \ldots, t_n$ are evaluable terms

Examples

```
plus(2,3)                      →  5
stringappend(“abc”,”def”)      →  "abcdef"
stringify(vinay)               →  "vinay"
symbolize(“vinay”)             →  vinay

min(plus(2,3),times(2,3))      →  5
```

Many predefined functions are variadic, e.g. `plus(2,3,4)`.

Possibly Surprising Results

Dataset `{h(a,2),w(a,3),h(b,4),w(b,2)}`

Possible Rule

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

Results

```
goal(a,times(2,3))
goal(b,times(4,2))
```

Evaluate Predicate

`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.

Nesting Okay

Example

```
goal(Z) :-
  evaluate(min(plus(2,3),times(2,3)),Z)
```

Result

```
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
```

Dataset `{p(a,b),p(a,c),p(b,d)}`

Examples

```
countofall(Z,p(a,Z))      →  2
setofall(Z,p(a,Z))        →  [b,c]
```

Aggregates

Dataset `{p(a,b),p(a,c),p(b,d)}`

Example

```
goal(X,L) :-
  p(X,Y) &
  evaluate(countofall(Z,p(X,Z)),L)
```

Result `{goal(a,2),goal(b,1)}`

Example

```
goal(X,L) :-
  p(X,Y) &
  evaluate(setofall(Z,p(X,Z)),L)
```

Result `{goal(a,[b,c]),goal(b,[d])}`

same, distinct, mutex

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!!!

`same`, `distinct`, `mutex`

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

`same(plus(2,2),4)`	is false
`distinct(plus(2,2),4)`	is true

NB: Use evaluate to get ordinary equality

`evaluate(plus(2,2),V) & same(V,4)`	is true
`evaluate(plus(2,2),4)`	is true

Documentation

http://epilog.stanford.edu/documentation/epilog/vocabulary.php

User Defined Functions

Epilog provides a means to define new evaluable functions in terms of existing functions.

Example

```
f(X) := plus(pow(X,2),times(2,X),1)

goal(Z) :- 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.

Sierra

http://epilog.stanford.edu/sierra/sierra.html