Computational Law
Definitions

Michael Genesereth
Computer Science Department
Stanford University

Kinship Example

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

More Data

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)

grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

Deduction

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

Base relation

+

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

=

```
grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

View relation

Benefits

Economy - fewer facts need to be stored

Less chance of getting out of sync

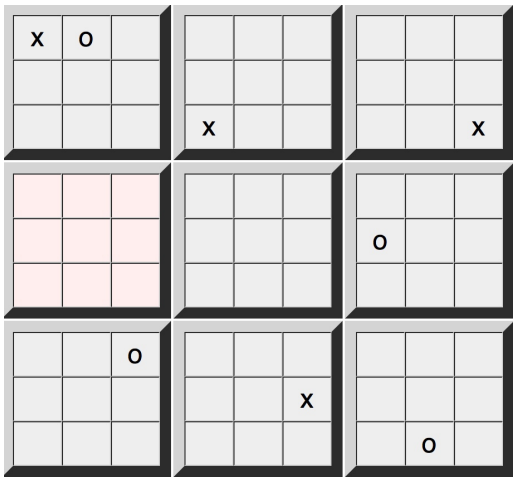
View definitions work for any number of objects

Uses

Defining additional domain relations

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

Defining what is legal



Defining what is illegal

Syntax

Constants and Variables

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

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

Same as before

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

```
X    Y23    Somebody    _
```

Terms

A **term** is either an object constant or a variable.

```
art    bob    X    Y23
```

NB: Epilog also allows users to write more complex terms. And this significantly increases its representational power. Not needed for our our purposes today.

Atoms, Negations, and Literals

Atoms

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

Negations

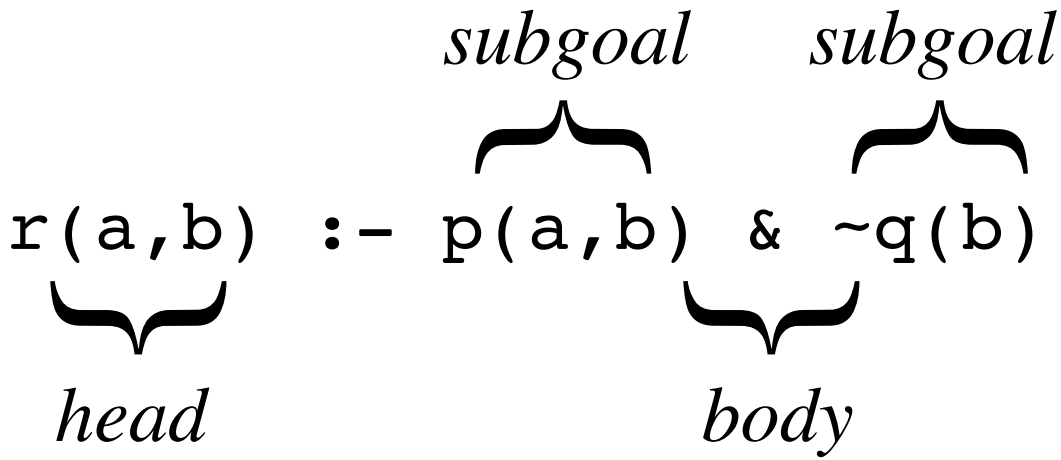
```
~p(a,b)
```

A *literal* is either an atom or a negation of an atom.

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

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

Rules



Rules with Variables

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

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

Safety

A rule is *safe* if and only if every variable in the head and every variable in any negative subgoal appears in some positive subgoal in the body.

Safe Rule:

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

Unsafe Rule:

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

Unsafe Rule:

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

Safety

A rule is *safe* if and only if every variable in the head and every variable in any negative subgoal appears in some positive subgoal in the body.

Safe Rule:

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

Unsafe Rule:

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

Unsafe Rule:

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

Rulesets

A **ruleset** is a finite set of rules.

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

Note that there can be more that one rule defining a relation.

Semipositive Rulesets

A **semipositive** ruleset is one in which negations apply only to base relations, i.e. there are no subgoals with negated views.

Base Relations: {m, n, p}
View Relations: {q, r}

Example:

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

Non-Example:

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

Semantics

Instances

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

Rule

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

Ground Terms

```
{a,b}
```

Instances

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

Rule Application

The **result of applying a rule** r **to a dataset** Δ is the set of all ψ such that (1) ψ is the head of an arbitrary instance of r, (2) every positive subgoal in the instance is in Δ, and (3) no negative subgoal in the instance is in Δ.

Example

Dataset

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

Result

```
r(a,b)
r(b,c)
```

Rule

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

Instances

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

Closure

Using this notion, we define the *closure* of a ruleset on a dataset to be the result of

(1) **applying** the rules in the ruleset to facts in the dataset,

(2) **adding** the results to the dataset, and

(3) **repeating** until nothing new is added.

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

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

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

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

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

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

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

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

```
q(a,b)
q(b,c)
q(c,d)
q(d,c)
q(b,a)
q(c,b)
r(c,d)
r(d,c)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)    q(a,b)    s(a,b)
edge(b,c)    q(b,c)    s(b,c)
edge(c,d)    q(c,d)    s(c,d)
edge(d,c)    q(d,c)    s(d,c)
p(a)         q(b,a)
p(b)         q(c,b)
p(c)         r(c,d)
p(d)         r(d,c)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)    q(a,b)    s(a,b)
edge(b,c)    q(b,c)    s(b,c)
edge(c,d)    q(c,d)    s(c,d)
edge(d,c)    q(d,c)    s(d,c)
p(a)         q(b,a)    s(a,c)
p(b)         q(c,b)    s(b,d)
p(c)         r(c,d)    s(c,c)
p(d)         r(d,c)    s(d,d)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

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

```
q(a,b)
q(b,c)
q(c,d)
q(d,c)
q(b,a)
q(c,b)
r(c,d)
r(d,c)
```

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

same and distinct

Identity

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

Difference

`distinct(t1,t2)` is true iff t1 and t2 are different

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

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

`evaluate(plus(2,2),4)` is true

Evaluable Terms

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 Predicate

`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)

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

NB: unbound variables allowed in second argument only

Aggregate Terms

Predefined Aggregate

```
countofall
```

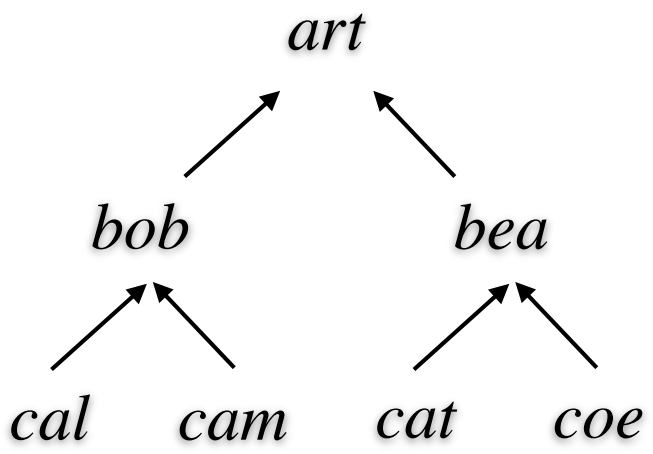
Example

```
numchildren(X,N) :-
   person(X) &
   evaluate(countofall(Y,parent(X,Y)),N)
```

