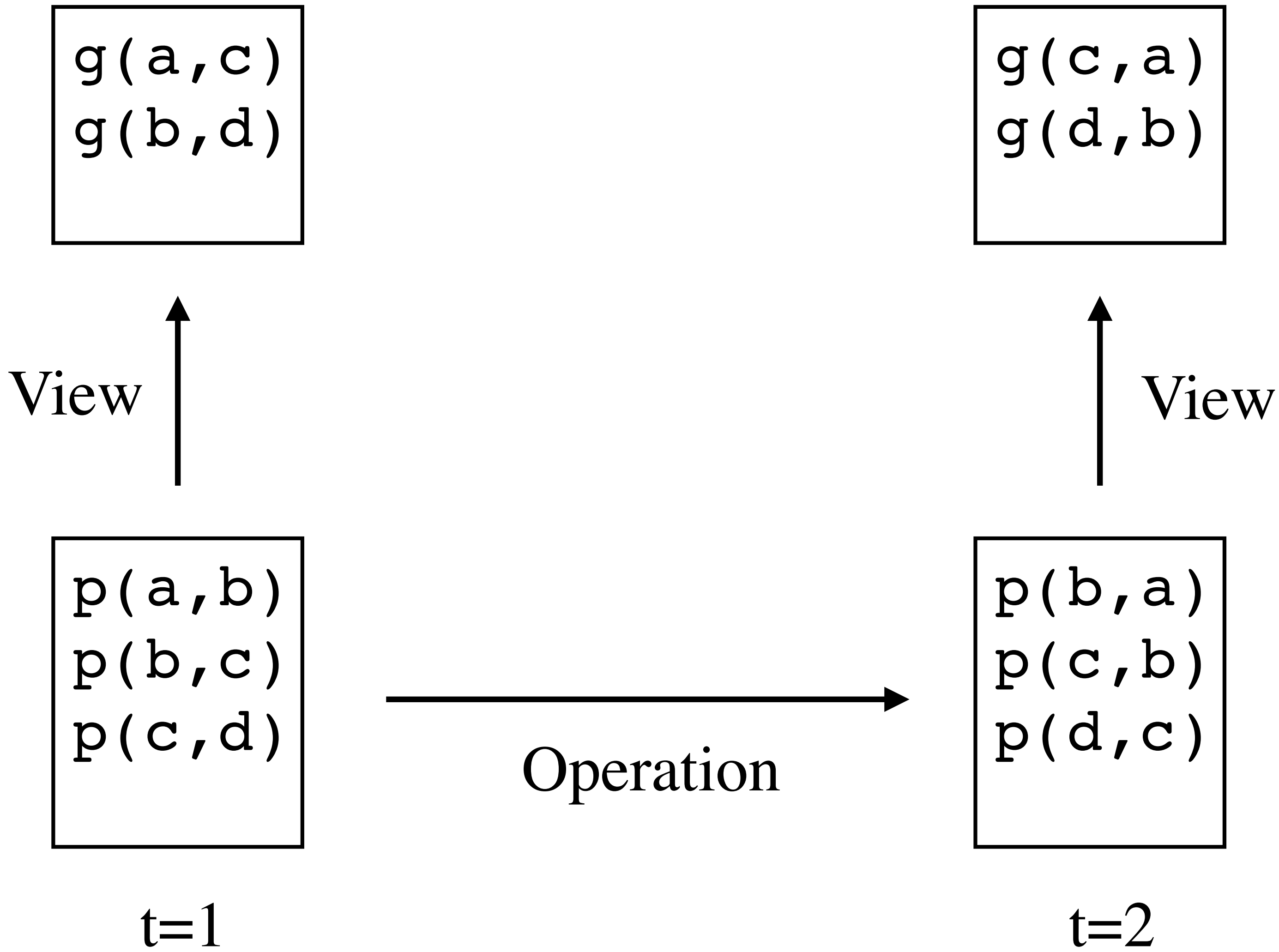
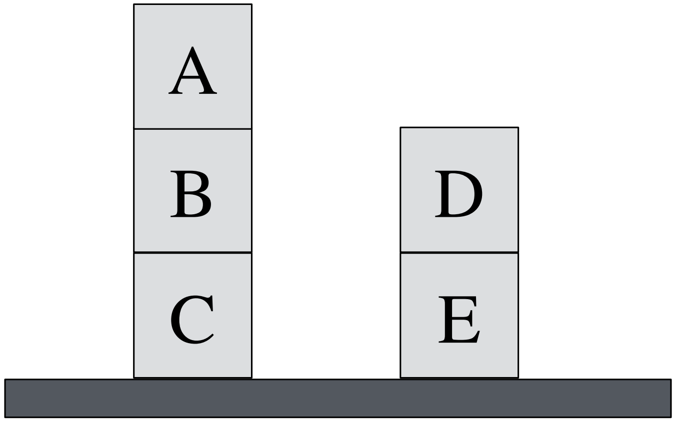
Logic Programming
Model Management

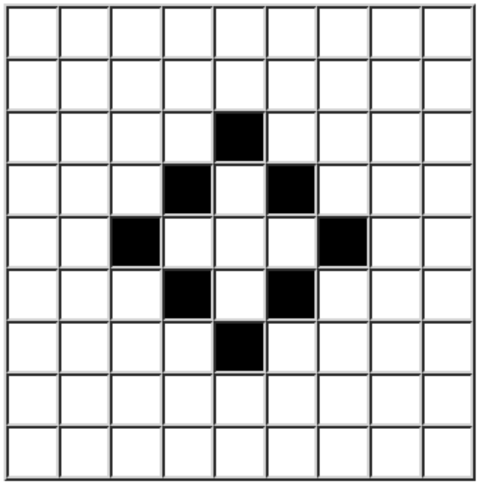
Michael Genesereth
Computer Science Department
Stanford University

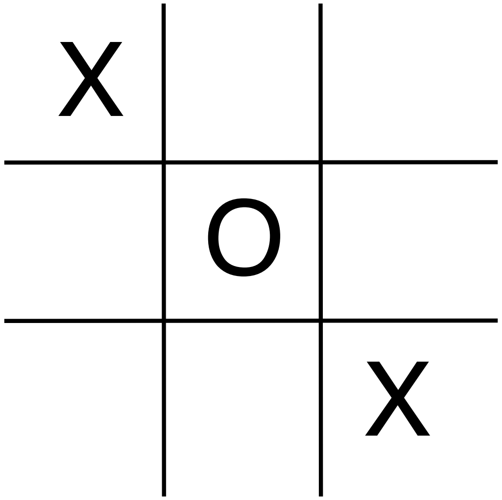
Operations


g(a,c)
g(b,d)
g(c,a)
g(d,b)
View
View
p(a,b)
p(b,c)
p(c,d)
Operation
p(b,a)
p(c,b)
p(d,c)
t=1
t=2

