Evaluation Methods

**Bottom-Up Evaluation**
- Starts with dataset
- Applies rules to produce closure
- Repeat up the stratum hierarchy
- Check whether query is in the resulting dataset

**Top-Down Evaluation**
- Starts with query to be answered
- Applies rules to reduce to subqueries
- Continues until reaches data level
- Match base level subgoal against dataset
Disadvantages of Bottom-Up
   Generates large numbers of irrelevant conclusions
   Does not work with infinite extensions

Disadvantages of Top-Down
   Slightly harder to understand
   Sometimes recomputes subgoals
   Susceptible to avoidable infinite loops
Top-Down Processing of Ground Goals and Rules

Unification

Top-Down Processing of Goals and Rules with Variables
Ground Goals and Rules
Given a query, a dataset, and a ruleset, do the following.

(1) If the predicate in the query is a **base predicate**, succeed if and only if query is in dataset.

(2) If the query is a **negation**, evaluate target and succeed if and only if fail to prove.

(3) If the query is a **conjunction**, succeed iff succeed on all conjuncts.

(4) If the predicate in the query is a **view predicate**, evaluate the body of each rule defining that predicate and succeed if and only if succeed on at least one rule.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ruleset</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a)</td>
<td>s(c) :- p(a) &amp; q(b)</td>
</tr>
<tr>
<td>p(b)</td>
<td>s(c) :- p(b) &amp; t(c)</td>
</tr>
<tr>
<td>p(c)</td>
<td>s(c) :- p(c) &amp; ~q(c)</td>
</tr>
<tr>
<td>q(d)</td>
<td>t(c) :- p(a) &amp; p(d)</td>
</tr>
</tbody>
</table>
Dataset

- \( p(a) \)
- \( p(b) \)
- \( p(c) \)
- \( q(d) \)

Ruleset

- \( s(c) \) :- \( p(a) \) & \( q(b) \)
- \( s(c) \) :- \( p(b) \) & \( t(c) \)
- \( s(c) \) :- \( p(c) \) & \( \neg q(c) \)
- \( t(c) \) :- \( p(a) \) & \( p(d) \)

Top Down Evaluation

```
\( p(a) \) & \( q(b) \)?  \( p(b) \) & \( t(c) \)?  \( p(c) \) & \( \neg q(c) \)?
\( X \)                   \( X \)                   \( V \)
```

Example
Unification
Unification is the process of determining whether two expressions can be unified, i.e. made identical by appropriate substitutions for their variables.

Example: \( p(a, y) \) and \( p(x, b) \) can be unified. If we replace \( x \) by \( a \) and \( y \) by \( b \), we end up with \( p(a, b) \) in both cases.
A substitution is a finite set of pairs of variables and terms, called replacements.

\[\{x \leftarrow a, y \leftarrow f(b), z \leftarrow V\}\]

Domain: \(\{x, y, z\}\)
Range: \(\{a, f(b), V\}\)
The result of applying a substitution $\sigma$ to an expression $\varphi$ is the expression $\varphi\sigma$ obtained from $\varphi$ by replacing every occurrence of every variable in the substitution by its replacement.

$q(X,Y) \{ X \leftarrow a, Y \leftarrow f(b), Z \leftarrow V \} = q(a,f(b))$
$q(X,X) \{ X \leftarrow a, Y \leftarrow f(b), Z \leftarrow V \} = q(a,a)$
$q(X,W) \{ X \leftarrow a, Y \leftarrow f(b), Z \leftarrow V \} = q(a,W)$
$q(Z,V) \{ X \leftarrow a, Y \leftarrow f(b), Z \leftarrow V \} = q(V,V)$
Cascaded Substitutions

\[ r\{x,y,z\}\{x\leftarrow a, y\leftarrow f(u), z\leftarrow v\} = r\{a,f(u),v\} \]
\[ r\{a,f(u),v\}\{u\leftarrow d, v\leftarrow e, z\leftarrow g\} = r(a,f(d),e) \]
\[ r\{x,y,z\}\{x\leftarrow a, y\leftarrow f(d), z\leftarrow e, u\leftarrow d, v\leftarrow e\} = r(a,f(d),e) \]
Composition of Substitutions

The *composition* of substitution $\sigma$ and $\tau$ is the substitution (written \(\text{compose}(\sigma,\tau)\) or, more simply, $\sigma\tau$) obtained by

1. applying $\tau$ to the replacements in $\sigma$
2. adding to $\sigma$ pairs from $\tau$ with different variables
3. deleting any assignments of a variable to itself.

\[
\{X \leftarrow a, Y \leftarrow U, Z \leftarrow V\}\{U \leftarrow d, V \leftarrow e, Z \leftarrow g\} \\
= \{X \leftarrow a, Y \leftarrow d, Z \leftarrow e\}\{U \leftarrow d, V \leftarrow e, Z \leftarrow g\} \\
= \{X \leftarrow a, Y \leftarrow d, Z \leftarrow e, U \leftarrow d, V \leftarrow e\}
\]
A substitution $\sigma$ is a unifier for an expression $\varphi$ and an expression $\psi$ if and only if $\varphi\sigma=\psi\sigma$.

