Logic Programming
Queries, Values, Transforms

Michael Genesereth
Computer Science Department
Stanford University

Relational 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?*

Functional Queries

Arithmetic:

e.g. 2 + 2?

e.g. *Concatenation of "abc" and "def"?*

Aggregates:

e.g. *Number of children of Art?*

e.g. *Number of people with daughters?*

Compo:

e.g. 2 x *number of children of Art?*

Transitions

Additions

e.g. *Add the fact that Art is the parent of Bill*.

Removals

e.g. *Drop the fact that Art is the parent of Bill*.

Conditional Changes:

e.g. *Add grandparent facts implied by parent facts*.

e.g. *Replace parent facts with corresponding child facts*.

Query Syntax

Vocabulary

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

```
a, b, c, 1, 2, 3, joe, cs151
f, g, pair, triple
p, q, r, person, parent, prefers
```

Same as before

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)
```

Atoms are like factoids in datasets
except that
they may contain variables.

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*.

Ground Query

$$\underbrace{\texttt{goal(a,b)}}_{\textit{head}}\ \texttt{:-}\ \underbrace{\overbrace{\texttt{p(a,b)}}^{\textit{subgoal}}\ \texttt{\&}\ \overbrace{\texttt{\textasciitilde q(b)}}^{\textit{subgoal}}}_{\textit{body}}$$

Intuitive meaning: `goal(a,b)` *is true if* `p(a,b)` *is true and* `q(b)` *is false*.

Queries May Contain Variables

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

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

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

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 of the instance* is in the dataset,

(3) no *negative subgoal of the instance* is in the dataset.

Example

Dataset

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

Query

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

Result

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

Instances

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

Example

Dataset

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

Query

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

Result

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

Instances

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

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)
```

Instances

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

Instances

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

Instances

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

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))
```

Instances

```
  goal(a,f(a)) :- p(a,a) & p(a,a)
  goal(a,f(a)) :- p(a,b) & p(b,a)
  goal(a,f(a)) :- p(a,c) & p(c,a)
  goal(a,f(a)) :- p(a,d) & p(d,a)
  goal(b,f(b)) :- p(b,a) & p(a,b)
  goal(b,f(b)) :- p(b,b) & p(b,b)
  goal(b,f(b)) :- p(b,c) & p(c,b)
  goal(b,f(b)) :- p(b,d) & p(d,b)
  goal(c,f(c)) :- p(c,a) & p(a,c)
  goal(c,f(c)) :- p(c,b) & p(b,c)
  goal(c,f(c)) :- p(c,c) & p(c,c)
→ goal(c,f(c)) :- p(c,d) & p(d,c)
  goal(d,f(d)) :- p(d,a) & p(a,d)
  goal(d,f(d)) :- p(d,b) & p(b,d)
→ goal(d,f(d)) :- p(d,c) & p(c,d)
  goal(d,f(d)) :- p(d,d) & p(d,d)
```

Non-Examples

Dataset

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

Query

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

***Not* Results**

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

Too few.

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

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)}
```

Queries

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

Result

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

NB: A query set is effectively a disjunction.

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

Herbrand Universe {a, b}

Dataset {p(a,a)}

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

Unbound Variables in Head

Rule

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

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

Dataset {`p(a,a)`}

Instances / **Results**

```
  goal(a,a) :- p(a,a)          goal(a,a)
  goal(a,b) :- p(a,a)          goal(a,b)
  goal(a,f(a)) :- p(a,a)       goal(a,f(a))
  goal(a,f(b)) :- p(a,a)       goal(a,f(b))
  goal(a,f(f(a))) :- p(a,a)    goal(a,f(f(a)))
           ...                        ...
```

Unbound Variables in Negation

Query

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

Herbrand Universe {a, b, c}

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

What is the result?

Possible Meanings

Query

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

Possible Meanings

Find all `X` such that `p(X,Y)` is true and there is *no* `Z` for which `p(Y,Z)` is *true*.

Find all `X` such that `p(X,Y)` is true and there is *some* `Z` for which `p(Y,Z)` is *false*.

Possible Meanings

Query

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

Herbrand Universe `{a, b, c}`

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

Results

Find all `X` such that `p(X,Y)` is true and there is *no* `Z` for which `p(Y,Z)` is *true*.

`{goal(b)}`

Find all `X` such that `p(X,Y)` is true and there is *some* `Z` for which `p(Y,Z)` is *false*.

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

Unbound Variables in Negation

Unsafe Rule

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

Herbrand Universe {a, b, c}

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

Instances **Results**

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

Predefined Concepts

Predefined Relations

Evaluable Relations

`less`, `leq`, `symleq`, `symless`

`same`, `distinct`, `mutex`

`exists`

`evaluate` *discussed in next lesson*

`less, leq, symless, symleq`

Comparison

`less(t1,t2)` is true iff t1 < t2

`leq(t1,t2)` is true iff t1 t1 <= t2

`symless(t1,t2)` is true iff t1 lexically less than t2

`symleq(t1,t2)` is true iff t1 is lexically less than t2

Examples

`less(2,3)`	is true
`leq(2,3)`	is true
`leq(2,2)`	is true

`symless("abc","def")`	is true
`symleq("abc","def")`	is true
`symleq("abc","abc")`	is true

Safetv: No unbound variables allowed!!!

