Logic Programming
Evaluation

Michael Genesereth
Computer Science Department
Stanford University

Query Semantics

Dataset: `{p(b),p(c),p(d),q(d)}`

Rule:

```
goal(X) :- p(X) & ~q(X)
```

Instances:

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

Result: `{goal(b),goal(c)}`

Query Semantics

Dataset: {`p(b),p(c),p(d),q(d)`}

Rule:

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

Instances:

```
          goal(b) :- p(b) & ~q(b)
    goal(f(b)) :- p(f(b)) & ~q(f(b))
goal(f(f(b))) :- p(f(f(b))) & ~q(f(f(b)))
                  ...
          goal(c) :- p(c) & ~q(c)
    goal(f(c)) :- p(f(c)) & ~q(f(c))
                  ...
```

Result: {`goal(f(b)),goal(f(c))`}

Programme

Evaluation Procedure
- Evaluating ground queries
- Matching
- Evaluating queries with variables

Computational Analysis
- Unindexed datasets
- Indexing

Query Optimization
- Dropping Rules
- Dropping Subgoals
- Ordering Subgoals

Evaluating Ground Queries

Evaluation of Ground Queries

Given a query rule, call the procedure **eval** on the body. The result is a boolean. The answer to the query is the singleton set of the query head if true; else, it is the empty set.

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

Query: `goal(c) :- p(c,d) & ~p(d,c)`
Body: `p(c,d) & ~p(d,c)`
Result: *true*

Answer: {`goal(c)`}

For ground queries, there is just one instance. Duh.

Evaluating Atoms

(1) If the body of a query rule is an **atom**, we check whether that atom is contained in our dataset. If so, the body is true.

Dataset: {`p(a,b)`, `p(a,c)`, `p(b,c)`, `p(c,d)`}
Query: `goal(a) :- p(a,b)`
Answer: {`goal(a)`}

Dataset: {`p(a,b)`, `p(a,c)`, `p(b,c)`, `p(c,d)`}
Query: `goal(a) :- p(b,a)`
Answer: {}

Evaluating Negations

(2) If the body is a **negation**, we check whether the atom is contained in our dataset. If so, the body is false. If the atom is not contained in our dataset, then the body is true.

Dataset: {`p(a,b)`, `p(a,c)`, `p(b,c)`, `p(c,d)`}
Query: `goal(b) :- ~p(b,c)`
Answer: {}

Dataset: {`p(a,b)`, `p(a,c)`, `p(b,c)`, `p(c,d)`}
Query: `goal(b) :- ~p(c,b)`
Answer: {`goal(b)`}

Evaluating Conjunctions

(3) If the body is a **conjunction** of literals, we execute this procedure on the first conjunct. If the answer is true, we move on to the next conjunct and so forth until we are done. If any one of the conjuncts is false, then the value of the body as a whole is false.

Dataset: `{p(a,b),p(a,c),p(b,c),p(c,d)}`
Query: `goal(c) :- p(c,d) & ~p(d,c)`
Answer: `{goal(c)}`

Dataset: `{p(a,b),p(a,c),p(b,c),p(c,d)}`
Query: `goal(c) :- p(c,d) & p(d,c)`
Answer: `{}`

Evaluation of Ground Queries

The value of a query with multiple rules is the union of the values of each of the rules in the query.

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

Query:

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

Answer: `{goal(a)}` ∪ `{}` ∪ `{goal(c)}` =
`{goal(a),goal(c)}`

Matching

Unification

Matching is the process of determining whether a *pattern* (an expression with or without variables) matches an instance (an expression without variables), i.e. whether the two expressions can be made identical by appropriate substitutions for the variables in the pattern.

Substitutions

A *substitution* is a finite set of pairs of variables and terms, called *replacements*.

$$\{X \leftarrow a, Y \leftarrow b\}$$

The result of *applying* a substitution σ to an expression ϕ is the expression ϕσ obtained from ϕ by replacing every occurrence of every variable with a binding in the substitution by the term to which it is bound.

$$p(X,b)\{X \leftarrow a, Y \leftarrow b\} = p(a,b)$$
$$q(X,Y,X)\{X \leftarrow a, Y \leftarrow b\} = q(a,b,a)$$

Matcher

A substitution σ is a *matcher* for a pattern and an instance if and only if applying the substitution to the pattern results in the given instance.

`p(X,b)`{`X`←`a`, `Y`←`b`} = `p(a,b)`

`q(X,Y,X)`{`X`←`a`, `Y`←`b`} = `q(a,b,a)`

Here, {`X`←`a`, `Y`←`b`} is a matcher for `p(X,b)` and `p(a,b)`. It is also a matcher for `q(X,Y,X)` and `q(a,b,a)`.

Matching Procedure

(1) If the pattern is a **symbol** and the instance is the *same* symbol, then the procedure succeeds, returning the unmodified substitution as result. If the pattern is a symbol and the instance is a *different* symbol or a compound expression, then the procedure fails.

(2) If the pattern is a **variable** *with a binding*, we compare the binding for the variable with the given instance. If they are identical, the procedure succeeds, returning the unmodified substitution as result; otherwise it fails. If the pattern is a variable *without a binding*, we include a binding for the variable in the given instance and we return that substitution as a result.

(3) If the pattern is a **compound expression** and the instance is a *compound expression of the same length*, we iterate across the pattern and the instance. If the pattern is a compound expression and the instance is *a symbol or a compound expression of a different length*, we fail.

Example

Compare: `p(X,Y)`, `p(a,b)`, `{}`

- **Compare:** `p`, `p`, `{}`
- **Result:** `{}` N.B.: `{}` *is not the same as false*.
- **Compare:** `X`, `a`, `{}`
- **Result:** `{X←a}`
- **Compare:** `Y`, `b`, `{X←a}`
- **Result:** `{X←a, Y←b}`

Result: `{X←a, Y←b}`

Example

- **Compare:** `p(X,X),p(a,a),{}`
  - **Compare:** `p`, `p`, `{}`
  - **Result:** `{}`
  - **Compare:** `X`, `a`, `{}`
  - **Result:** `{X←a}`
  - **Compare:** `X`, `a`, `{X←a}`
    - **Compare:** `a`, `a`, `{X←a}`
    - **Result:** `{X←a}`
  - **Result:** `{X←a}`
- **Result:** `{X←a}`

Example

```
Compare: p(X,X),p(a,b),{}
    Compare: p, p, {}
    Result: {}
    Compare: X, a, {}
    Result: {X←a}
    Compare: X, b, {X←a}
        Compare: a, b, {X←a}
        Result: false
    Result: false
Result: false
```

Evaluation with Variables

Evaluation with Variables

Given a query rule, call the procedure **eval** (to be described) on the body and an empty substitution. The result is the **set** of substitutions that satisfy the body. The answer is obtained by applying the substitutions to the head of the rule.

Dataset: {`p(a,b)`, `p(a,c)`, `p(b,c)`, `p(c,d)`}
Query: `goal(Y) :- p(a,Y) & p(Y,Z)`

Call eval: `p(a,Y) & p(Y,Z)`, {}
Exit eval: {{Y←b, Z←c}, {Y←c, Z←d}}

Answer: {`goal(b)`, `goal(c)`}

Evaluating Atoms

(1) If the expression is an **atom**, we try matching the atom to the factoids in our dataset. For each factoid that matches the atom, we add the corresponding substitution to our answer set; and we return the set of all substitutions obtained in this way.

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

Call eval: `p(a,Y)`, {}
Exit eval: {{`Y`←`b`}, {`Y`←`c`}} (two results)

Evaluating Negations

(2) If the expression is a **negation**, we call eval on the target of the negation and the given substitution. If the result is a non-empty set, then the negation is false and we return the empty set. If the result of the recursive call is the empty set, then the negation is true and we return the singleton set containing the input substitution as a result.

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

Call eval: `~p(Y,d)`, `{Y←b}`
Exit eval: {`{Y←b}`} (just one result)

Call eval: `~p(Y,d)`, `{Y←c}`
Exit eval: {} (no results)

Evaluating Conjunctions

(3) If the expression is a **conjunction**, we call eval on the first conjunct and the given substitution. We iterate over the list of answers, for each calling eval on the remaining conjuncts.

