Computational Law
Ontologies

Michael Genesereth
Computer Science Department
Stanford University

Conceptualization

Objects - e.g. people, companies, cities

concrete (*person*) or abstract (*number*, *set*, *justice*)

primitive (*computer chip*) or composite (*circuit*)

real (*earth*) or fictitious (*Sherlock Holmes*)

Relationships

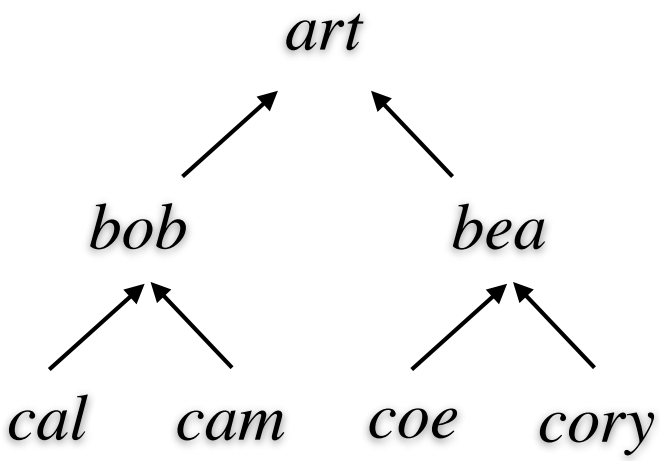
properties of objects or relationships among objects

e.g. Joe *is a person*

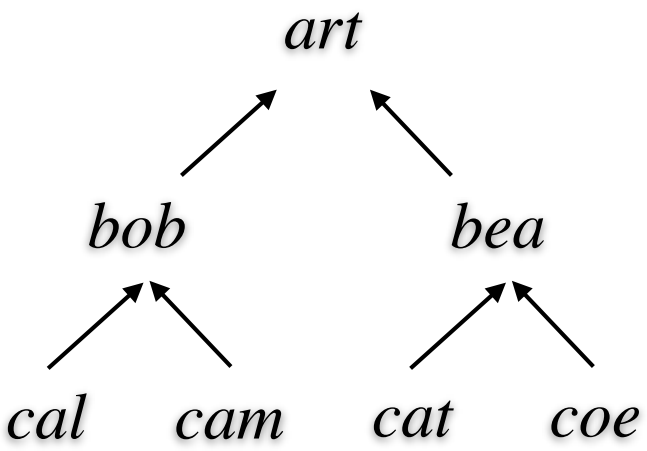
e.g. Joe *is the parent of* Bill

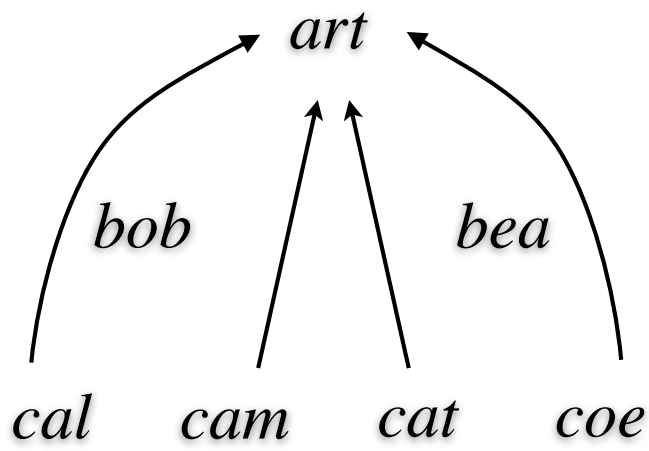
e.g. Joe *likes* Bill *more than* Harry

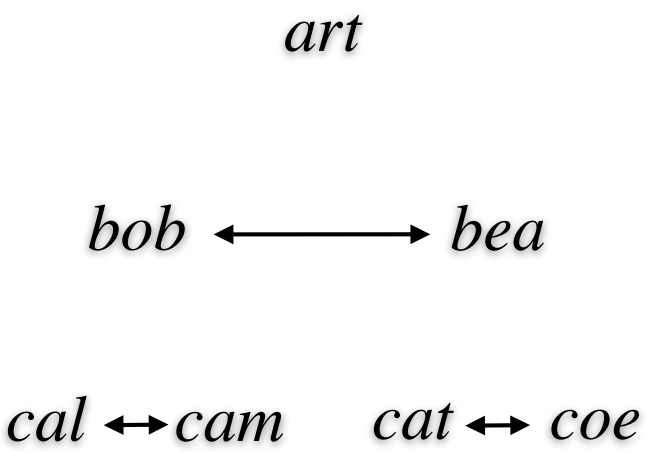
Parentage

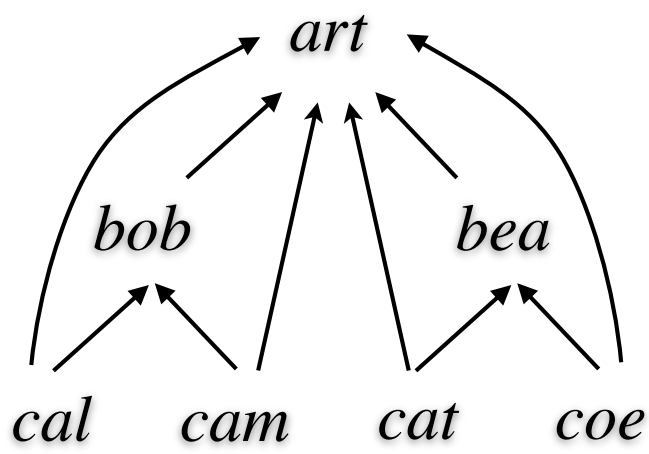


Kinship Relations

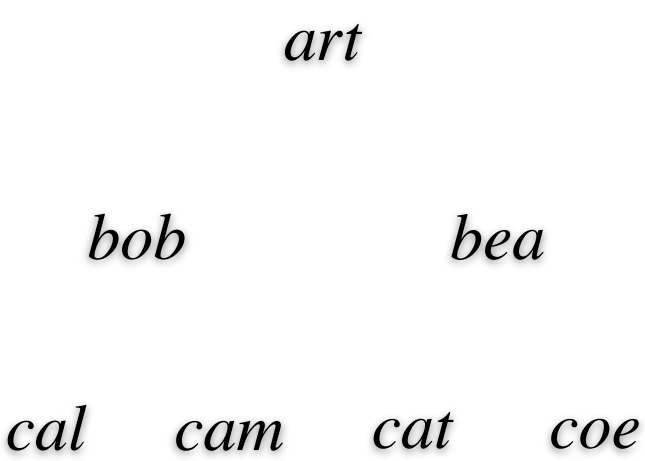


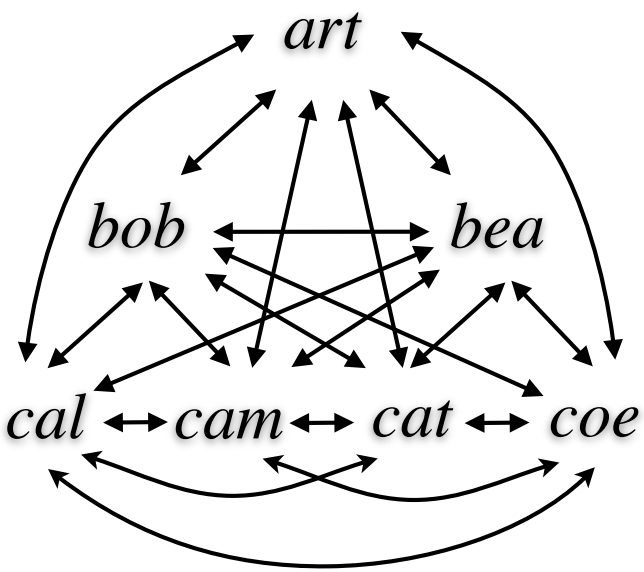




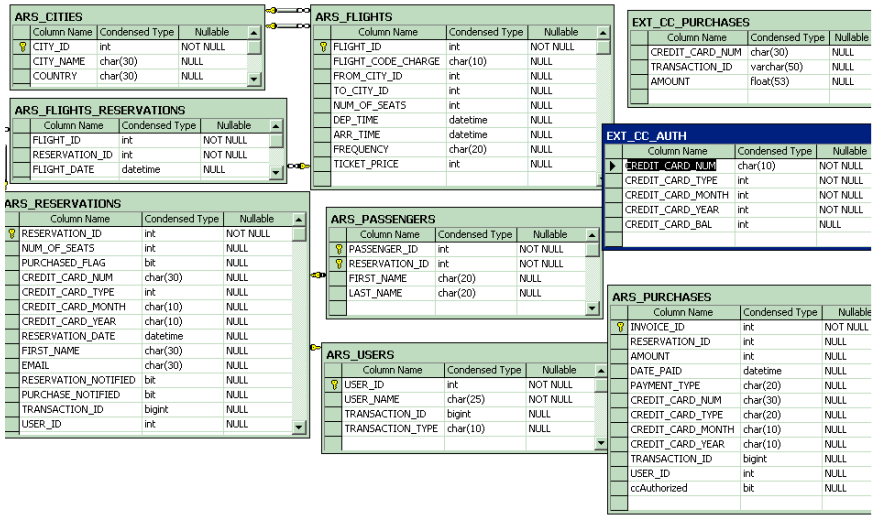


Degenerate Relations





Tables and Graphs



ARS_CITIES

Column Name	Condensed Type	Nullable
CITY_ID	int	NOT NULL
CITY_NAME	char(30)	NULL
COUNTRY	char(30)	NULL

