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

Constants

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!”`

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.

Arity

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): `person(joe)`
Binary predicate (2 arguments): `parent(art,bob)`
Ternary predicate (3 arguments): `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. `male/1`, `parent/2`, and `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   _
```

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)
```

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*.

Query

subgoal *subgoal*

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

head *body*

Sample Queries with Variables

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

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

Queries

A **query** is a non-empty, finite set of query rules.

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

NB: The IDEs for most Logic Programming systems (including Sierra) do *not* support queries with multiple rules. Reasons discussed later.

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 in the instance is in the dataset,

(3) no negative subgoal is in the dataset.

Example

Dataset

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

Result

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

Rule

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

Positive instances (2)

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

Negative instances (14)

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

Example

Dataset

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

Result

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

Rule

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

Positive instances (2)

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

Negative instances (14)

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

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

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

Result

```
goal(c,b)
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

```
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)`}

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

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

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

Herbrand Universe {a, b}

Dataset {p(a,a)}

Instances

```
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)
```

Results

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

Unbound Variables in Head

Rule

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

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

Dataset {p(a,a)}

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

Unbound Variables in Negation

Unsafe Rule

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

{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*.

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

Unbound Variables in Negation

Unsafe Rule

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

Herbrand Universe {a, b, c, d}

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

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

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

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

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

NB: Many predefined functions are variadic, e.g. plus.

Unexpected 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
```

Aggregates

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

Example

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

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

Example

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

`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

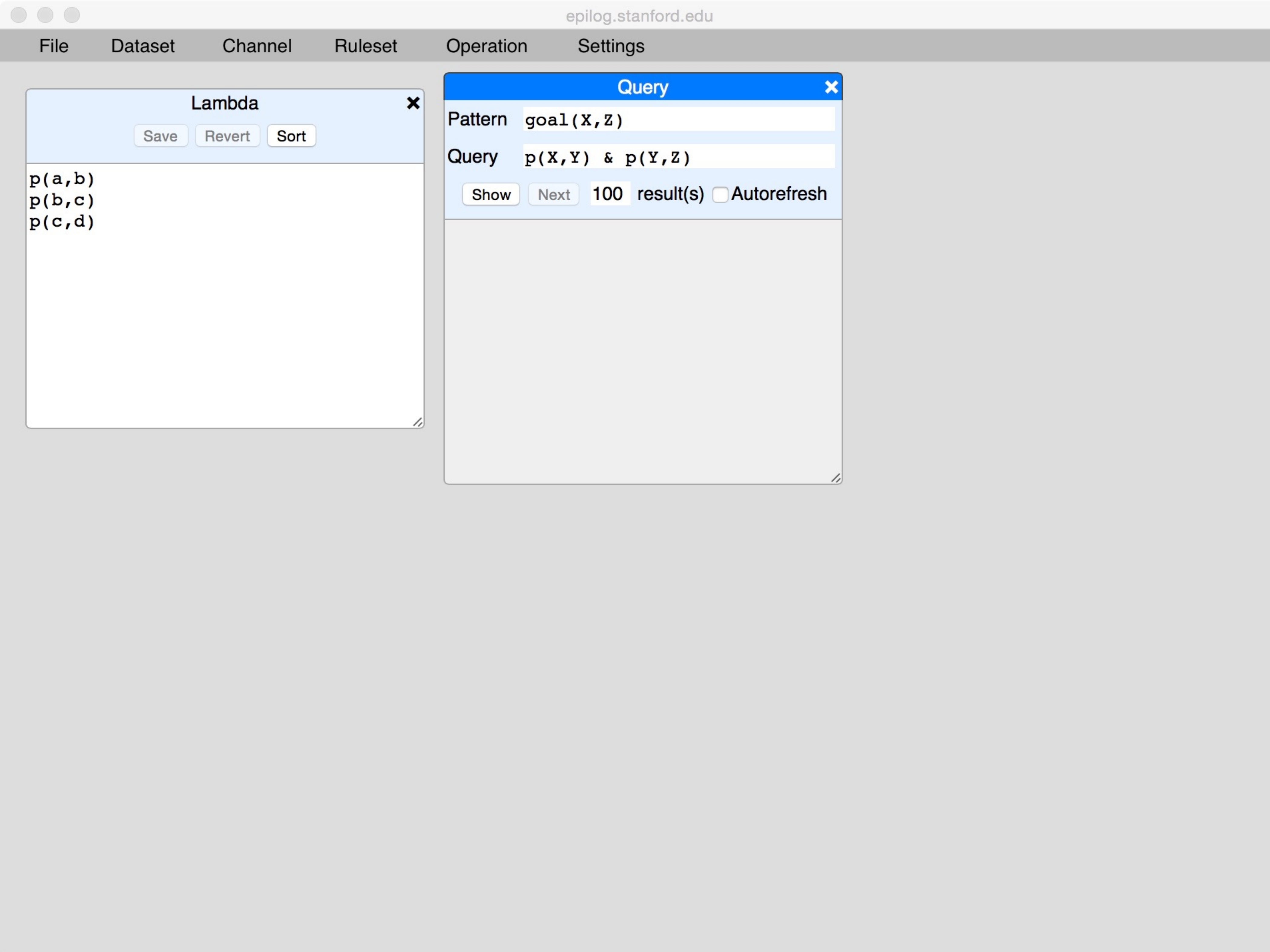
```
f(X) := plus(pow(X,2),times(2,X),1)