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

Call eval: `p(a,Y) & ~p(Y,d)`, {}

    Call eval: `p(a,Y)`, {}

    Exit eval: {{`Y←b`}, {`Y←c`}}

    Call eval: `~p(Y,d)`, {`Y←b`}

    Exit eval: {{`Y←b`}} (just one result)

    Call eval: `~p(Y,d)`, {`Y←c`}

    Exit eval: {} (no results)

Exit eval: {{`Y←b`}} (just one result)

Evaluation with Variables

Given a query rule, call the procedure **eval** (to be described) on the body and an empty substitution. The result is a list of substitutions that satisfy the body. The value of the rule is obtained by applying the substitutions to the head of the rule.

Dataset: `{p(a,b),p(a,c),p(b,c),p(c,d)}`
Query: `goal(Y) :- p(a,Y) & ~p(Y,d)`

Call eval: `p(a,Y) & ~p(Y,d),{}`
Exit eval: `{{Y←b}}`

Answer: `{goal(b)}`

Computational Analysis

Assumptions

Worst Case Analysis based on *number of unifications*

n objects in Herbrand Universe

Assumptions:

All rules applied

Subgoals processed left to right

(1) We will first consider analysis with *no indexing*.

(2) Then we will look at analysis with *indexed data*.

Optimizations not considered (until next lesson)*:*

Dropping redundant rules or subgoals

Reordering of subgoals

Caching

Example

Example

```
goal(a,c) :- p(a,Y) & p(Y,c)
```

Cost of computing whether `goal(a,c)` is true

Example

Example

```
goal(a,c) :- p(a,Y) & p(Y,c)
```

Cost of computing whether `goal(a,c)` is true

$$n\text{^}2 + n*n\text{^}2 = n\text{^}2 + n\text{^}3$$

Suppose $n = 3$

$$3\text{^}2 + 3*3\text{^}2 = 3\text{^}2 + 3\text{^}3 = 36$$

Example

Example

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

Cost of computing *all* instances of `goal`

Example

Example

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

Cost of computing *all* instances of `goal`

$$n\text{^}2 + n\text{^}2*n\text{^}2 = n\text{^}2 + n\text{^}4$$

Suppose $n = 3$

$$3\text{^}2 + 3\text{^}2*3\text{^}2 = 3\text{^}2 + 3\text{^}4 = 90$$

Full Indexing

In *full indexing*, each factoid appears on the list of factoids associated with each constant in that factoid.

Example: {`p(a,b)`, `p(b,c)`, `q(b)`, `q(c)`}

Index on `p`: {`p(a,b)`, `p(b,c)`}
Index on `q`: {`q(b)`, `q(c)`}

Index on `a`: {`p(a,b)`}
Index on `b`: {`p(a,b)`, `p(b,c)`, `q(b)`}
Index on `c`: {`p(b,c)`, `q(c)`}

NB: No *compound* indices (e.g. all factoids with `a` *and* `b`)
NB: No *positional* indices (e.g. `p(a,b)` vs. `p(b,a)`)

Worst Case

Example:

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

Index on `p`: {`p(a,a)`, `...` , `p(c,c)`}

Index on `a`: {`p(a,a)`, `p(a,b)`, `p(a,c)`, `p(b,a)`, `p(c,a)`}

Index on `b`: {`p(a,b)`, `p(b,a)`, `p(b,b)`, `p(b,c)`, `p(c,b)`}

Index on `c`: {`p(a,c)`, `p(b,c)`, `p(c,a)`, `p(c,b)`, `p(c,c)`}

Example with Symbol Indexing

Example

```
goal(a,c) :- p(a,Y) & p(Y,c)
```

Cost of computing `goal(a,c)` *without* indexing

$$n\text{^}2 + n*n\text{^}2 = n\text{^}2 + n\text{^}3$$

Suppose $n = 3$

$$3\text{^}2 + 3*3\text{^}2 = 3\text{^}2 + 3\text{^}3 = 36$$

Cost of computing whether `goal(a,c)` *with* indexing

$$(2n\text{-}1) + n*(2n\text{-}1) = 2n\text{^}2 + n - 1$$

Suppose $n = 3$

$$2*3\text{^}2 + 3 - 1 = 18 + 2 = 20$$

Example with Indexing

Example

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

Cost of computing `goal(X,Z)` *without* indexing

n^2 + $n^{2*}n$^2 = n^2 + n^4

Suppose n = 3

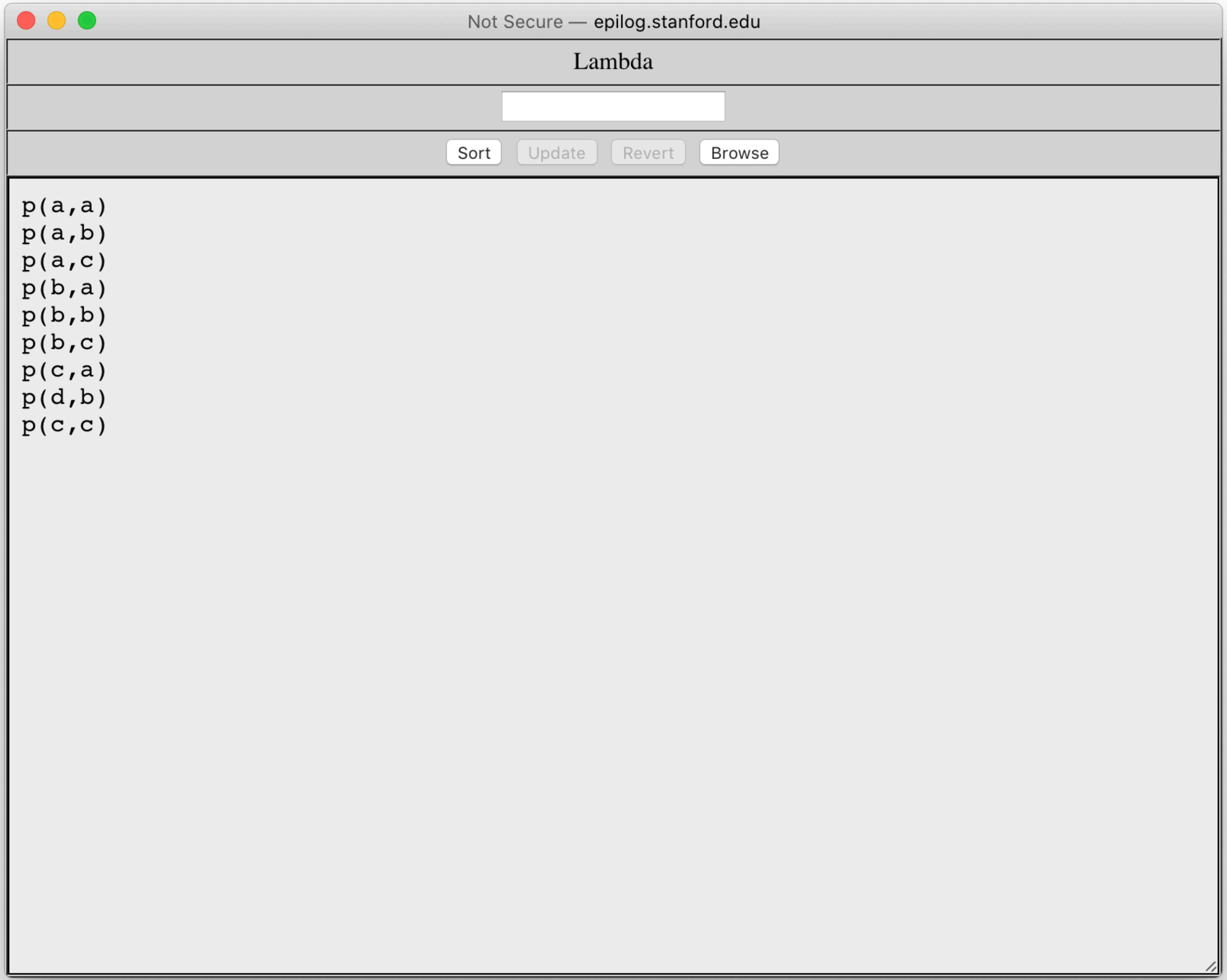
3^2 + 3^2*3^2 = 3^2 + 3^4 = 90

Cost of computing *all* instances *with* indexing:

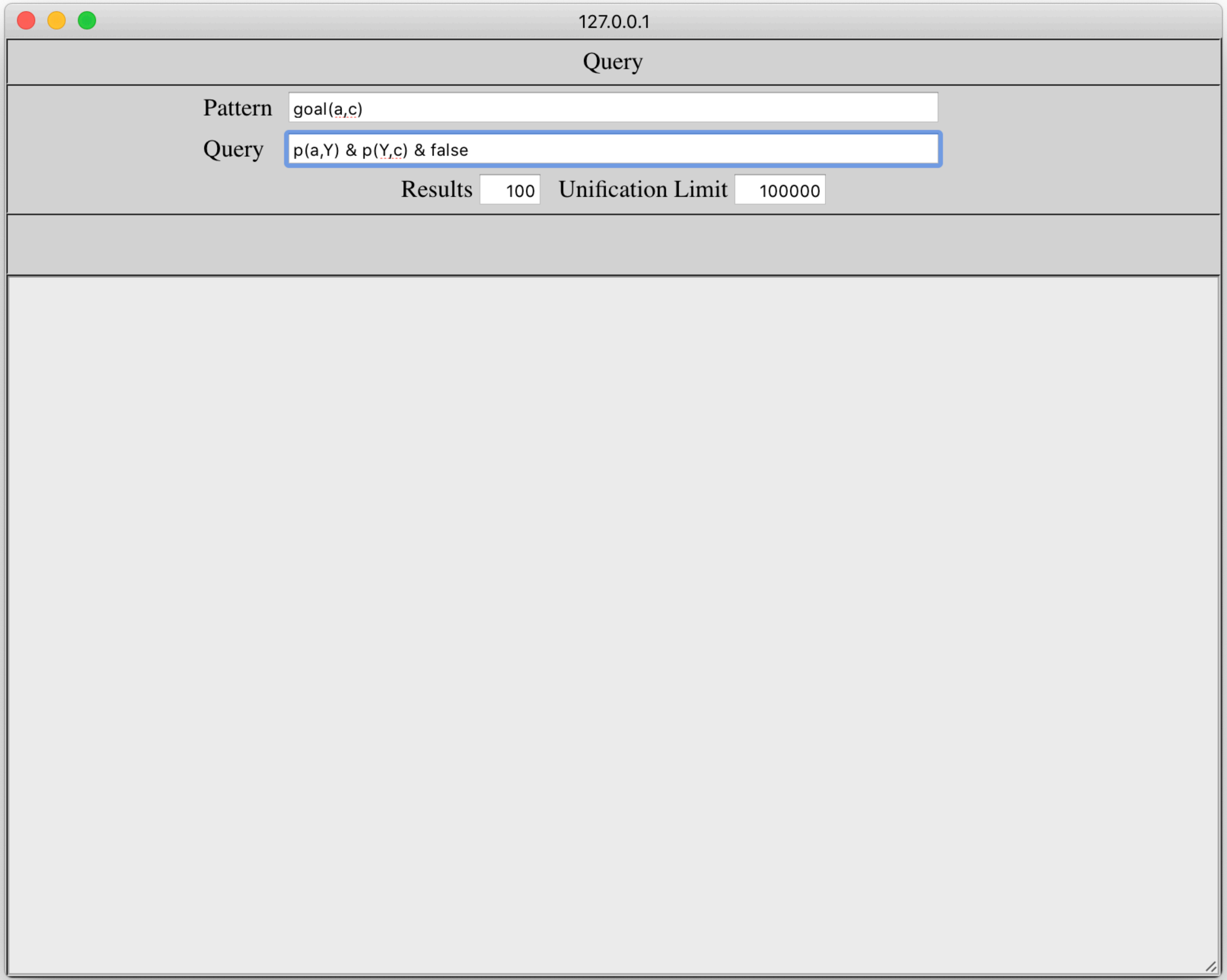
n^2 + n^2*(2n-1) = n^2 + 2n^3 - n^2 = 2n^3

Suppose n = 3

2*3^3 = 54



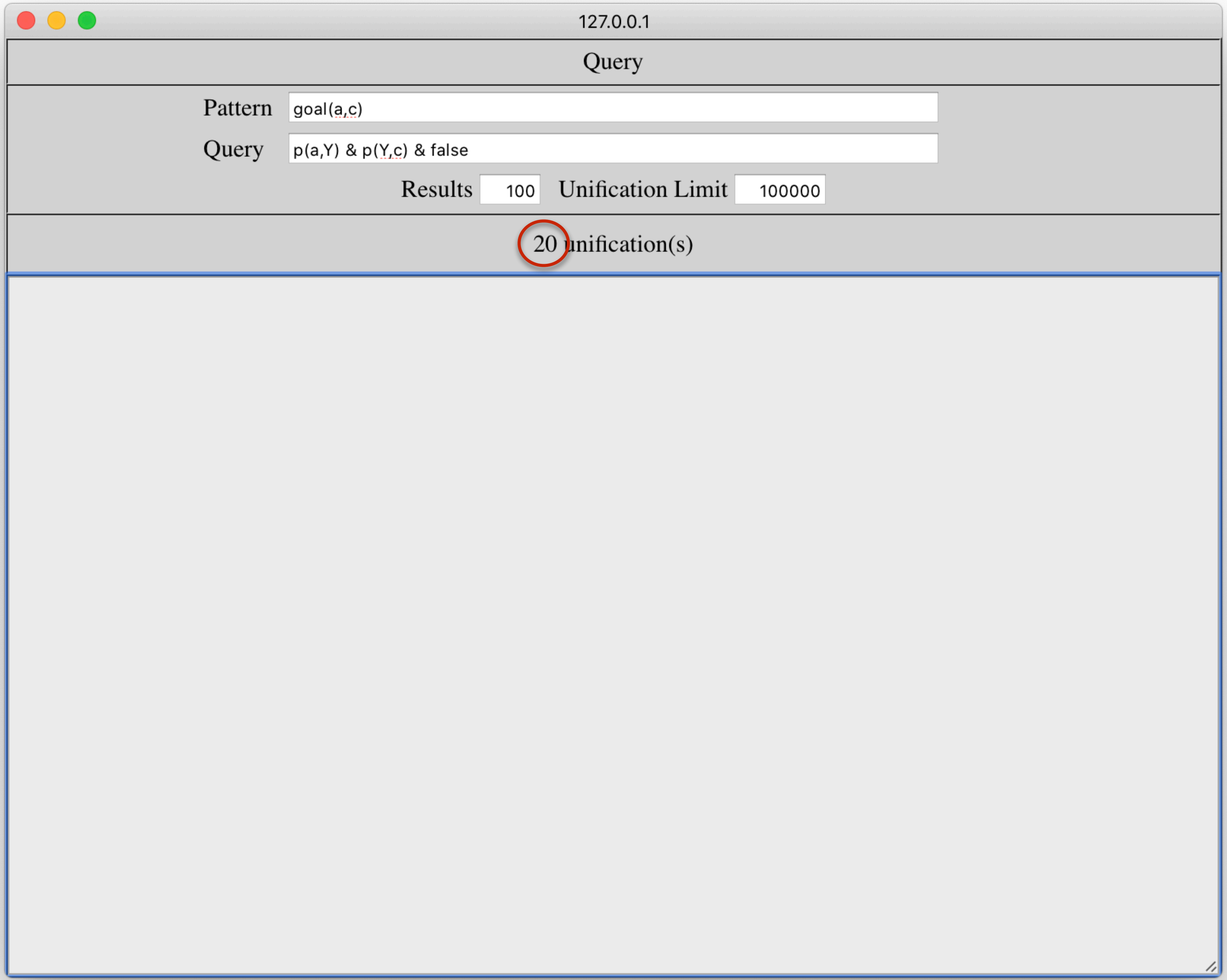
```
p(a,a)
p(a,b)
p(a,c)
p(b,a)
p(b,b)
p(b,c)
p(c,a)
p(d,b)
p(c,c)
```



Pattern goal(a,c)