Kinship

Datasets

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```



Example

Grandparents:

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

Example

Grandparents:

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

Example

Ancestors:

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,coe)
parent(bea,cat)
parent(bea,coe)
```

View:

```
ancestor(art,bob)
ancestor(art,bea)
ancestor(bob,cal)
ancestor(bob,cam)
ancestor(bea,cat)
ancestor(bea,coe)
ancestor(art,cal)
ancestor(art,cam)
ancestor(art,cat)
ancestor(art,coe)
```

Example

Ancestors:

```
ancestor(X,Z) :- parent(X,Z)
ancestor(X,Z) :- parent(X,Y) & ancestor(Y,Z)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
ancestor(art,bob)
ancestor(art,bea)
ancestor(bob,cal)
ancestor(bob,cam)
ancestor(bea,cat)
ancestor(bea,coe)
ancestor(art,cal)
ancestor(art,cam)
ancestor(art,cat)
ancestor(art,coe)
```

Example

Personhood:

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
person(art)
person(bob)
person(cal)
person(cam)
person(bea)
person(cat)
person(coe)
```

Example

Personhood:

```
person(X) :- parent(X,Y)
person(Y) :- parent(X,Y)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
person(art)
person(bob)
person(cal)
person(cam)
person(bea)
person(cat)
person(coe)
```

Example

Childlessness:

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
childless(cal)
childless(cam)
childless(cat)
childless(coe)
```

Example

Childlessness:

```
childless(X) :-
  evaluate(countofall(Y,parent(X,Y)),0)
```

Data:

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

View:

```
childless(cal)
childless(cam)
childless(cat)
childless(coe)
```

Constraints

Up to now - unconstrained datasets

any dataset from its Herbrand base is acceptable

Not always the case

A person cannot be own parent

A person cannot be both dead and alive

every parent in parent relation must be in adult relation

`illegal`

Boolean / 0-ary predicate

true if and only if dataset violates at least one constraint

We encode constraints by defining `illegal`.

Examples

A person cannot be his own parent.

```
illegal :- parent(X,X)
```

A person cannot be both dead and alive.

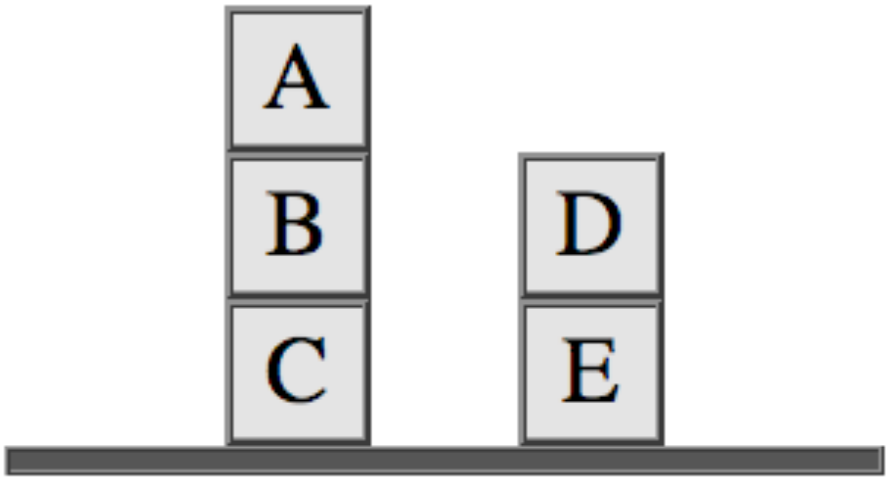
```
illegal :- dead(X) & alive(X)
```

Every parent is an adult.

```
illegal :- parent(X,Y) & ~adult(X)
```

Blocks World

Blocks World



Vocabulary

Symbols: `a`, `b`, `c`, `d`, `e`

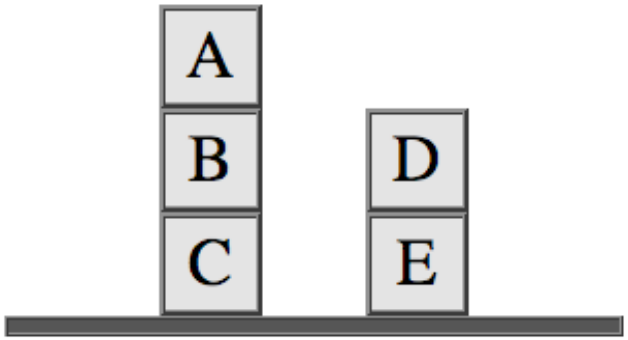
Unary Predicate:

`block`

Binary Predicate:

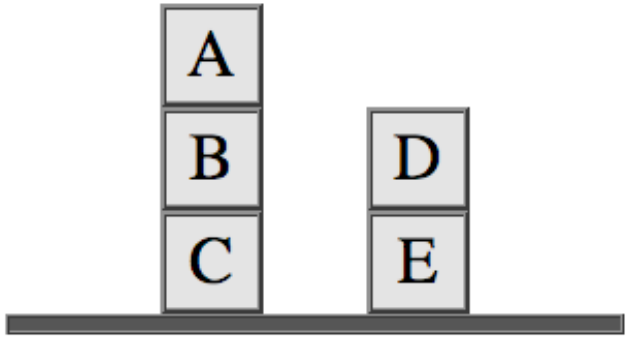
`on` - pairs of blocks in which first is on the second

Data




```
block(a)
block(b)        on(a,b)
block(c)        on(b,c)
block(d)        on(d,e)
block(e)
```

Blocks World - cluttered

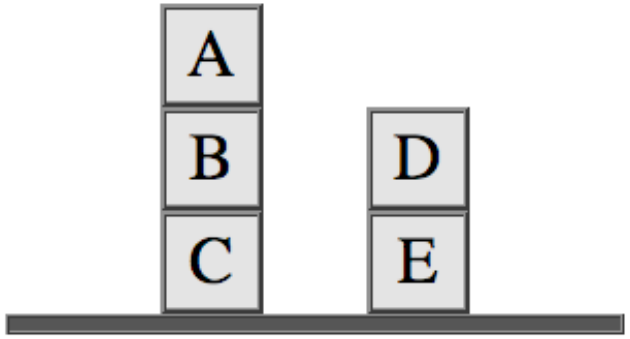



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
cluttered(Y) :- on(X,Y)
```

```
cluttered(b)
cluttered(c)
cluttered(e)
```

Blocks World - clear

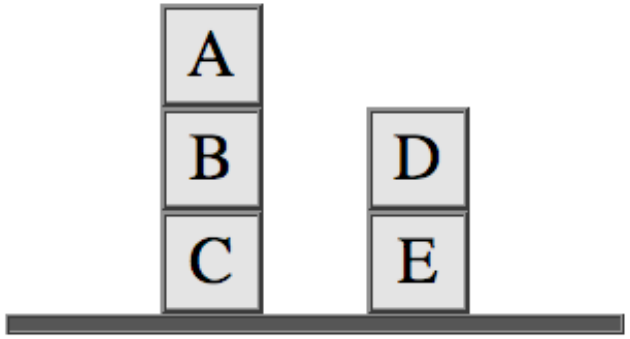



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
clear(Y) :- block(Y) & countofall(X,on(X,Y),0)
```

```
clear(a)
clear(d)
```

Blocks World - supported

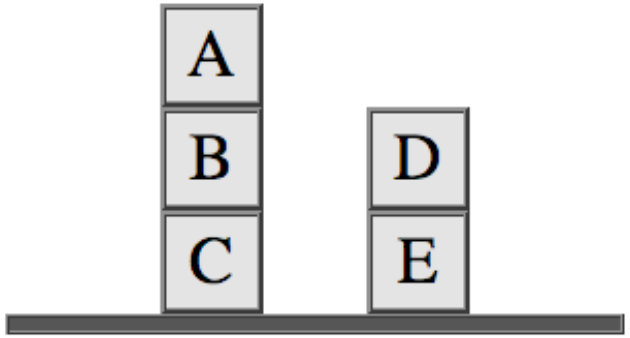



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

???

