Logic Programming
Query 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(a) :- p(a) & ~q(a)
goal(f(a)) :- p(f(a)) & ~q(f(a))
goal(f(f(a))) :- p(f(f(a))) & ~q(f(f(a)))
...
```

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

Programme

- Evaluation Procedure
  - Evaluating ground queries
  - Matching
  - Evaluating queries with variables
- Computational Analysis
  - Unindexed datasets
  - Indexing

Evaluating Ground Queries

Evaluation of Ground Queries

Given a query rule, call the procedure **eval** on the body. The result is a boolean. The result is the singleton set of the head if true; else, 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)`
Result: {`goal(a)`}

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

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)`
Result: `{}`

Dataset: `{p(a,b),p(a,c),p(b,c),p(c,d)}`
Query: `goal(b) :- ~p(c,b)`
Result: `{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 the answer to 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)`
Result: {`goal(c)`}

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

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)
```

Result: {`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*.

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

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.

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

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.

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

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

Here, $\{\texttt{X}\leftarrow\texttt{a}, \texttt{Y}\leftarrow\texttt{b}\}$ 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`$\leftarrow$`a`}
- **Compare:** `X`, `b`, {`X`$\leftarrow$`a`}
  - **Compare:** `a`, `b`, {`X`$\leftarrow$`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 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,Z)`

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

Value of query: {`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`).