Examples

`less(2,3)`	is true
`leq(2,3)`	is true
`less(2,2)`	is false
`leq(2,2)`	is true
`symless("abc","def")`	is true
`symleq("abc","def")`	is true
`symless("abc","abc")`	is false
`symleq("abc","abc")`	is true
`less("abc","def")`	is false (not numbers)
`symless(abc,def)`	is true
`less(10,2)`	is false
`symless(10,2)`	is true

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!!!

Existential Quantification

Identity

`exists(t,p)` is true iff
there is some binding for the variables in `t`
for which `p` is true.

Data

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

Examples

`p(X) & ~q(X,Y)`	is unsafe
`p(X) & ~exists(Y,q(X,Y))`	is true for `X=a`

Safety: No unbound variables allowed in the second argument except those in the first argument.

Kinship

Dataset

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

$$\begin{tikzcd}
 & & art & & \\
 & bob \arrow[ur] & & bea \arrow[ul] & \\
cal \arrow[ur] & cam \arrow[u] & & cat \arrow[u] & coe \arrow[ul]
\end{tikzcd}$$

Grandparents

Query

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

Dataset:

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

Result:

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

People

Query:

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

Dataset:

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

Result:

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

Siblings

Query

```
???
```

Dataset:

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

Result:

```
goal(bob,bea)
goal(bea,bob)
goal(cal,cam)
goal(cam,cal)
goal(cat,coe)
goal(coe,cat)
```

Siblings

Query

```
goal(Y,Z):-parent(X,Y) & parent(X,Z) & distinct(Y,Z)
```

Dataset:

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

Result:

```
goal(bob,bea)
goal(bea,bob)
goal(cal,cam)
goal(cam,cal)
goal(cat,coe)
goal(coe,cat)
```

Children

Dataset:

```
parent(art,bob)
parent(art,bea)
parent(art,ben)
parent(bob,eli)
```

Query: find every person with at least one child

Children

Dataset:

```
parent(art,bob)
parent(art,bea)
parent(art,ben)
parent(bob,eli)
```

Query: find every person with at least one child

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

Children

Dataset:

```
parent(art,bob)
parent(art,bea)
parent(art,ben)
parent(bob,eli)
```

Query: find every person with at least two children

Children

Dataset:

```
  parent(art,bob)
  parent(art,bea)
  parent(art,ben)
  parent(bob,eli)
```

Query: find every person with at least two children

```
goal(X) :-
  parent(X,Y) &
  parent(X,Z) &
  distinct(Y,Z)
```

Children

Dataset:

```
parent(art,bob)
parent(art,bea)
parent(art,ben)
parent(bob,eli)
```

Query: find every person with exactly two children

Children

Dataset:

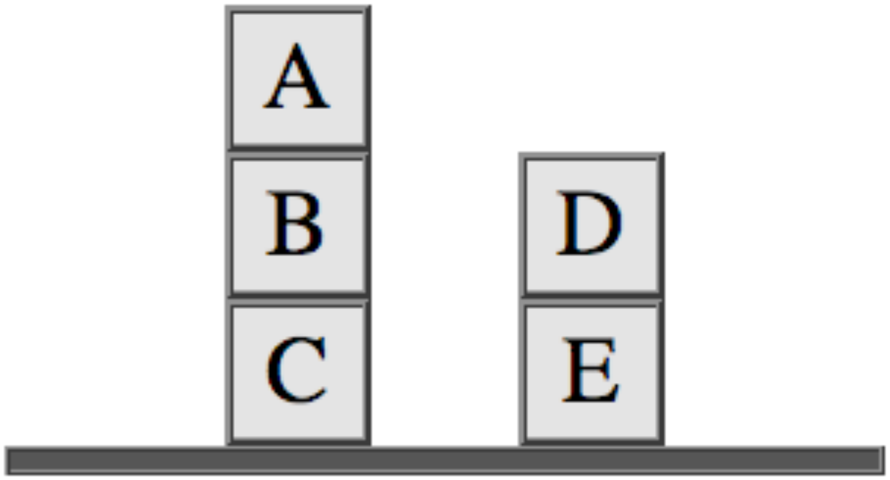
```
parent(art,bob)
parent(art,bea)
parent(art,ben)
parent(bob,eli)
```

Query: find every person with exactly two children

```
goal(X) :-
  parent(X,Y) &
  parent(X,Z) &
  distinct(Y,Z) &
  ~exists(W,parent(X,W) & mutex(W,Y,Z))
```

Blocks World

Blocks World



Vocabulary

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

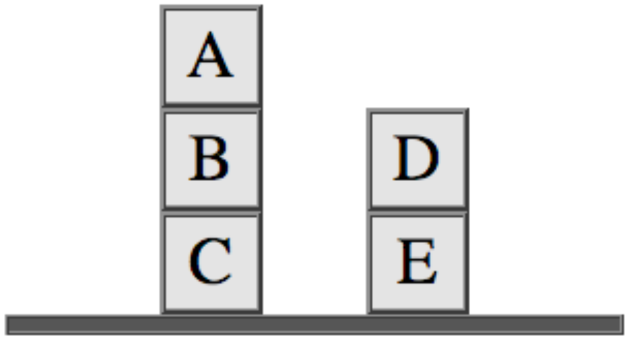
Unary Predicate:

- `block`

Binary Predicate:

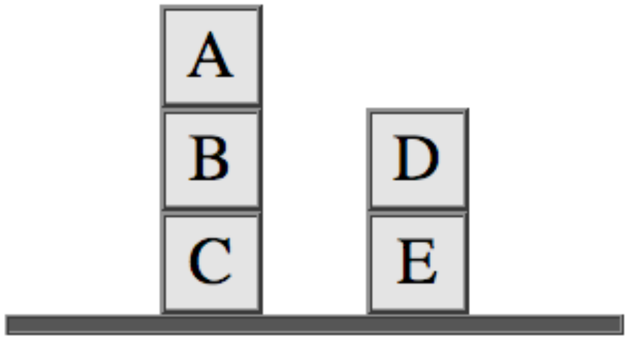
- `on` - pairs of blocks in which first is on the second
- `table` - blocks resting on the table
- `clear` - blocks with nothing on top

Data



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

Blocks World - cluttered

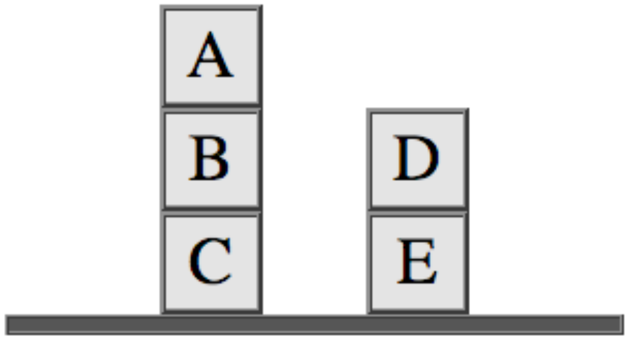


```
block(a)
block(b)        on(a,b)
block(c)        on(b,c)
block(d)        on(d,e)
block(e)
```

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

```
goal(b)
goal(c)
goal(e)
```

Blocks World - clear

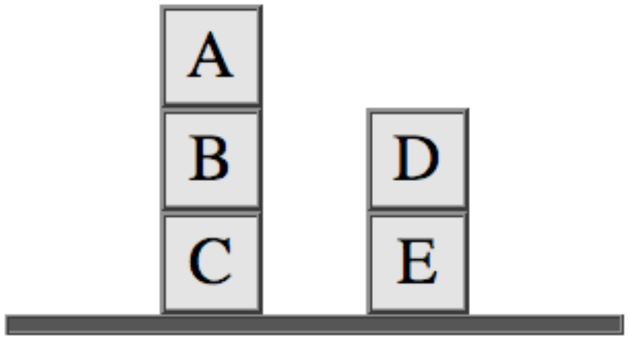


```
block(a)
block(b)        on(a,b)
block(c)        on(b,c)
block(d)        on(d,e)
block(e)
```

```
goal(Y) :-  block(Y) & notexist(X,on(X,Y))
```

```
goal(a)
goal(d)
```

Blocks World - supported

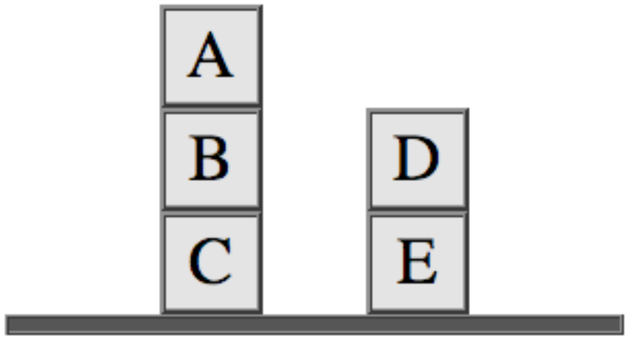


```
block(a)
block(b)        on(a,b)
block(c)        on(b,c)
block(d)        on(d,e)
block(e)
```

```
???
```

```
goal(a)
goal(b)
goal(d)
```

Blocks World - supported

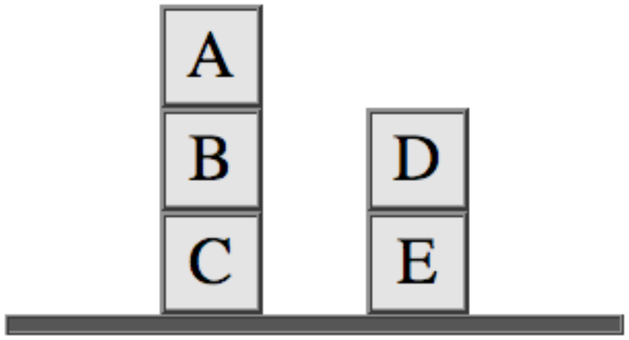


```
block(a)
block(b)        on(a,b)
block(c)        on(b,c)
block(d)        on(d,e)
block(e)
```

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

```
goal(a)
goal(b)
goal(d)
```

Blocks World - table

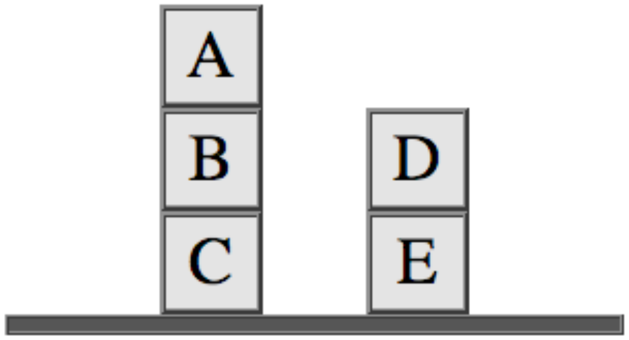


```
block(a)
block(b)     on(a,b)
block(c)     on(b,c)
block(d)     on(d,e)
block(e)
```

```
goal(X) :- block(X) & notexist(Y,on(X,Y))
```

```
goal(c)
goal(e)
```