Query p(a,Y) & p(Y,c) & false

Results 100 Unification Limit 100000


127.0.0.1

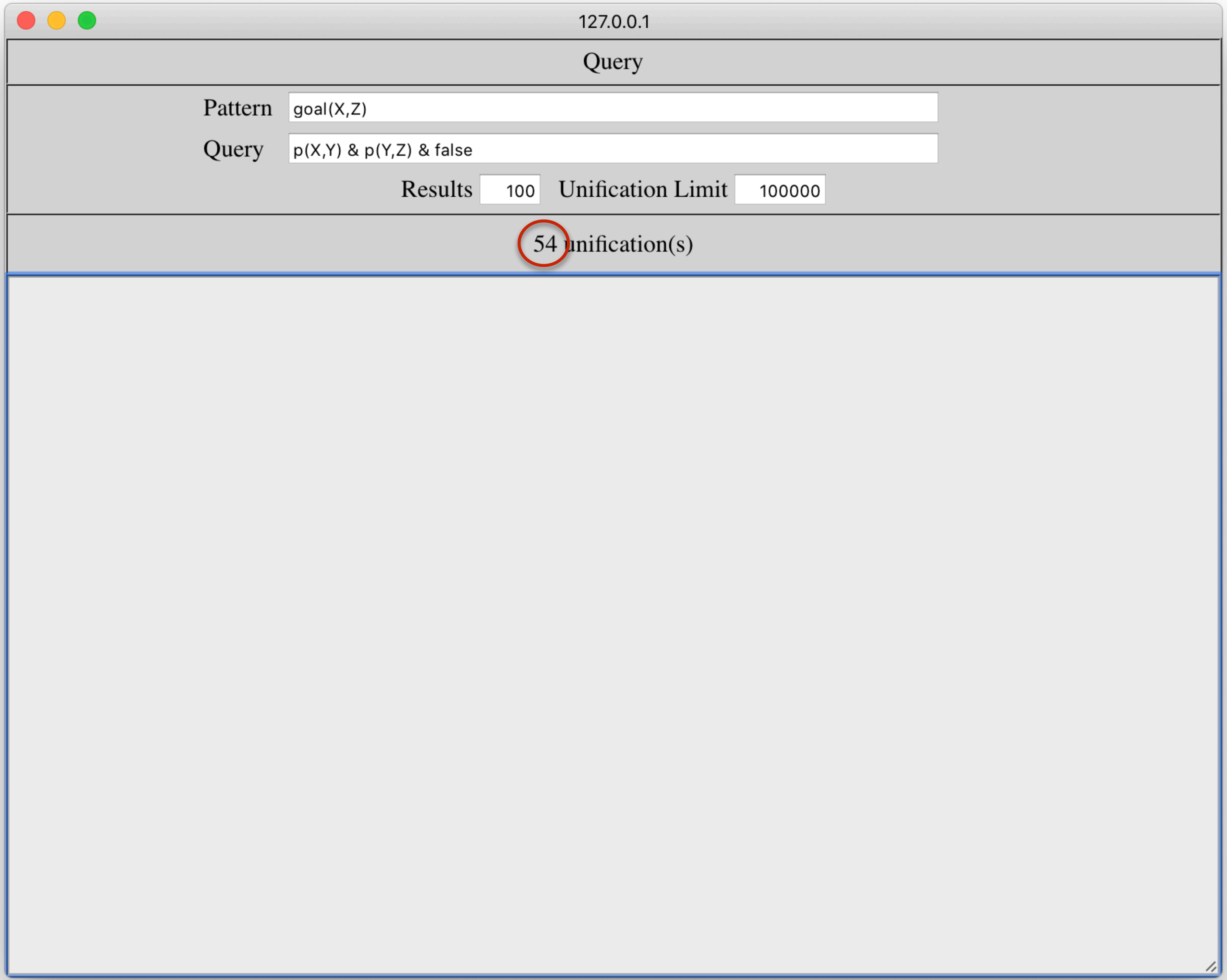
Query

Pattern goal(a,c)

Query p(a,Y) & p(Y,c) & false

Results 100 Unification Limit 100000

20 unification(s)


127.0.0.1

Query

Pattern goal(X,Z)

Query p(X,Y) & p(Y,Z) & false

Results 100 Unification Limit 100000

54 unification(s)

Pipelining

Basic Idea

Normal Evaluation of Conjuncts: (1) Call eval on the first conjunct and a given substitution. (2) Collect all answers. (3) Then iterate over answers, for each calling eval on the remaining conjuncts.

All *answers to the first conjunct are computed before working on subsequent conjuncts.*

Pipelined Evaluation of Conjuncts: (1) Call eval on the first conjunct and a given substitution. (2) Compute just one substitution. (3) Call eval on remaining conjuncts with the resulting substitution. Once done, back up and compute another solution to the first conjunct and repeat.

One *answer to the first conjunct is computed and then used before generating additional answers to the first conjunct.*