ARS_FLIGHTS

Column Name	Condensed Type	Nullable
FLIGHT_ID	int	NOT NULL
FLIGHT_CODE_CHARGE	char(10)	NULL
FROM_CITY_ID	int	NULL
TO_CITY_ID	int	NULL
NUM_OF_SEATS	int	NULL
DEP_TIME	datetime	NULL
ARR_TIME	datetime	NULL
FREQUENCY	char(20)	NULL
TICKET_PRICE	int	NULL

EXT_CC_PURCHASES

Column Name	Condensed Type	Nullable
CREDIT_CARD_NUM	char(30)	NULL
TRANSACTION_ID	varchar(50)	NULL
AMOUNT	float(53)	NULL

ARS_FLIGHTS_RESERVATIONS

Column Name	Condensed Type	Nullable
FLIGHT_ID	int	NOT NULL
RESERVATION_ID	int	NOT NULL
FLIGHT_DATE	datetime	NULL

EXT_CC_AUTH

Column Name	Condensed Type	Nullable
CREDIT_CARD_NUM	char(10)	NOT NULL
CREDIT_CARD_TYPE	int	NOT NULL
CREDIT_CARD_MONTH	int	NOT NULL
CREDIT_CARD_YEAR	int	NOT NULL
CREDIT_CARD_BAL	int	NULL

ARS_RESERVATIONS

Column Name	Condensed Type	Nullable
RESERVATION_ID	int	NOT NULL
NUM_OF_SEATS	int	NULL
PURCHASED_FLAG	bit	NULL
CREDIT_CARD_NUM	char(30)	NULL
CREDIT_CARD_TYPE	int	NULL
CREDIT_CARD_MONTH	char(10)	NULL
CREDIT_CARD_YEAR	char(10)	NULL
RESERVATION_DATE	datetime	NULL
FIRST_NAME	char(30)	NULL
EMAIL	char(30)	NULL
RESERVATION_NOTIFIED	bit	NULL
PURCHASE_NOTIFIED	bit	NULL
TRANSACTION_ID	bigint	NULL
USER_ID	int	NULL

ARS_PASSENGERS

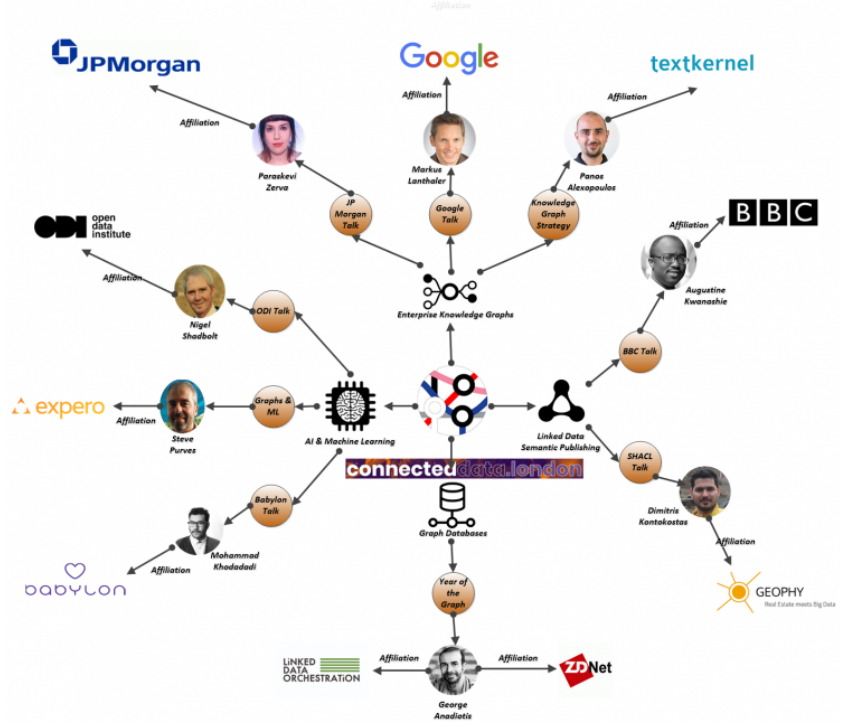
Column Name	Condensed Type	Nullable
PASSENGER_ID	int	NOT NULL
RESERVATION_ID	int	NOT NULL
FIRST_NAME	char(20)	NULL
LAST_NAME	char(20)	NULL

ARS_USERS

Column Name	Condensed Type	Nullable
USER_ID	int	NOT NULL
USER_NAME	char(25)	NOT NULL
TRANSACTION_ID	bigint	NULL
TRANSACTION_TYPE	char(10)	NULL

ARS_PURCHASES

Column Name	Condensed Type	Nullable
INVOICE_ID	int	NOT NULL
RESERVATION_ID	int	NULL
AMOUNT	int	NULL
DATE_PAID	datetime	NULL
PAYMENT_TYPE	char(20)	NULL
CREDIT_CARD_NUM	char(30)	NULL
CREDIT_CARD_TYPE	char(20)	NULL
CREDIT_CARD_MONTH	char(10)	NULL
CREDIT_CARD_YEAR	char(10)	NULL
TRANSACTION_ID	bigint	NULL
USER_ID	int	NULL
ccAuthorized	bit	NULL



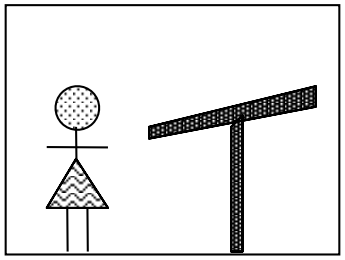
Natural Language

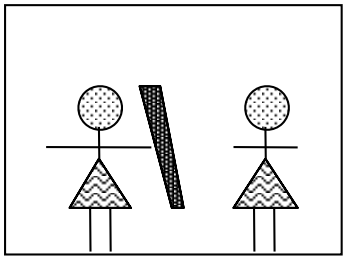
Sentences

e.g. Joe *is a person*.
e.g. Joe *works for* Apple.
e.g. Joe *is in office* B-122.

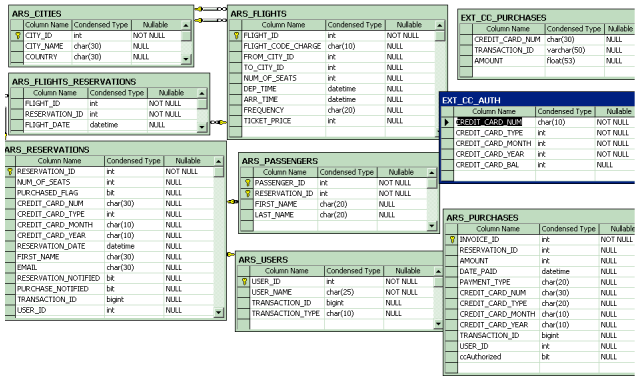
But natural language is ambiguous and difficult to process

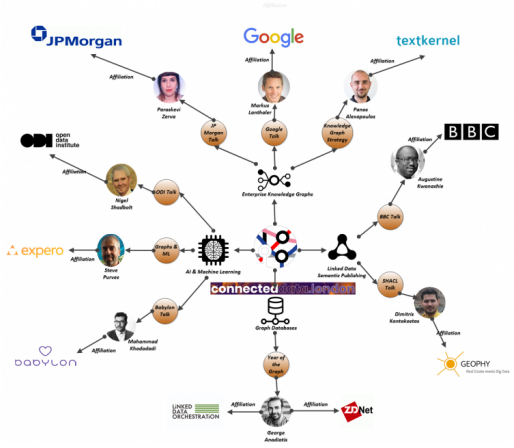
There's a girl in the room with a telescope.





Mathematical Notation





Joe *is a person*.
Joe *works for* Apple.
Joe *is in office* B-122.

```
person(joe)
worksfor(joe,apple)
office(joe,b122)
```

Sentential Representation

Constants

Constants are strings of lower case letters, digits, underscores, and periods *or* strings of arbitrary 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
```

A set of constants is called a **vocabulary**.

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!”`

Predicates / relation constants represent relations.

`person`, `parent`, `prefers`

Arity

The **arity** of 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`.

Data

A **datum / factoid / fact** is an expression formed from an *n*-ary predicate and *n* ground terms enclosed in parentheses and separated by commas.

Symbols: `a, b`
Predicate: `p/2, q/1`

Sample Datum: `p(a,b)`
Sample Datum: `q(a)`

The **Herbrand base** for a vocabulary is the set of all factoids that can be formed from the vocabulary.

Datasets

A **dataset** is any set of factoids that can be formed from a vocabulary, i.e. a subset of the Herbrand base.

Symbols: `a, b`
Predicates: `p/2, q/1`
Dataset: {`p(a,b), p(b,a), q(a)`}
Dataset: {}
Dataset: {`p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b)`}

We use datasets to characterize states of the world. The facts in a dataset are assumed to be true and those that are not in the dataset are assumed to be false.

Exercise

Vocabulary

Symbols: `a`, `b`

Predicates: `p/2`, `q/1`

Questions

How many elements in the Herbrand universe?

How many elements in the Herbrand base?

How many possible datasets?

Formality and Informality

In some logic programming languages (e.g. Prolog), types and arities determine syntactic legality; and they are enforced by interpreters and compilers.

In other languages (e.g. Epilog), types and arities suggest their intended use. However, they do not determine syntactic legality, and they are not enforced by interpreters and compilers.

In our examples, we use Epilog; but, in this course, we specify types and arities where appropriate and we *try* to adhere to them.

Note on Spelling

Spelling carries no meaning in logic programming (except as informal documentation for programmers).

