Logic Programming

Query Evaluation

Michael Genesereth
Computer Science Department
Stanford University
Dataset: \{p(b), p(c), p(d), q(d)\}

Rule:
\[
\text{goal}(X) \leftarrow p(X) \land \neg q(X)
\]

Instances:
\[
\begin{align*}
goal(a) & \leftarrow p(a) \land \neg q(a) \\
goal(b) & \leftarrow p(b) \land \neg q(b) \\
goal(c) & \leftarrow p(c) \land \neg q(c) \\
goal(d) & \leftarrow p(d) \land \neg q(d)
\end{align*}
\]

Result: \{goal(b), goal(c)\}
Dataset: \{ p(b), p(c), p(d), q(d) \}

Rule:

good(f(X)) :- p(X) & \neg q(X)

Instances:

good(a) :- p(a) & \neg q(a)
good(f(a)) :- p(f(a)) & \neg q(f(a))
good(f(f(a))) :- p(f(f(a))) & \neg 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.*
(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**: \{\}
(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:** {}
(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:** \{\text{p(a,b)}, \text{p(a,c)}, \text{p(b,c)}, \text{p(c,d)}\}
**Query:** \text{goal(c) :- p(c,d) & \neg p(d,c)}
**Result:** \{goal(c)\}

**Dataset:** \{\text{p(a,b)}, \text{p(a,c)}, \text{p(b,c)}, \text{p(c,d)}\}
**Query:** \text{goal(c) :- p(c,d) & p(d,c)}
**Result:** \{\}
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)\} \cup \{\} \cup \{goal(c)\}

{goal(a), goal(c)}
Matching
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.
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 \( \sigma \) to an expression \( \phi \) is the expression \( \phi \sigma \) obtained from \( \phi \) by replacing every occurrence of every variable with a binding in the substitution by the term to which it is bound.

\[
p(X, b)\{X \leftarrow a, Y \leftarrow b\} = p(a, b) \\
q(X, Y, X)\{X \leftarrow a, Y \leftarrow b\} = q(a, b, a)
\]
A substitution $\sigma$ is a *matcher* for a pattern and an instance if and only if applying the substitution to the pattern results in the given instance.

\[
p(X, b)\{X\leftarrow a, Y\leftarrow b\} = p(a, b)
\]

\[
q(X, Y, X)\{X\leftarrow a, Y\leftarrow b\} = q(a, b, a)
\]

Here, \{x\leftarrow a, y\leftarrow b\} is a matcher for $p(x, b)$ and $p(a, b)$. It is also a matcher for $q(x, y, x)$ and $q(a, b, a)$. 
(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: \( \{x\leftarrow a\} \)

Compare: \( x, a, \{x\leftarrow a\} \)
  
  Compare: \( a, a, \{x\leftarrow a\} \)
  
  Result: \( \{x\leftarrow a\} \)

Result: \( \{x\leftarrow a\} \)

Result: \( \{x\leftarrow a\} \)
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: \texttt{goal(Y) :- p(a,Y) \& p(Y,Z)}

Call eval: \quad p(a,Y) \& p(Y,Z), \{\}
Exit eval: \quad \{\{Y\leftarrow b, Z\leftarrow c\}, \{Y\leftarrow c, Z\leftarrow d\}\}

Value of query: \{\texttt{goal(b)}, \texttt{goal(c)}\}
(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)
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:** \(\sim p(Y,d), \{Y\leftarrow b\}\)
**Exit eval:** \{\{Y\leftarrow b\}\} (just one result)

**Call eval:** \(\sim p(Y,d), \{Y\leftarrow c\}\)
**Exit eval:** {} (no results)
(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:** \(p(a,Y) \land \neg p(Y,d), \{\}\)
  - **Call eval:** \(p(a,Y), \{\}\)
  - **Exit eval:** \(\{\{Y\leftarrow b\}, \{Y\leftarrow c\}\}\)

  **Call eval:** \(\neg p(Y,d), \{Y\leftarrow b\}\)
  - **Exit eval:** \(\{\{Y\leftarrow b\}\}\) (just one result)

  **Call eval:** \(\neg p(Y,d), \{Y\leftarrow c\}\)
  - **Exit eval:** \(\{\}\) (no results)

  **Exit eval:** \(\{\{Y\leftarrow b\}\}\) (just one result)
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:** \( \text{goal}(Y) :\neg p(a,Y) \land \neg p(Y,d) \)

**Call eval:** \( p(a,Y) \land \neg p(Y,d), \emptyset \)

**Exit eval:** \( \{Y\leftarrow b\} \)

**Answer:** \( \{\text{goal}(b)\} \)
Computational Analysis
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

g\text{oal}(a,c) \text{ :- } p(a,Y) \text{ } \& \text{ } p(Y,c)

Cost of computing whether \textit{g}o\texttt{a}l\texttt{(}a,c\texttt{)} is true
Example

\[ \text{goal}(a, c) \ :- \ p(a, Y) \ & \ p(Y, c) \]

Cost of computing whether \( \text{goal}(a, c) \) is true

\[ n^2 + n \cdot n^2 = n^2 + n^3 \]

