Logic Programming
Model Management

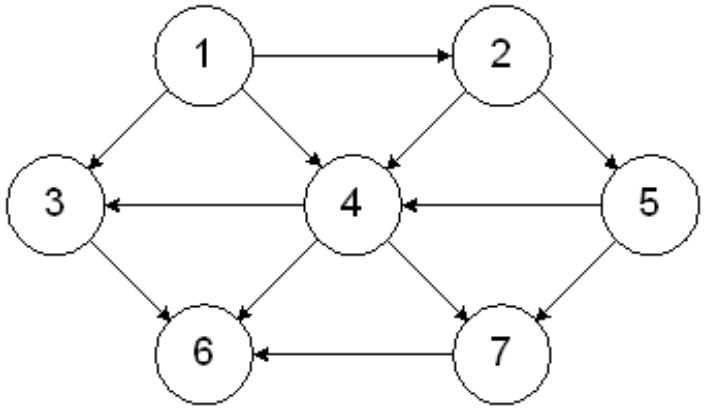
Michael Genesereth
Computer Science Department
Stanford University

Programme

Tables vs 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

Graphs



Constraint Satisfaction

University

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

Graphs

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 incoming arcs to `X`.

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

Example

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)`}

Example

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)`}

Example

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

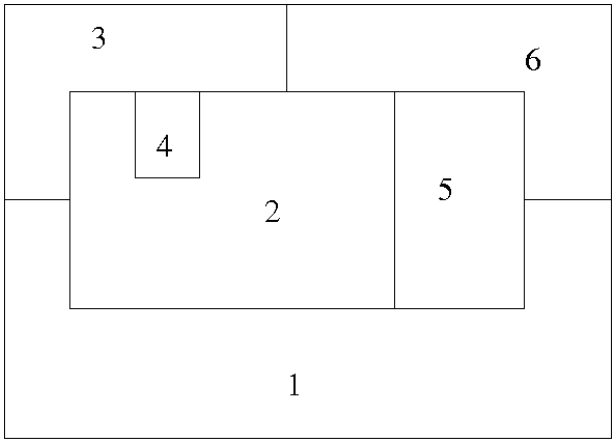
Save Revert

```
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

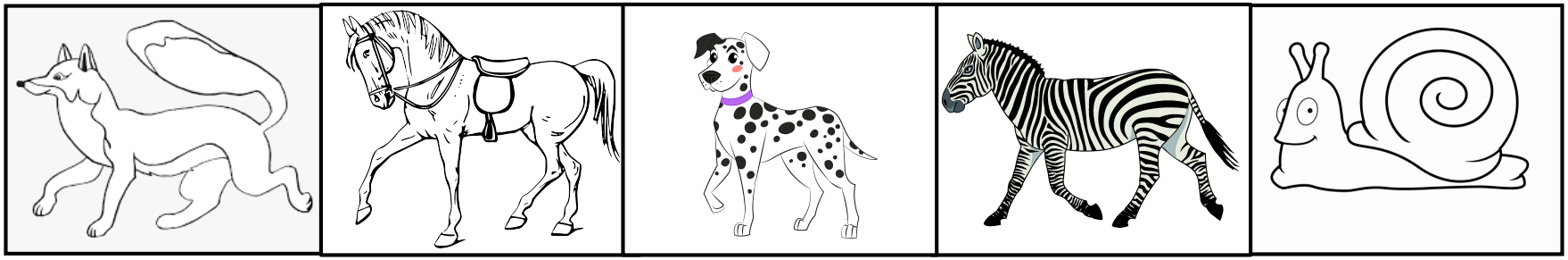
Constraint Propagation

Constraint Satisfaction Problems



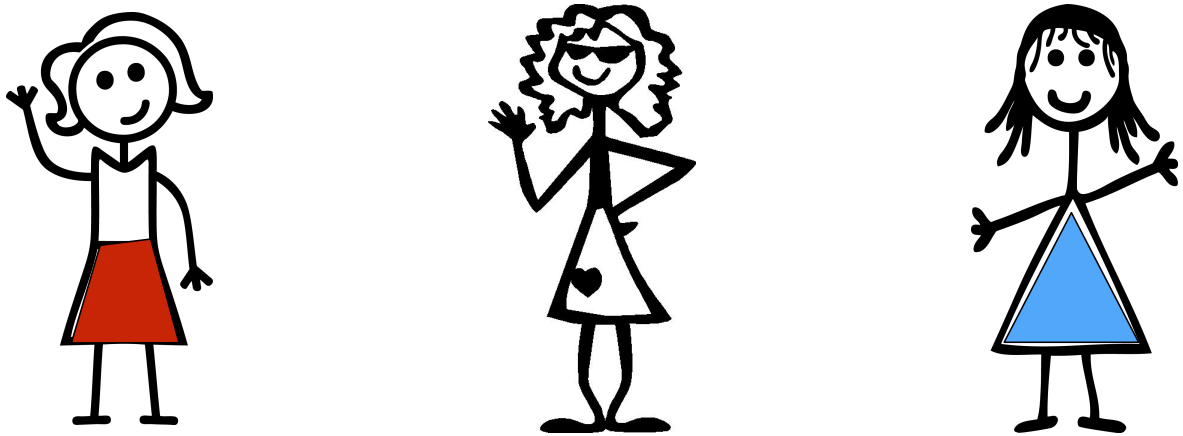