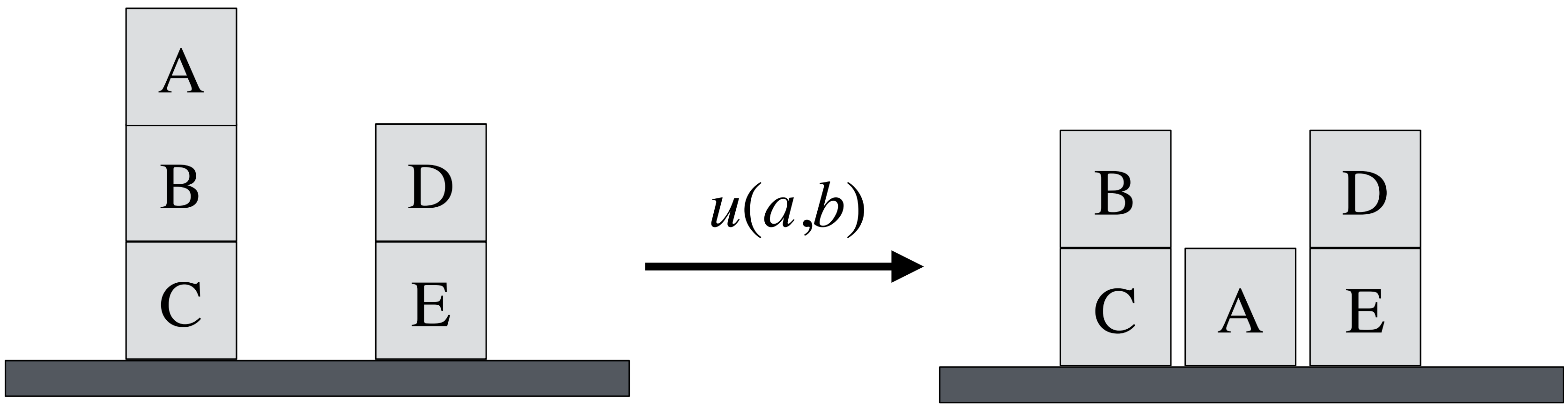
Dynamic Worlds







Changing Worlds

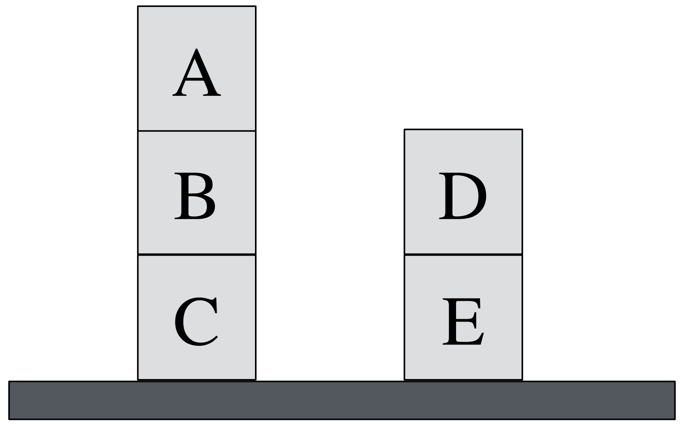



```
on(a,b)
on(b,c)
on(d,e)
```

$\xrightarrow{\texttt{u(a,b)}}$

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

Changing Models




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

augment →

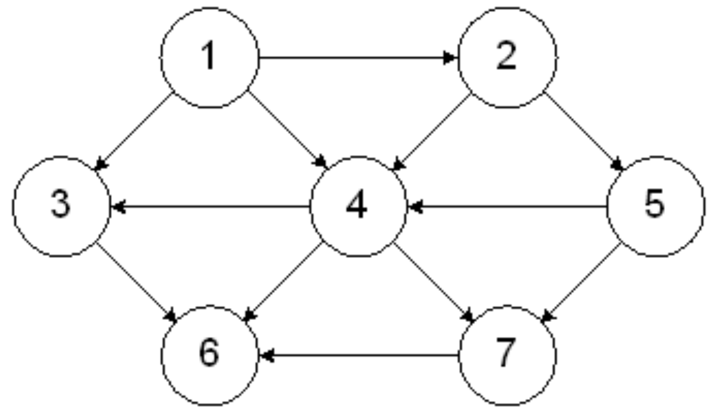
```
on(a,b)
on(b,c)
on(d,e)
clear(a)
clear(d)
table(c)
table(e)
```

Programme

Tables versus Triples

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

Graph Editing



View Materialization

Order table

Partition key	Row key	Order date	Shipping address	Total invoice	Order status
001 (Customer ID)	1 (Order ID)	11082013	One Microsoft way Redmond, WA 98052	$400	In process
005	2	11082013	One Microsoft way Redmond, WA 98052	$200	Shipped

OrderItem table

Partition key	Row key	Product	Unit Price	Amount	Total
1 (Order ID)	001_1 (OrderItem ID)	XX	$100	2	$200
1	001_2	YY	$40	5	$200
2	002_1	ZZ	$200	1	$200

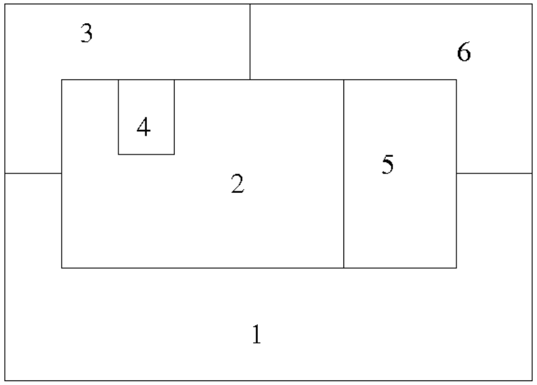
Customer table

Partition key	Row key	Billing Information	Shipping address	Gender	Age
US East (region)	001 (Customer ID)	*****0001	One Microsoft way Redmond, WA 98052	Female	30
US East	002	*****2006	One Microsoft way Redmond, WA 98052	Male	40

Materialized View

Partition key	Row key	Product Name	Total sold	Number of customers
Electronics (Product category)	001 (Product ID)	XX	$30,000	500
Electronics	002	YY	$100,000	400

Constraint Propagation



Tables versus Triples

University

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

Tables

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)
```

Triples

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

student.major(belinda,computers)
student.advisor(belinda,cathy)
student.year(belinda,sophomore)

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

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

Convert to Triples

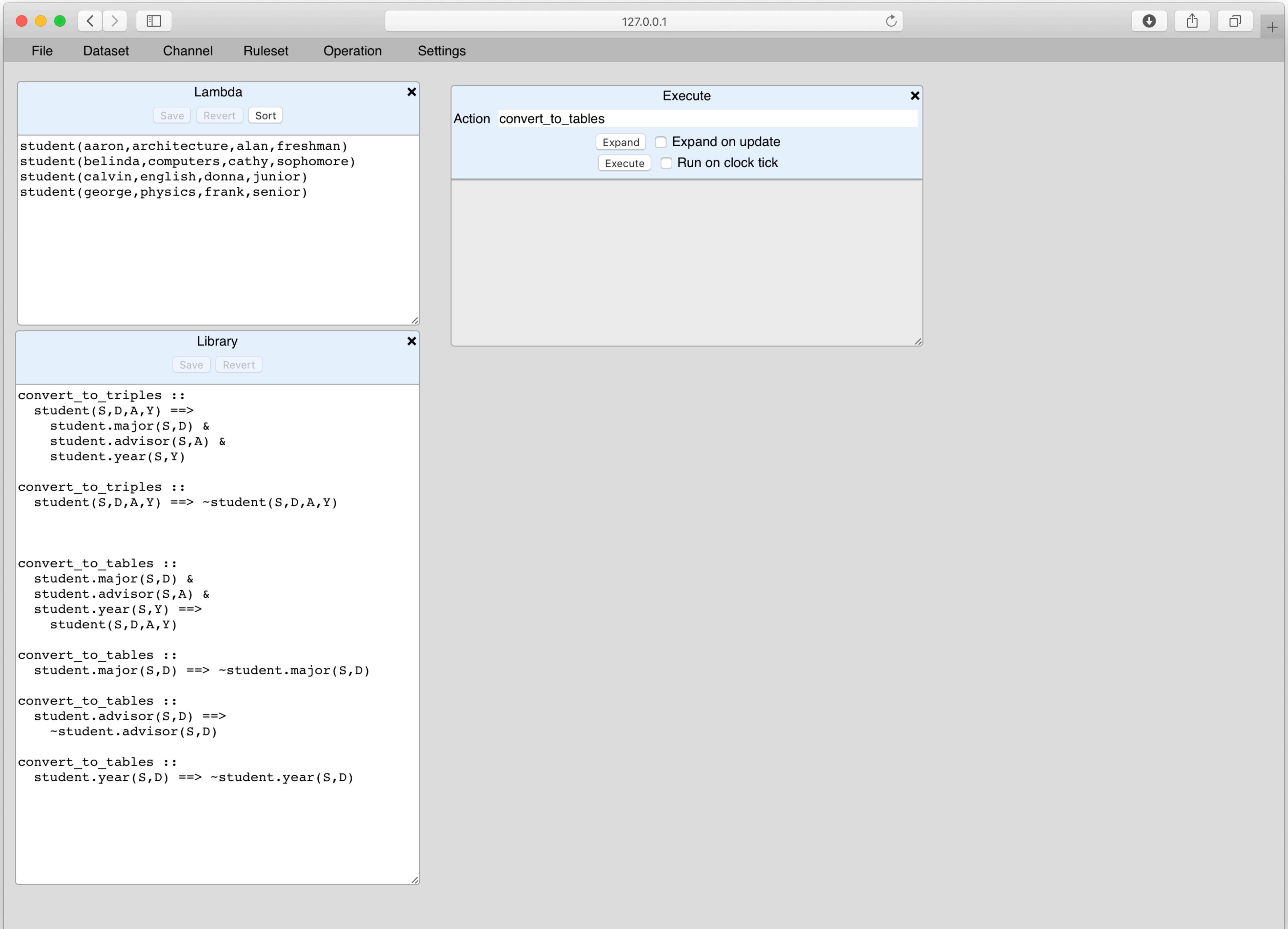
```
convert_to_triples ::
  student(S,D,A,Y) ==>
    student.major(S,D) &
    student.advisor(S,A) &
    student.year(S,Y)