Suppose \( n = 3 \)

\[ 3^2 + 3 \cdot 3^2 = 3^2 + 3^3 = 36 \]
Example

Example

\[
\text{goal}(X,Z) :- \ p(X,Y) \ & \ p(Y,Z)
\]

Cost of computing *all* instances of \text{goal}
Example

\[ \text{goal}(X, Z) :- \ p(X, Y) \ & \ p(Y, Z) \]

Cost of computing *all* instances of goal

\[ n^2 + n^2 \cdot n^2 = n^2 + n^4 \]

Suppose \( n = 3 \)

\[ 3^2 + 3^2 \cdot 3^2 = 3^2 + 3^4 = 90 \]
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 \(b\)).
Example:

<table>
<thead>
<tr>
<th></th>
<th>p(a,a)</th>
<th>p(b,a)</th>
<th>p(c,a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a,b)</td>
<td></td>
<td>p(b,b)</td>
<td></td>
</tr>
<tr>
<td>p(a,c)</td>
<td></td>
<td>p(b,c)</td>
<td></td>
</tr>
<tr>
<td>p(a,b)</td>
<td></td>
<td></td>
<td>p(c,b)</td>
</tr>
<tr>
<td>p(b,c)</td>
<td></td>
<td></td>
<td>p(c,c)</td>
</tr>
</tbody>
</table>

Index on p: \{p(a,a), \ldots, 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

\[ \text{goal}(a,c) := p(a,Y) \land p(Y,c) \]

Cost of computing \text{goal}(a,c) \textit{ without indexing}\n
\[ n^2 + n \cdot n^2 = n^2 + n^3 \]

Suppose \( n = 3 \)

\[ 3^2 + 3 \cdot 3^2 = 3^2 + 3^3 = 36 \]

Cost of computing whether \text{goal}(a,c) \textit{ with indexing} \n
\[ (2n-1) + n \cdot (2n-1) = 2n^2 + n - 1 \]

Suppose \( n = 3 \)

\[ 2 \cdot 3^2 + 3 - 1 = 18 + 2 = 20 \]
Example

g\text{oal}(X,Z) :- p(X,Y) \& p(Y,Z)

Cost of computing \text{goal}(X,Z) \textit{without} indexing:
\[ n^2 + n^2 \times n^2 = n^2 + n^4 \]

Suppose \( n = 3 \)
\[ 3^2 + 3^2 \times 3^2 = 3^2 + 3^4 = 90 \]

Cost of computing \textit{all} instances \textit{with} indexing:
\[ n^2 + n^2 \times (2n-1) = n^2 + 2n^3 - n^2 = 2n^3 \]

Suppose \( n = 3 \)
\[ 2 \times 3^3 = 54 \]
\( p(a,a) \)
\( p(a,b) \)
\( p(a,c) \)
\( p(b,a) \)
\( p(b,b) \)
\( p(b,c) \)
\( p(c,a) \)
\( p(d,b) \)
\( p(c,c) \)
<table>
<thead>
<tr>
<th>Pattern</th>
<th>goal(a,c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td><code>p(a,Y) &amp; p(Y,c) &amp; false</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Results</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unification Limit</td>
<td>100000</td>
</tr>
<tr>
<td>Pattern</td>
<td>goal(a,c)</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Query</td>
<td>p(a,Y) &amp; p(Y,c) &amp; false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Results</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unification Limit</td>
<td>100000</td>
</tr>
</tbody>
</table>

20 unification(s)
<table>
<thead>
<tr>
<th>Pattern</th>
<th>goal(X,Z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>p(X,Y) &amp; p(Y,Z) &amp; false</td>
</tr>
<tr>
<td>Results</td>
<td>100</td>
</tr>
<tr>
<td>Unification Limit</td>
<td>100000</td>
</tr>
</tbody>
</table>

54 unification(s)
Pipelining
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.
Dataset: \{p(a,b), p(b,c), q(b), q(c)\}

Call eval: \(p(X,Y) \land q(Y), \emptyset\)
  Call eval: \(p(X,Y), \emptyset\)
  Exit eval: \\{\{X\leftarrow a, Y\leftarrow b\}, \{X\leftarrow b, Y\leftarrow c\}\}\}

Call eval: \(q(Y), \{X\leftarrow a, Y\leftarrow b\}\)
Exit eval: \\{\{X\leftarrow a, Y\leftarrow b\}\}\}
Call eval: \(q(Y), \{X\leftarrow b, Y\leftarrow c\}\)
Exit eval: \\{\{X\leftarrow b, Y\leftarrow c\}\}\}

Exit eval: \\{\{X\leftarrow a, Y\leftarrow b\}, \{X\leftarrow b, Y\leftarrow 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\}\}
\[ p(a, b) \]
\[ p(b, c) \]
\[ q(b) \]
\[ q(c) \]
Query: \( p(X,Y) \land q(Y) \)

Results: 1  Unification Limit: 100000

5 unification(s)

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

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(X,Y)
Exit: p(b,c)
Call: q(c)
Exit: q(c)
Query: \( p(X,Y) \) \& q(Y) \\

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: \( p(X,Y) \)