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 Indexing

Example

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

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

$$n^2 + n*n^2 = n^2 + n^3$$

Suppose $n = 3$

$$3^2 + 3*3^2 = 3^2 + 3^3 = 36$$

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

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

Suppose $n = 3$

$$2*3^2 + 3 - 1 = 18 + 2 = 20$$

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 $n = 3$

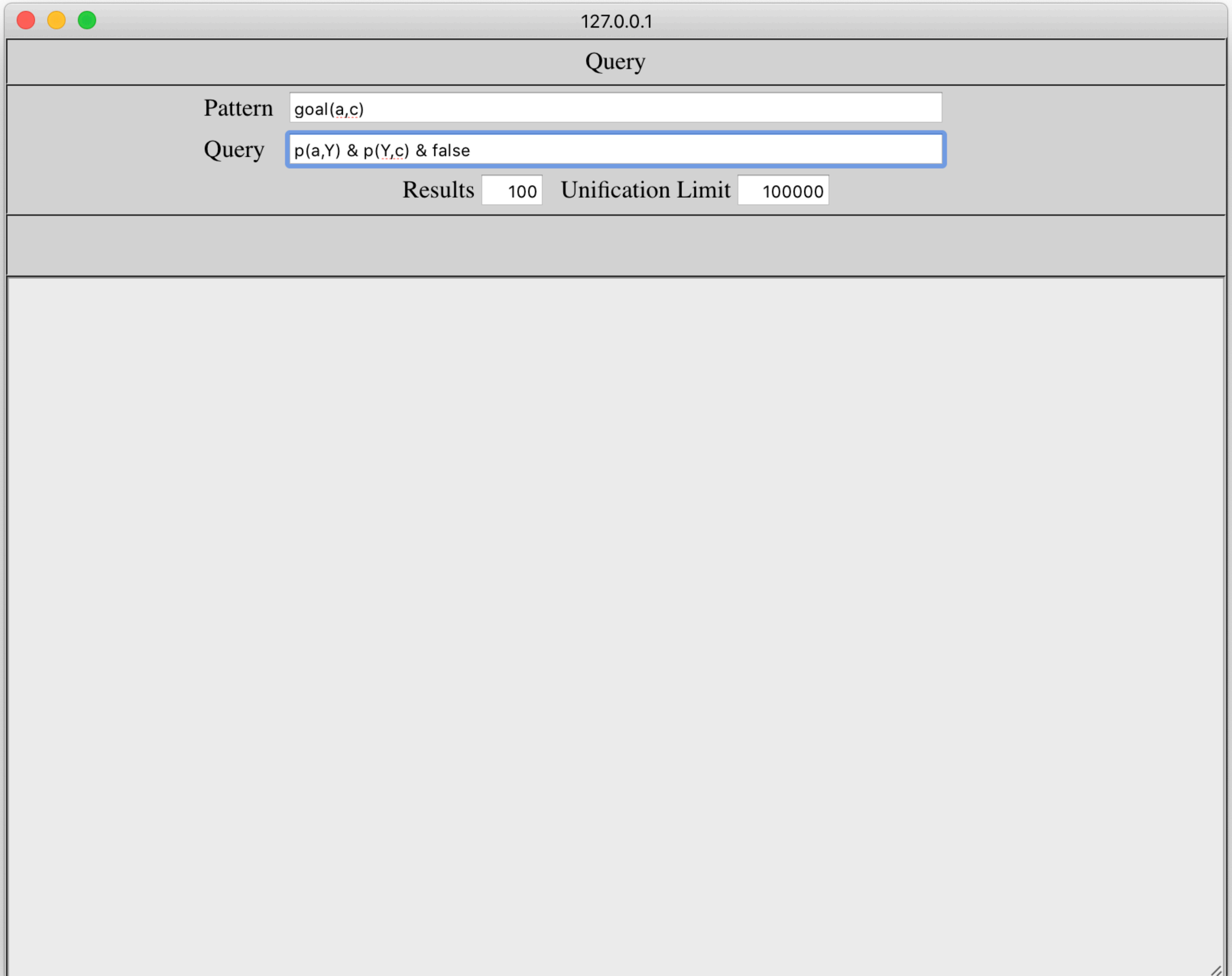
$$2*3^3 = 54$$

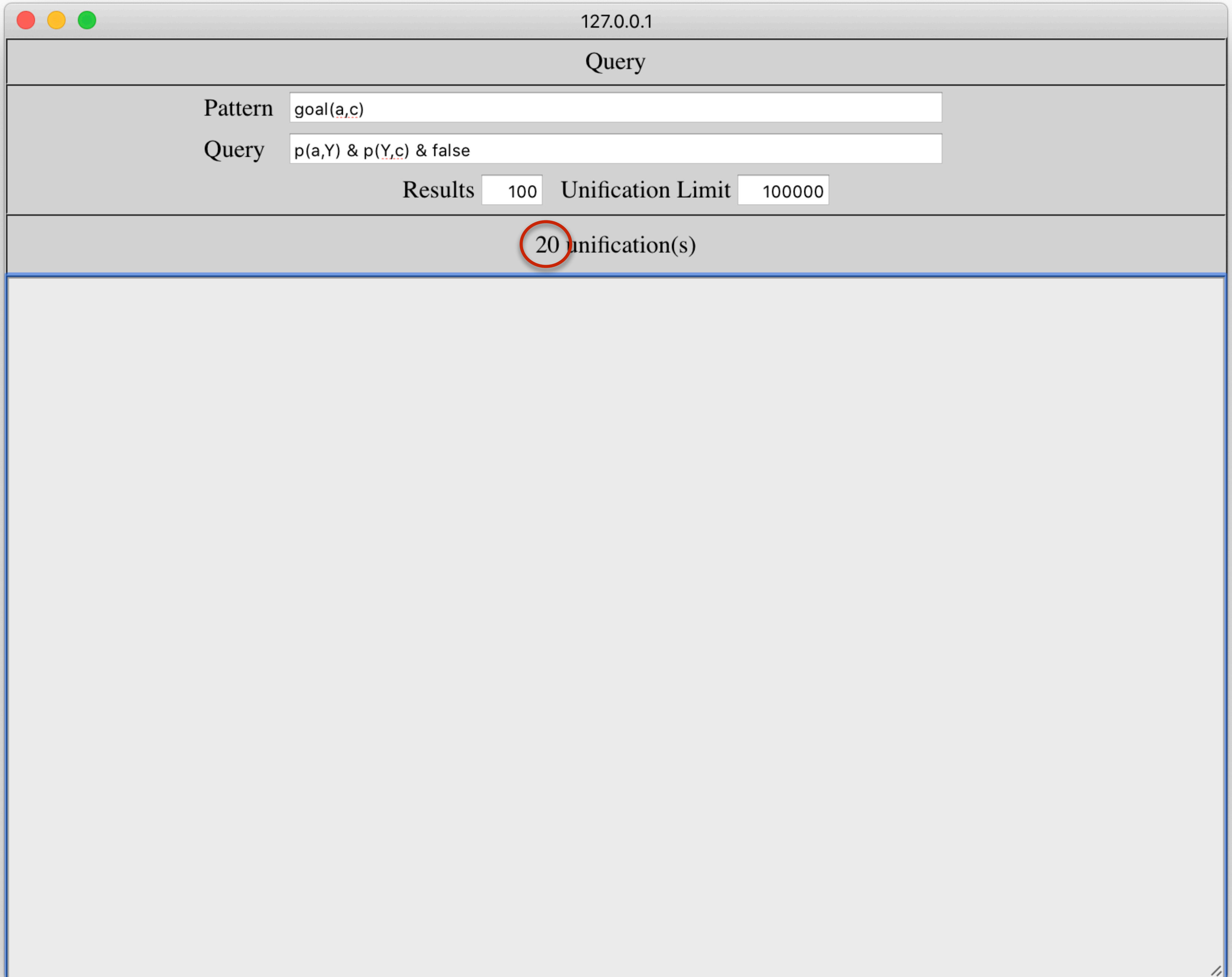
Not Secure — epilog.stanford.edu

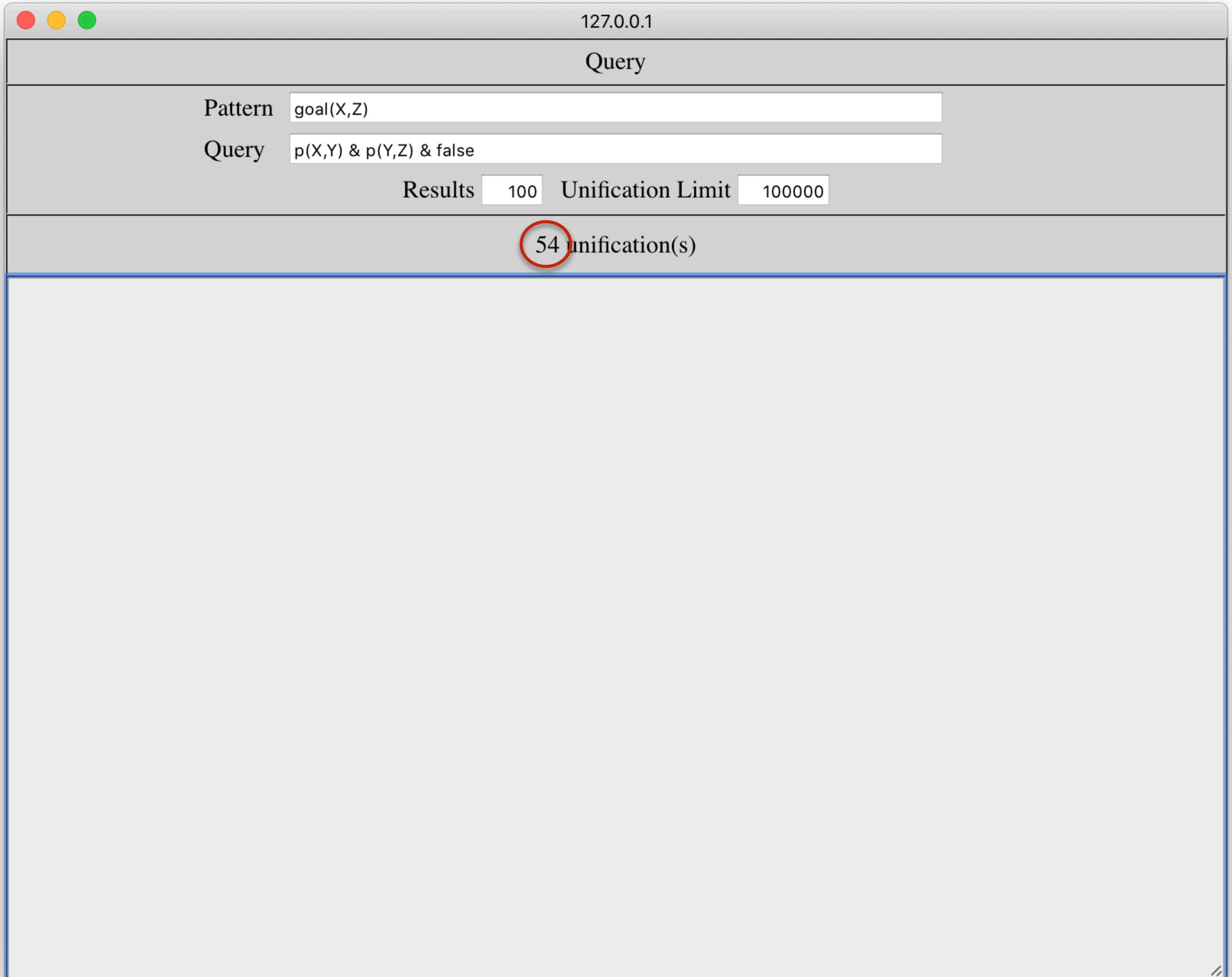
Lambda

Sort Update Revert Browse

```
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)
```