convert_to_triples ::
  student(S,D,A,Y) ==> ~student(S,D,A,Y)
```

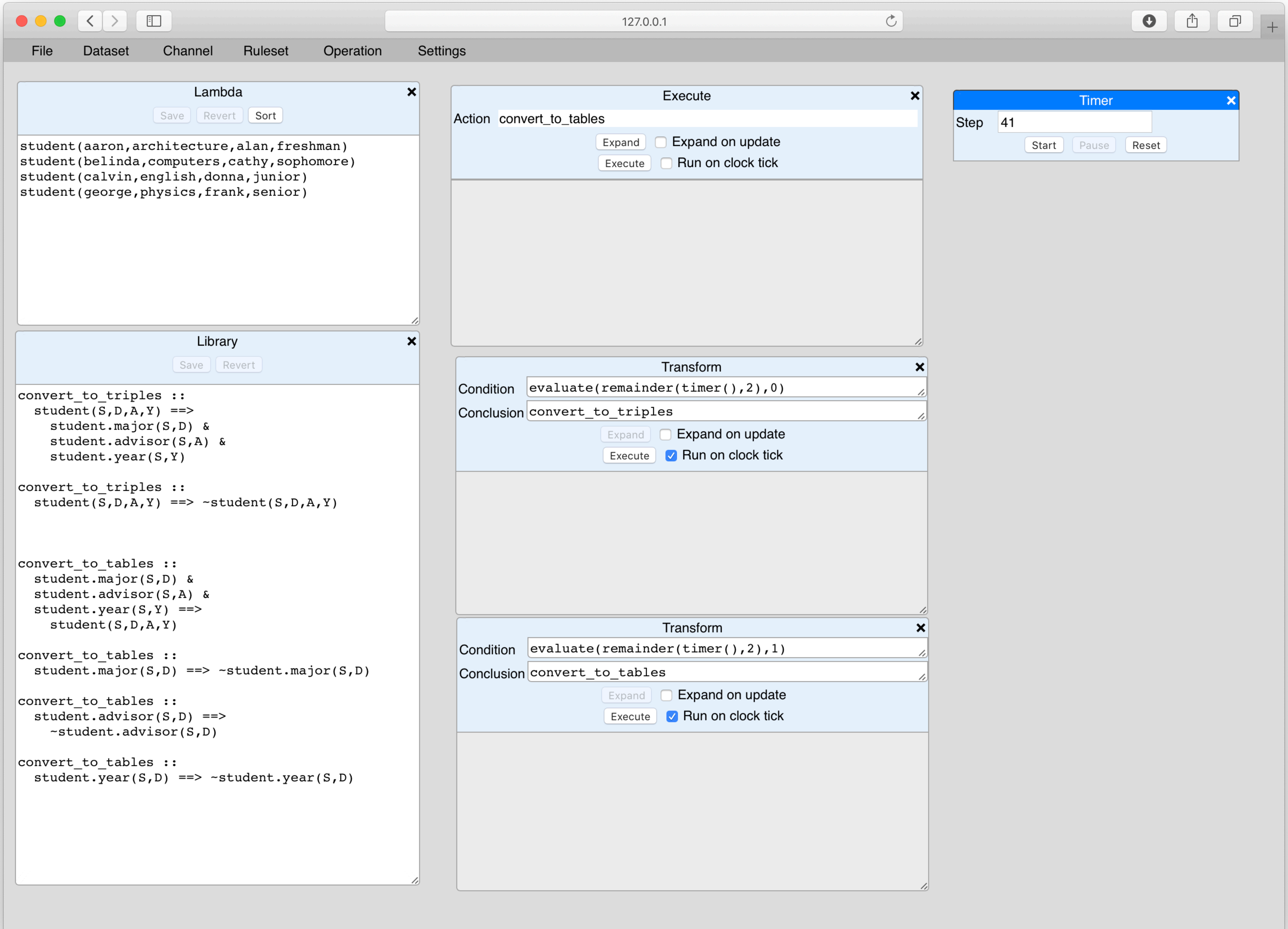
- or -

```
convert_to_triples ::
  student(S,D,A,Y) ==>
    ~student(S,D,A,Y) &
    student.major(S,D) &
    student.advisor(S,A) &
    student.year(S,Y)
```

Convert to Tables

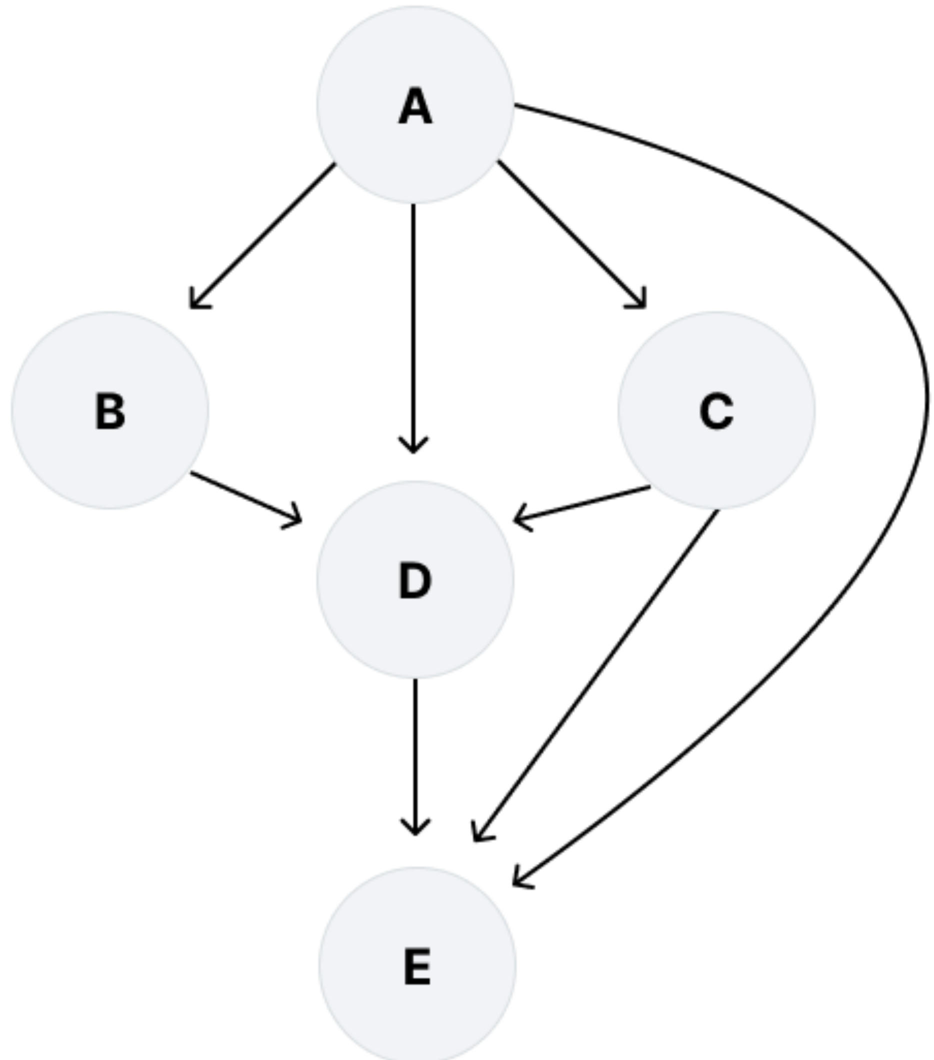
```
convert_to_tables ::
  student.major(S,D) &
  student.advisor(S,A) &
  student.year(S,Y) ==>
    student(S,D,A,Y)

convert_to_tables ::
  student.major(S,D) ==> ~student.major(S,D)

convert_to_tables ::
  student.advisor(S,D) ==>
    ~student.advisor(S,D)

convert_to_tables ::
  student.year(S,D) ==> ~student.year(S,D)
```


127.0.0.1
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
student(aaron,architecture,alan,freshman)
student(belinda,computers,cathy,sophomore)
student(calvin,english,donna,junior)
student(george,physics,frank,senior)
Library
Save
Revert
convert_to_triples ::
  student(S,D,A,Y) ==>
    student.major(S,D) &
    student.advisor(S,A) &
    student.year(S,Y)
convert_to_triples ::
  student(S,D,A,Y) ==> ~student(S,D,A,Y)
convert_to_tables ::
  student.major(S,D) &
  student.advisor(S,A) &
  student.year(S,Y) ==>
    student(S,D,A,Y)
convert_to_tables ::
  student.major(S,D) ==> ~student.major(S,D)
convert_to_tables ::
  student.advisor(S,D) ==>
    ~student.advisor(S,D)
convert_to_tables ::
  student.year(S,D) ==> ~student.year(S,D)
Execute
Action convert_to_tables
Expand
Expand on update
Execute
Run on clock tick


127.0.0.1
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
student(aaron,architecture,alan,freshman)
student(belinda,computers,cathy,sophomore)
student(calvin,english,donna,junior)
student(george,physics,frank,senior)
Library
Save
Revert
convert_to_triples ::
student(S,D,A,Y) ==>
student.major(S,D) &
student.advisor(S,A) &
student.year(S,Y)
convert_to_triples ::
student(S,D,A,Y) ==> ~student(S,D,A,Y)
convert_to_tables ::
student.major(S,D) &
student.advisor(S,A) &
student.year(S,Y) ==>
student(S,D,A,Y)
convert_to_tables ::
student.major(S,D) ==> ~student.major(S,D)
convert_to_tables ::
student.advisor(S,D) ==>
~student.advisor(S,D)
convert_to_tables ::
student.year(S,D) ==> ~student.year(S,D)
Execute
Action convert_to_tables
Expand
Expand on update
Execute
Run on clock tick
Transform
Condition evaluate(remainder(timer(),2),0)
Conclusion convert_to_triples
Expand
Expand on update
Execute
Run on clock tick
Transform
Condition evaluate(remainder(timer(),2),1)
Conclusion convert_to_tables
Expand
Expand on update
Execute
Run on clock tick
Timer
Step 41
Start
Pause
Reset

Directed Graphs

Example



Example

Dataset:

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

Operations:

`copy(X,Y)` - copy outgoing links from `X` to `Y`.

`invert(X)` - reverse the outgoing arcs from `X`.

`insert(X,Y)` - add arc from `X` to `Y` and all `Y` successors.

copy(X,Y)

Dataset:

`{edge(a,b),edge(b,d),edge(b,e)}`

Operation Definition:

`copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)`

Action: `copy(b,c)`

Positive Updates: `{edge(c,d),edge(c,e)}`

Negative Updates: `{}`

Result:

`{edge(a,b),edge(b,d),edge(b,e),edge(c,d),edge(c,e)}`

invert(X)

Dataset:

{`edge(a,b),edge(b,d),edge(b,e),edge(c,d),edge(c,e)`}

Operation Definition:

`invert(X)` - reverse the outgoing arcs from `X`.

invert(X)

Dataset:

`{edge(a,b),edge(b,d),edge(b,e),edge(c,d),edge(c,e)}`

Operation Definition:

```
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
```

Action: `invert(c)`

Positive Updates: `{edge(d,c),edge(e,c)}`

Negative Updates: `{edge(c,d),edge(c,e)}`

Result:

`{edge(a,b),edge(b,d),edge(b,e),edge(d,c),edge(e,c)}`

insert(X,Y)

Dataset:

{`edge(a,b),edge(b,d),edge(b,e),edge(d,c),edge(e,c)`}

Operation Definition:

`insert(X,Y)` - add arc from `X` to `Y` and all successors of `Y`.

insert(X,Y)

Dataset:

{edge(a,b), edge(b,d), edge(b,e), edge(d,c), edge(e,c)}

Operation Definition:

```
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
```

Action: insert(w,b)

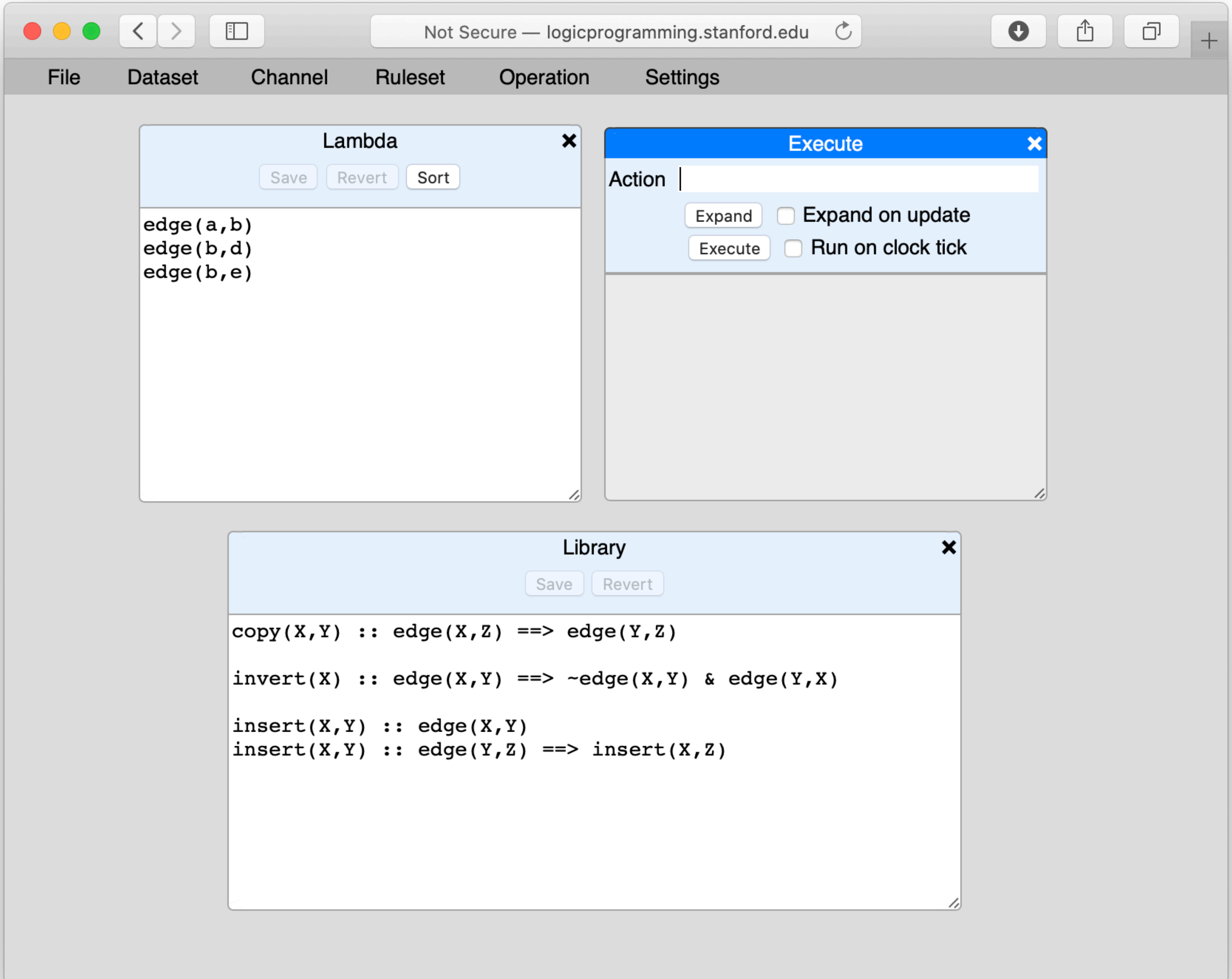
Expansion:

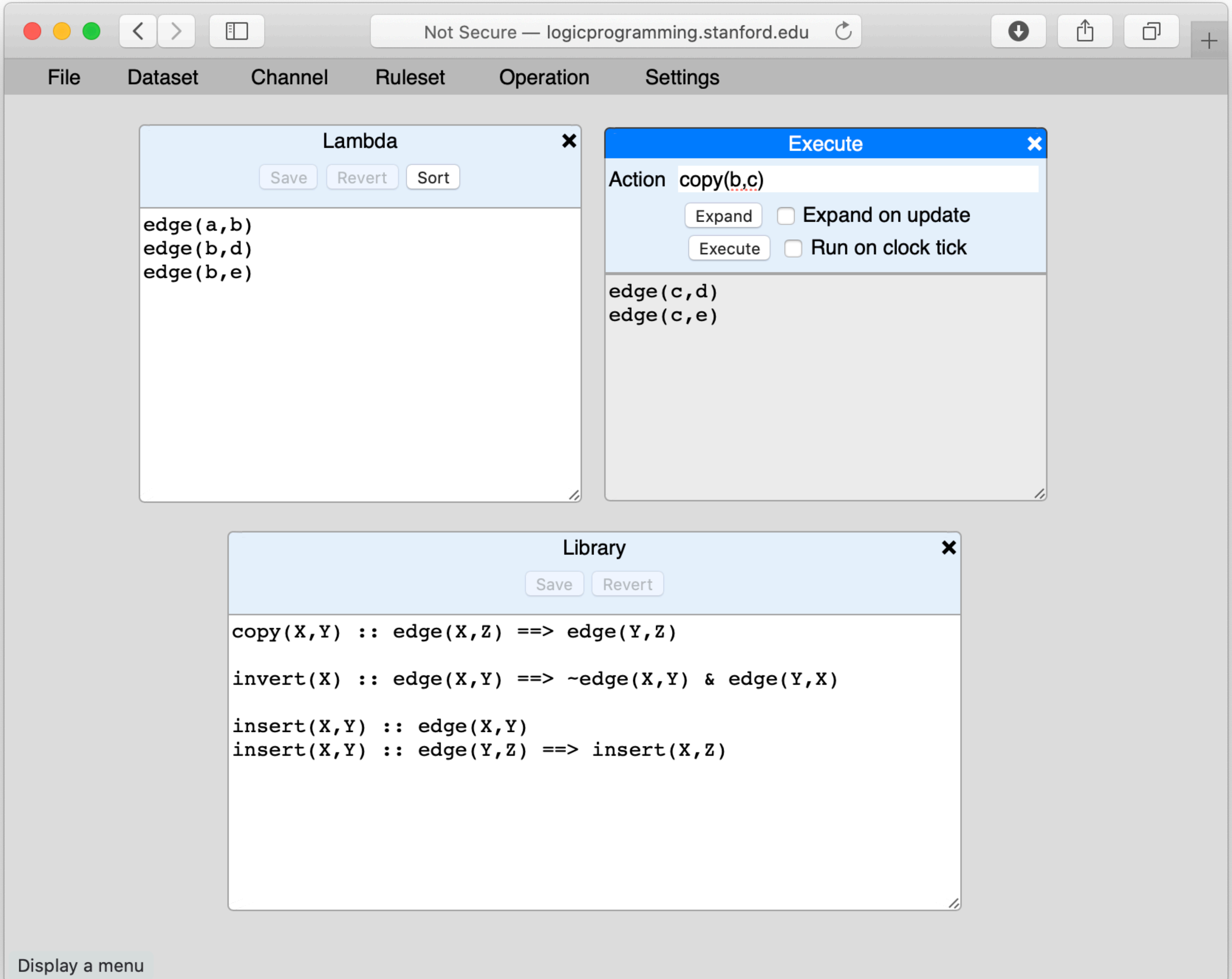
{insert(w,b), edge(w,b), insert(w,d), insert(w,e), edge(w,d), edge(w,e), insert(w,c), edge(w,c)}

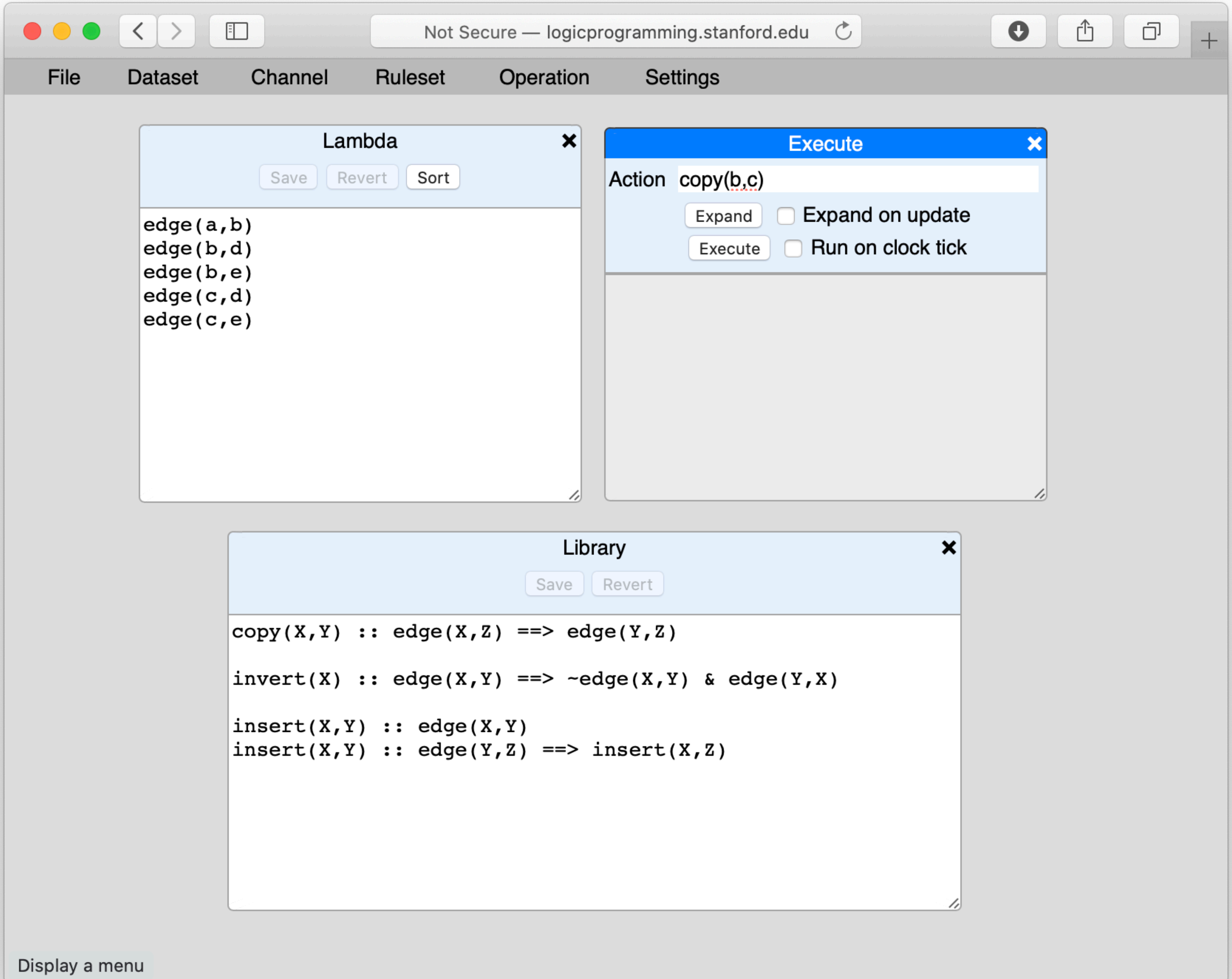
Negative Updates: {}

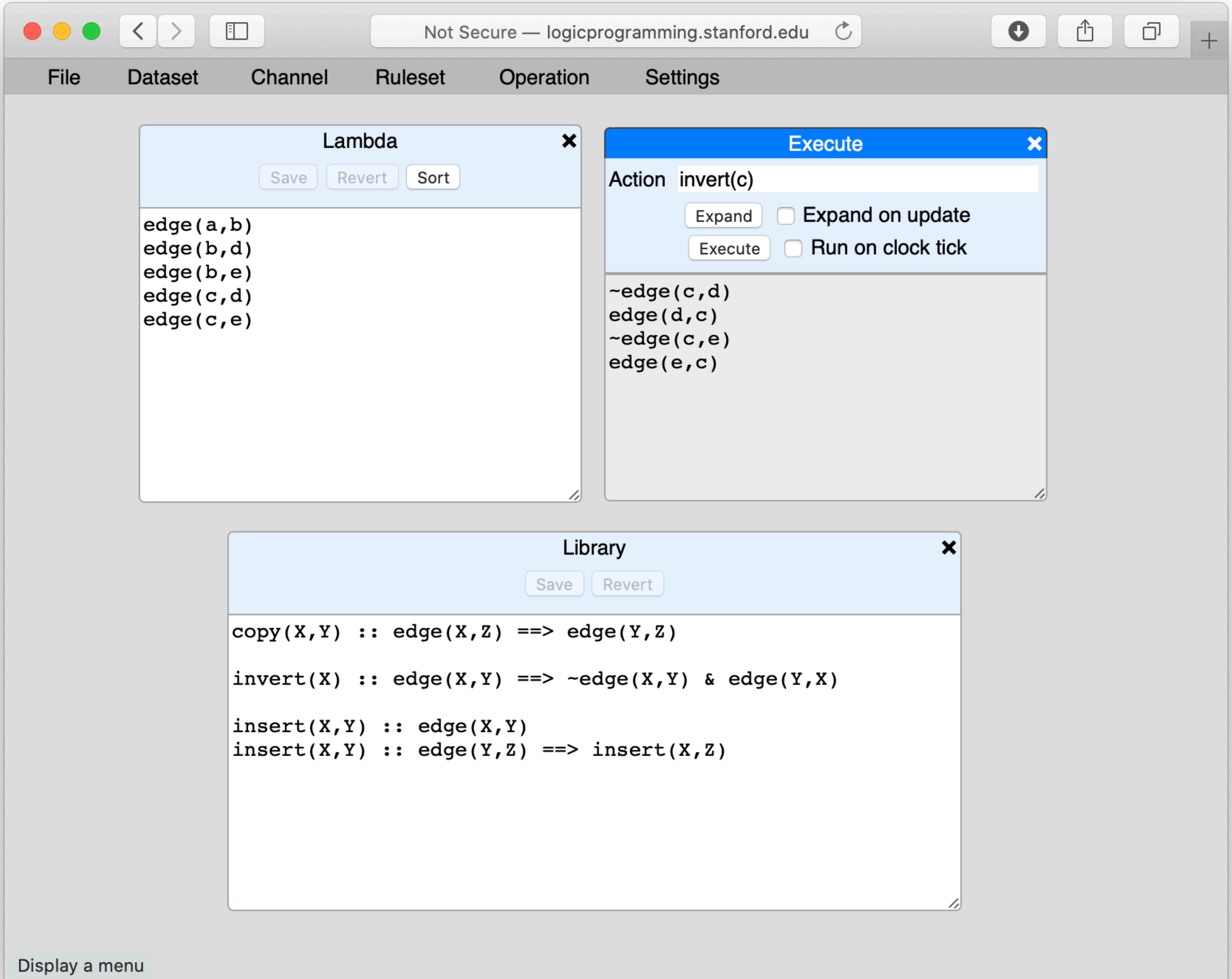
Positive Updates:

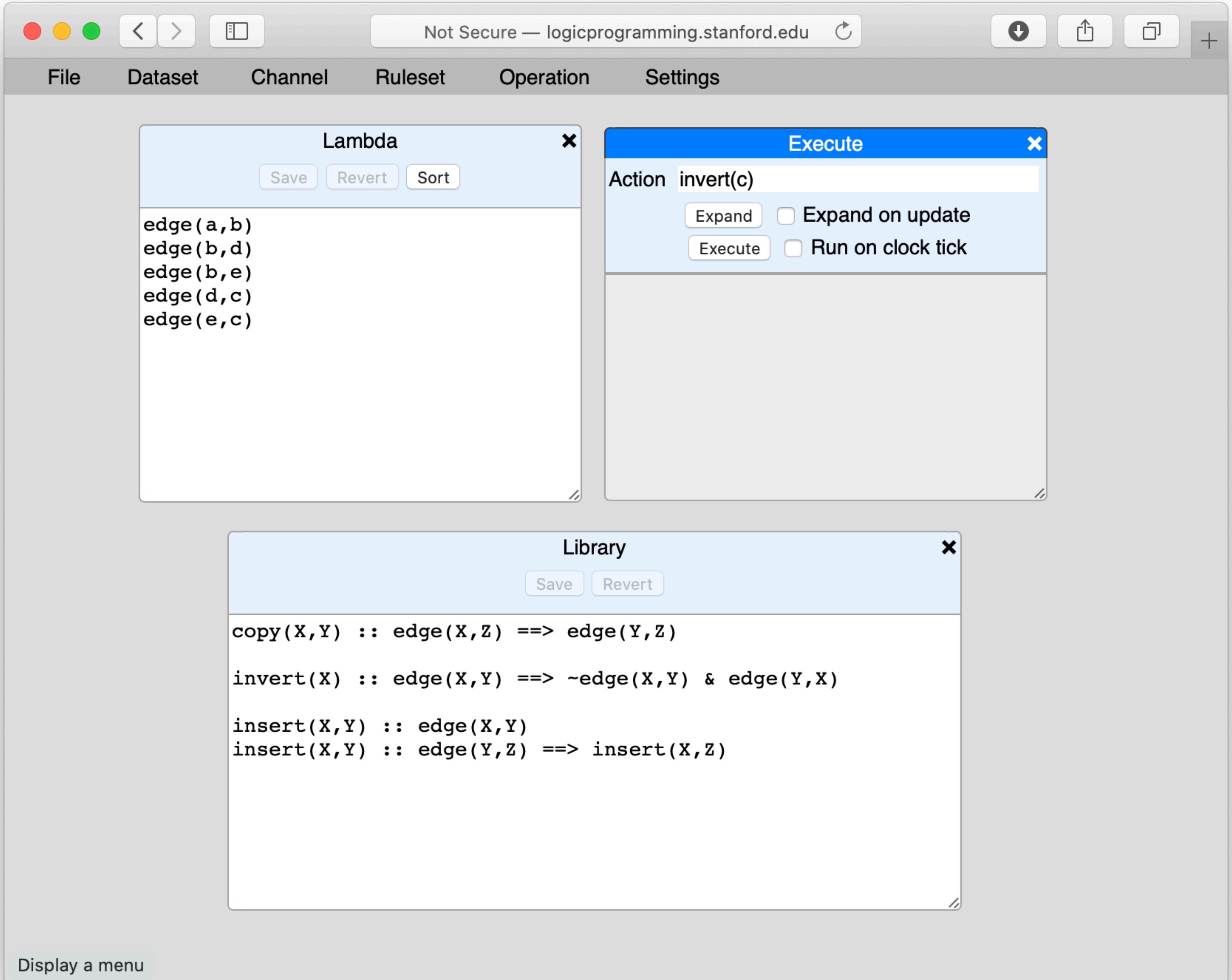
{edge(w,b), edge(w,d), edge(w,e), edge(w,c)}

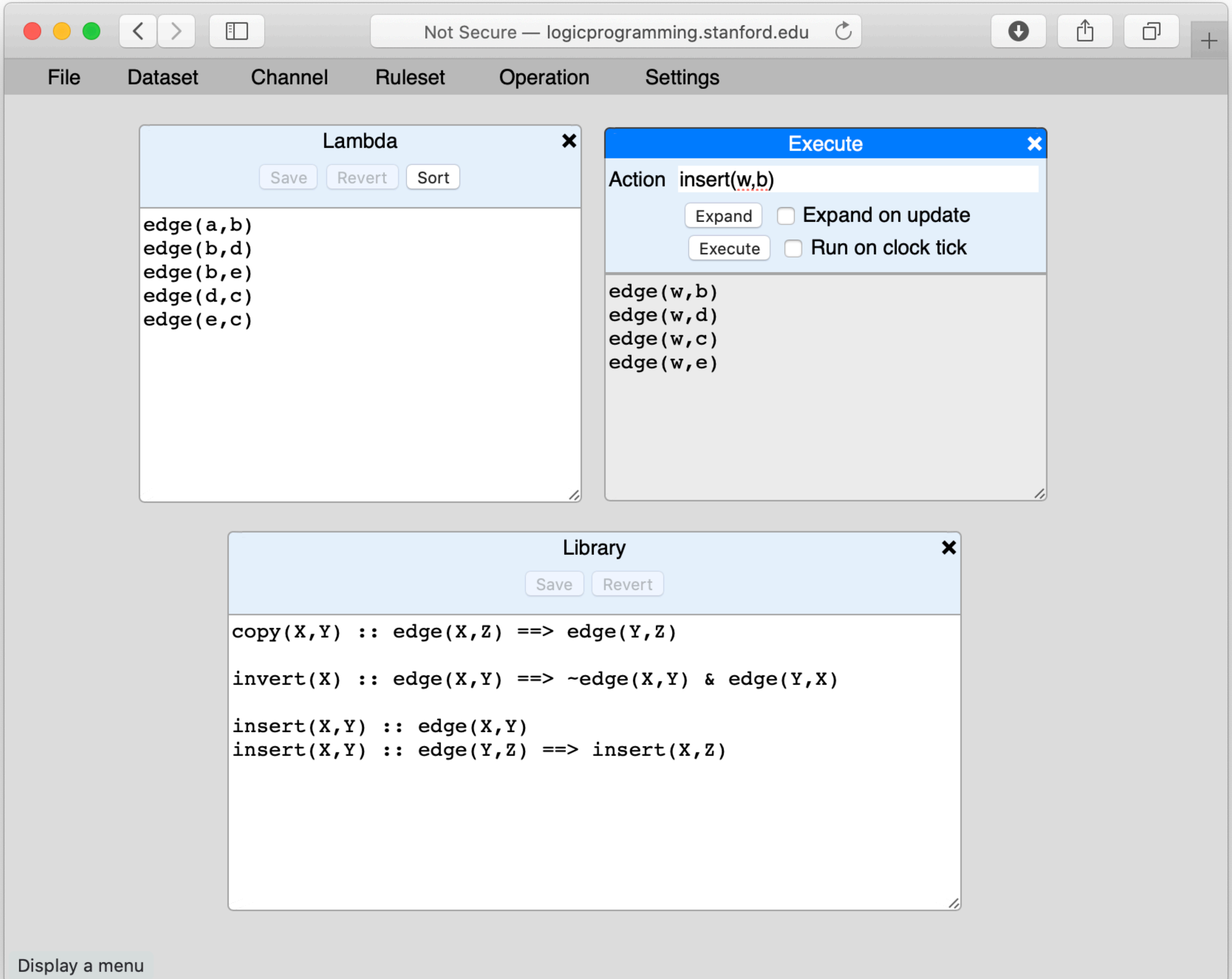

Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
Execute
Action
Expand
Expand on update
Execute
Run on clock tick
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)

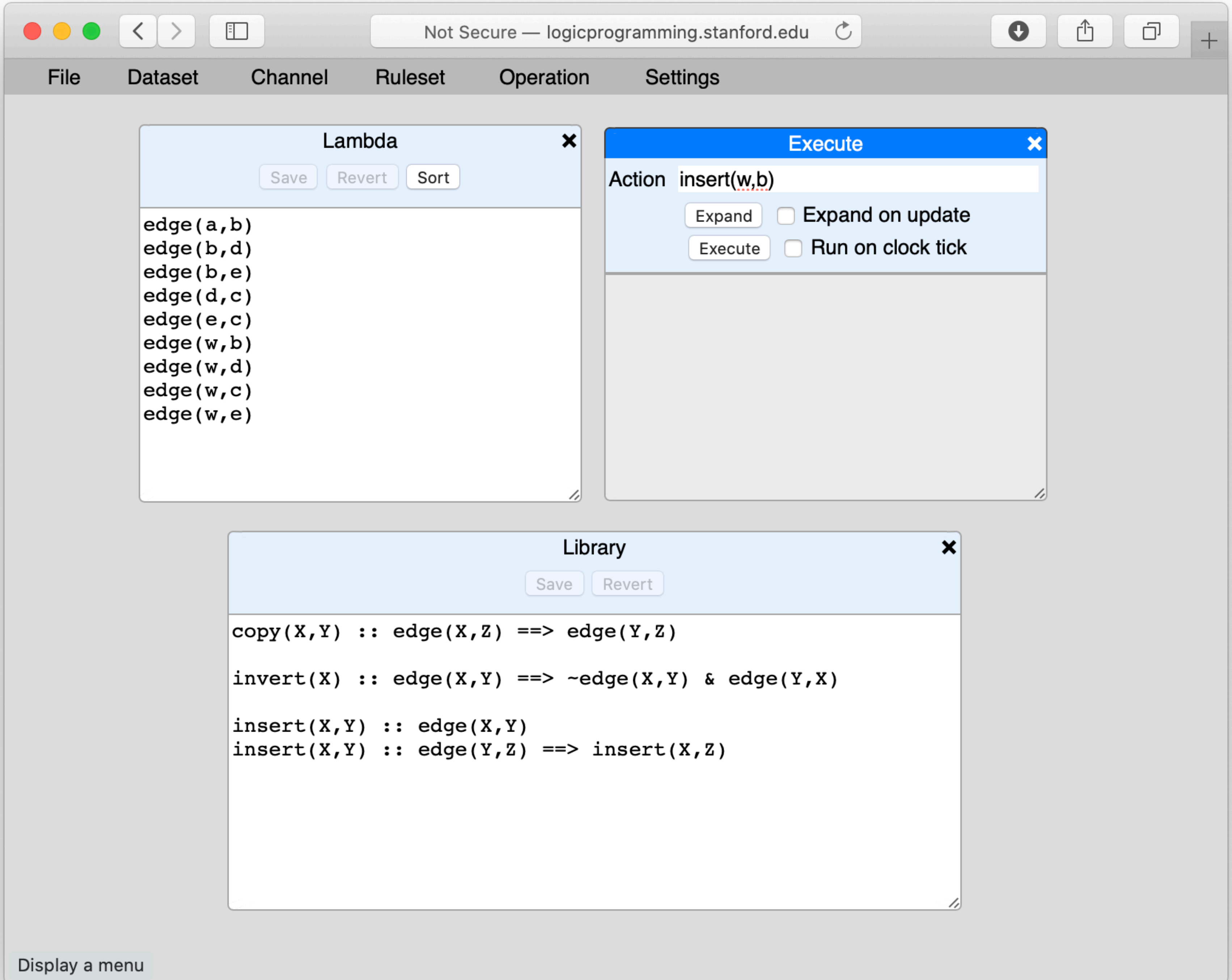

Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
Execute
Action copy(b,c)
Expand
Expand on update
Execute
Run on clock tick
edge(c,d)
edge(c,e)
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
Display a menu


Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
edge(c,d)
edge(c,e)
Execute
Action
copy(b,c)
Expand
Expand on update
Execute
Run on clock tick
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
Display a menu


Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
edge(c,d)
edge(c,e)
Execute
Action invert(c)
Expand
Expand on update
Execute
Run on clock tick
~edge(c,d)
edge(d,c)
~edge(c,e)
edge(e,c)
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
Display a menu


Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
edge(d,c)
edge(e,c)
Execute
Action
invert(c)
Expand
Expand on update
Execute
Run on clock tick
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
Display a menu


Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
edge(d,c)
edge(e,c)
Execute
Action
insert(w,b)
Expand
Expand on update
Execute
Run on clock tick
edge(w,b)
edge(w,d)
edge(w,c)
edge(w,e)
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
Display a menu



Not Secure — logicprogramming.stanford.edu
File
Dataset
Channel
Ruleset
Operation
Settings
Lambda
Save
Revert
Sort
edge(a,b)
edge(b,d)
edge(b,e)
edge(d,c)
edge(e,c)
edge(w,b)
edge(w,d)
edge(w,c)
edge(w,e)
Execute
Action insert(w,b)
Expand
Expand on update
Execute
Run on clock tick
Library
Save
Revert
copy(X,Y) :: edge(X,Z) ==> edge(Y,Z)
invert(X) :: edge(X,Y) ==> ~edge(X,Y) & edge(Y,X)
insert(X,Y) :: edge(X,Y)
insert(X,Y) :: edge(Y,Z) ==> insert(X,Z)
Display a menu

View Materialization

Materialized Views

A **materialized view** is a view relation that is stored explicitly in the database.

Ruleset:

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

Base Data:

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

Materialized View:

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

Analysis

Ruleset

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

Computation Cost for s:

`r` computed multiple times

for n=2, unifications = 1242

for n=3, unifications = 41636

where n is the number of objects

Storage for p:

O(n^2) facts stored

Example with s Materialized

Ruleset

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

Computation Cost for s

for n=2, unifications = 8

for n=3, unifications = 27

Storage for s

O(n^3) in worst case

Example with r Materialized

Ruleset

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

Computation Cost for s

for n=2, unifications = 15
for n=3, unifications = 40
where n is the number of objects

Computation Cost for r

for n=2, unifications = 17
for n=3, unifications = 55

Storage for r

O(n) in worst case

Materialization

Naive Approach:

On receipt of update,
discard materialized data and
rematerialize.

Differential Approach:

Write operation definitions for the materialized view.
Remove rules defining the materialized view.
On receipt of update, run newly defined operations.

Example

Old View Definitions

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

Operation Definitions