```
supported(a)
supported(b)
supported(d)
```

Blocks World - supported

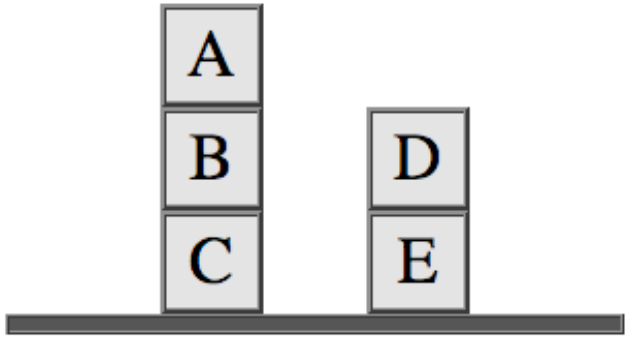



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
supported(X) :- on(X,Y)
```

```
supported(a)
supported(b)
supported(d)
```

Blocks World - table

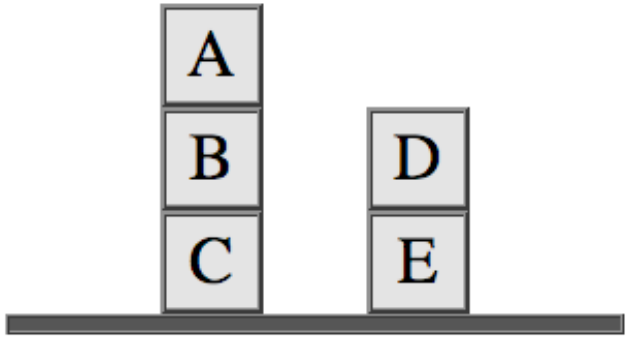



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

???

```
table(c)
table(e)
```

Blocks World - table

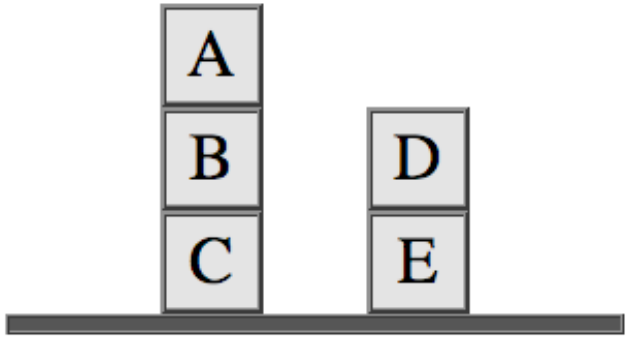



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
table(X) :- block(X) & countofall(Y,on(X,Y),0)
```

```
table(c)
table(e)
```

Blocks World - stack

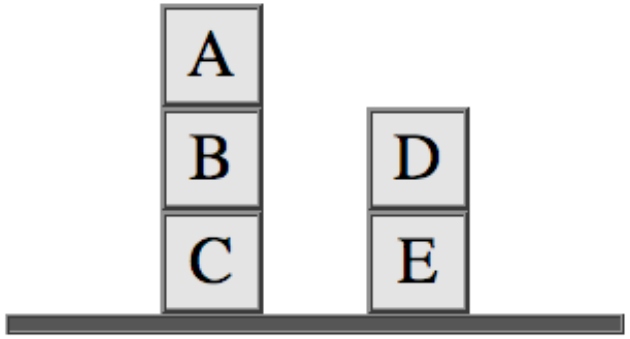



```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
stack(X,Y,Z) :- on(X,Y) & on(Y,Z)
```

```
stack(a,b,c)
```

Blocks World - above




```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
above(X,Y) :- on(X,Y)
above(X,Z) :- on(X,Y) & above(Y,Z)
```