```
parent(art,bob)
parent(bob,cal)

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

coulish(widget,gadget)
coulish(gadget,framis)
```

The *meaning* of a constant in logic programming is determined solely by the sentences that mention it.

Note on Order of Arguments

The order of arguments in an instance of a relation is determined by one's understanding of the relation.

Example:

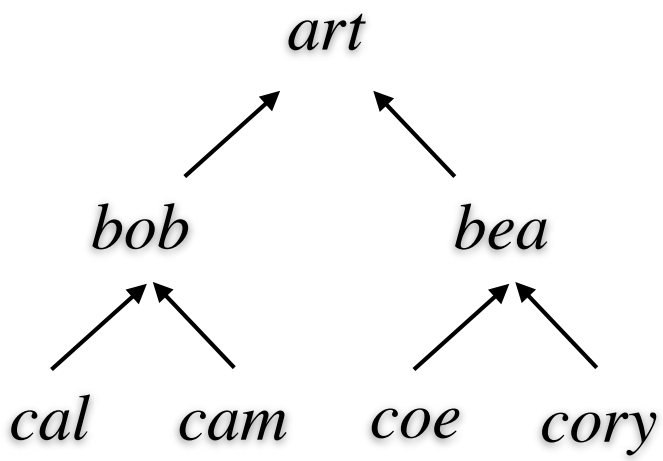
```
prefers(art,bea,bob)
```

For me, this sentence means that Art prefers Bea to Bob. Other interpretations are possible; the important thing is to be consistent - once you choose, stick with it.

Kinship

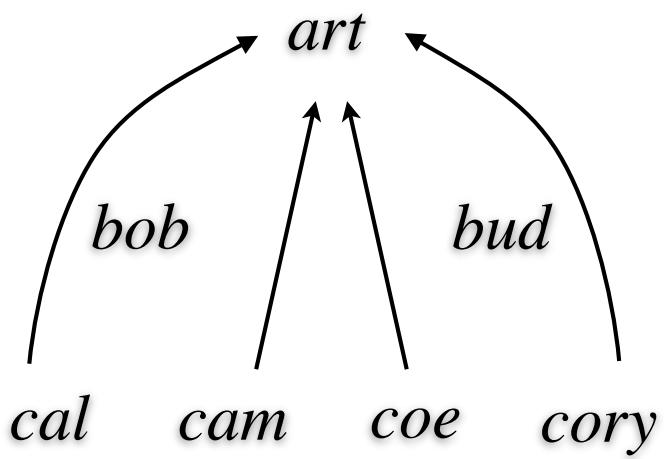
Datasets

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bud,coe)
parent(bud,cory)
```



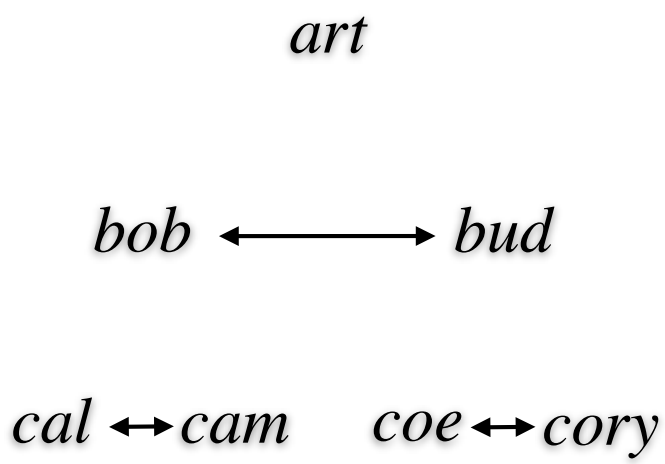
Datasets

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



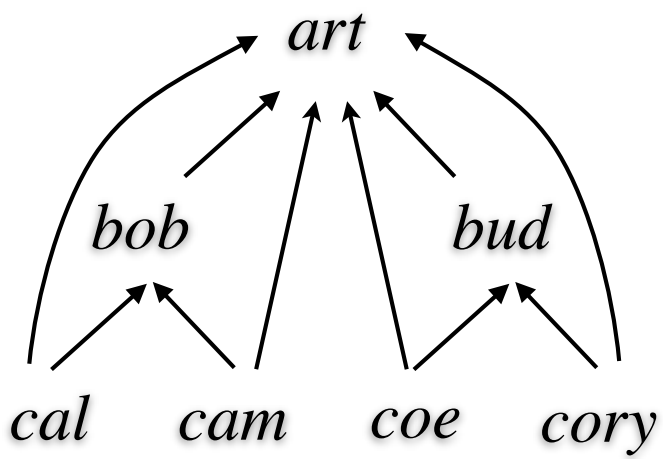
Datasets

```
sibling(bob,bud)
sibling(bud,bob)
sibling(cal,cam)
sibling(cam,cal)
sibling(coe,cory)
sibling(cory,coe)
```



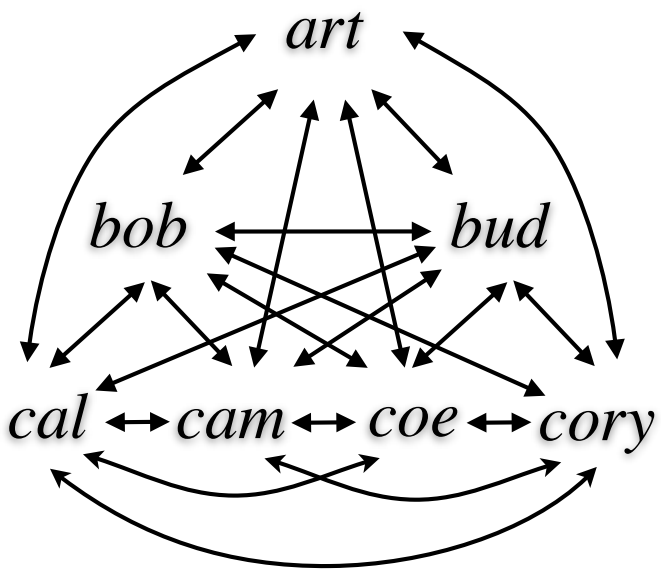
Datasets

```
ancestor(art,bob)
ancestor(art,bud)
ancestor(art,cal)
ancestor(art,cam)
ancestor(art,coe)
ancestor(art,cory)
ancestor(bob,cal)
ancestor(bob,cam)
ancestor(bud,coe)
ancestor(bud,cory)
```



Datasets

```
related(art,bob)
related(art,bud)
related(art,cal)
related(art,cam)
related(art,coe)
related(art,cory)
    . . .
related(cal,cam)
related(cal,coe)
related(cal,cory)
related(cam,coe)
related(cam,cory)
related(coe,cory)
```



Definability

Some relations definable in terms of others

- e.g. we can define `grandparent` in terms of `parent`
- e.g. we can define `sibling` in terms of `parent`
- e.g. we can define `ancestor` in terms of `parent`
- e.g. we can define `parent` in terms of `ancestor`
- See upcoming material on **view definitions**

This is where it gets interesting.

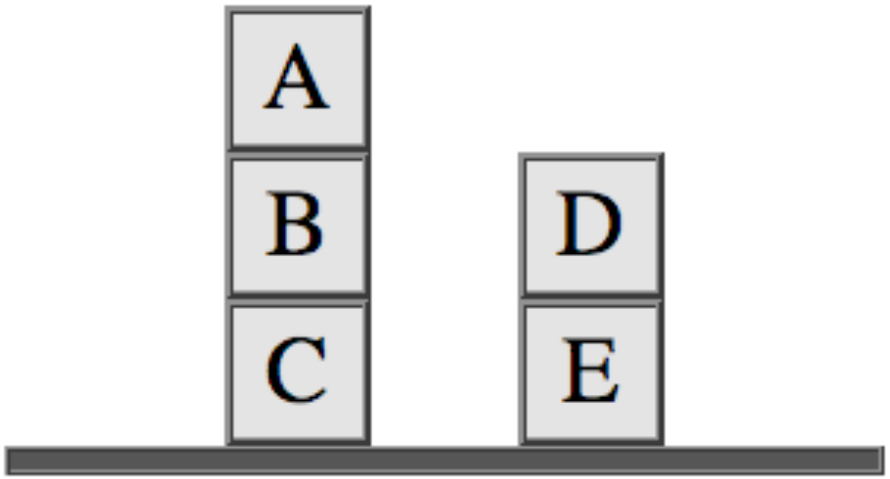
Constraints

Some combinations of arguments do not make sense

- e.g. `parent(art,art)`
- e.g. `parent(art,bob)` and `parent(bob,art)`
- e.g. `male(art)` and `female(art)`
- See upcoming material on **constraints**

Blocks World

Blocks World



Vocabulary

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

Unary Predicates:

- `clear` - blocks with no blocks on top.
- `table` - blocks on the table.

Binary Predicates:

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

Ternary Predicates:

- `stack` - triples of blocks arranged in a stack.