```
+p(X,Y) :: p(X,Y)
+p(X,Y) :: p(Y,Z) ==> r(X)
+p(Y,Z) :: p(X,Y) ==> r(X)
```

Exercise for viewer: What are the deletion rules?

New View Definition (remove definition of `r`)

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

Execution

Old Dataset: `{p(a,b),p(b,a),r(a),r(b)}`

Rules:

```
s(X,Y,Z) :- r(X) & r(Y) & r(Z)
+p(X,Y) :: p(X,Y)
+p(X,Y) :: p(Y,Z) ==> r(X)
+p(Y,Z) :: p(X,Y) ==> r(X)
```

Change Request: `+p(d,c)`
Update: `{p(d,c)}`

New Dataset:

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

Execution

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

Rules:

```
s(X,Y,Z) :- r(X) & r(Y) & r(Z)
+p(X,Y) :: p(X,Y)
+p(X,Y) :: p(Y,Z) ==> r(X)
+p(Y,Z) :: p(X,Y) ==> r(X)
```

Change Request: `+p(c,d)`
Update: `{p(c,d),r(c),r(d)}`

New Dataset:

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

Update Through Views

View Definition

```
r(X) :- p(X)
r(X) :- q(X)
```

Request: `+r(bob)`

Positive Update:

```
{p(bob)}?
{q(bob)}?
{p(bob),q(bob)}?
{}?
```

What if p is dead and q is alive?

Update Policies

View Definition

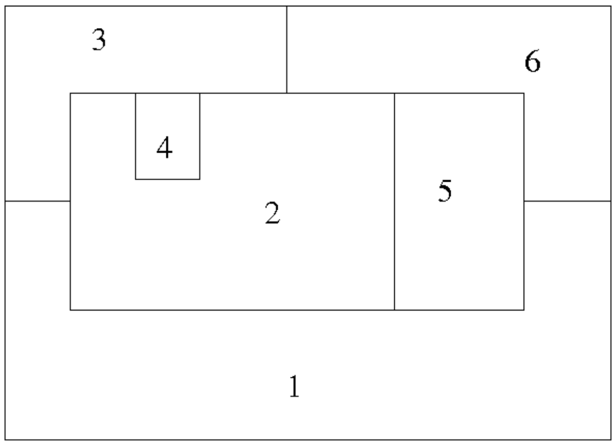
```
r(X) :- p(X)
r(X) :- q(X)
```

Update Rule

```
+r(X) :: ~q(X) ==> p(X)
```

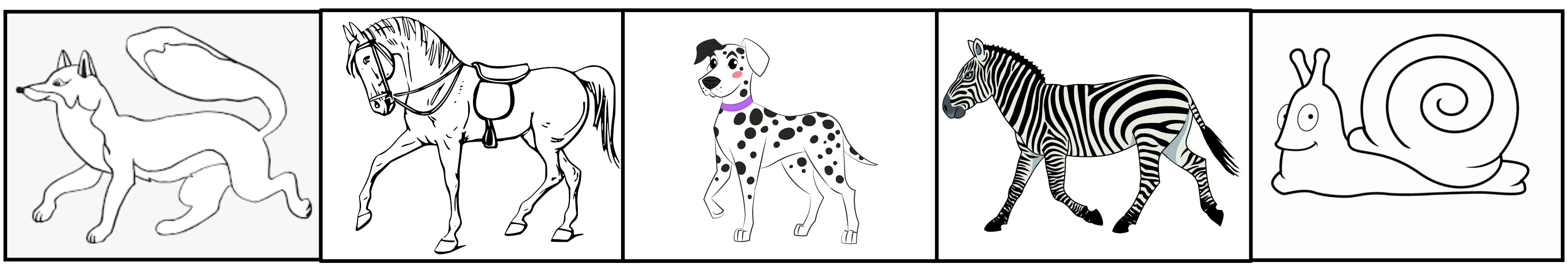
Constraint Propagation

Constraint Satisfaction Problems



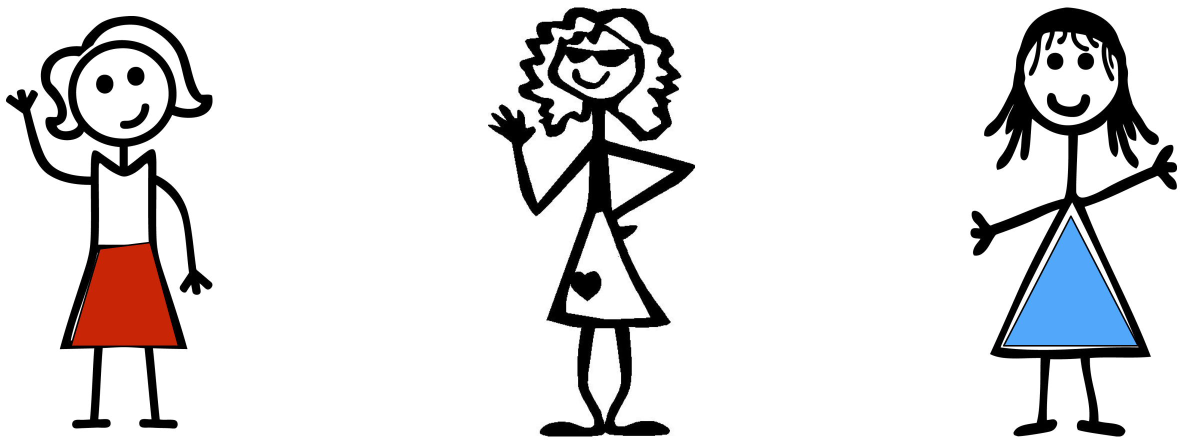