Normal Evaluation

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

Call eval: `p(X,Y) & q(Y)`, {}

 Call eval: `p(X,Y)`, {}

 Exit eval: {{`X`←`a`,`Y`←`b`}, {`X`←`b`,`Y`←`c`}}

 Call eval: `q(Y)`, {`X`←`a`,`Y`←`b`}

 Exit eval: {{`X`←`a`,`Y`←`b`}}

 Call eval: `q(Y)`, {`X`←`b`,`Y`←`c`}

 Exit eval: {{`X`←`b`,`Y`←`c`}}

Exit eval: {{`X`←`a`,`Y`←`b`}, {`X`←`b`,`Y`←`c`}}

Pipelined Evaluation

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

Call eval: `p(X,Y) & q(Y)`, {}

    Call eval: `p(X,Y)`, {}

    Exit eval: `{X←a,Y←b}`

    Call eval: `q(Y)`, `{X←a,Y←b}`

    Exit eval: `{X←a,Y←b}`

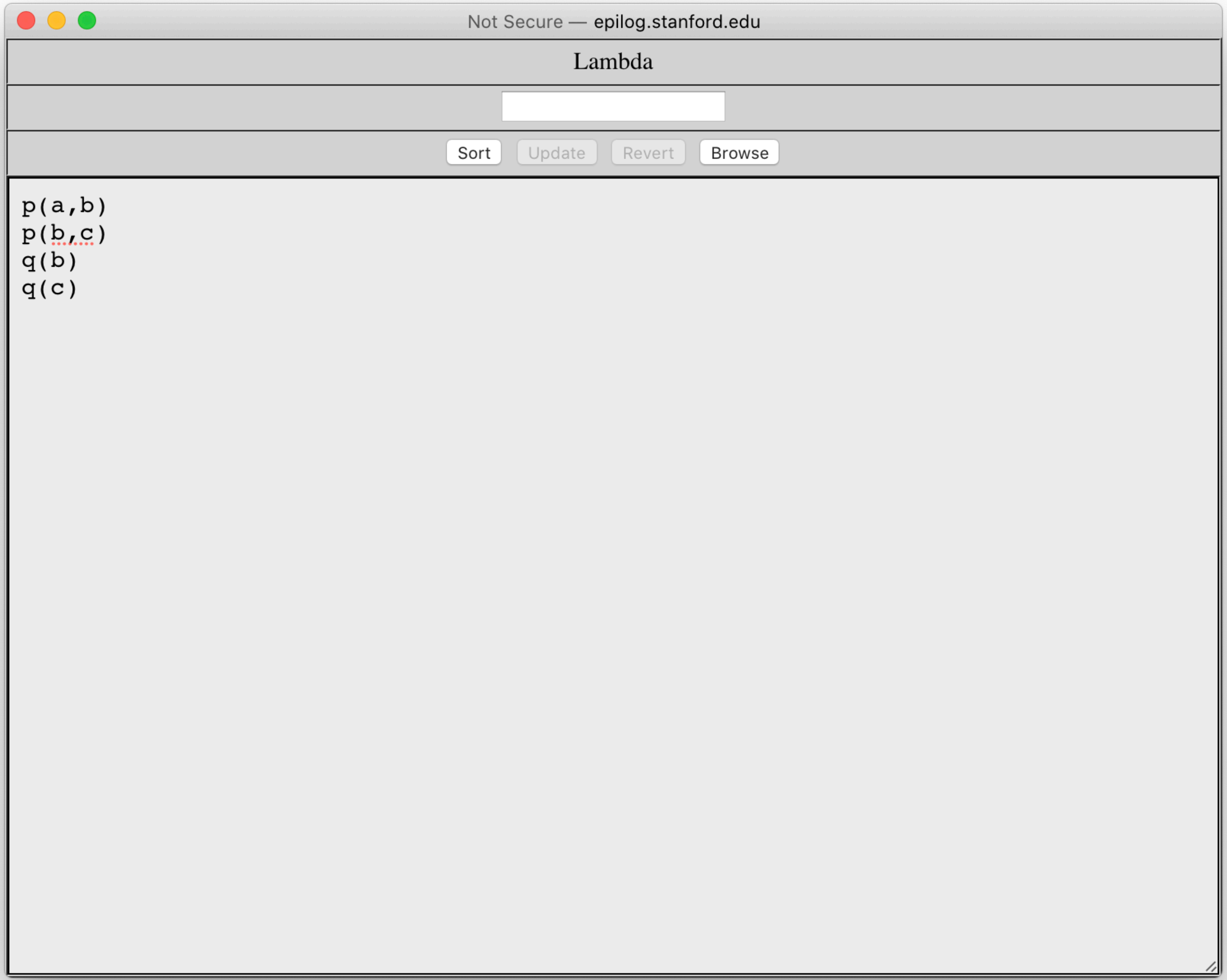
    Redo eval: `p(X,Y)`, {}

    Exit eval: `{X←b,Y←c}`

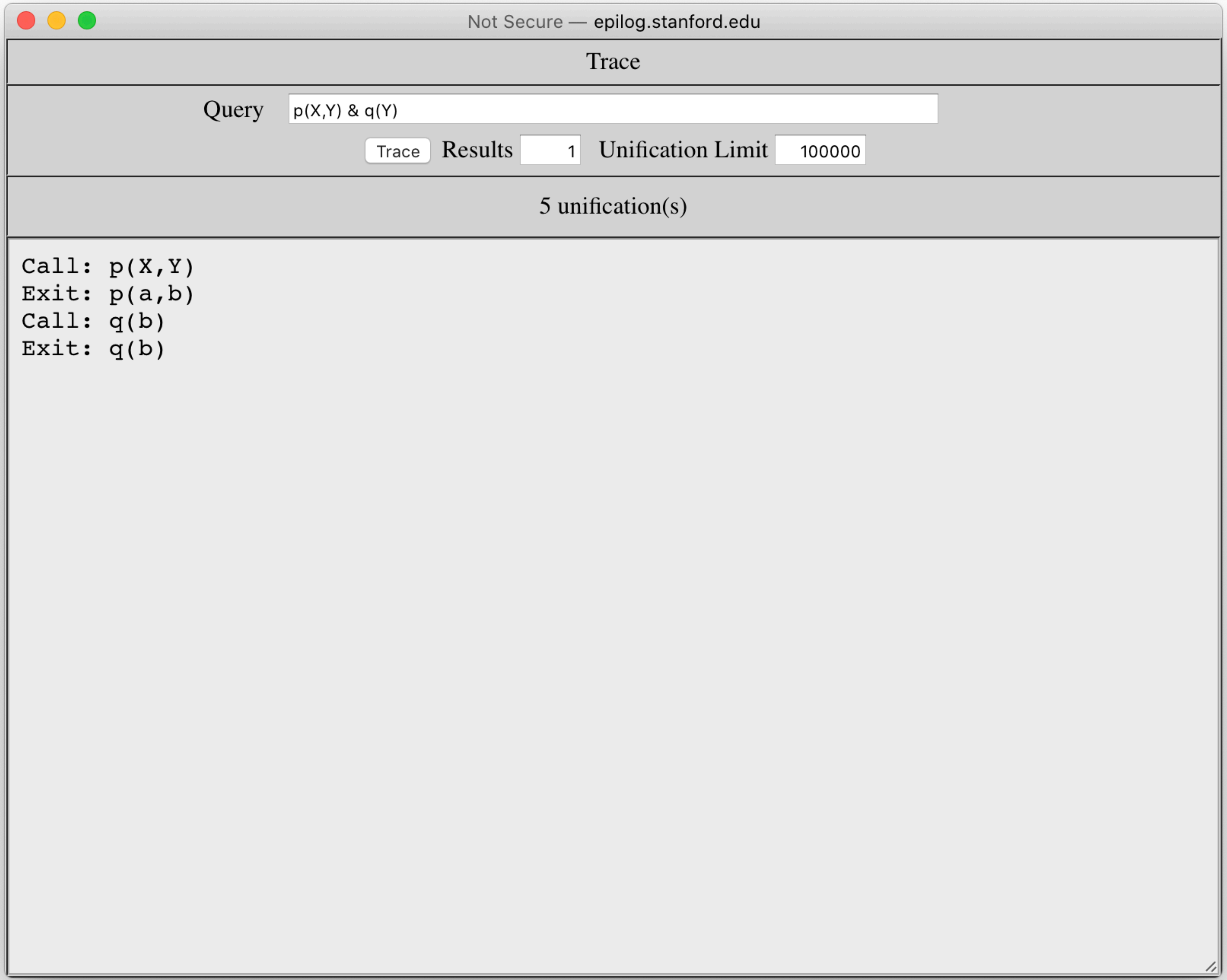
    Call eval: `q(Y)`, `{X←b,Y←c}`

    Exit eval: `{X←b,Y←c}`

Exit eval: {`{X←a,Y←b}`, `{X←b,Y←c}`}


Not Secure — epilog.stanford.edu
Lambda
Sort
Update
Revert
Browse

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



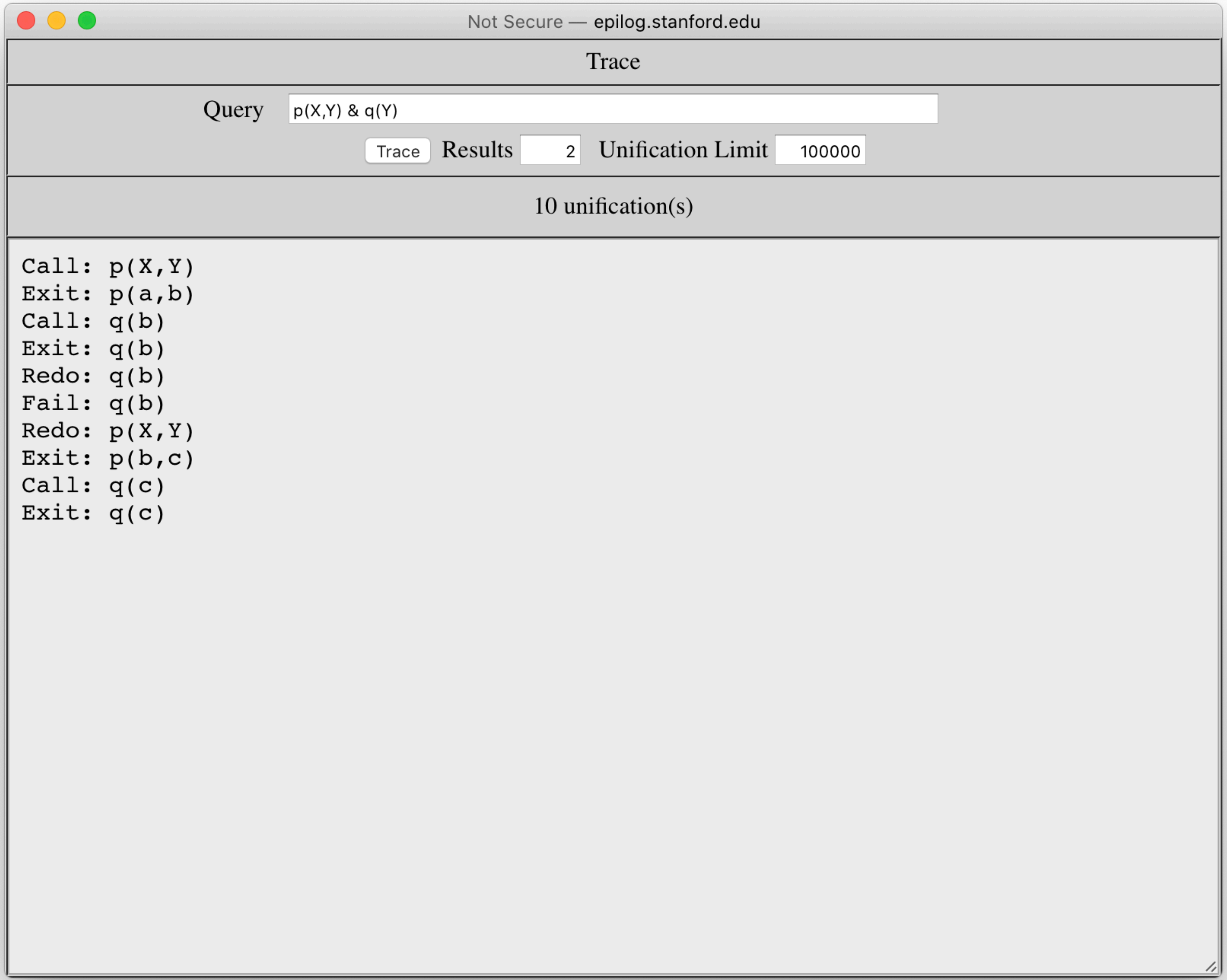
Query: p(X,Y) & q(Y)