```
above(a,b)
above(b,c)
above(a,c)
above(d,e)
```

Sierra

Sierra

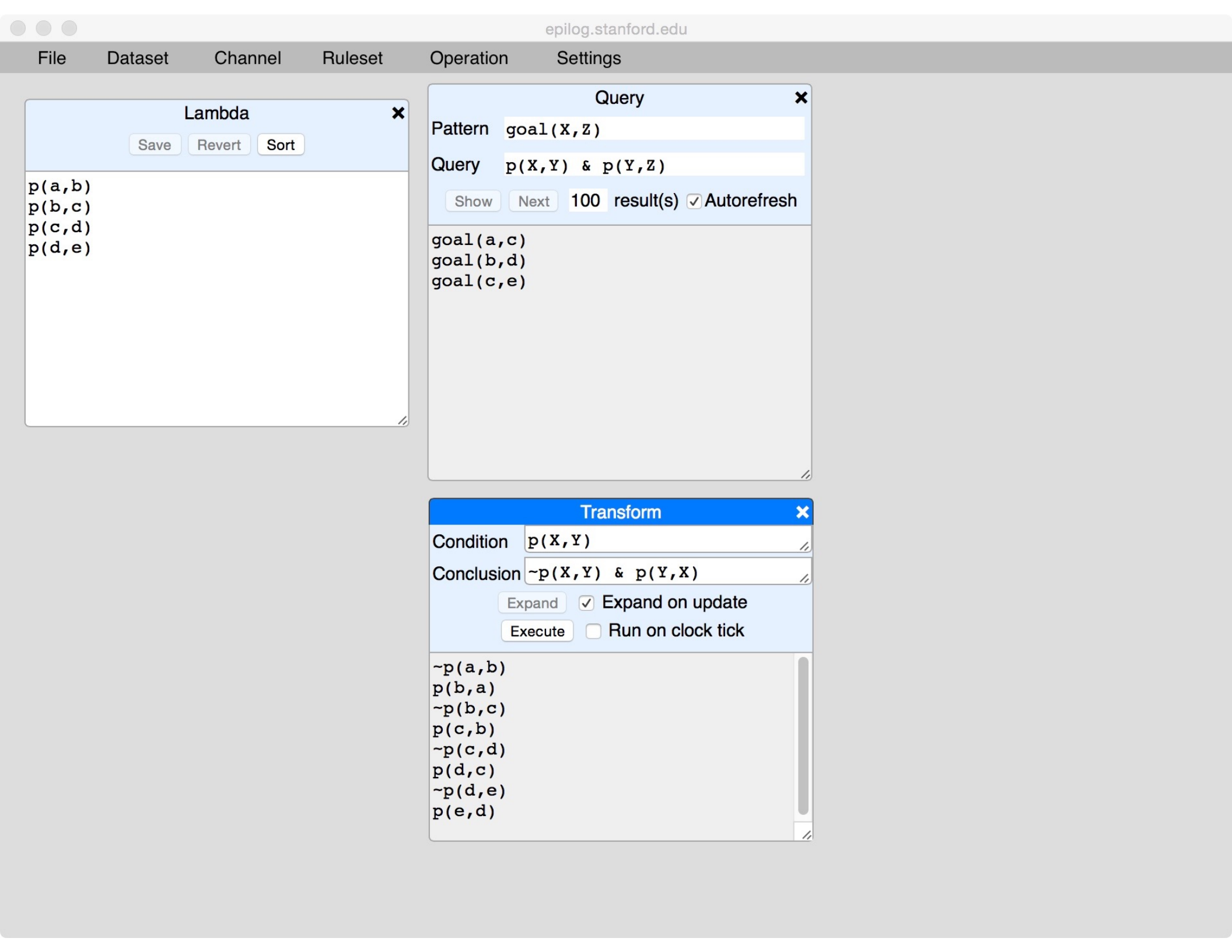
Sierra is browser-based IDE (interactive development environment) for Epilog.

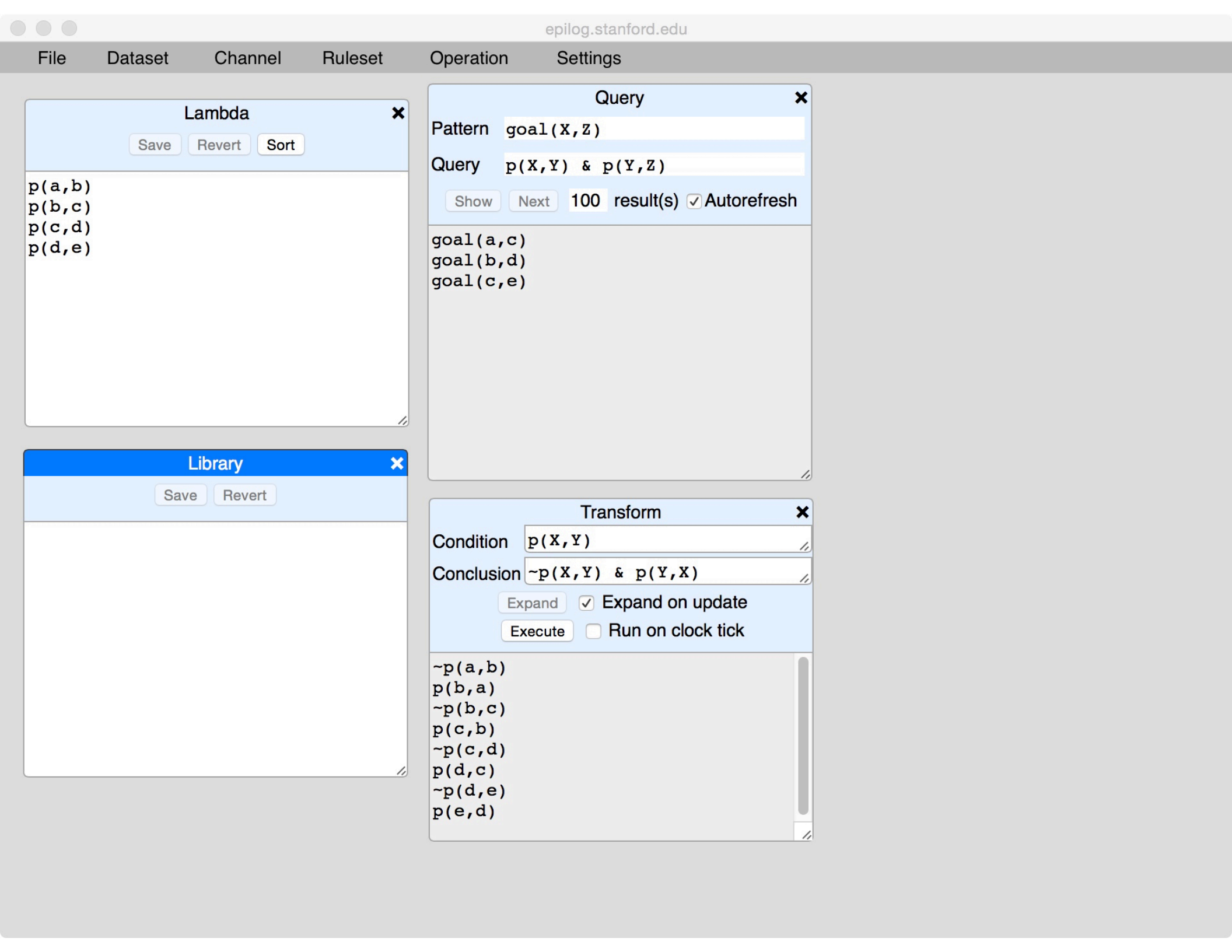
Saving and loading files

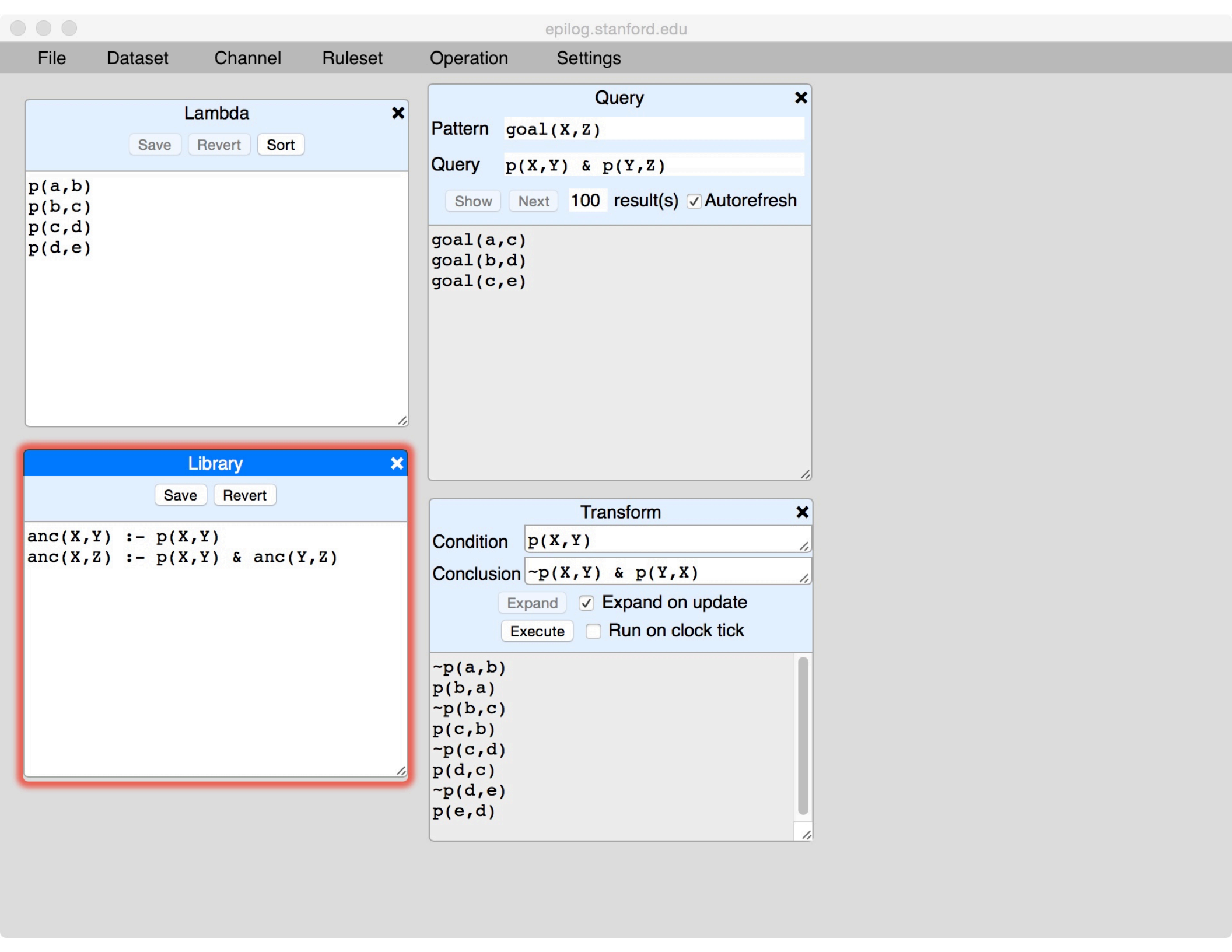
Viewing and Editing datasets
Querying datasets
Transformation tools for datasets

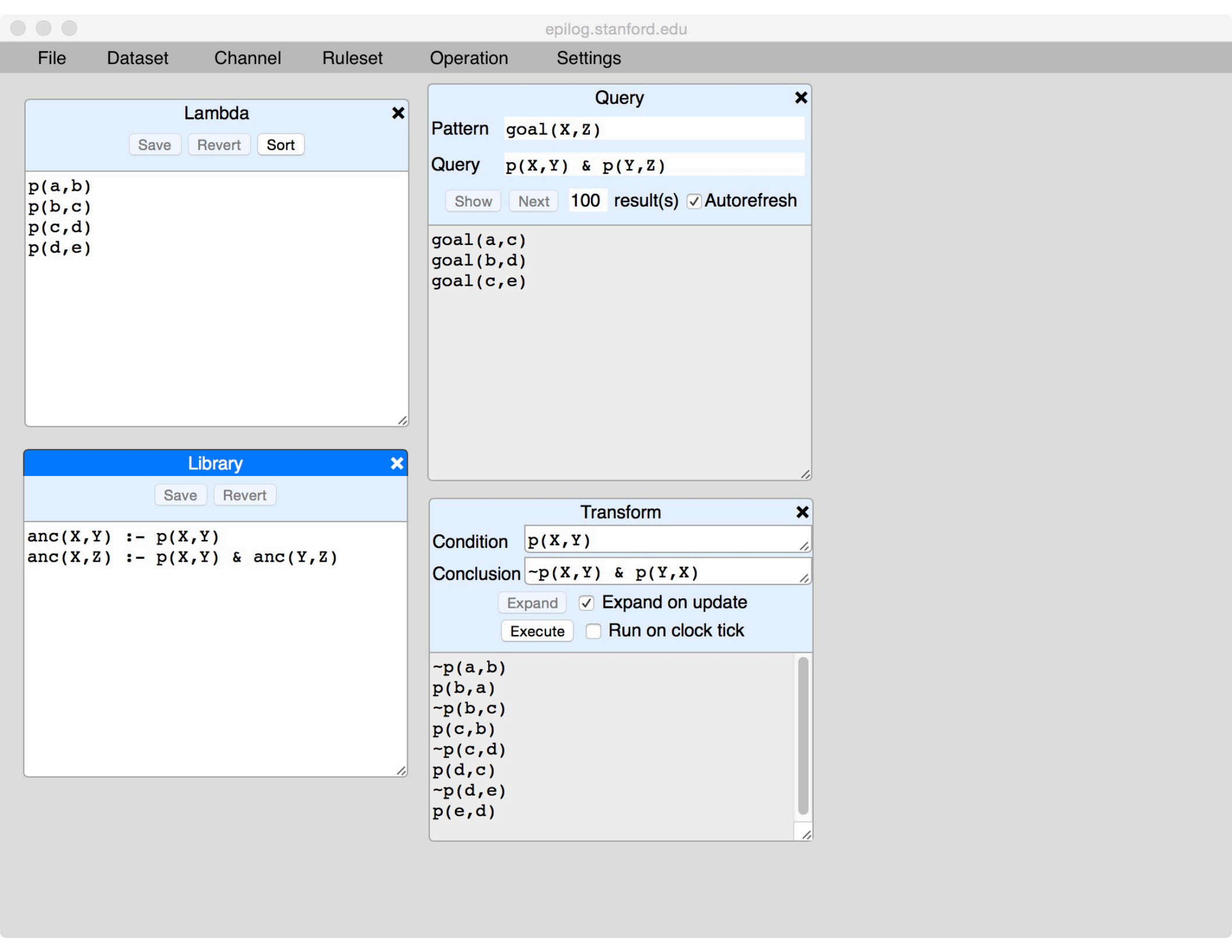
Interpreter (for view definitions, action definitions)
Trace capability (useful for debugging rules)
Analysis tools (error checking and optimizing rules)

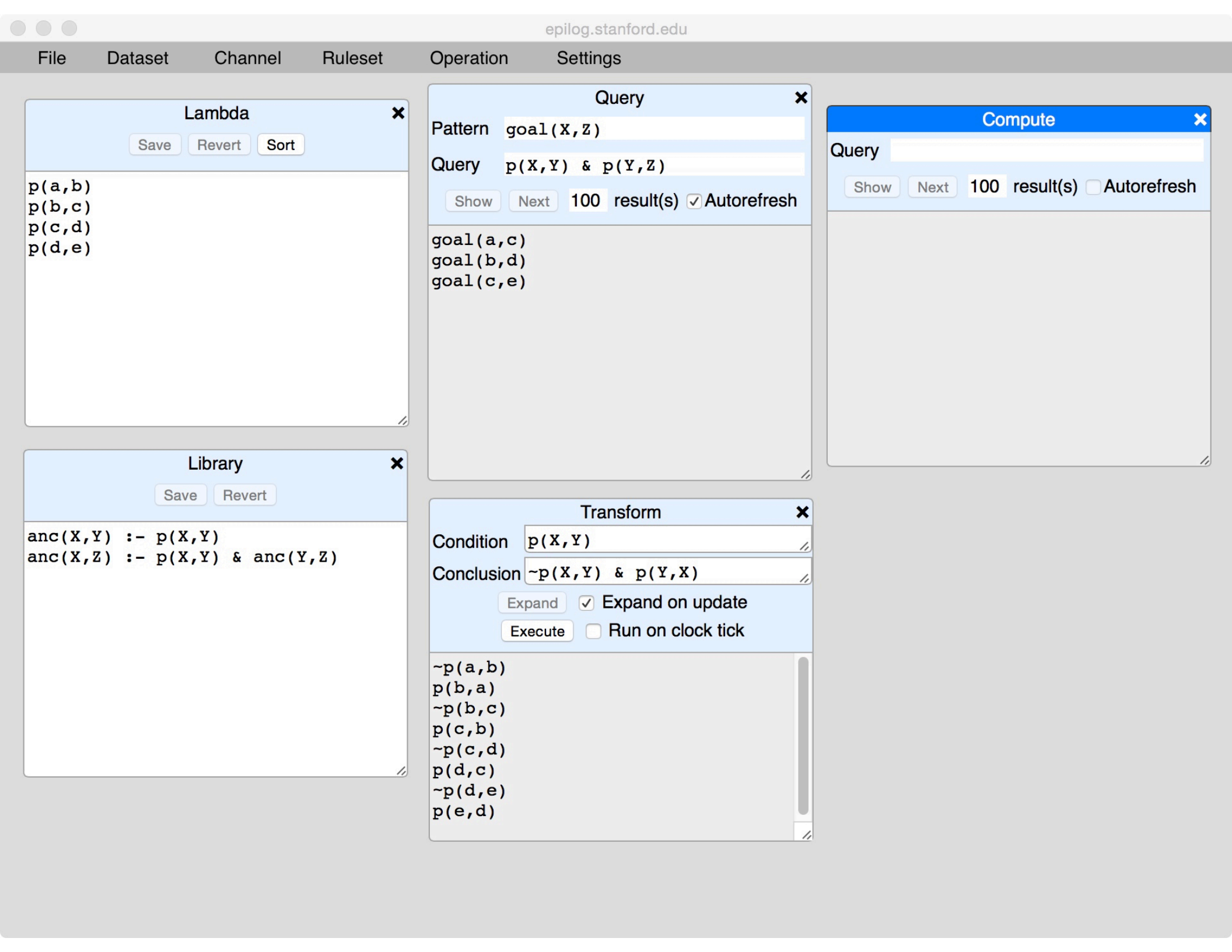
http://epilog.stanford.edu/homepage/sierra.php

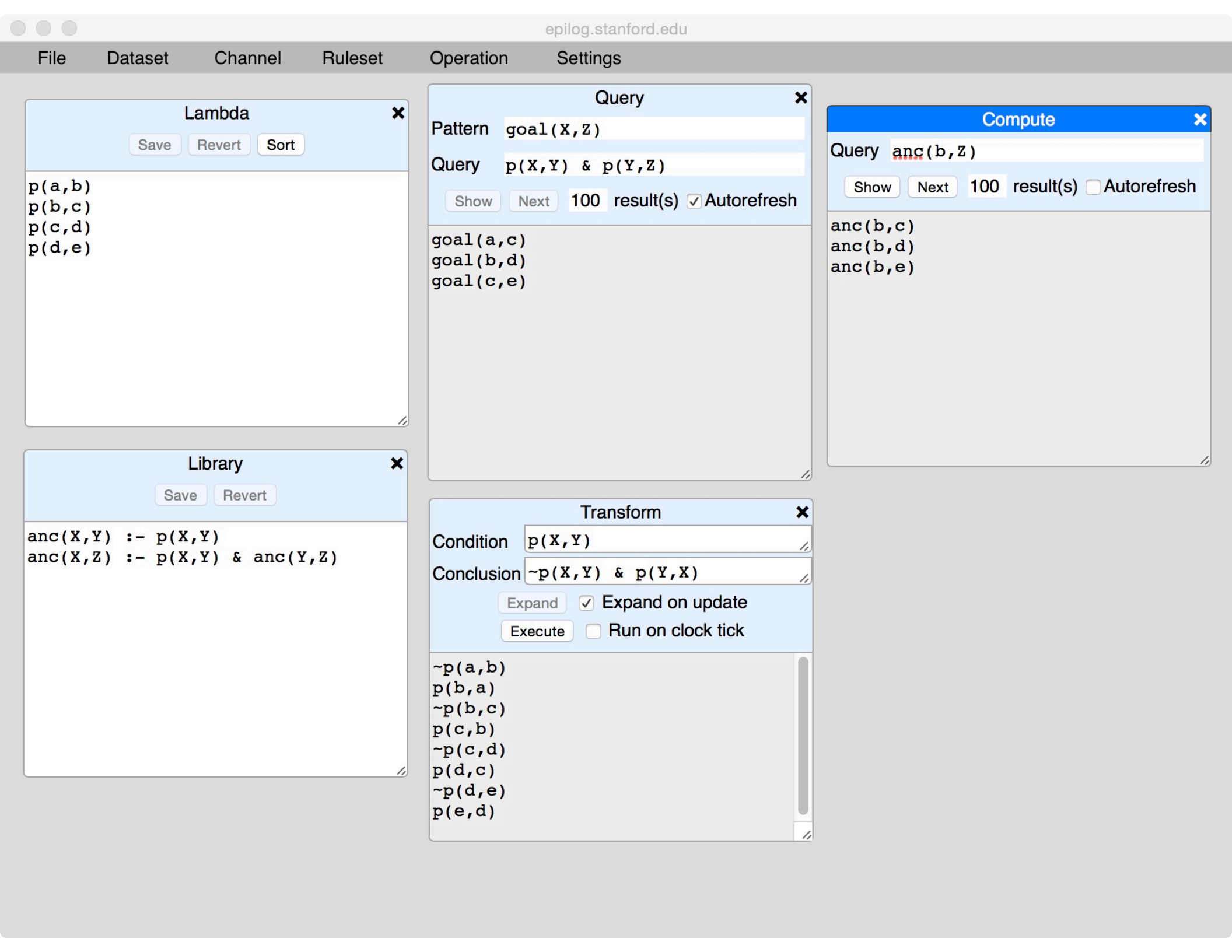

epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
p(d,e)
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)
goal(c,e)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,d)

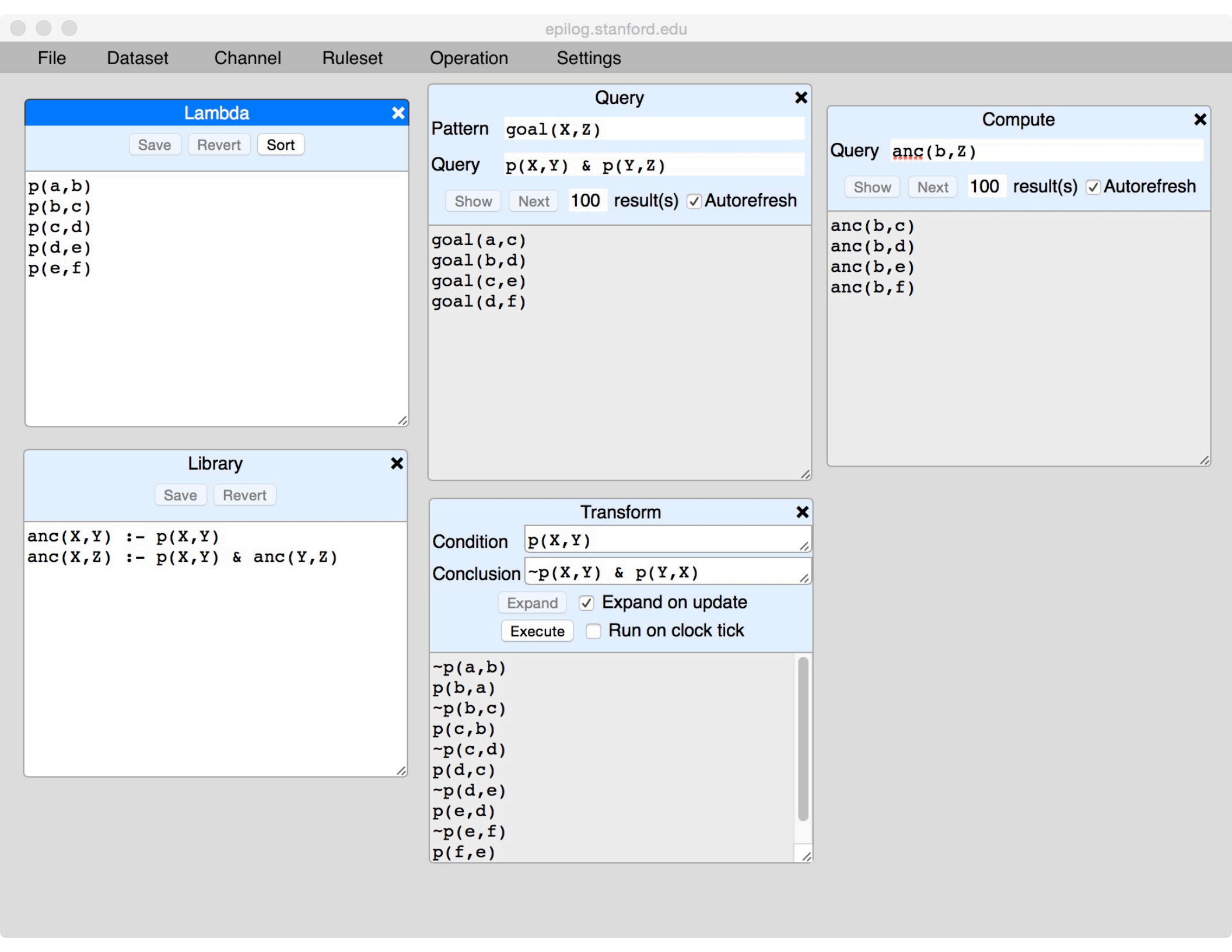

epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
p(d,e)
Library
Save
Revert
Query
Pattern
goal(X,Z)
Query
p(X,Y) & p(Y,Z)
Show
Next
100
result(s)
Autorefresh
goal(a,c)
goal(b,d)