goal(Z) :- evaluate(f(3),Z)
```

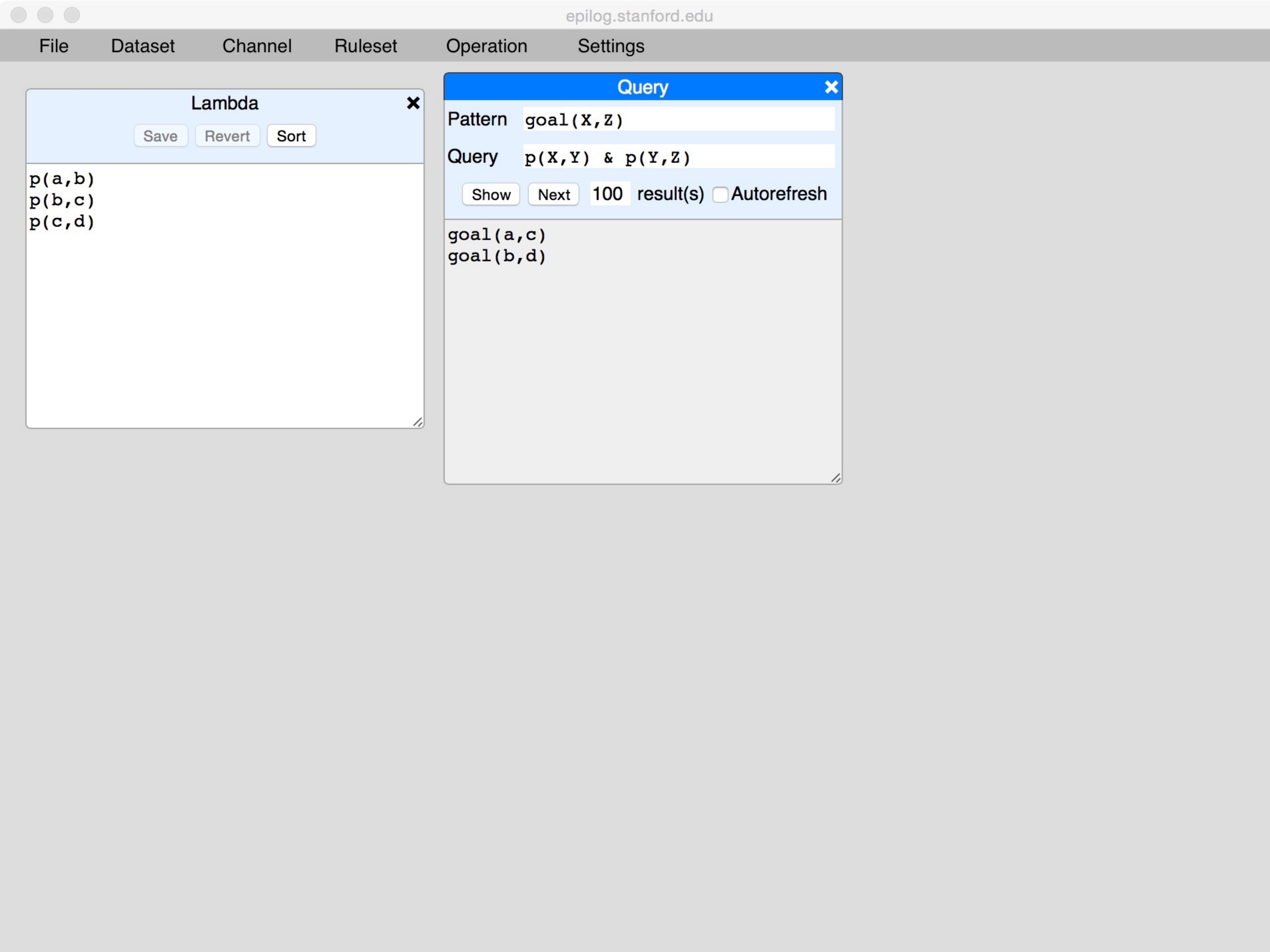
User-defined functions are quite useful in practice because they make some rules more