Dataset

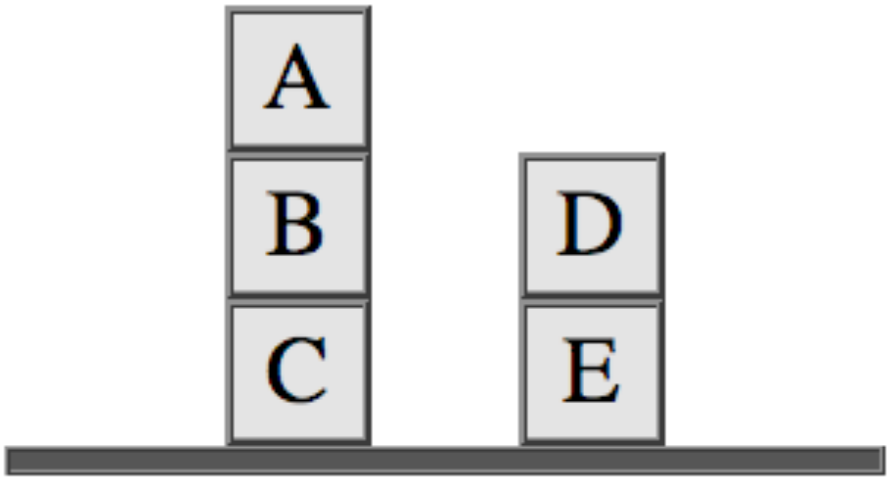
```
clear(a)
clear(d)

table(c)
table(e)

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

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

stack(a,b,c)
```



Definability

Some relations definable in terms of others

- e.g. we can define `clear` in terms of `on`
- e.g. we can define `table` in terms of `on`
- e.g. we can define `stack` in terms of `on`
- e.g. we can define `above` in terms of `on`

Constraints

Physical constraints

e.g. `on(a,a)` X

e.g. `{...,on(a,b),...,on(b,a),...}` X

Regulatory constraints

e.g. no highrises

`{...,on(a,b),...,on(b,c),...,on(c,d),...}` X

University

University

Students:	Departments:	Faculty:	Years:
aaron	architecture	alan	freshman
belinda	computers	cathy	sophomore
calvin	english	donna	junior
george	physics	frank	senior

Predicate:

```
student(Student,Department,Advisor,Year)
```

Dataset:

```
student(aaron,architecture,alan,freshman)
student(belinda,computers,cathy,sophomore)
student(calvin,english,donna,junior)
student(george,physics,frank,senior)
```

Missing Values

Suppose a student has not declared a major.
What if a student does not have an advisor?

Leave out fields (syntactically illegal):

```
student(aaron,,,freshman)
```

Add suitable values to vocabulary (new symbol):

```
student(aaron,undeclared,orphan,freshman)
```

Database nulls (new linguistic feature):

```
student(aaron,null,null,freshman)
```

Multiple Values

Suppose a student has *two* majors.

Multiple Rows (storage, update inconsistencies):

```
student(calvin,english,junior)
student(calvin,physics,junior)
```

Multiple fields (storage, extensibility?):

```
student(calvin,english,physics,junior)
student(george,physics,physics,senior)
```

Use compound names:

```
student(calvin,pair(english,physics),junior)
```

Triples

Represent wide relations as collections of binary relations.

Wide Relation:

```
student(Student,Department,Advisor,Year)
```

Binary Relations:

```
student.major(Student,Department)
student.advisor(Student,Faculty)
student.year(Student,Year)
```

Always works when there is a field of the wide relation (called the **key**) that uniquely specifies the values of the other elements. If none exists, possible to create one.

Triples

```
student.major(aaron,architecture)
student.advisor(aaron,alan)
student.year(aaron,freshman)

student.year(belinda,sophomore)

student.major(calvin,english)
student.major(calvin,physics)
student.advisor(calvin,donna)
student.year(calvin,senior)

student.major(george,physics)
student.advisor(george,frank)
student.year(george,senior)
```

Terminology

Classes

`student, department, faculty, year`

Attributes (binary relations associated with a class):

```
student.major(Student,Department)
student.advisor(Student,Faculty)
student.year(Student,Year)
```

Properties of Attributes:

domain is class of objects in first position (arguments)
range is class of objects in second position (values)
unique if *at most* one value for each argument
total if *at least* one value for each argument

Subtlety

Missing information

there is a value but we do not know it.

e.g. Aaron has an advisor but we do not know who it is.

Non-existent value

there is no value

e.g. Aaron does not have an advisor.

For now, in talking about datasets, we assume full info. If a value is missing, there is none.

Sales

Sales Ledgers

In 2015, Art sold Arborhouse to Bob for $1000000.
In 2016, Bob sold Pelicanpoint to Carl for $2000000.
In 2016, Carl sold Ravenswood to Dan in $2000000.
In 2017, Dan sold Ravenswood to Art for $3000000.

Real Estate Ledger

People:	Properties:	Years:	Money:
art	arborhouse	2015	1000000
bob	pelicanpoint	2016	2000000
carl	ravenswood	2017	3000000
dan			

Relation Constant:

```
sale(Year,Seller,Property,Buyer,Amount)
```

Dataset:

```
sale(2015,art,arborhouse,bob,1000000)
sale(2016,art,pelicanpoint,bob,2000000)
sale(2016,carl,ravenswood,dan,2000000)
sale(2017,dan,arborhouse,art,3000000)
```

Sales Ledgers

In 2015, Art sold Arborhouse to Bob for $1000000.
In 2016, Bob sold Pelicanpoint to Carl for $2000000.
In 2016, Carl sold Ravenswood to Dan in $2000000.
In 2017, Dan sold Ravenswood to Art for $3000000.

In 2015, Art sold Bob a widget for $10.
In 2016, Art sold Bob a gadget for $20.
In 2016, Art sold Bob another gadget for $20.
In 2017, Art sold Bob a framis for $30.

Sales Ledger

People:	**Items:**	**Years:**	**Money:**
art	widget	2015	10
bob	gadget	2016	20
carl	framis	2017	30
dan			

Relation Constant:

```
sale(Year,Seller,Item,Buyer,Amount)
```

Dataset:

```
sale(2015,art,widget,bob,10)
sale(2016,art,gadget,bob,20)
sale(2016,art,gadget,bob,20)
sale(2017,art,framis,bob,30)
```

Duplicate factoid!?

Sales Ledger

Sales:	People:	Items:	Years:	Money:
t1	art	widget	2015	10
t2	bob	gadget	2016	20
t3	carl	framis	2017	30
t4	dan			

Relation Constant:

```
sale(Sale,Year,Seller,Item,Buyer,Amount)
```

Dataset:

```
sale(t1,2015,art,widget,bob,10)
sale(t2,2016,art,gadget,bob,20)
sale(t3,2016,art,gadget,bob,20)
sale(t4,2017,art,framis,bob,30)
```

Sierra

Sierra

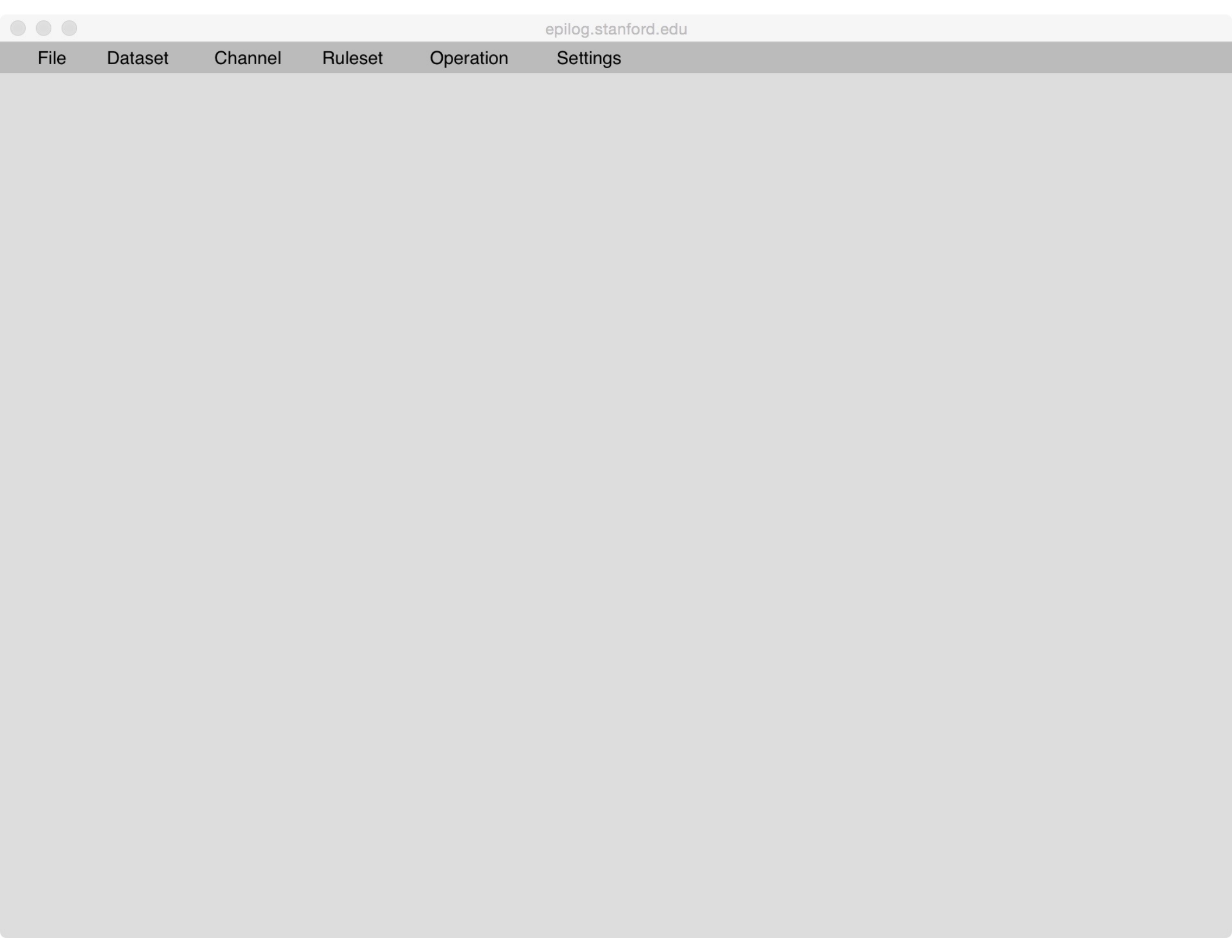
Sierra is browser-based IDE (interactive development environment) for Epilog.