Results: 1

Unification Limit: 100000

5 unification(s)

```
Call: p(X,Y)
Exit: p(a,b)
Call: q(b)
Exit: q(b)
```



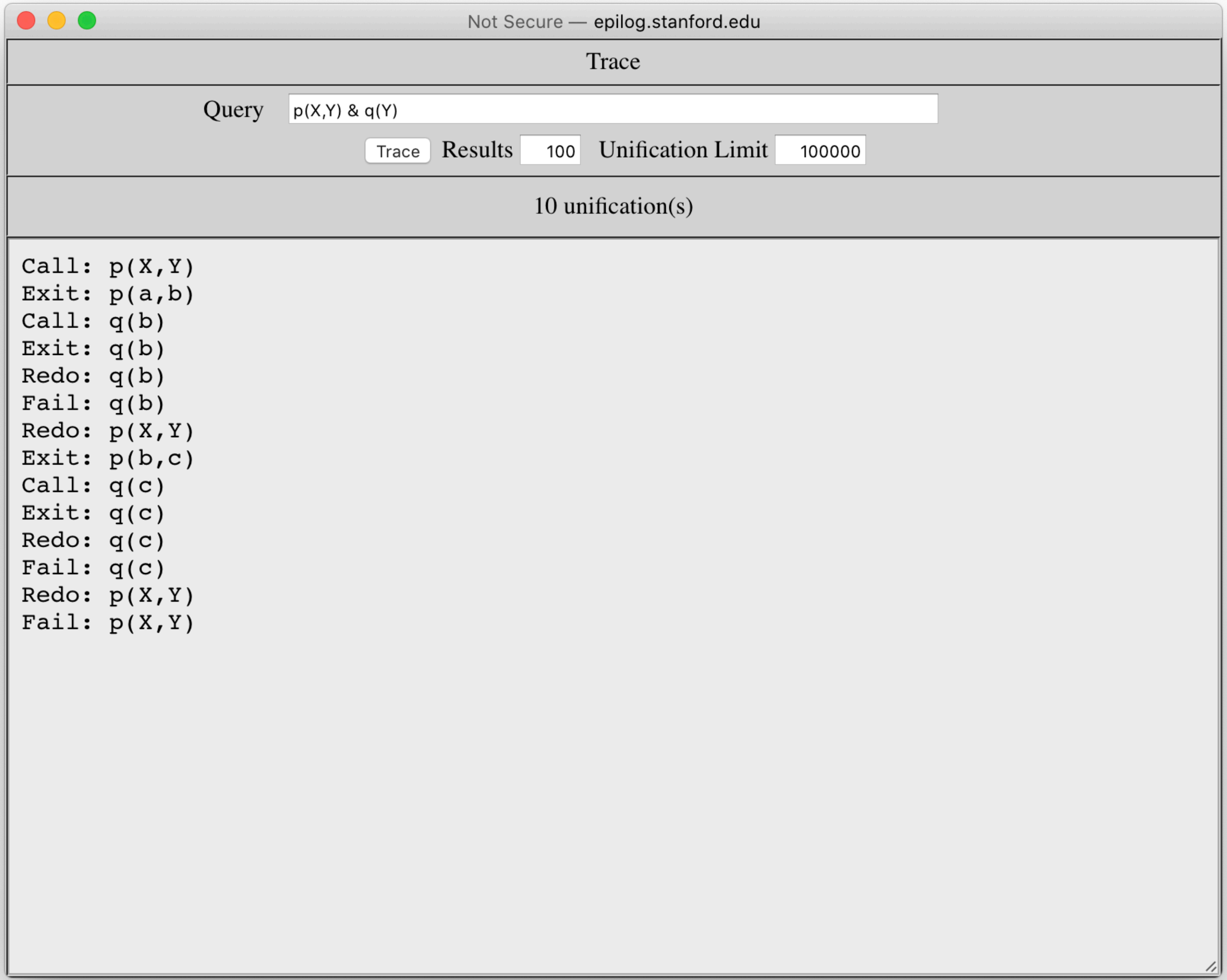
Query: p(X,Y) & q(Y)

Results: 2

Unification Limit: 100000

10 unification(s)

```
Call: p(X,Y)
Exit: p(a,b)
Call: q(b)
Exit: q(b)
Redo: q(b)
Fail: q(b)
Redo: p(X,Y)
Exit: p(b,c)
Call: q(c)
Exit: q(c)
```



Query p(X,Y) & q(Y)

Trace Results 100 Unification Limit 100000

10 unification(s)

```
Call: p(X,Y)
Exit: p(a,b)
Call: q(b)
Exit: q(b)
Redo: q(b)
Fail: q(b)
Redo: p(X,Y)
Exit: p(b,c)
Call: q(c)
Exit: q(c)
Redo: q(c)
Fail: q(c)
Redo: p(X,Y)
Fail: p(X,Y)
```

Query Optimization

Semantic Equivalence

Two queries are **semantically equivalent** if and only if they produce identical results for every dataset.

Query 1:

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

Query 2:

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

Computational Disparity

Syntactically different but semantically equivalent queries may have different computational properties.

Query 1: O(*n*^4)

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

Query 2: O(*n*^3)

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

Optimization

Logical - deleting and/or rearranging subgoals and rules

Rule Removal

Subgoal Removal

Subgoal Ordering

Conceptual - changing vocabulary

Triples vs Wide Relations

Minimal Spanning Trees

Reification and Relationalization

Rule Removal

Useless Rules

Example:

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

Example:

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

Useless rules produce no results.

Redundant Rules

Example:

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

Redundant rules produce only results produced by other rules, i.e. answers to one rule are a subset of answers to the other.

Trickier Cases

Redundant Rules:

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

Non-Redundant Rules:

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

Subsumption

A rule *r1* **subsumes** a rule *r2* if and only if it is possible to replace some or all of the variables of *r1* in such a way that the heads are the same and all of subgoals of *r1* are members of the body of *r2*.

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

Here, the first rule subsumes the second. We just replace `X` in the first rule by itself and replace `Y` by `b`, with the following result.

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

Subsumption Technique

Start with rule 1 and rule 2 as inputs where (a) the heads are identical and (b) neither rule contains any negations.

(1) Create a substitution in which each variable in rule 2 is bound to a distinct, new symbol.

(2) Create a **canonical dataset** consisting of the *subgoals of rule 2* where all variables are replaced by these bindings.

(3) Substitute the bindings for the head variables in rule 1.

(4) Evaluate this modified rule on the dataset created in (2). If there are answers, then rule 1 subsumes rule 2. If not, then rule 1 does *not* subsume rule 2.

Example

Inputs

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

Substitution: {`X←c1`, `Z←c3`}

Canonical Dataset: {`p(c1,b)`, `q(b)`, `r(c3)`}

Evaluate: `p(c1,Y) & q(Y)`
Result: {`Y←b`}

The first rule *does* subsume the second.