Blocks World - stack

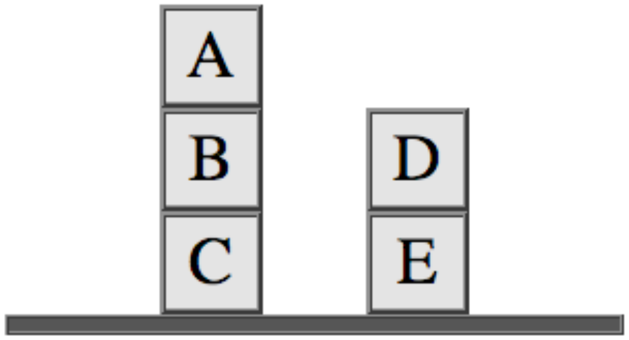


```
block(a)
block(b)        on(a,b)
block(c)        on(b,c)
block(d)        on(d,e)
block(e)
```

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

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

Blocks World - above



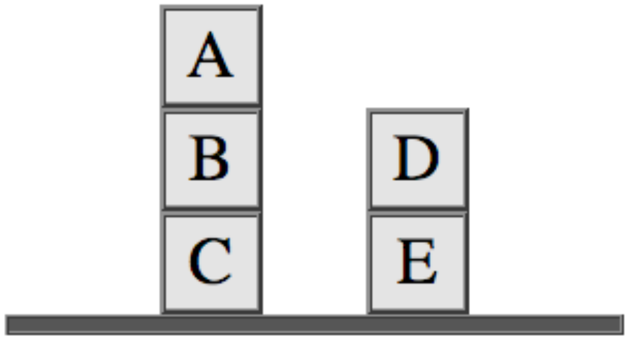
```
block(a)
block(b)      on(a,b)
block(c)      on(b,c)
block(d)      on(d,e)
block(e)
```

```
        goal(X,Y) :- on(X,Y)
    goal(X,Z) :- on(X,Y) & on(Y,Z)
goal(X,W) :- on(X,Y) & on(Y,Z) & on(Z,W)
                  ...

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

See next lecture to see how to do this in general.

Blocks World - above



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

```
goal(X,Y) :- on(X,Y)
goal(X,Z) :- on(X,Y) & on(Y,Z)
goal(X,W) :- on(X,Y) & on(Y,Z) & on(Z,W)
...

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

See unit on views to see how to do this using just two rules.

Food World

YUM!
tea
Eat
FOOD
café
bakery
coffee
kitchen
cucina

Vocabulary

Symbols:

`calamari`, `greek`, `caesar`, `green`
`puree`, `consomme`, `vichyssoise`
`beef`, `lamb`, `chicken`, `trout`
`baklava`, `icecream`, `shortcake`, `souffle`, `tiramisu`
`mon`, `tue`, `wed`, `thu`, `fri`

Constructors:

`three/3`, `four/4`, `five/5` - meals of different sizes

Predicates:

`food/1` - type predicate
`day/1` - type predicate
`menu/2` - day and meal

Menu

Dataset

```
menu(mon,three(calamari,beef,shortcake))
menu(mon,three(puree,beef,icecream))
menu(tue,three(puree,beef,icecream))
menu(tue,four(consomme,greek,lamb,baklava))
menu(wed,four(consomme,greek,lamb,baklava))
menu(thu,five(vichyssoise,caesar,trout,chicken,tiramisu))
menu(fri,five(vichyssoise,green,trout,beef,souffle))
```

Meals Served on Monday

Dataset

```
menu(mon,three(calamari,beef,shortcake))
menu(mon,three(puree,beef,icecream))
menu(tue,three(puree,beef,icecream))
menu(tue,four(consomme,greek,lamb,baklava))
menu(wed,four(consomme,greek,lamb,baklava))
menu(thu,five(vichyssoise,caesar,trout,chicken,tiramisu))
menu(fri,five(vichyssoise,green,trout,beef,souffle))
```

Query

```
  goal(M) :- menu(mon,M)
```

Result

```
  goal(three(calamari,beef,shortcake))
  goal(three(puree,beef,icecream))
```

Days with Three Course Meals

Dataset

```
menu(mon,three(calamari,beef,shortcake))
menu(mon,three(puree,beef,icecream))
menu(tue,three(puree,beef,icecream))
menu(tue,four(consomme,greek,lamb,baklava))
menu(wed,four(consomme,greek,lamb,baklava))
menu(thu,five(vichyssoise,caesar,trout,chicken,tiramisu))
menu(fri,five(vichyssoise,green,trout,beef,souffle))
```

Query

```
  goal(D) :- menu(D,three(X,Y,Z))
```

Result

```
  goal(mon)
  goal(tue)
```

Dietary Versions of Five Course Meals

Dataset

```
menu(mon,three(calamari,beef,shortcake))
menu(mon,three(puree,beef,icecream))
menu(tue,three(puree,beef,icecream))
menu(tue,four(consomme,greek,lamb,baklava))
menu(wed,four(consomme,greek,lamb,baklava))
menu(thu,five(vichyssoise,caesar,trout,chicken,tiramisu))
menu(fri,five(vichyssoise,green,trout,beef,souffle))
```

Query

```
  goal(three(V,Y,Z)) :- menu(D,five(U,V,X,Y,Z))
```

Result

```
  goal(three(caesar,chicken,tiramisu))
  goal(three(green,beef,souffle))
```