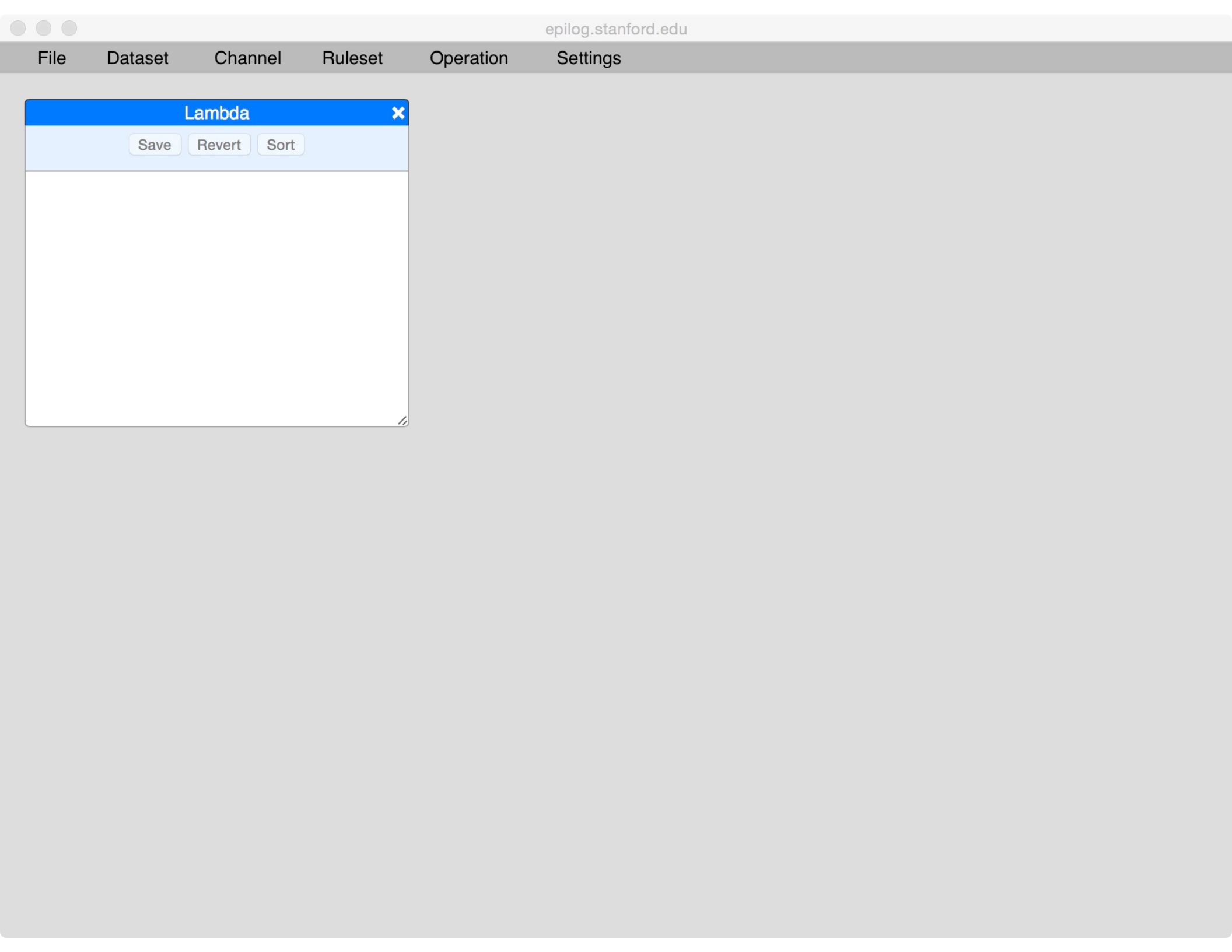
Saving and loading files

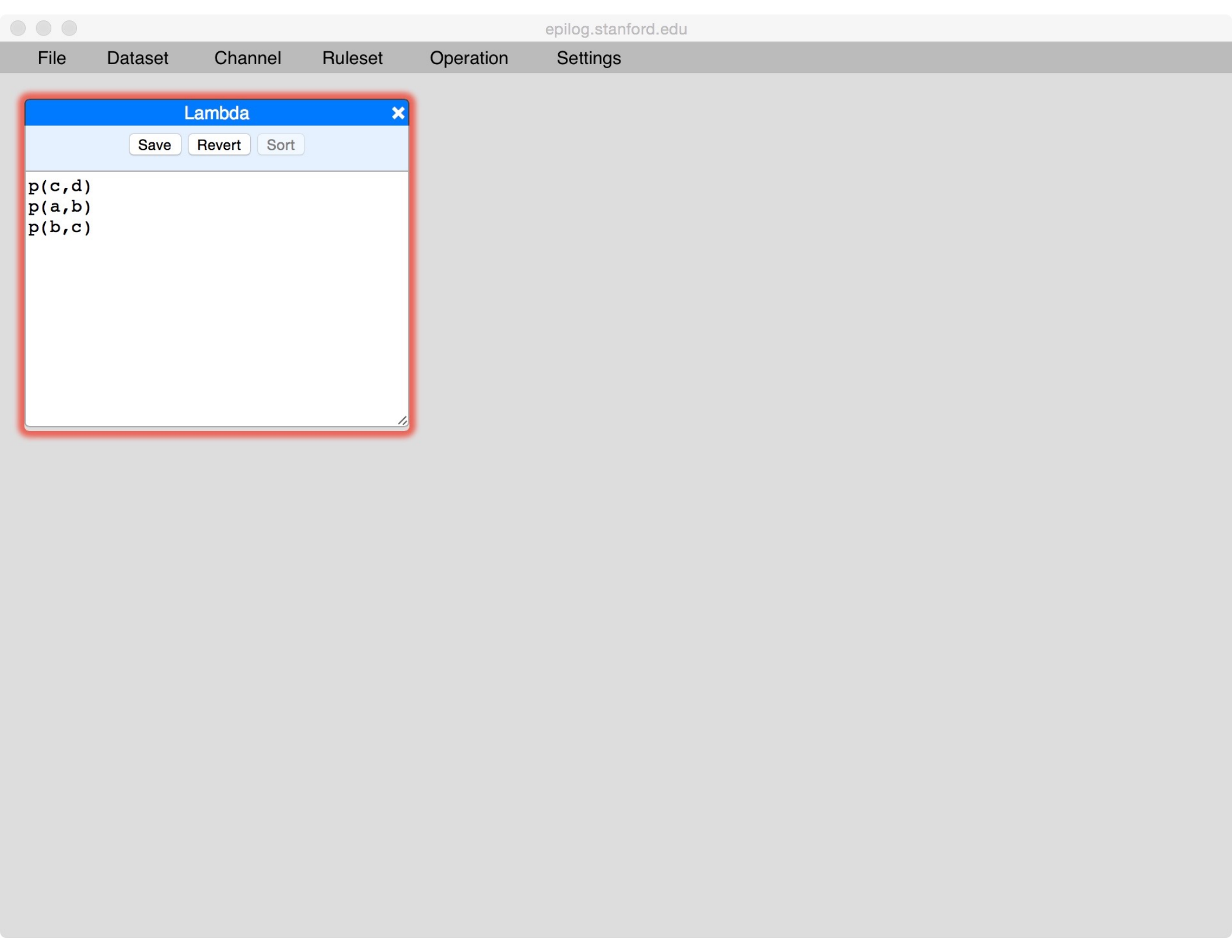
Visualization of datasets
Searching datasets
Updating datasets