readable and they can be evaluated very efficiently.

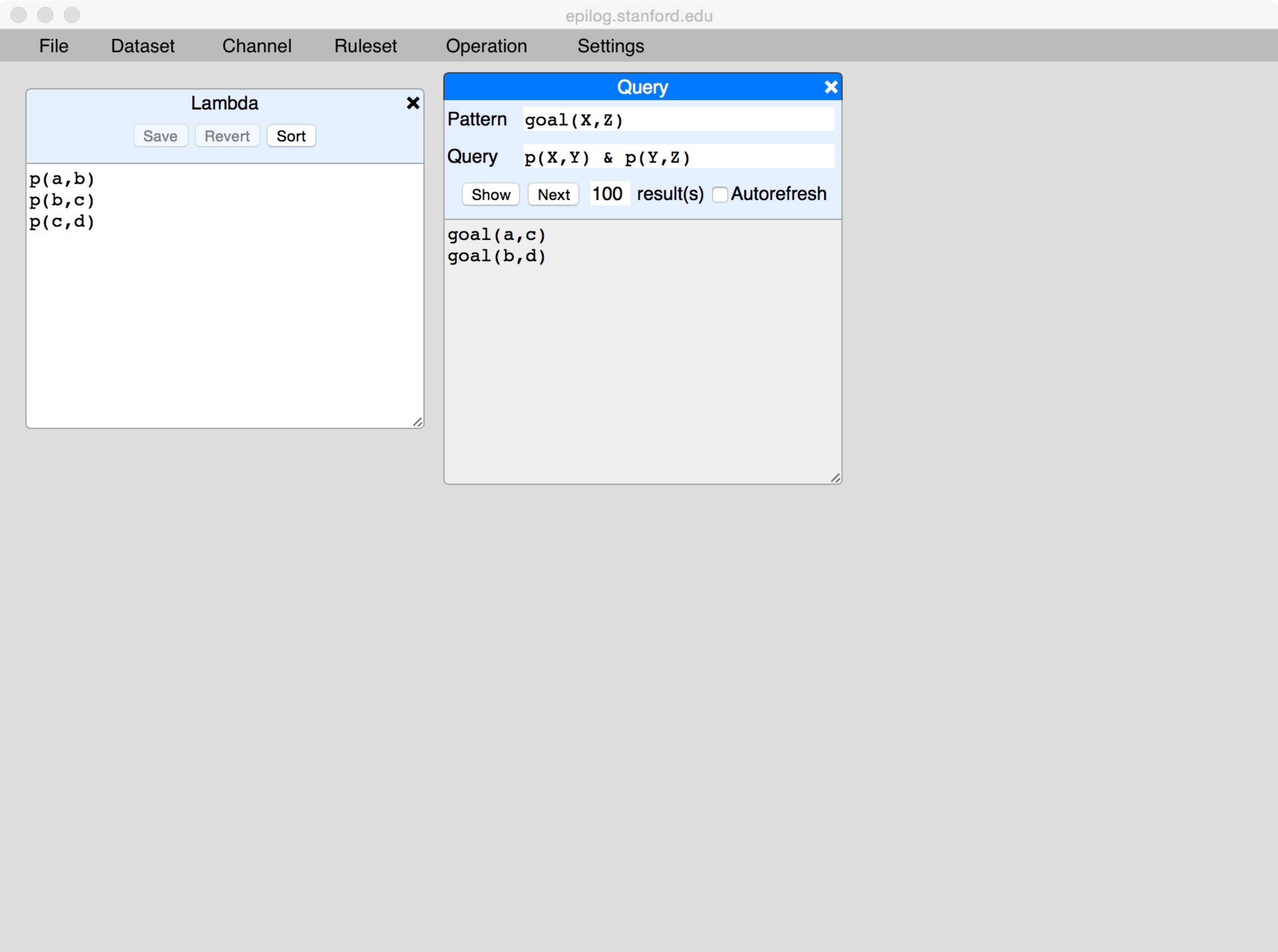
NB: We won't be talking more about user-defined functions.