Example

Inputs

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

Substitution: {`X←c1`, `Y←c2`}

Canonical Dataset: {`p(c1,c2)`, `q(c2)`}

Evaluate: `p(c1,b) & q(b) & r(Z)`
Result: failure

The first rule *does not* subsume the second.

Rule Removal Technique

Compare every rule to every other rule (quadratic). If one rule subsumes another, it is okay to drop the subsumed rule.

NB: Applies only to rules with *no negative subgoals* and *no predefined relations*.

NB: The technique is *sound* in that it is guaranteed to produce an equivalent query.

NB: In *the absence of any constraints* on datasets to which the rules are applied, it is also guaranteed to be *complete* in that all surviving rules are needed for some dataset.

NB: In *the face of constraints*, it may be possible to drop rules that are not detected by this method, i.e. *not complete*.

Extensions

If heads are not identical, they can sometimes be made identical by consistently replacing variables while avoiding clashes.

Original rules:

```
  goal(X) :- p(X,b) & q(b) & r(Z)
  goal(U) :- p(U,V) & q(V)
```

Equivalent rules:

```
  goal(X) :- p(X,b) & q(b) & r(Z)
  goal(X) :- p(X,V) & q(V)
```

There are other extensions for dealing with rules involving *negations* and *built-ins* and *constraints*.

Subgoal Removal

Subgoal Removal

Original Rule:

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

Equivalent Reformulation:

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

Subgoal Removal Technique

Accept query rule as input.

(1) Delete a subgoal.
(2) Check whether the resulting rule subsumes the original.
(3) If yes, continue. If no, try a different subgoal.

Output the result.

Example

Original Rule:

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

Delete first subgoal - does **not** subsume (and not safe):

```
goal(X,Y) :- q(Y) & q(Z)
```
✗

Delete second subgoal - does **not** subsume:

```
goal(X,Y) :- p(X,Y) & q(Z)
```
✗

Delete third subgoal - subsumes original:

```
goal(X,Y) :- p(X,Y) & q(Y)
```
✓

Soundness

This technique is *sound* in that it is guaranteed to produce an equivalent query.

Completeness

In *the absence of any constraints*, this method is guaranteed to be *complete* in that all surviving subgoals are needed for some dataset.

In *the presence of constraints*, it may be possible to drop more subgoals, i.e. *not complete*.

```
goal(X,Y) :- father(X,Y) & male(X)
```

There are extensions that deal with constraints. See literature on *the chase*.

Subgoal Ordering

Subgoal Ordering

Original Rule

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

Reformulation

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

Analysis

Original Rule

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

$(n\^2 + 2n) + n*((n\^2 + 2n) + n*(n\^2 + 2n)) =$
$n\^4 + 3n\^3 + 3n\^2 + 2n$

Reformulation

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

Analysis

Original Rule

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

$(n\text{^}2 + 2n) + n*((n\text{^}2 + 2n) + n*(n\text{^}2 + 2n)) =$
$n\text{^}4 + 3n\text{^}3 + 3n\text{^}2 + 2n$

Reformulation

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

$(n\text{^}2 + 2n) + n*((n\text{^}2 + 2n) + 1*(n\text{^}2 + 2n)) =$
$2n\text{^}3 + 5n\text{^}2 + 2n$

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

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

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

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

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

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

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

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

Narrow and Wide Relations

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.

Abstract Example

Wide Relation:

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

Triples:

```
p1(a,d)    p2(a,e)
p1(b,d)    p2(b,e)
p1(c,d)    p2(c,e)
```

Analysis

Wide Relation:

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

Query: `goal(X) :- p(X,d,e)`
Cost without indexing: 3 **Cost with indexing:** 3

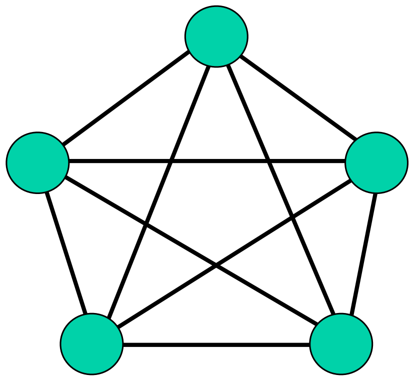
Triples:

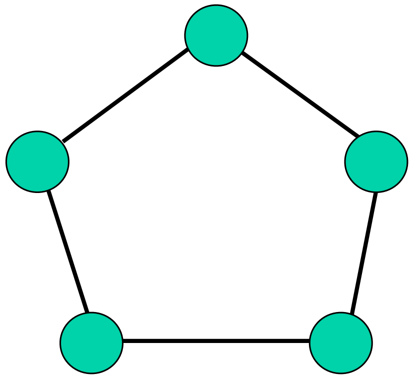
```
p1(a,d)    p2(a,e)
p1(b,d)    p2(b,e)
p1(c,d)    p2(c,e)
```

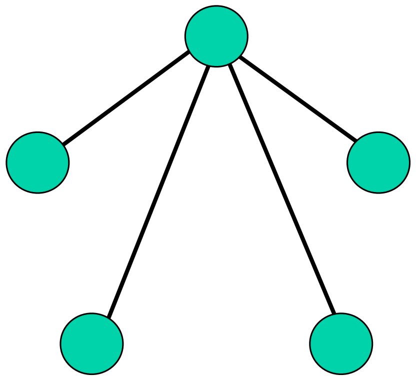
Query: `goal(X) :- p1(X,d) & p2(X,e)`
Cost without indexing: 24 **Cost with indexing:** 9

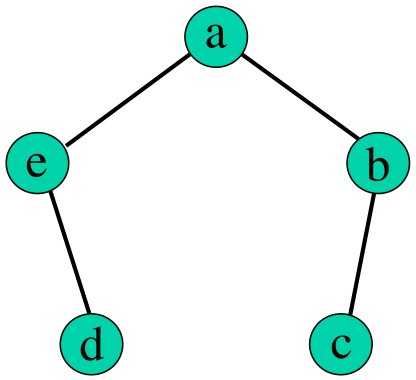
Minimal Spanning Trees

Social Isolation Cells









Representation

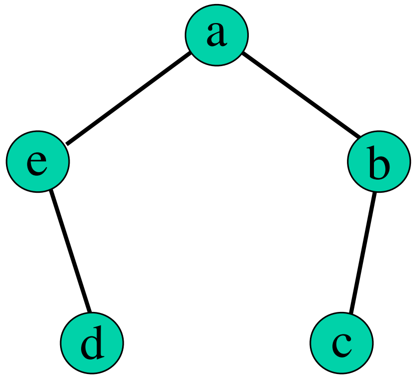
Vocaulary:

People - `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, `j`, ...

Interaction - `r`/2

Example:

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



NB: Possible to represent undirected with only one factoid per arc rather than two, but we will ignore that for now.

Computational Analysis

Are two people are in the same cell?

```
goal(a,e) :- r(a,e)
goal(a,e) :- r(a,Y) & r(Y,e)
goal(a,e) :- r(a,Y1) & r(Y1,Y2) & r(Y2,e)
goal(a,e) :- r(a,Y1) & r(Y1,Y2) & r(Y2,Y3) & r(Y3,e)
```

Number of unifications for `goal(a,e)` (with indexing):

$$8$$

$$8 + 4*8 \ = 40$$

$$8 + 4*(8 + 4*8)) \ = 168$$

$$8 + 4*(8 + 4*(8 + 4*8)) \ = 680$$

Total: 896

Alternative Representation

Precompute and store the transitive closure of `r`

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

Are two people are in the same cell?

```
goal(a,e) :- r(a,e)
```

Number of unifications for `goal(a,e)` (with indexing):

8

Number of factoids for *n* objects:

$8*n$

MST Representation

Assign a number for each group and store with people

```
r(a,1)    r(f,2)    ...
r(b,1)    r(g,2)    ...
r(c,1)    r(h,2)    ...
r(d,1)    r(i,2)    ...
r(e,1)    r(j,2)    ...
```

Are two people are in the same cell?