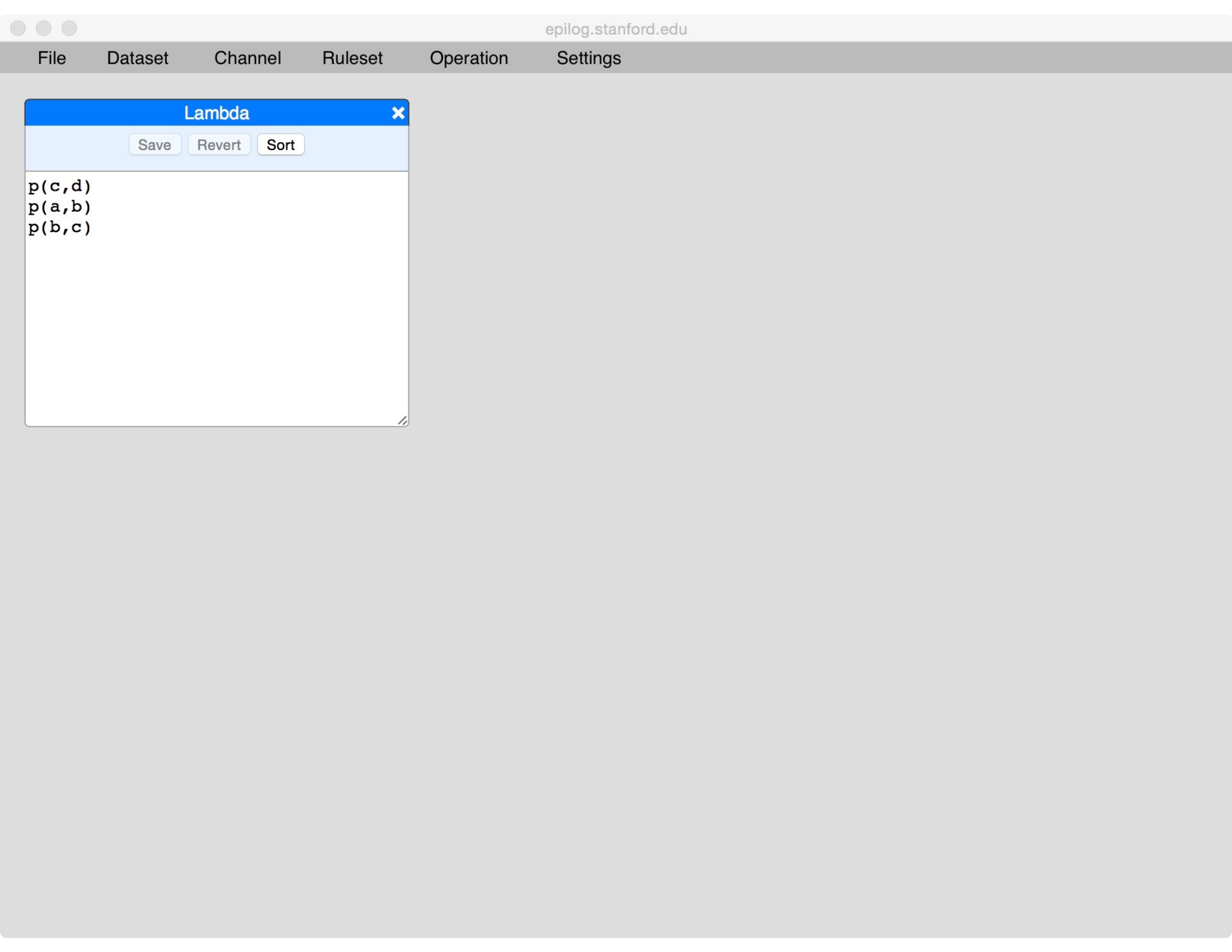
Interpreter (for executing queries and updates)
Trace capability (for debugging rules)
Analysis tools (error checking and optimization)

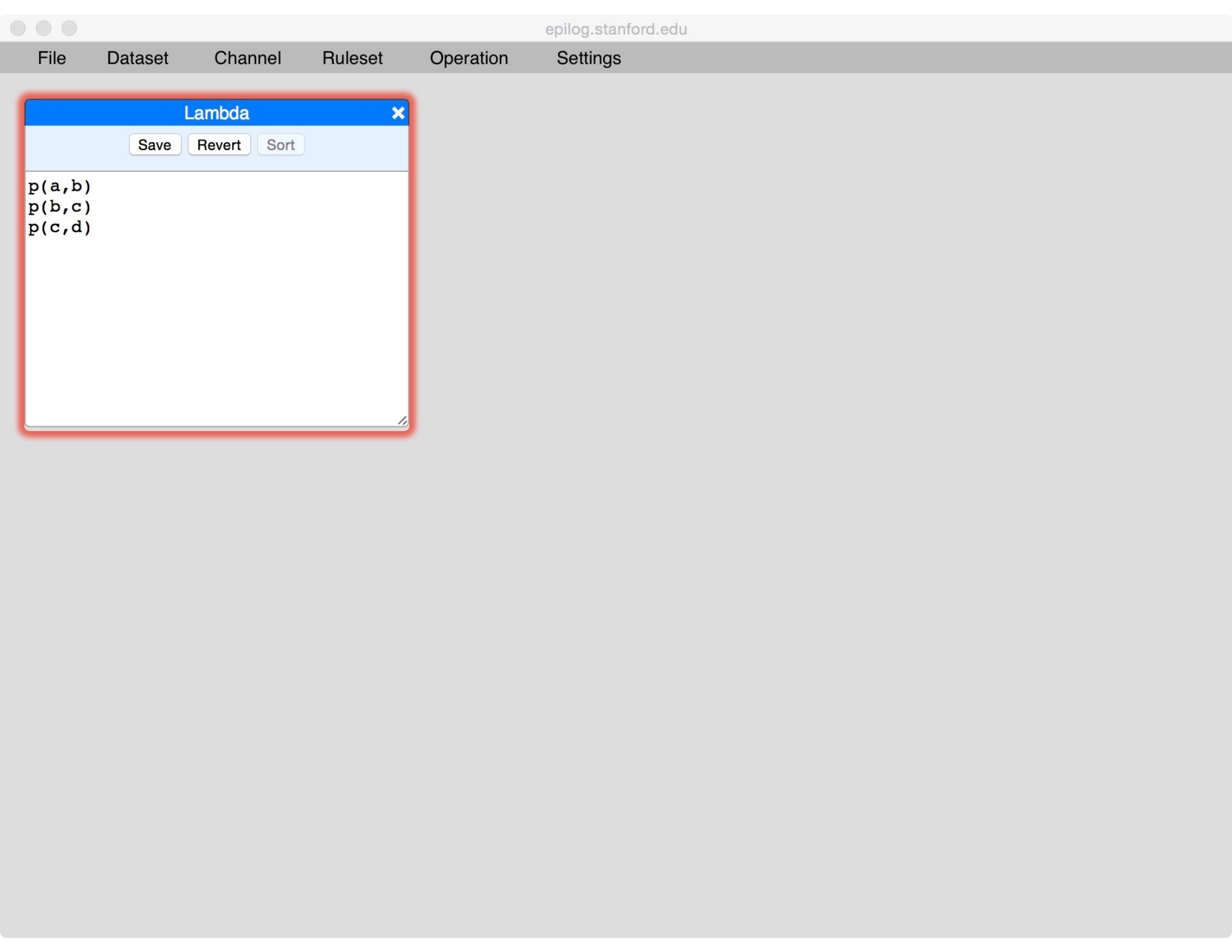
http://epilog.stanford.edu/homepage/sierra.php