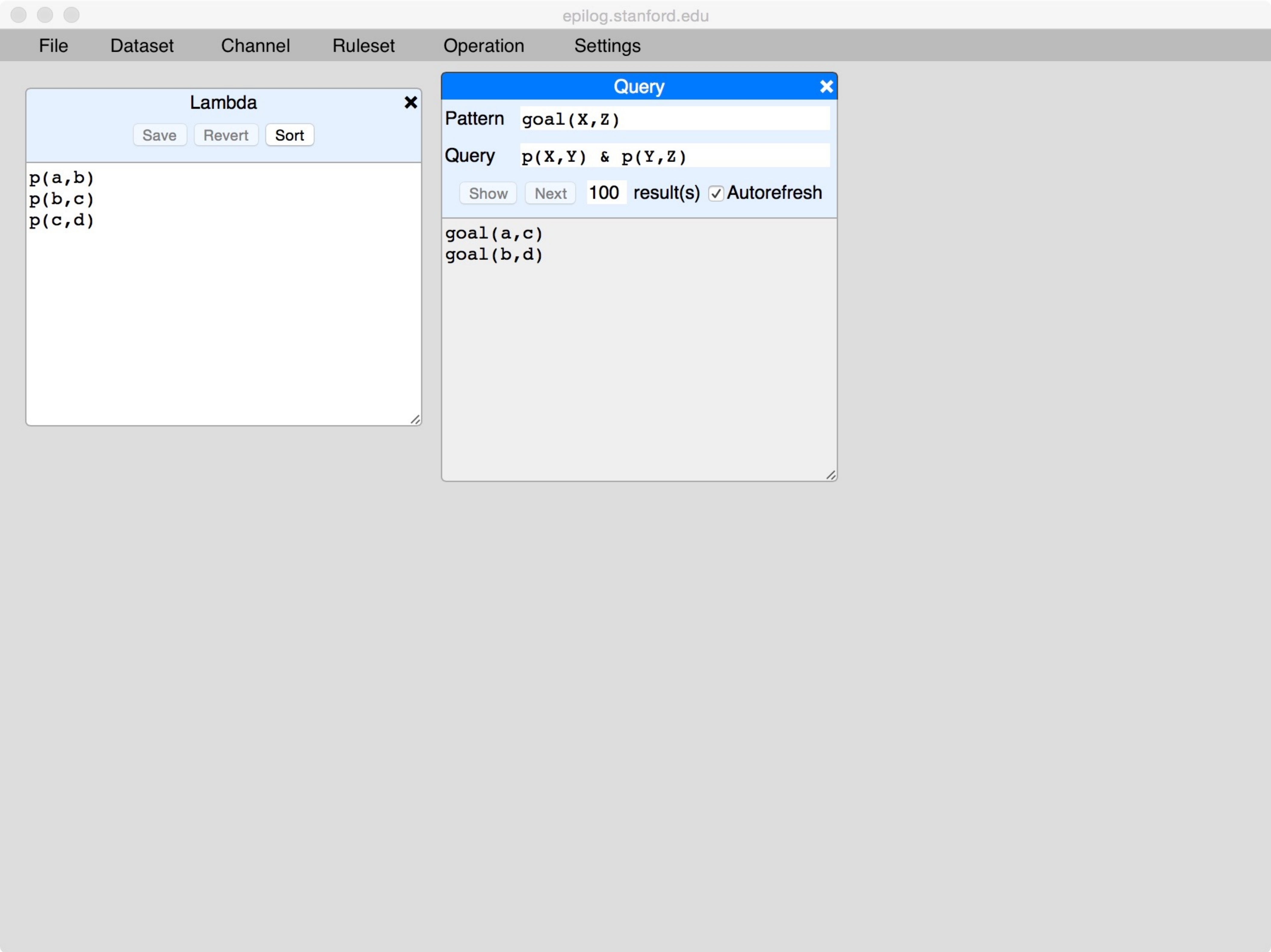
Sierra

http://epilog.stanford.edu/homepage/sierra.php

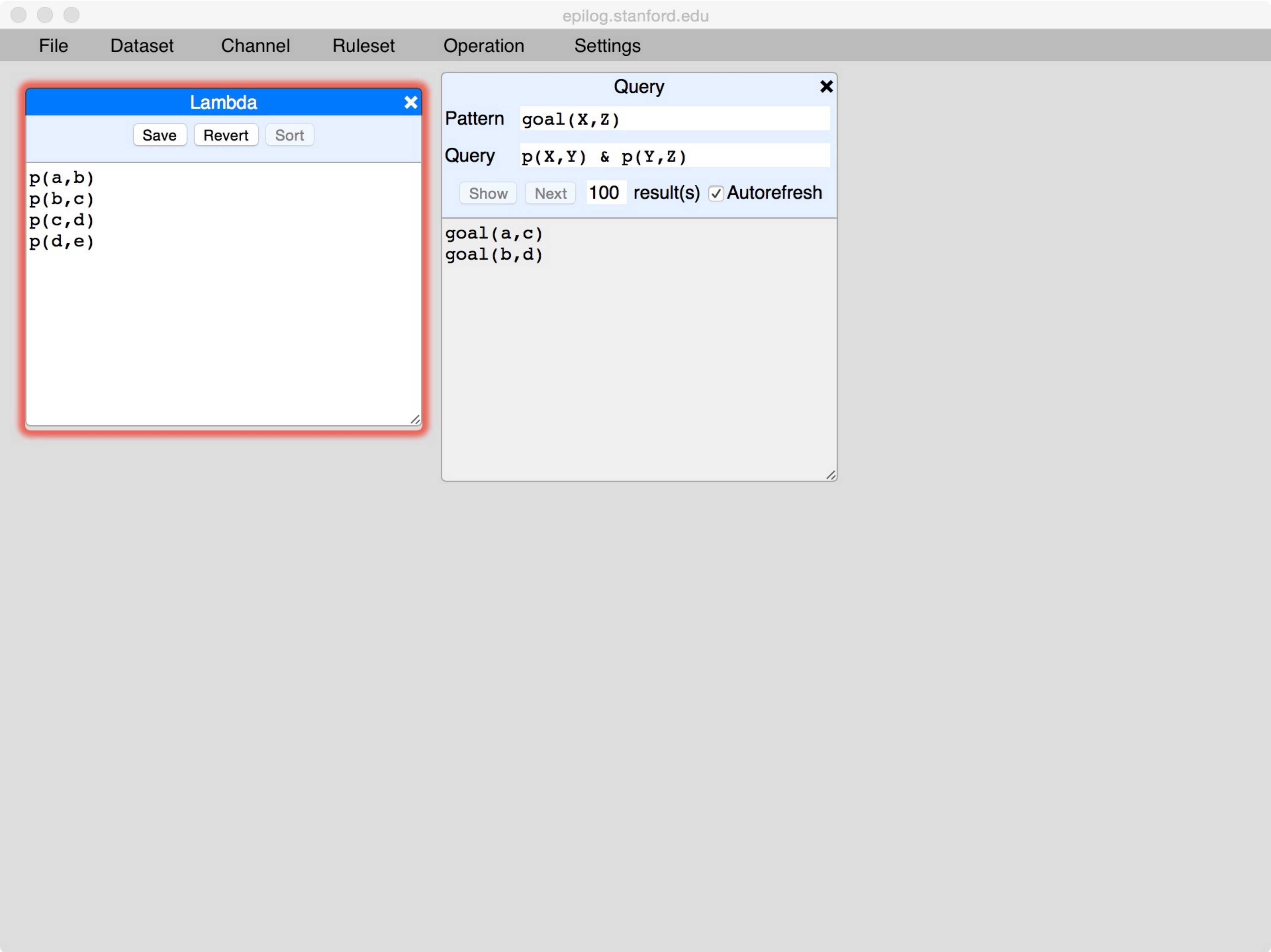

epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh

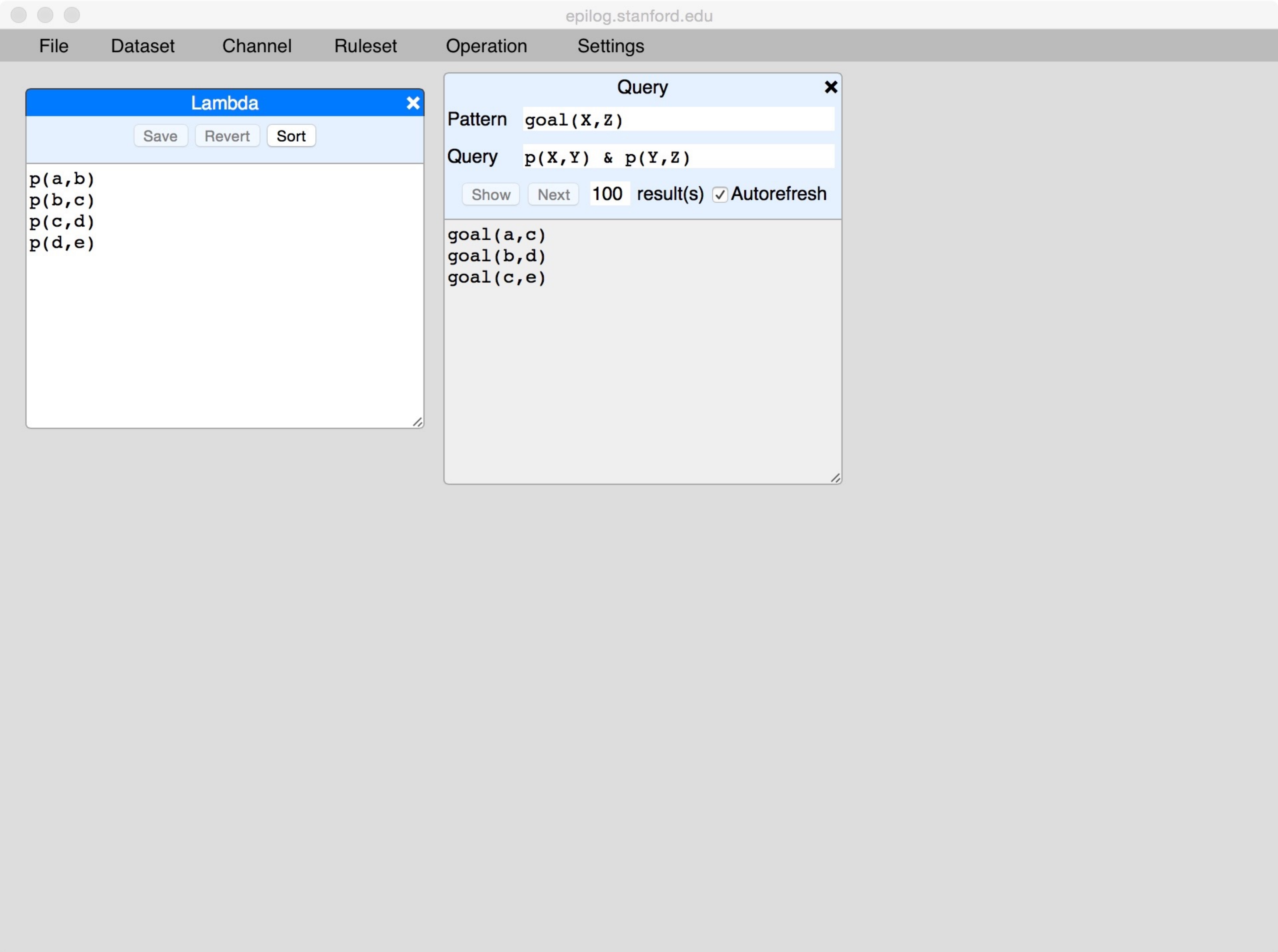

epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)



epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
p(d,e)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
p(d,e)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)
goal(c,e)