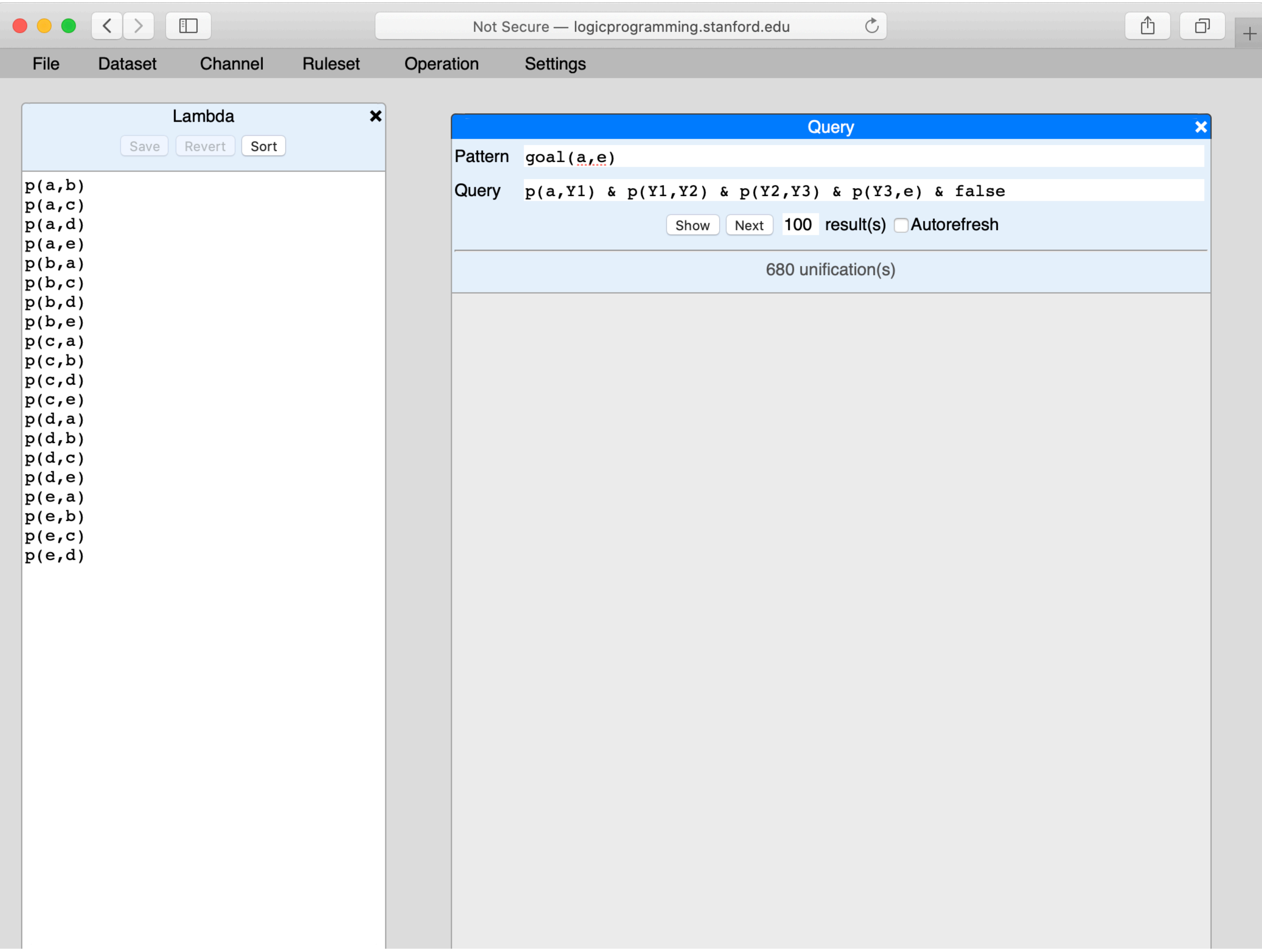
```
goal(a,e) :- r(a,N) & r(e,N)
```

Number of unifications for `goal(a,e)` (with indexing):

2

Number of factoids for *n* objects:

n



Not Secure — logicprogramming.stanford.edu

File Dataset Channel Ruleset Operation Settings

Lambda

Save Revert Sort

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

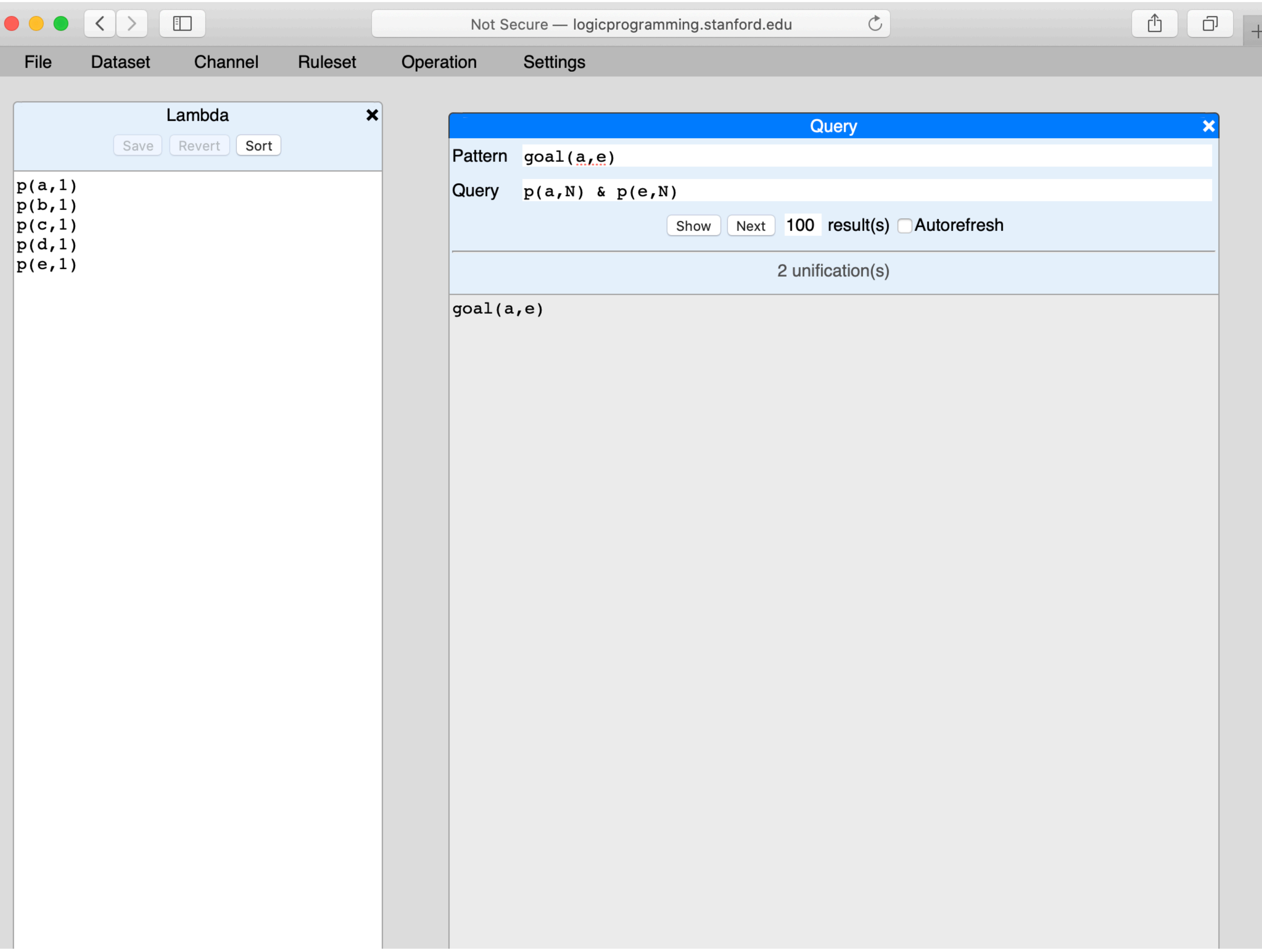
Query

Pattern `goal(a,e)`

Query `p(a,Y1) & p(Y1,Y2) & p(Y2,Y3) & p(Y3,e) & false`

Show Next 100 result(s) Autorefresh

680 unification(s)



```
p(a,1)
p(b,1)
p(c,1)
p(d,1)
p(e,1)
```

Pattern `goal(a,e)`

Query `p(a,N) & p(e,N)`

100 result(s)

2 unification(s)

```
goal(a,e)
```