127.0.0.1
Query
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}

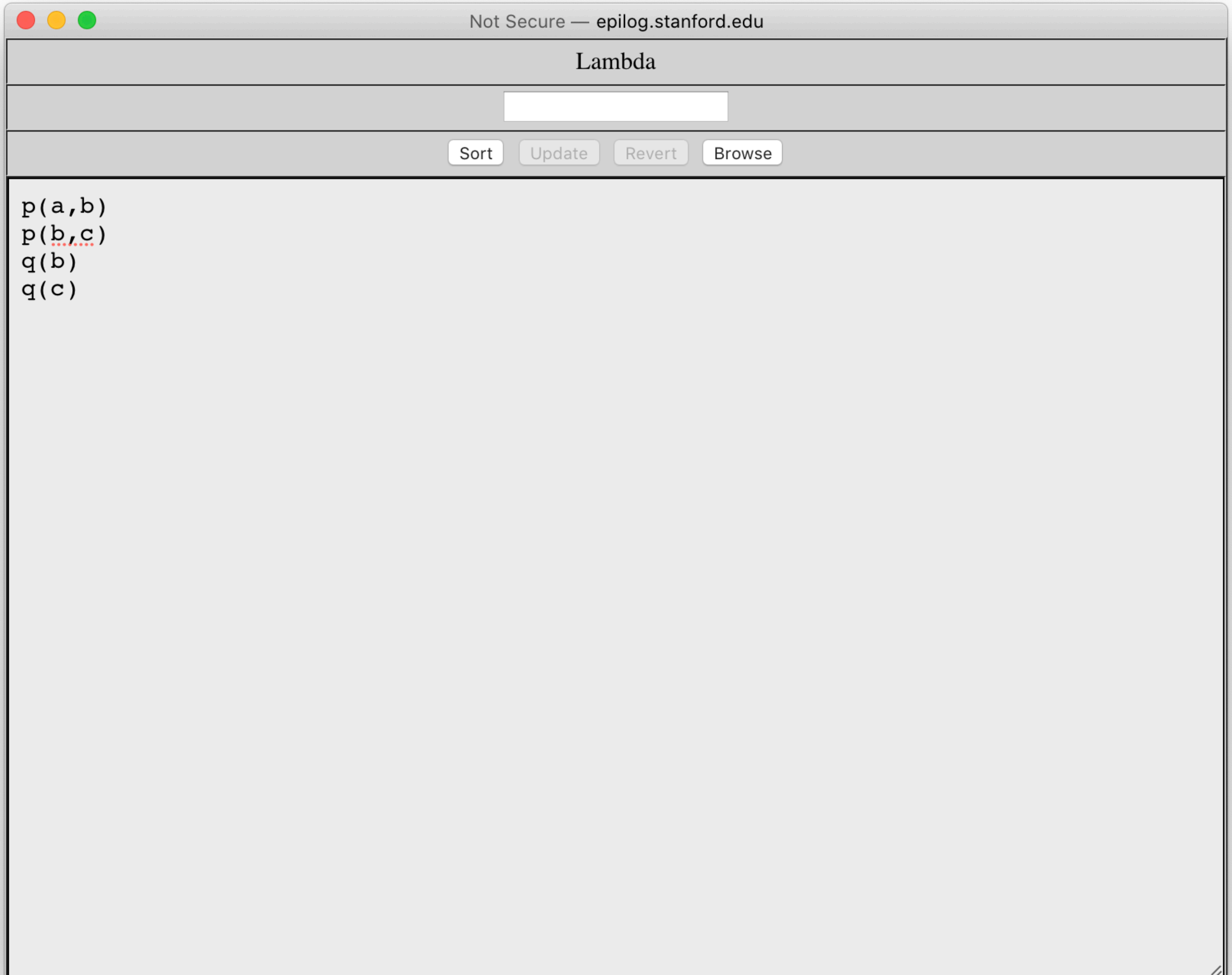
  Redo eval: p(X,Y),{}

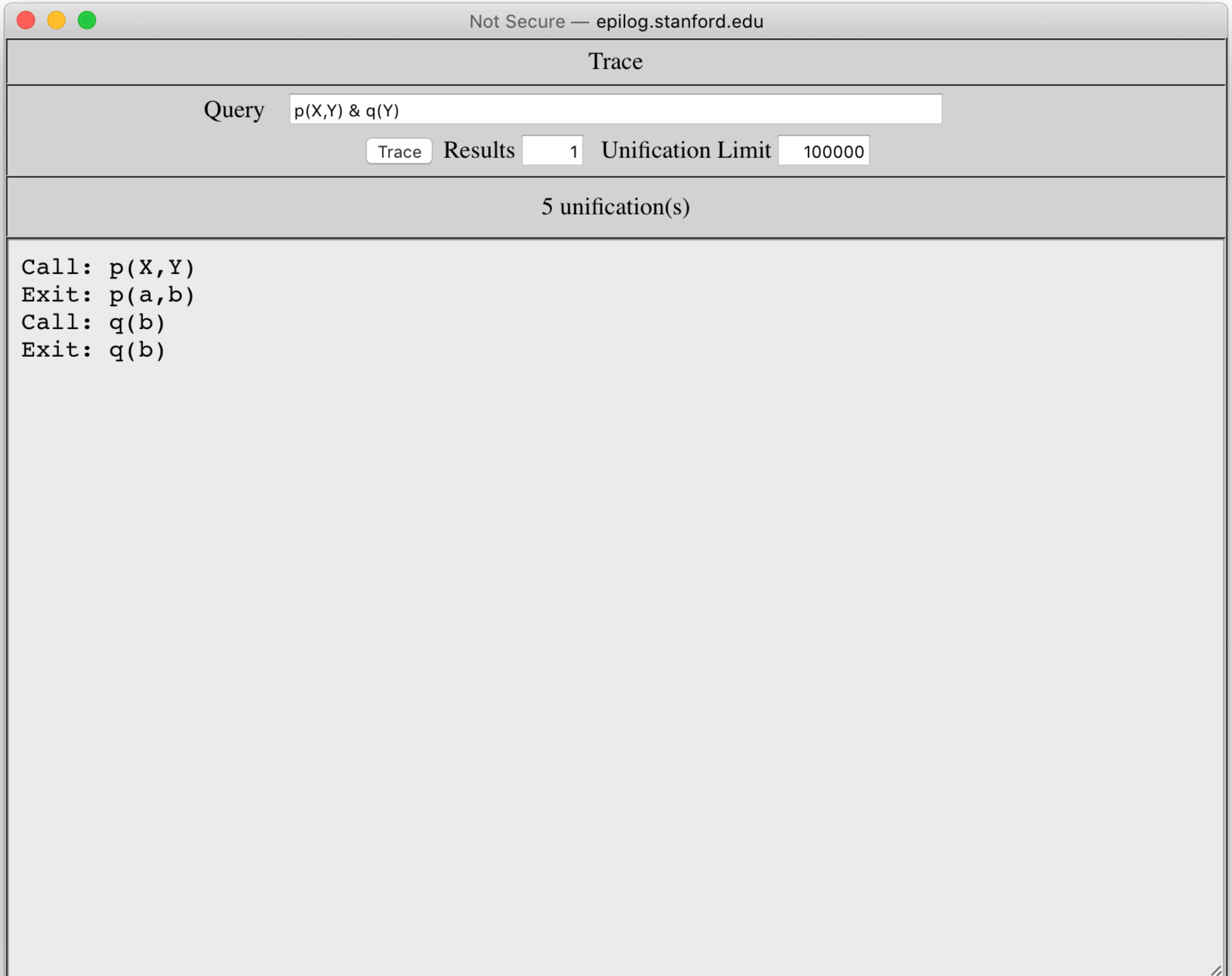
  Exit eval: {X←b,Y←c}

  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)


Not Secure — epilog.stanford.edu
Trace
Query
p(X,Y) & q(Y)
Trace
Results
1
Unification Limit
100000
5 unification(s)
Call: p(X,Y)
Exit: p(a,b)
Call: q(b)
Exit: q(b)

Not Secure — epilog.stanford.edu

Trace

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

Trace 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)
```

Not Secure — epilog.stanford.edu

Trace

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)
```