Days When Beef Is Served

Dataset

```
menu(mon,three(calamari,beef,shortcake))
menu(mon,three(puree,beef,icecream))
menu(tue,three(puree,beef,icecream))
menu(tue,four(consomme,greek,lamb,baklava))
menu(wed,four(consomme,greek,lamb,baklava))
menu(thu,five(vichyssoise,caesar,trout,chicken,tiramisu))
menu(fri,five(vichyssoise,green,trout,beef,souffle))
```

Query

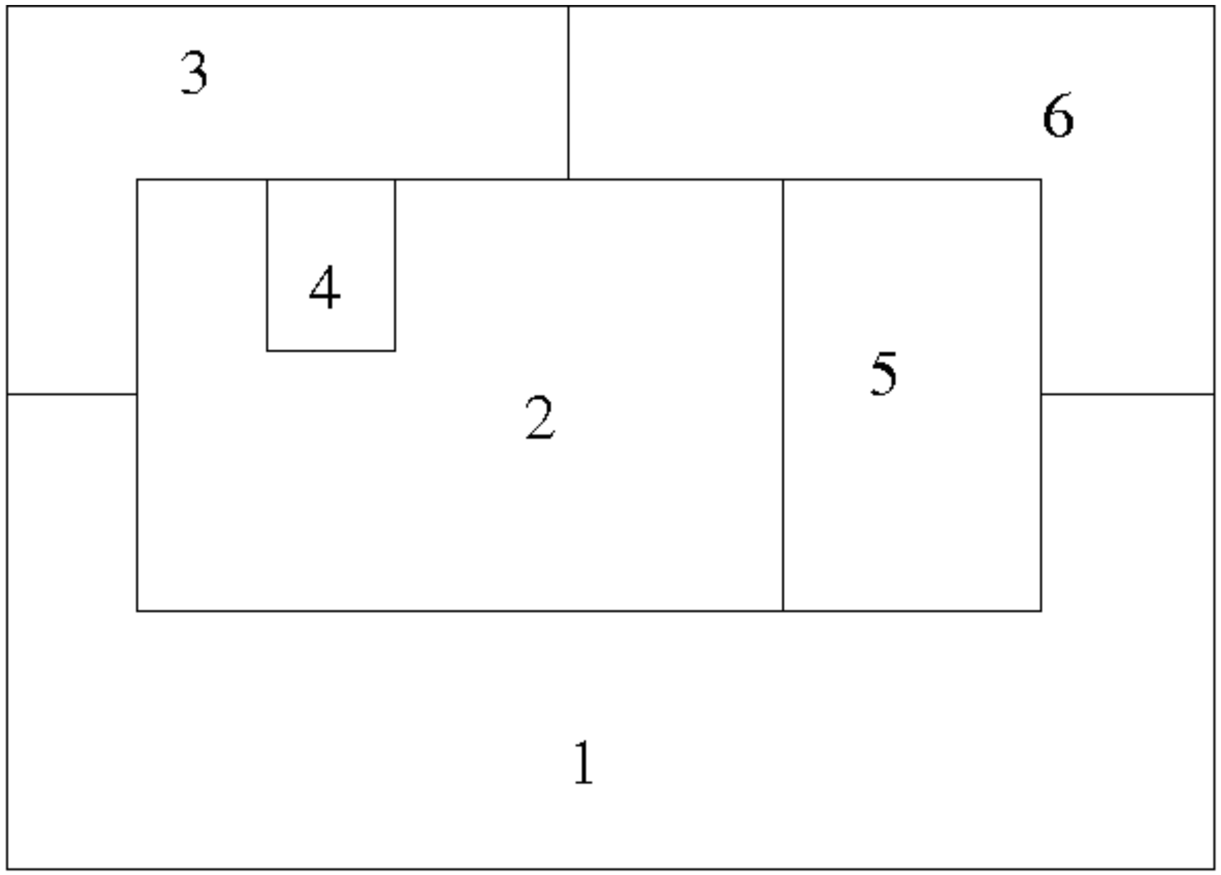
```
  goal(D) :- menu(D,three(X,beef,Z))
  goal(D) :- menu(D,four(X,Y,beef,Z))
  goal(D) :- menu(D,five(X,Y,Z,beef,W))
```

Result

```
  goal(mon)
  goal(tue)
  goal(fri)
```

Map Coloring

Map



Approach 1

```
hue(red)
hue(green)
hue(blue)
hue(purple)

region(r1)
region(r2)
region(r3)
region(r4)
region(r5)
region(r6)
```

```
adjacent(r1,r2)
adjacent(r1,r3)
adjacent(r1,r5)
adjacent(r1,r6)
adjacent(r2,r3)
adjacent(r2,r4)
adjacent(r2,r5)
adjacent(r2,r6)
adjacent(r3,r4)
adjacent(r3,r6)
adjacent(r5,r6)
```

```
color(R,H) :- region(R) & hue(H) & ???
```

Approach 1

```
hue(red)                      adjacent(r1,r2)
hue(green)                    adjacent(r1,r3)
hue(blue)                     adjacent(r1,r5)
hue(purple)                   adjacent(r1,r6)
                              adjacent(r2,r3)
region(r1)                    adjacent(r2,r4)
region(r2)                    adjacent(r2,r5)
region(r3)                    adjacent(r2,r6)
region(r4)                    adjacent(r3,r4)
region(r5)                    adjacent(r3,r6)
region(r6)                    adjacent(r5,r6)
```