goal(c,e)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,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)
Library
Save
Revert
anc(X,Y) :- p(X,Y)
anc(X,Z) :- p(X,Y) & anc(Y,Z)
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)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,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)
Library
Save
Revert
anc(X,Y) :- p(X,Y)
anc(X,Z) :- p(X,Y) & anc(Y,Z)
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)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,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)
Library
Save
Revert
anc(X,Y) :- p(X,Y)
anc(X,Z) :- p(X,Y) & anc(Y,Z)
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)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,d)
Compute
Query
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)
p(d,e)
Library
Save
Revert
anc(X,Y) :- p(X,Y)
anc(X,Z) :- p(X,Y) & anc(Y,Z)
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)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,d)
Compute
Query
anc(b,Z)
Show
Next
100
result(s)
Autorefresh
anc(b,c)
anc(b,d)
anc(b,e)


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)
p(e,f)
Library
Save
Revert
anc(X,Y) :- p(X,Y)
anc(X,Z) :- p(X,Y) & anc(Y,Z)
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)
goal(d,f)
Transform
Condition
p(X,Y)
Conclusion
~p(X,Y) & p(Y,X)
Expand
Expand on update
Execute
Run on clock tick
~p(a,b)
p(b,a)
~p(b,c)
p(c,b)
~p(c,d)
p(d,c)
~p(d,e)
p(e,d)
~p(e,f)
p(f,e)
Compute
Query
anc(b,Z)
Show
Next
100
result(s)
Autorefresh
anc(b,c)
anc(b,d)
anc(b,e)
anc(b,f)

Assignments

Readings

Reading 3.1 - Definitions

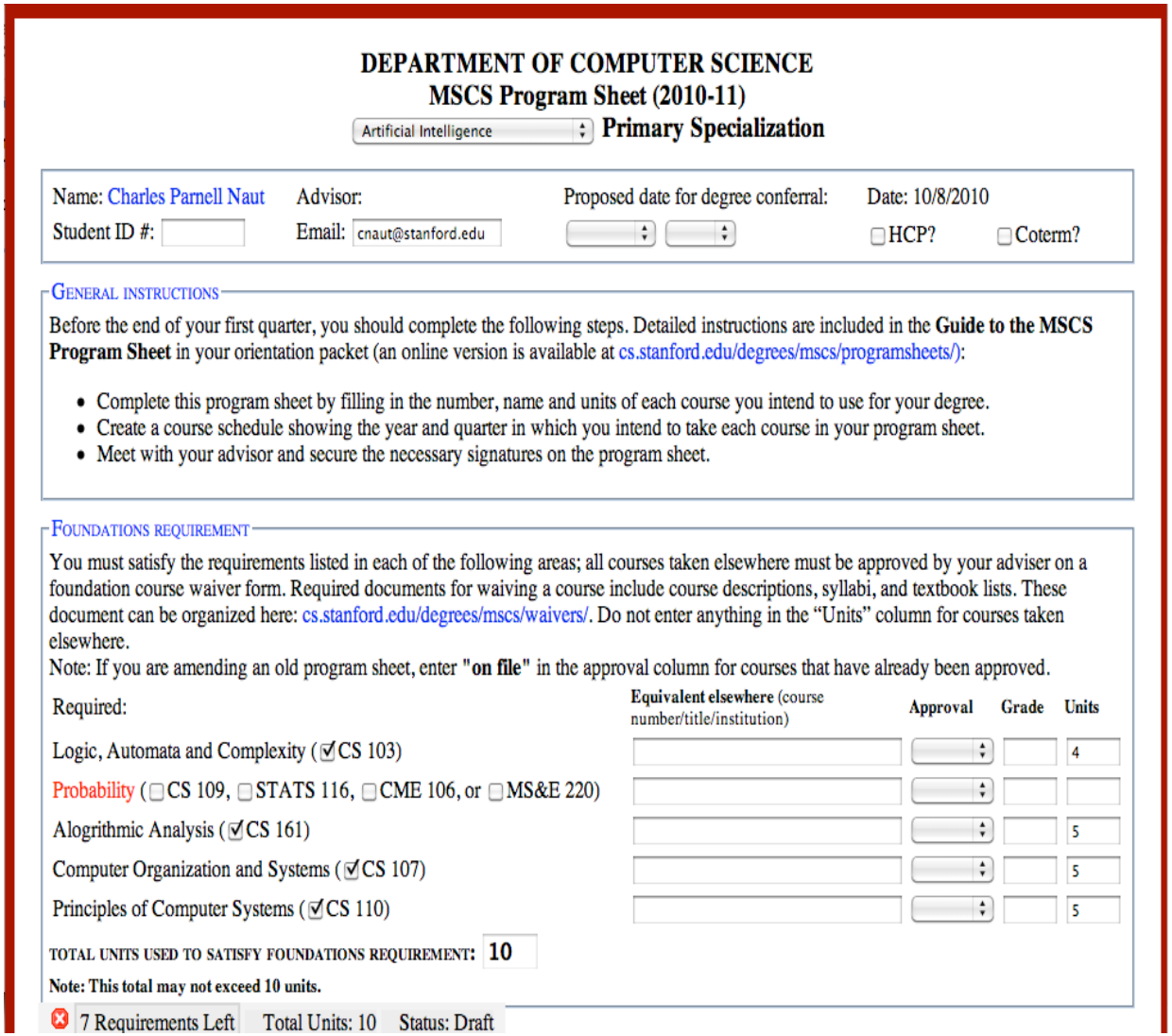
Assignment - Sierra

The goal of this exercise is for you to familiarize yourself with the Sierra capabilities for editing and using view definitions. Go to http://epilog.stanford.edu and click on the Sierra link.

In a separate window, open the documentation for Sierra. To access the documentation, go to http://epilog.stanford.edu, click on Documentation, and then click on the Sierra item on the resulting drop-down menu.

Read though section 6 of the documentation and reproduce the examples in the Sierra window you opened earlier. Once you have done this, experiment on your own. Try different data and different views.

Assignment - Program Sheets