```
  SEND
+MORE
-----
MONEY
```

	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.

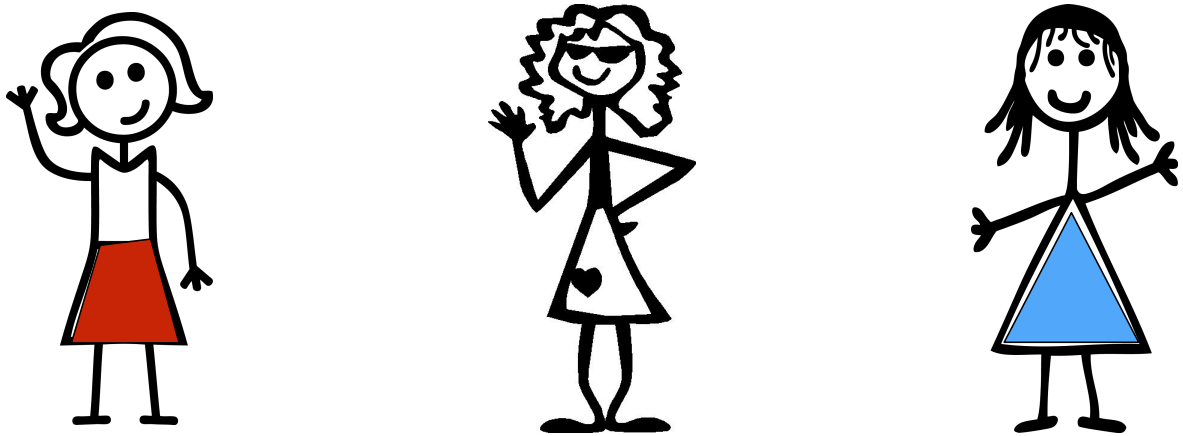


Betty to woman in white skirt: *Have you noticed that our skirts have colors different from our names?*

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:

`human(P)` is true if P is a person.

`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.

People and Colors

```
human(red)
human(white)
human(blue)

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.

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

Every color is worn.

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

Uniqueness Constraints

Nobody is wearing two colors.

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

No color is worn by two people.

```
update ::
  c(P1,C,v) & person(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

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

update ::
  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

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

update ::
  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

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

update ::
  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

→

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

Small "Zebra" Problem

Background Data