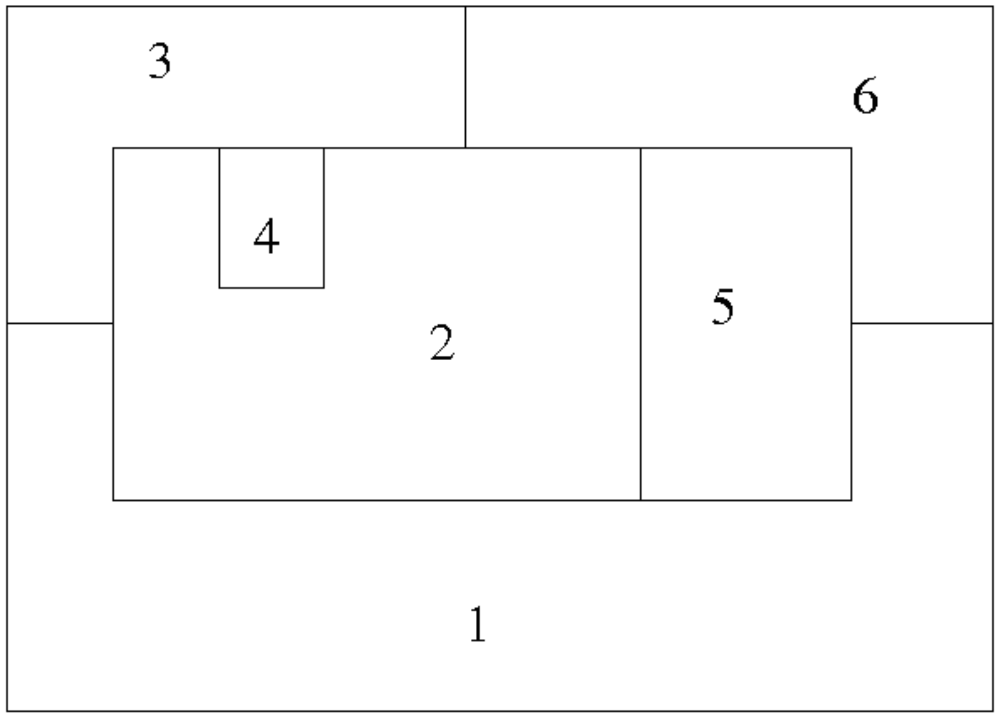
```
color(R,H) :-
  region(R) & hue(H) &
  evaluate(countofall(S,adjacent(R,S) & ???),0)
```

Nope.

Approach 2 - Dataset

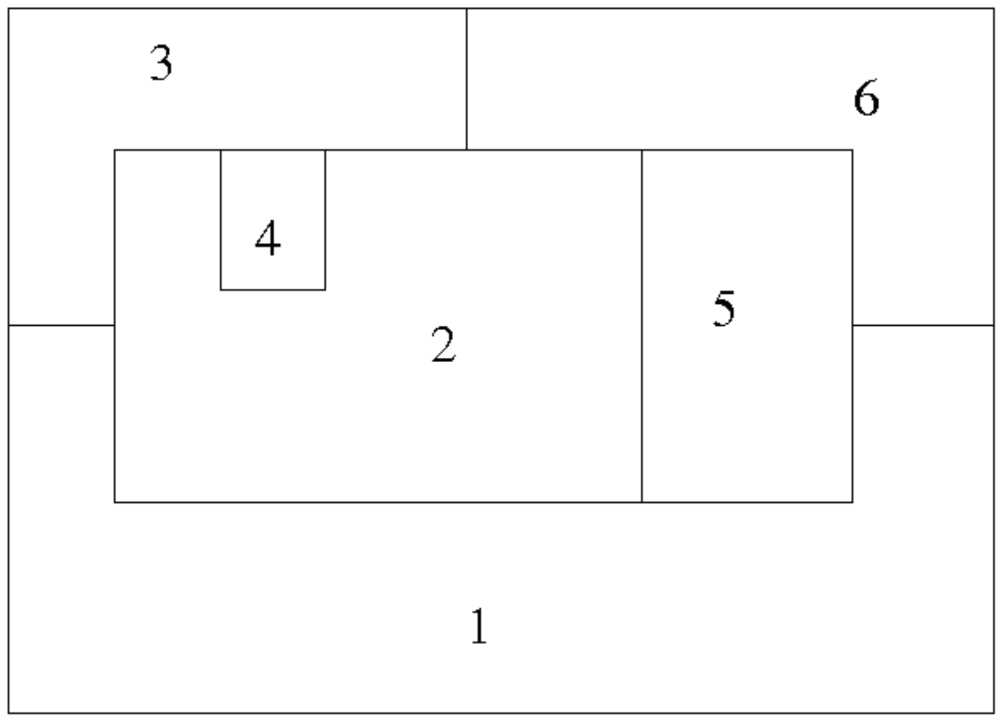
```
hue(red)
hue(green)
hue(blue)
hue(purple)
```