DEPARTMENT OF COMPUTER SCIENCE
MSCS Program Sheet (2010-11)
Artificial Intelligence **Primary Specialization**

Name: Charles Parnell Naut Advisor: Proposed date for degree conferral: Date: 10/8/2010
Student ID #: Email: cnaut@stanford.edu HCP? Coterm?

GENERAL INSTRUCTIONS

Before the end of your first quarter, you should complete the following steps. Detailed instructions are included in the **Guide to the MSCS Program Sheet** in your orientation packet (an online version is available at cs.stanford.edu/degrees/mscs/programsheets/):

- Complete this program sheet by filling in the number, name and units of each course you intend to use for your degree.
- Create a course schedule showing the year and quarter in which you intend to take each course in your program sheet.
- Meet with your advisor and secure the necessary signatures on the program sheet.

FOUNDATIONS REQUIREMENT

You must satisfy the requirements listed in each of the following areas; all courses taken elsewhere must be approved by your adviser on a foundation course waiver form. Required documents for waiving a course include course descriptions, syllabi, and textbook lists. These document can be organized here: cs.stanford.edu/degrees/mscs/waivers/. Do not enter anything in the "Units" column for courses taken elsewhere.
Note: If you are amending an old program sheet, enter "**on file**" in the approval column for courses that have already been approved.

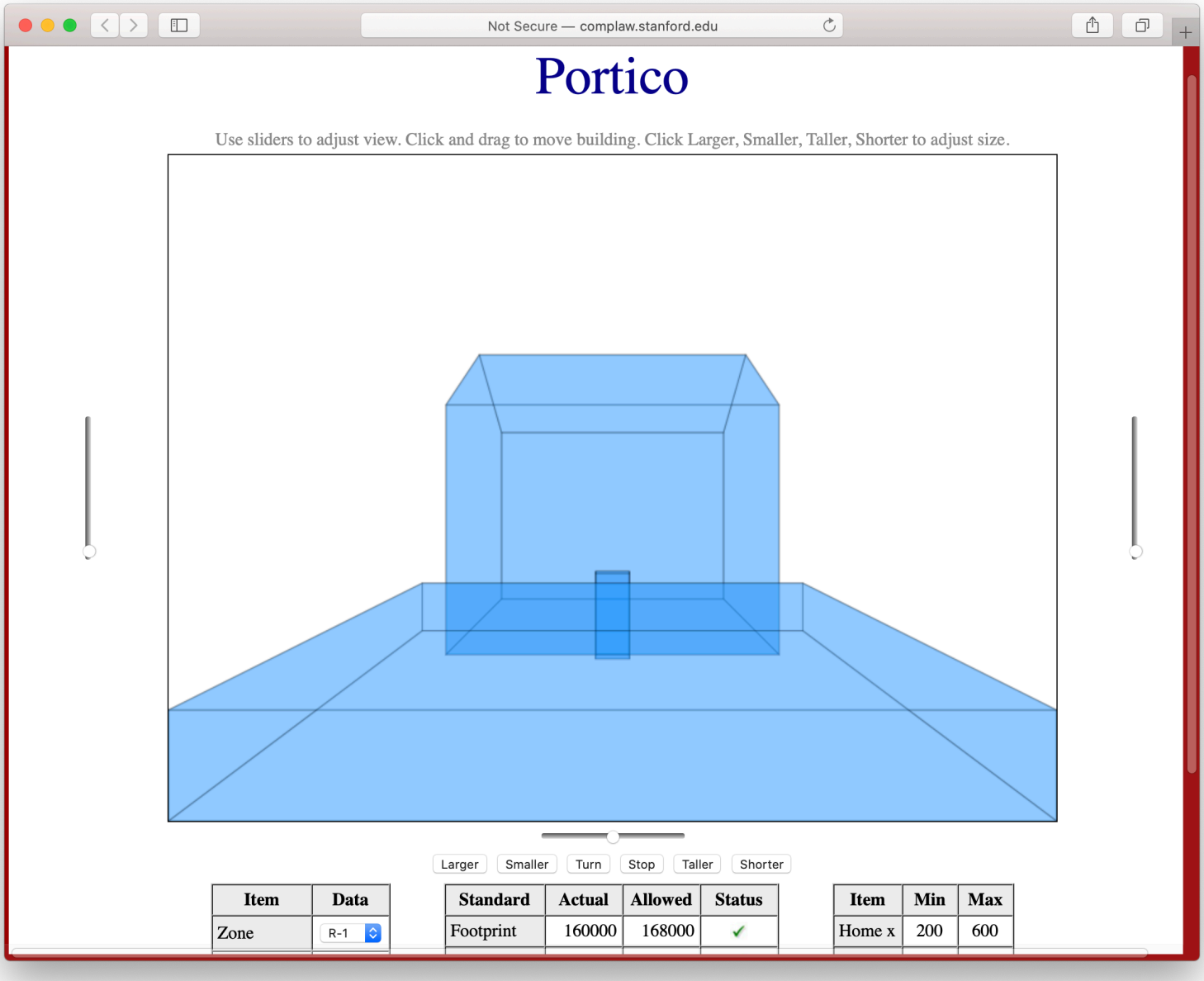
Required:	**Equivalent elsewhere** (course number/title/institution)	Approval	Grade	Units
Logic, Automata and Complexity (☑CS 103)				4
Probability (☐CS 109, ☐STATS 116, ☐CME 106, or ☐MS&E 220)				
Alogrithmic Analysis (☑CS 161)				5
Computer Organization and Systems (☑CS 107)				5
Principles of Computer Systems (☑CS 110)				5

TOTAL UNITS USED TO SATISFY FOUNDATIONS REQUIREMENT: 10
Note: This total may not exceed 10 units.

7 Requirements Left Total Units: 10 Status: Draft

http://complaw.stanford.edu/assignments/program_overview.html

Assignment - Portico



http://complaw.stanford.edu/assignments/portico_overview.html

CODEX

The Stanford Center for Legal Informatics

Legal Empowerment through Information Technology

CODEX

The Stanford Center for Legal Informatics

Legal Empowerment through Information Technology