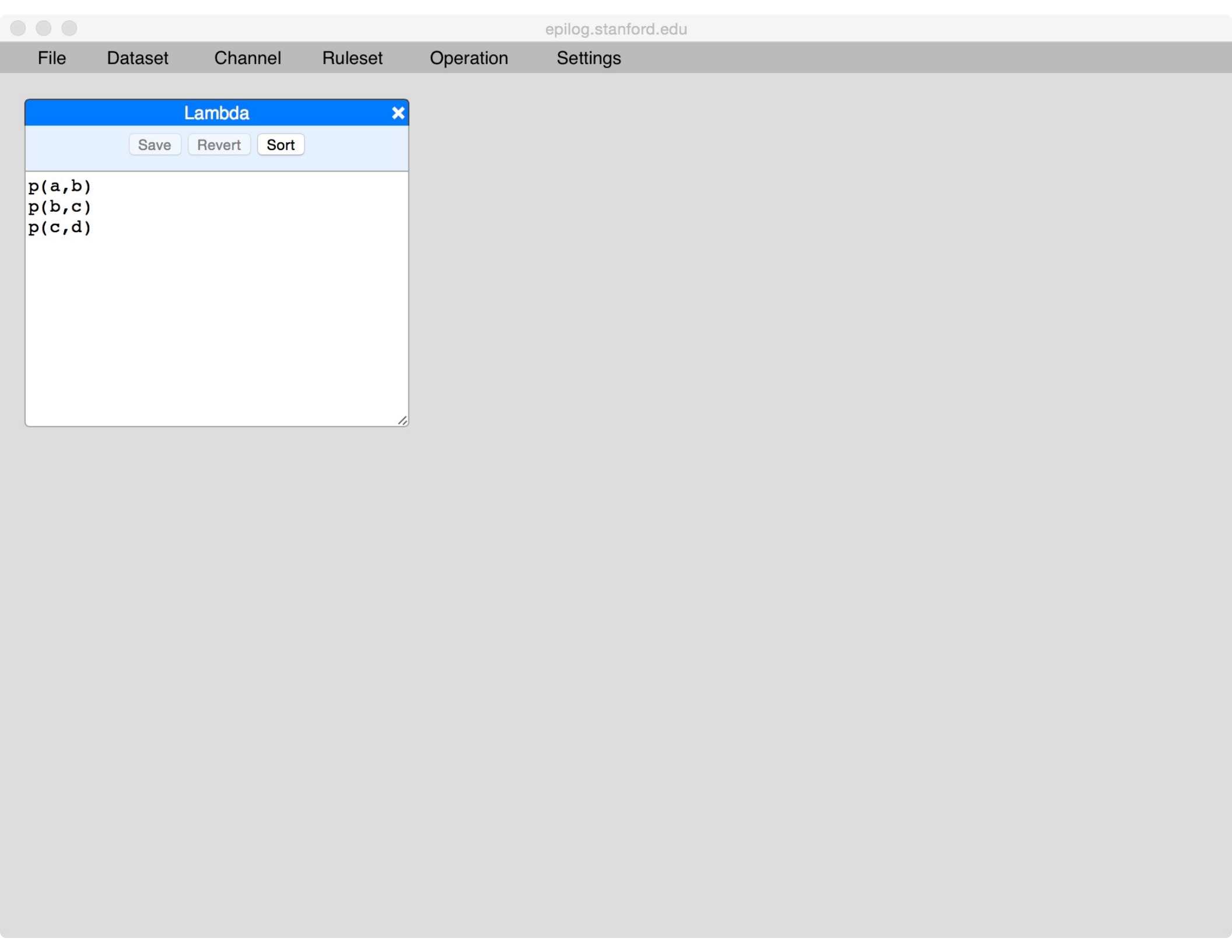

epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings

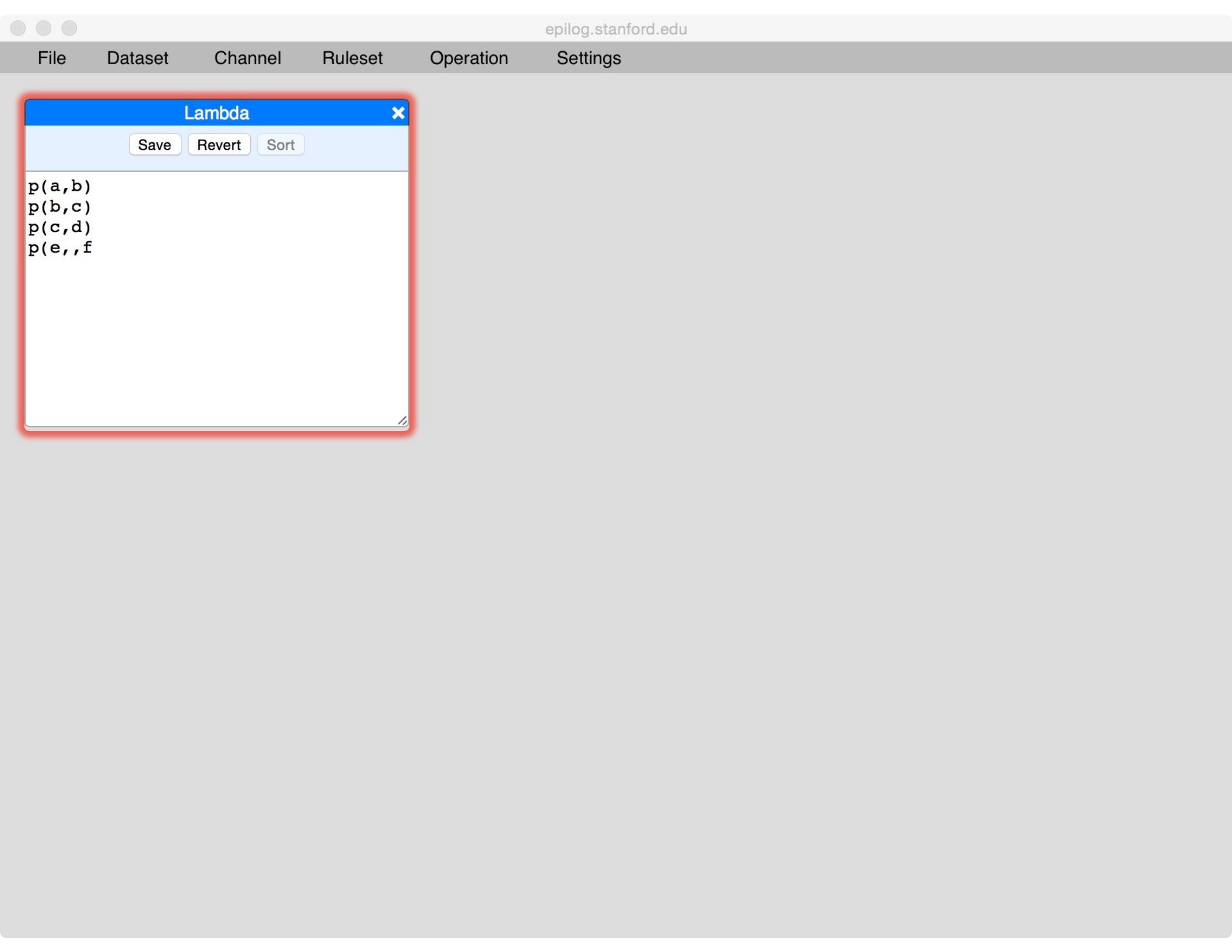

epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(c,d)
p(a,b)
p(b,c)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(c,d)
p(a,b)
p(b,c)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)
p(e,,f