Approach 2 - Query



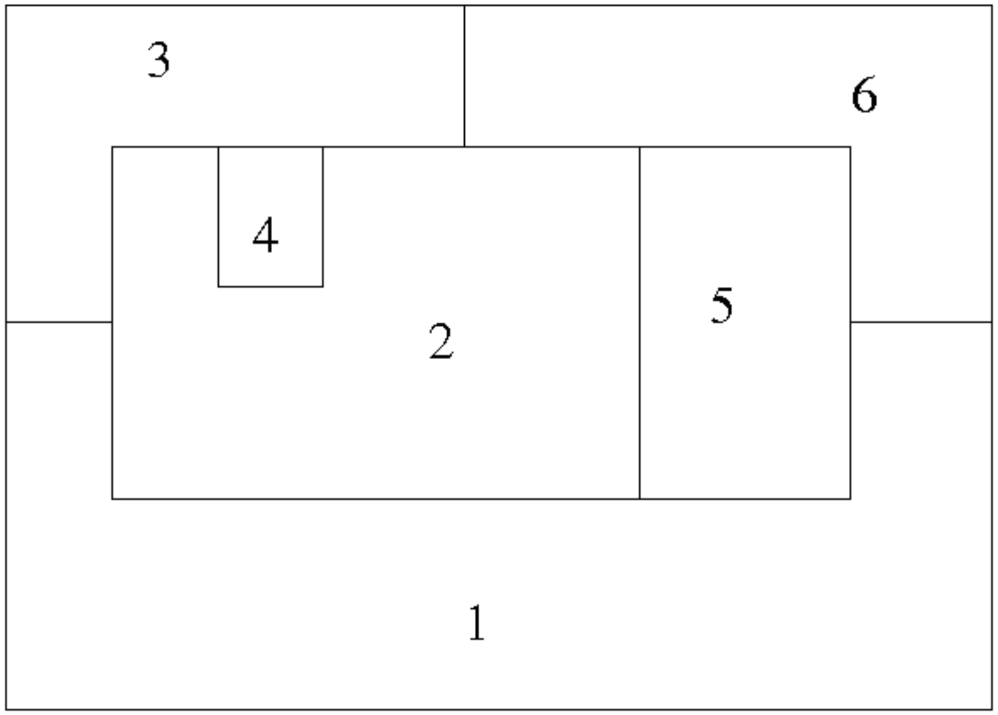
```
goal(C1,C2,C3,C4,C5,C6) :- ???
```

Approach 2 - Query



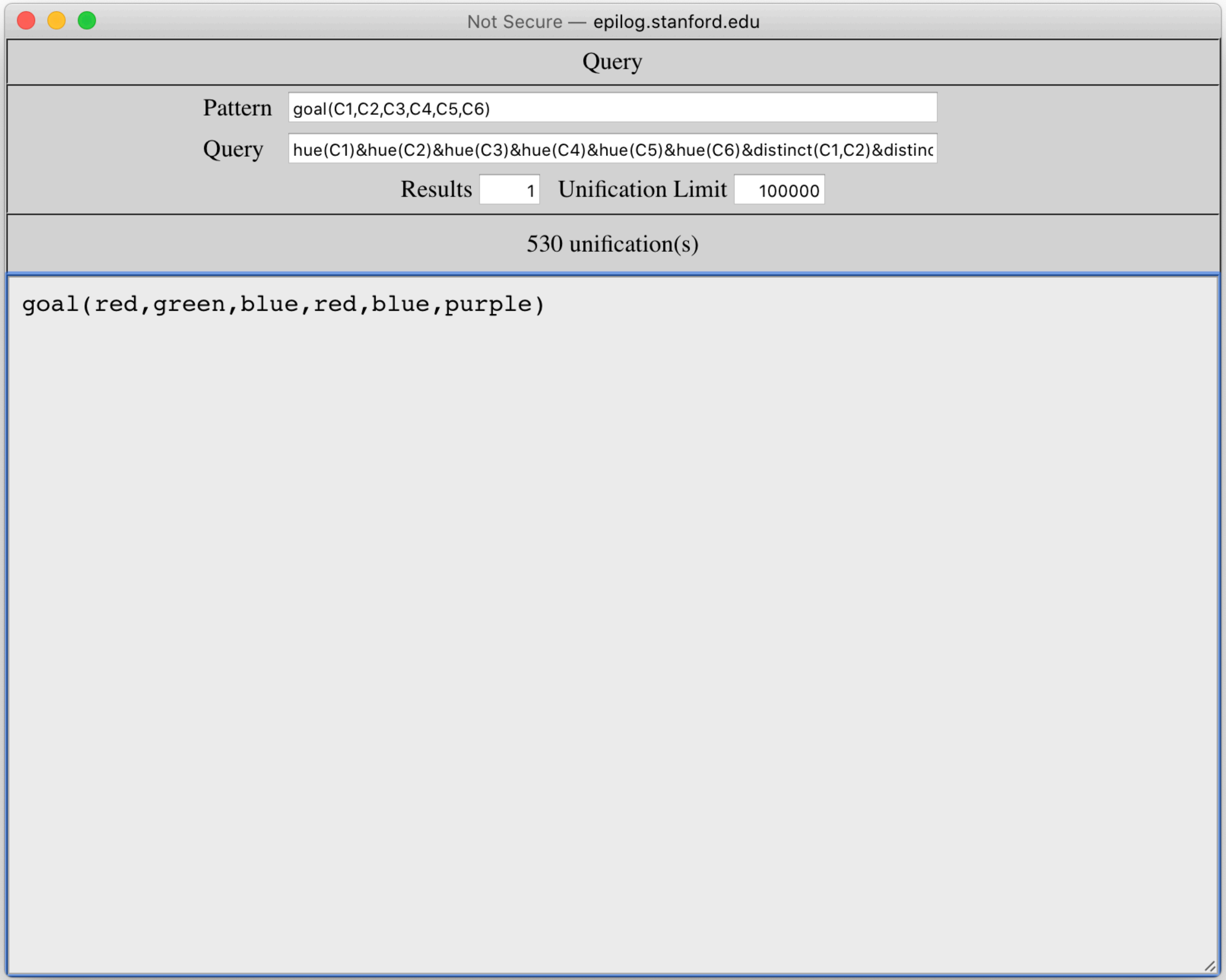
```
goal(C1,C2,C3,C4,C5,C6) :-
  hue(C1) & hue(C2) & hue(C3) & hue(C4) & hue(C5) & hue(C6) &
  ???
```