```
  SEND
+MORE
-----
MONEY
```

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	



Problem Statement

Ruby Red, Willa White, Betty Blue are having lunch.
One is wearing a red skirt, one white, one blue.

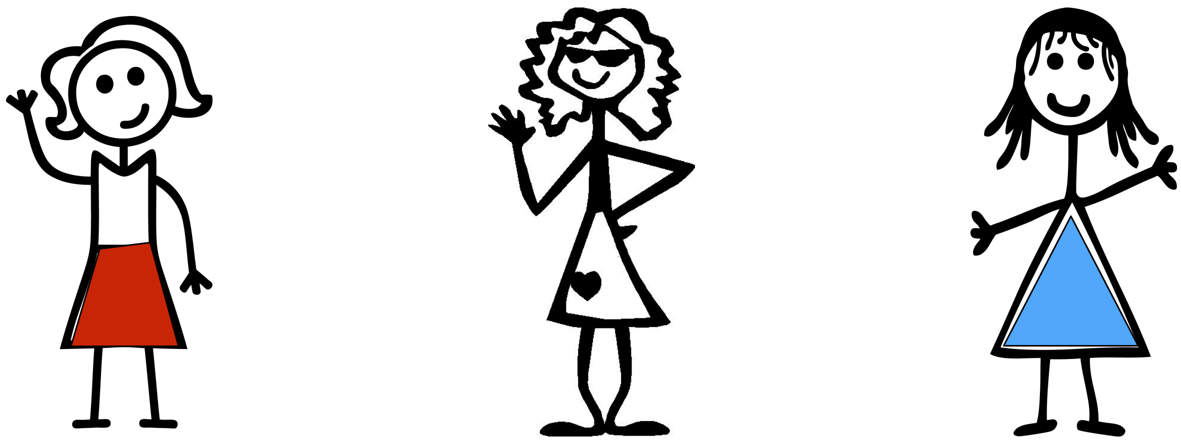


Betty to woman in white skirt: *Have you noticed that our skirts have colors different from our names?*

Which woman is wearing which skirt?

Problem Statement

Ruby Red, Willa White, Betty Blue are having lunch.
One is wearing a red skirt, one white, one blue.



Betty to woman in white skirt: *Have you noticed that our skirts have colors different from our names?*

Why is this problem so easy to solve?

Vocabulary

Symbols: `red`, `white`, `blue`, `v`, `x`

Unary predicates Predicates:

`color(C)` is true if C is a color.

Ternary Predicates:

`c(P,C,v)` is true if person P is wearing color C.

`c(P,C,x)` is true if person P is not wearing color C.

Operations:

`c1` - No person is wearing same color as name.

`c2` - Betty Blue spoke to person in white.

`c3` - Every person has one color and vice versa.

Colors

```
color(red)
color(white)
color(blue)
```

First Given

No person is wearing same color as name.

```
c1 :: color(C) ==> c(C,C,x)
```

Second Given

Betty Blue spoke to person in white.

```
c2 :: c(blue,white,x)
```

Existence Constraints

Every person is wearing some color.

```
c3 ::
  c(P,C1,x) & c(P,C2,x) & color(C3) & mutex(C1,C2,C3)
  ==> c(P,C3,v)
```

Every color is worn.

```
c3 ::
  c(P1,C,x) & c(P2,C,x) & color(P3) & mutex(P1,P2,P3)
  ==> c(P3,C,v)
```

Uniqueness Constraints

Nobody is wearing two colors.

```
c3 ::
  c(P,C1,v) & color(C2) & distinct(C1,C2)
  ==> c(P,C2,x)
```

No color is worn by two people.

```
c3 ::
  c(P1,C,v) & color(P2) & distinct(P1,P2)
  ==> c(P2,C,x)
```

Initial State

Color

Person

	r	w	b
r			
w			
b			

First Given

No person is wearing same color as name.

```
c1 :: color(C) ==> c(C,C,x)
```

	r	w	b
r			
w			
b			

	r	w	b
r	X		
w		X	
b			X

Second Given

Betty Blue spoke to person in white.

```
c2 :: c(blue,white,x)
```

	r	w	b
r	X		
w		X	
b			X

	r	w	b
r	X		
w		X	
b		X	X

Existence Constraints

```
c3 ::
   c(P,C1,x) & c(P,C2,x) & color(C3) & mutex(C1,C2,C3)
   ==> c(P,C3,v)

c3 ::
   c(P1,C,x) & c(P2,C,x) & human(P3) & mutex(P1,P2,P3)
   ==> c(P3,C,v)
```

	r	w	b
r	X		
w		X	
b		X	X

	r	w	b
r	X	V	
w		X	
b	V	X	X

Uniqueness Constraints

```
c3 ::
   c(P,C1,v) & color(C2) & distinct(C1,C2)
   ==> c(P,C2,x)

c3 ::
   c(P1,C,v) & person(P2) & distinct(P1,P2)
   ==> c(P2,C,x)
```

	r	w	b
r	X	V	
w		X	
b	V	X	X

→

	r	w	b
r	X	V	X
w	X	X	
b	V	X	X

Existence Constraints

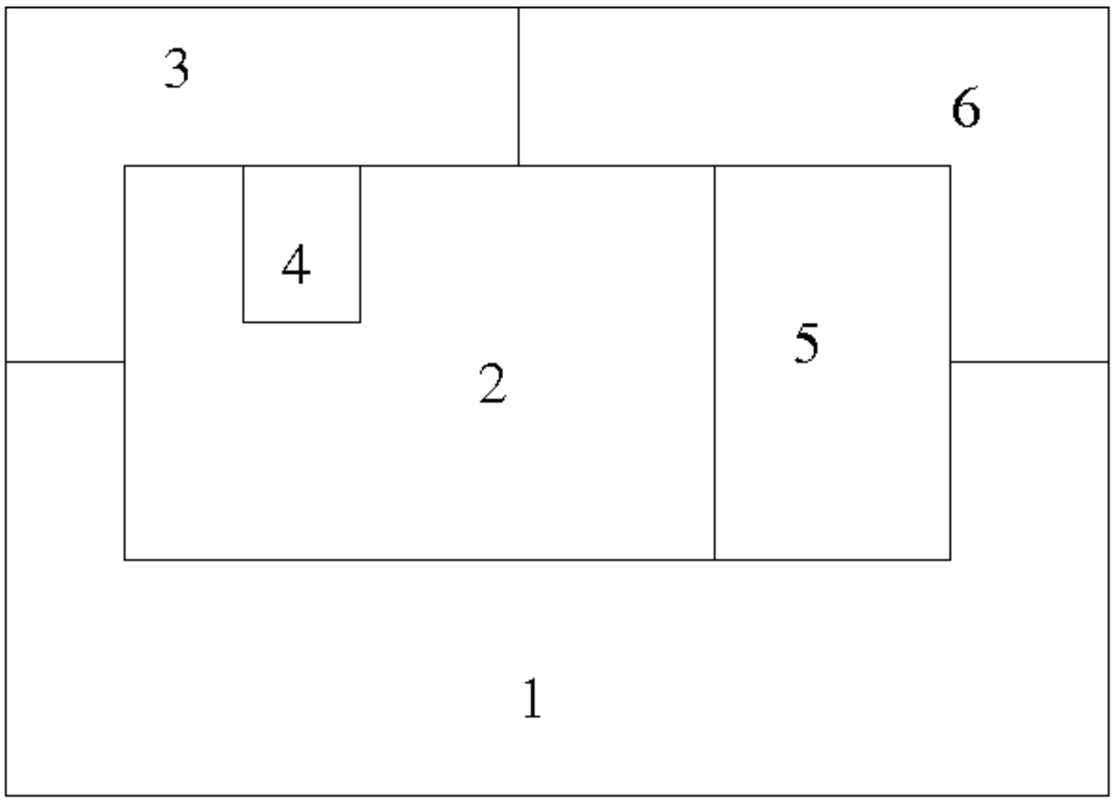
```
c3 ::
   c(P,C1,x) & c(P,C2,x) & color(C3) & mutex(C1,C2,C3)
   ==> c(P,C3,v)

c3 ::
   c(P1,C,x) & c(P2,C,x) & human(P3) & mutex(P1,P2,P3)
   ==> c(P3,C,v)
```

	r	w	b
r	X	V	X
w	X	X	
b	V	X	X

$\longrightarrow$

	r	w	b
r	X	V	X
w	X	X	V
b	V	X	X

Map Coloring



Sudoku

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7