File Dataset Channel Ruleset Operation Settings

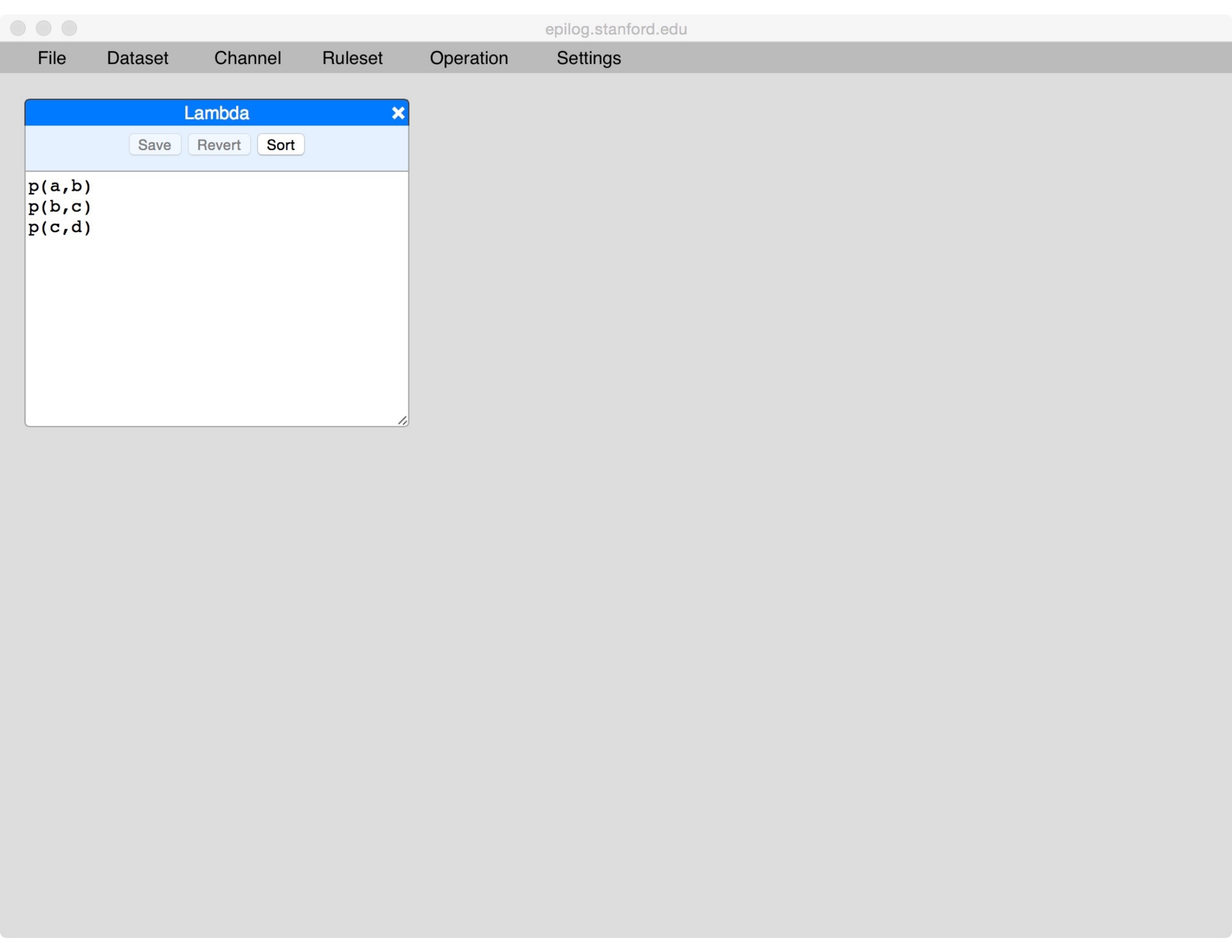
Lambda

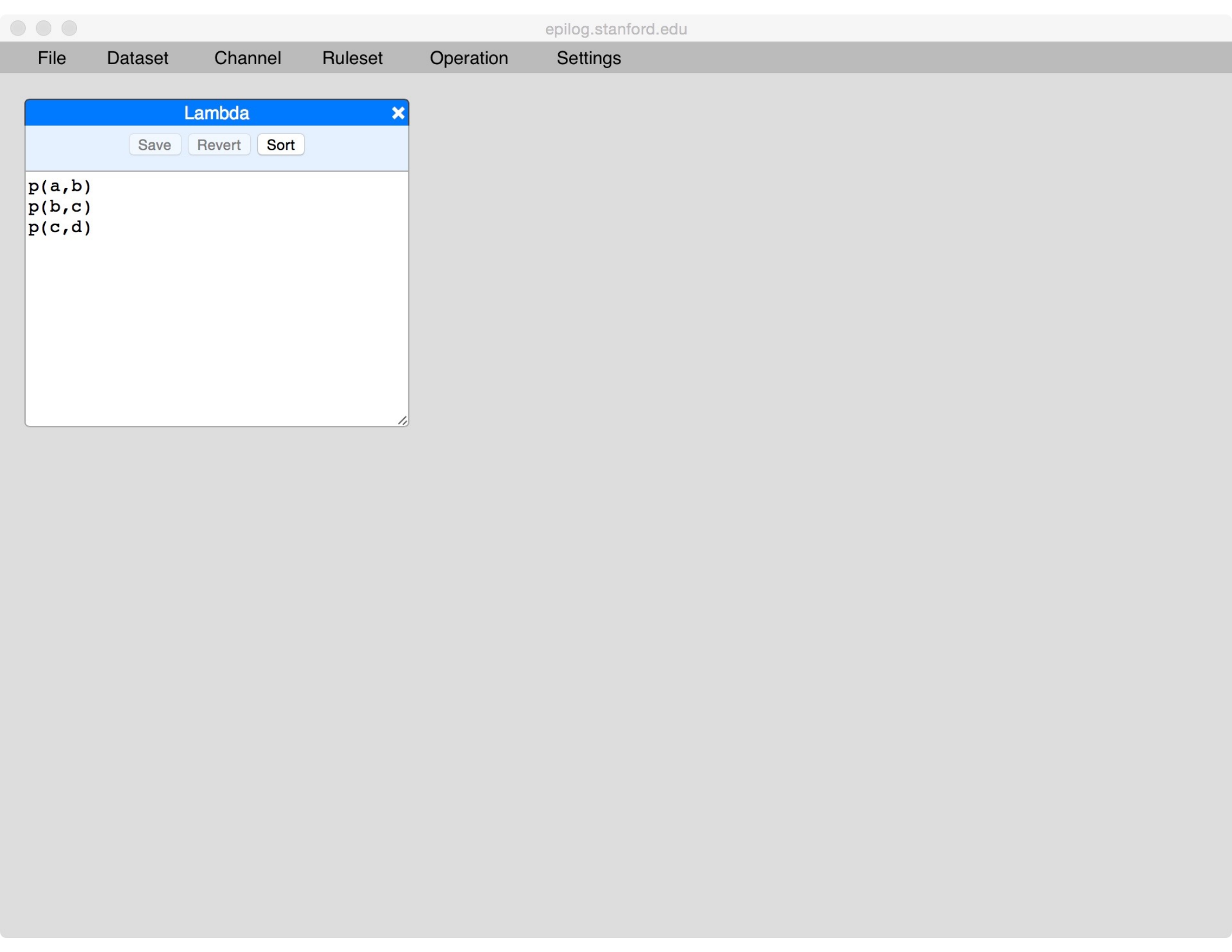
Save Revert Sort

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

Syntax error.

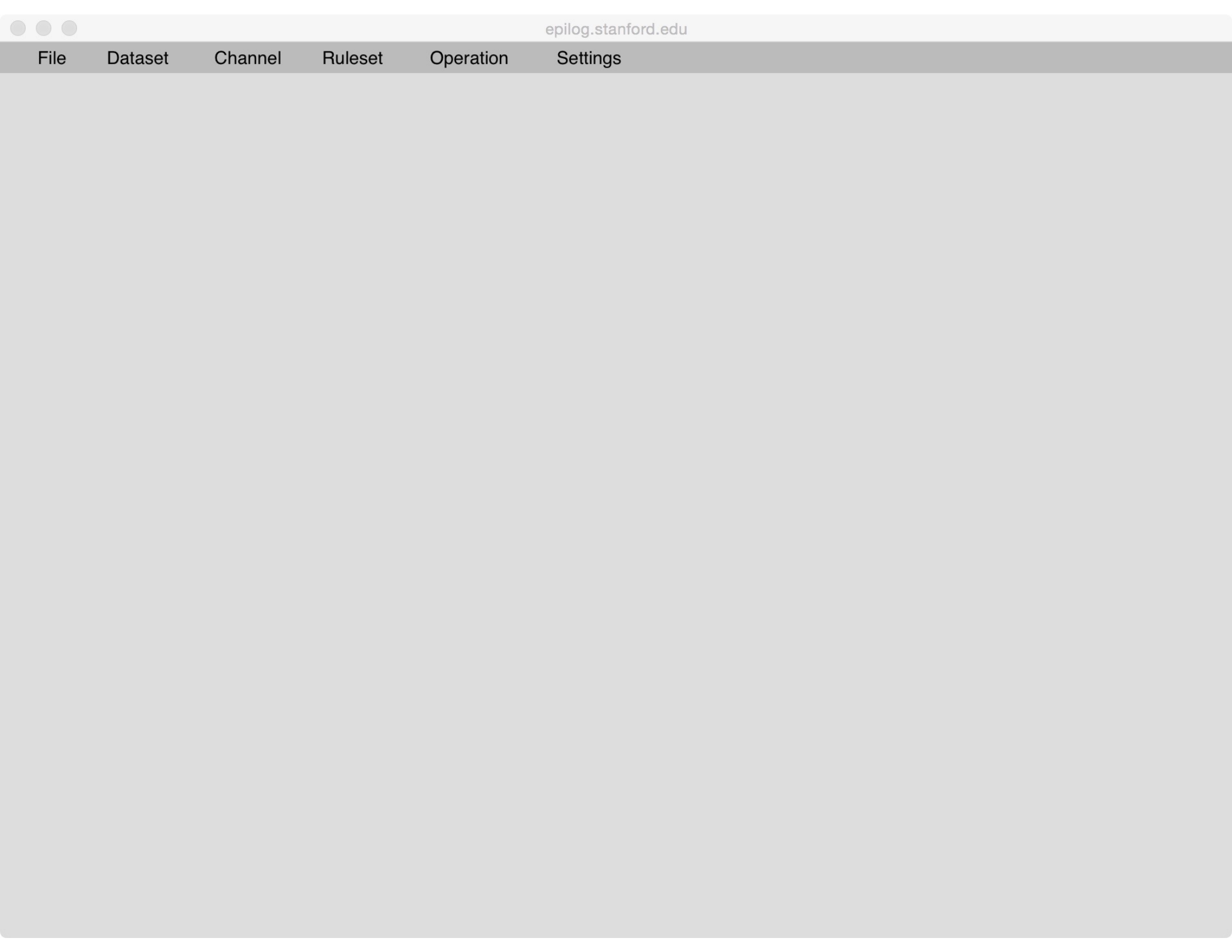
Close


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
p(a,b)
p(b,c)
p(c,d)

New Dataset

Save Configuration


epilog.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings

Load Configuration

Assignments

Readings

Reading 2.1 - Ontologies

Assignment - Sierra

The goal of this exercise is for you to familiarize yourself with the updates mechanism of Sierra. As always, 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 Sections 1-5 of the documentation and reproduce the examples in the Sierra window you opened earlier. Read section 9 and play around with saving and loading data and configurations.

Assigment - Movies

Consider a vocabulary that includes the following relations.

`movie.instance(`x`)` means that x is a movie.
`actor.instance(`x`)` means that x is an actor.
`director.instance(`x`)` means that x is a director.
`year.instance(`x`)` means that x is a year.
`title.instance(`x`)` means that x is a title.

`movie.star(`x`,`y`)` means that movie x stars actor y.
`movie.director(`x`,`y`)` means that movie x was directed by y.
`movie.year(`x`,`y`)` means that movie x was released in year y.
`movie.title(`x`,`y`)` means that movie x has the title y.

Choose symbols for a few movies, actors, directors, years, and titles, and encode the relevant data about these entities using this vocabulary.

Unauthorized Practice of Law

CA Business & Professions Code Chapter 4 Article 7:

* No person can "practice law" or advertise such, unless an active member of CA State Bar (§6125)

* Penalty for UPL: Fine of $1,000 or 1 year in jail (§6126(a))

* Court as fiduciary can take over unlicensed attorney's office and case (§6126(b – m)

* Damage remedies for UPL (§6126.5)

* Any attorney dealing with an UPL person is guilty of a misdemeanor (§6128)

Question

Suppose that a **machine** were able to satisfy the requirements for the admission to the Bar.

Would it be permitted to offer legal advice without running afoul of UPL restrictions?

Licensure Requirements

Graduation from accredited law school

* Bar exam (multi-state or state-specific)

Ethics Exam

Registration for admission to the Bar

Examination of applicant's moral character

Multi-State Bar Exam

Format:

200 multiple choice questions

Subject Matter:

civil procedure, contracts, criminal law, commercial transactions, torts, etc.

Assignment - Bar Exam

The Multistate Bar Exam (MBE) consists of 200 multiple choice questions covering a wide range of legal topics such as civil procedure, contracts, criminal law and procedure, commercial transactions, evidence, land use/real property, and negligence/torts. We are providing you with a selection of typical questions. Your mission on this assignment is to invent an ontology for one of these questions (your choice) and to use your ontology in encoding the *facts* relevant to your chosen question that will allow someone who knows the relevant *law* to answer the question.

Example

A woman from State A filed an action against a retailer in a state court in State B. The complaint alleged that the retailer had not delivered $100,000 worth of goods for which the woman had paid. Twenty days after being served, the retailer, which is incorporated in State C and has its principal place of business in State B, filed a notice of removal in a federal district court in State B. Was the action properly removed?

(A) No, because the notice of removal was not timely filed.
(B) No, because the retailer is a citizen of State B.
(C) Yes, because the parties are citizens of different states and more than $75,000 is in controversy.
(D) Yes, because the retailer is a citizen of State B and C.

Example

A man sued a railroad for personal injuries suffered when his car was struck by a train at an unguarded crossing. A major issue is whether the train sounded its whistle before arriving at the crossing. The railroad has offered the testimony of a resident who has lived near the crossing for 15 years. Although she was not present on the occasion in question, she will testify that, whenever she is home, the train always sounds its whistle before arriving at the crossing. Is the resident's testimony admissible?

(A) No, due to the resident's lack of personal knowledge regarding the incident in question.
(B) No, because habit evidence is limited to the conduct of persons, not businesses.
(C) Yes, as evidence of a routine practice.
(D) Yes, as a summary of her present sense impressions.

CODEX

The Stanford Center for Legal Informatics

Legal Empowerment through Information Technology

CODEX

The Stanford Center for Legal Informatics

Legal Empowerment through Information Technology