```
house(h1)
house(h2)
house(h3)

hue(red)
hue(white)
hue(blue)

nat(italian)
nat(spanish)
nat(norwegian)
```

Uniqueness and Existence

```
cb :: res(H,N1,yes) & nat(N2) & distinct(N1,N2)
==> res(H,N2,no)

cb :: res(H1,N,yes) & house(H2) &
distinct(H1,H2) ==> res(H2,N,no)

cb :: res(H1,N,no) & res(H2,N,no) & house(H3) &
mutex(H1,H2,H3) ==> res(H3,N,yes)


cb :: color(H,C1,yes) & hue(C2) &
distinct(C1,C2) ==> color(H,C2,no)

cb :: color(H1,C,yes) & house(H2) &
distinct(H1,H2) ==> color(H2,C,no)

cb :: color(H1,C,no) & color(H2,C,no) &
house(H3) & mutex(H1,H2,H3) ==> color(H3,C,yes)
```

Constraints

```
c1 :: res(h2,italian,yes)

c2 :: res(h1,spanish,no)
c3 :: res(H,spanish,yes) ==> color(H,red,no)

c4 :: res(H,norwegian,yes) ==> color(H,blue,yes)
```

Not Secure — epilog.stanford.edu

File Dataset Channel Ruleset Operation Settings

Lambda

Save Revert Sort

Execute

Action cb

Expand

Start Step

Execute

Action

Expand

Start Step

Library

Save Revert

```
house(h1)
house(h2)
house(h3)

hue(red)
hue(white)
hue(blue)

nat(italian)
nat(spanish)
nat(norwegian)

cb :: res(H,N1,yes) & nat(N2) & distinct(N1,N2) ==> res(H,N2,no)
cb :: res(H1,N,yes) & house(H2) & distinct(H1,H2) ==> res(H2,N,no)
cb :: res(H1,N,no) & res(H2,N,no) & house(H3) & mutex(H1,H2,H3) ==> res(H3,N,yes)

cb :: color(H,C1,yes) & hue(C2) & distinct(C1,C2) ==> color(H,C2,no)
cb :: color(H1,C,yes) & house(H2) & distinct(H1,H2) ==> color(H2,C,no)
cb :: color(H1,C,no) & color(H2,C,no) & house(H3) & mutex(H1,H2,H3) ==> color(H3,C,yes)

c1 :: res(h2,italian,yes)
c2 :: res(h1,spanish,no)
c3 :: res(H,spanish,yes) ==> color(H,red,no)
c4 :: res(H,norwegian,yes) ==> color(H,blue,yes)
```

Not Secure — epilog.stanford.edu

File Dataset Channel Ruleset Operation Settings

Lambda

Save Revert Sort

```
res(h2,italian,yes)
res(h2,spanish,no)
res(h2,norwegian,no)
res(h1,italian,no)
res(h3,italian,no)
res(h1,spanish,no)
res(h3,spanish,yes)
res(h3,norwegian,no)
res(h1,norwegian,yes)
color(h1,blue,yes)
color(h1,red,no)
color(h1,white,no)
color(h2,blue,no)
color(h3,blue,no)
color(h3,red,no)
color(h2,red,yes)
color(h2,white,no)
color(h3,white,yes)
```

Execute

Action cb

Expand

Start Step

Execute

Action c3

Expand

Start Step

Library

Save Revert

```
house(h1)
house(h2)
house(h3)

hue(red)
hue(white)
hue(blue)

nat(italian)
nat(spanish)
nat(norwegian)

cb :: res(H,N1,yes) & nat(N2) & distinct(N1,N2) ==> res(H,N2,no)
cb :: res(H1,N,yes) & house(H2) & distinct(H1,H2) ==> res(H2,N,no)
cb :: res(H1,N,no) & res(H2,N,no) & house(H3) & mutex(H1,H2,H3) ==> res(H3,N,yes)

cb :: color(H,C1,yes) & hue(C2) & distinct(C1,C2) ==> color(H,C2,no)
cb :: color(H1,C,yes) & house(H2) & distinct(H1,H2) ==> color(H2,C,no)
cb :: color(H1,C,no) & color(H2,C,no) & house(H3) & mutex(H1,H2,H3) ==> color(H3,C,yes)

c1 :: res(h2,italian,yes)
c2 :: res(h1,spanish,no)
c3 :: res(H,spanish,yes) ==> color(H,red,no)
c4 :: res(H,norwegian,yes) ==> color(H,blue,yes)
```