Approach 2 - Query



```
goal(C1,C2,C3,C4,C5,C6) :-
  hue(C1) & hue(C2) & hue(C3) & hue(C4) & hue(C5) & hue(C6) &
  distinct(C1,C2) & distinct(C1,C3) & distinct(C1,C5) &
  distinct(C1,C6) & distinct(C2,C3) & distinct(C2,C4) &
  distinct(C2,C5) & distinct(C2,C6) & distinct(C3,C4) &
  distinct(C3,C6) & distinct(C5,C6)
```

Approach 2 - Sierra


Not Secure — epilog.stanford.edu

Query

Pattern goal(C1,C2,C3,C4,C5,C6)

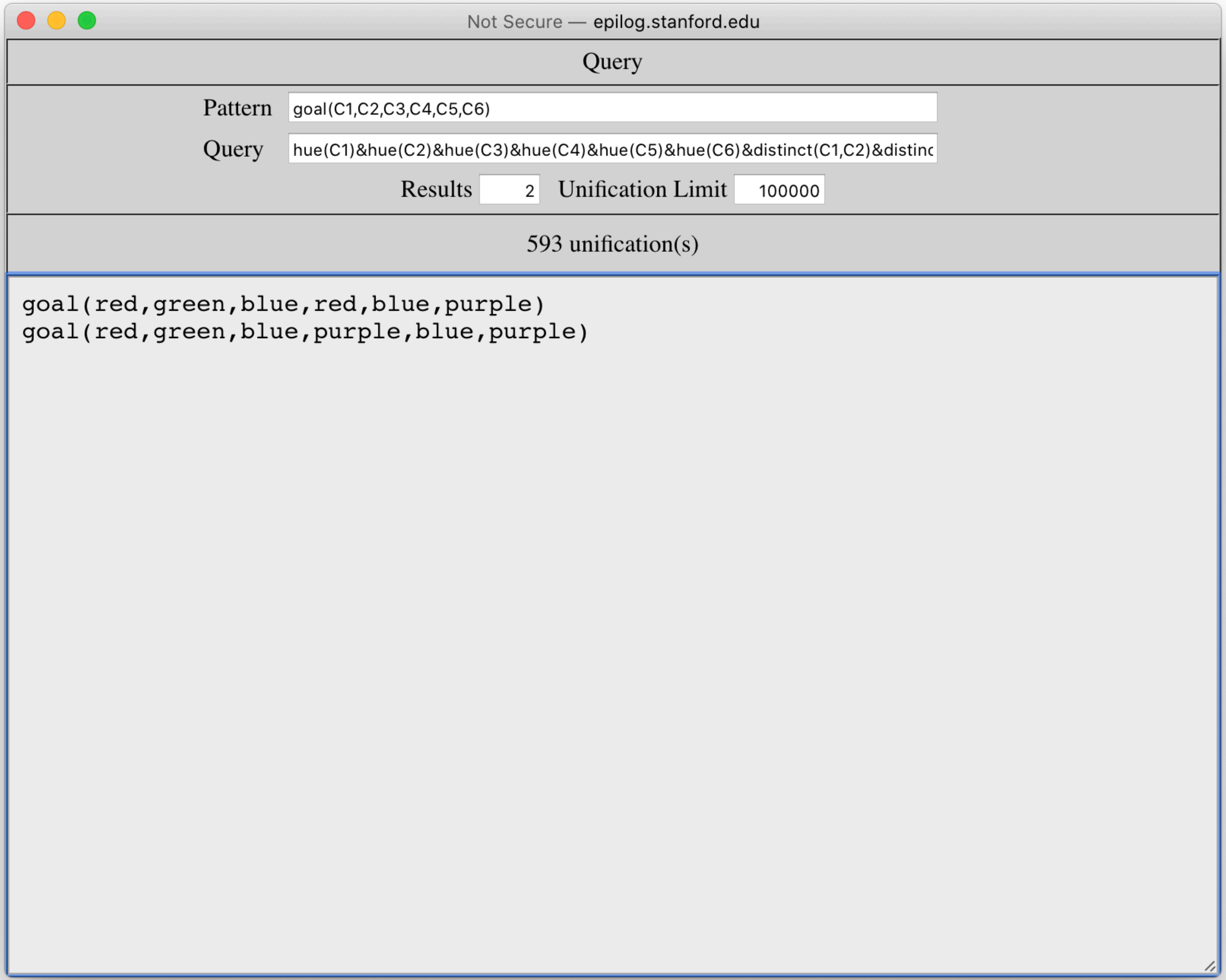
Query hue(C1)&hue(C2)&hue(C3)&hue(C4)&hue(C5)&hue(C6)&distinct(C1,C2)&distinc

Results 1 Unification Limit 100000

530 unification(s)

```
goal(red,green,blue,red,blue,purple)
```

Approach 2 - Sierra



Not Secure — epilog.stanford.edu

Query

Pattern goal(C1,C2,C3,C4,C5,C6)

Query hue(C1)&hue(C2)&hue(C3)&hue(C4)&hue(C5)&hue(C6)&distinct(C1,C2)&distinc

Results 2 Unification Limit 100000

593 unification(s)

```
goal(red,green,blue,red,blue,purple)
goal(red,green,blue,purple,blue,purple)
```

Sierrabase

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