\[
p(x,y)\{x\leftarrow a,y\leftarrow b,v\leftarrow b\}=p(a,b)
p(a,v)\{x\leftarrow a,y\leftarrow b,v\leftarrow b\}=p(a,b)
\]

If two expressions have a unifier, they are said to be unifiable. Otherwise, they are nonunifiable.

\[
p(x,x)
p(a,b)
\]
NonUniqueness of Unification

Unifier 1:
\[
p(x,y)\{x\leftarrow a, y\leftarrow b, v\leftarrow b\} = p(a, b)
\]
\[
p(a,v)\{x\leftarrow a, y\leftarrow b, v\leftarrow b\} = p(a, b)
\]

Unifier 2:
\[
p(x,y)\{x\leftarrow a, y\leftarrow f(w), v\leftarrow f(w)\} = p(a, f(w))
\]
\[
p(a,v)\{x\leftarrow a, y\leftarrow f(w), v\leftarrow f(w)\} = p(a, f(w))
\]

Unifier 3:
\[
p(x,y)\{x\leftarrow a, y\leftarrow v\} = p(a, v)
\]
\[
p(a,v)\{x\leftarrow a, y\leftarrow v\} = p(a, v)
\]
A substitution $\sigma$ is a *most general unifier* (mgu) of two expressions if and only if it is as general as or more general than any other unifier.

Theorem: If two expressions are unifiable, then they have an mgu that is unique up to variable permutation.

\[
p(x, y)\{x \leftarrow a, y \leftarrow v\} = p(a, v)
\]
\[
p(a, v)\{x \leftarrow a, y \leftarrow v\} = p(a, v)
\]

\[
p(x, y)\{x \leftarrow a, v \leftarrow y\} = p(a, y)
\]
\[
p(a, v)\{x \leftarrow a, v \leftarrow y\} = p(a, y)
\]
One good thing about our language is that there is a simple and inexpensive procedure for computing a most general unifier of any two expressions if it exists.
Each expression is treated as a sequence of its immediate subexpressions.

Linear Version:
\[ p(a, f(b, c), d) \]

Structured Version:
(1) If two expressions being compared are identical, succeed.

(2) If neither is a variable and at least one is a constant, fail.

(3) If one of the expressions is a variable, proceed as described shortly.

(4) If both expressions are sequences, iterate across the expressions, comparing as described above.
If one of the expressions is a variable, check whether the variable has a binding in the current substitution.

(a) If so, try to unify the binding with the other expression.

(b) If no binding, check whether the other expression contains the variable. If the variable occurs within the expression, fail; otherwise, set the substitution to the composition of the old substitution and a new substitution in which variable is bound to the other expression.
Example

Call: \( p(X, b), p(a, Y), \{ \} \)

Call: \( p, p, \{ \} \)
Exit: \( \{ \} \)

Call: \( X, a, \{ \} \)
Exit: \( \{ \} \{ x \leftarrow a \} = \{ x \leftarrow a \} \)

Call: \( b, Y, \{ x \leftarrow a \} \)
Exit: \( \{ x \leftarrow a \} \{ y \leftarrow b \} = \{ x \leftarrow a, y \leftarrow b \} \)

Exit: \( \{ x \leftarrow a, y \leftarrow b \} \)
Call: \( p(X, X), p(a, Y), {} \)

Call: \( p, p, {} \)
Exit: {} 

Call: \( X, a, {} \)
Exit: \( \{ x \leftarrow a \} = \{ x \leftarrow a \} \)

Call: \( X, Y, \{ x \leftarrow a \} \)
  Call: \( a, Y, \{ x \leftarrow a \} \)
  Exit: \( \{ x \leftarrow a \} \{ y \leftarrow a \} = \{ x \leftarrow a, y \leftarrow a \} \)
Exit: \( \{ x \leftarrow a, y \leftarrow a \} \)

Exit: \( \{ x \leftarrow a, y \leftarrow a \} \)
Example

Call: \( p(x, x), p(a, b), \{\} \)

Call: \( p, p, \{\} \)
Exit: \( \{\} \)

Call: \( x, a, \{\} \)
Exit: \( \{\} \{x \leftarrow a\} = \{x \leftarrow a\} \)

Call: \( x, b, \{x \leftarrow a\} \)
   Call: \( a, b, \{x \leftarrow a\} \)
   Exit: \( false \)
Exit: \( false \)

Exit: \( false \)
Example

Call: \( p(X, X), p(Y, f(Y)), {} \)

Call: \( p, p, {} \)
Exit: \( {} \)

Call: \( X, Y, {} \)
Exit: \( {}\{X \leftarrow Y\} = \{X \leftarrow Y\} \)

Call: \( X, f(Y), \{X \leftarrow Y\} \)
   Call: \( Y, f(Y), \{X \leftarrow Y\} \)
   Exit: \( false \)
Exit: \( false \)

Exit: \( false \)
Reason

Circularity Problem:
\{X \leftarrow f(Y), Y \leftarrow f(Y)\}

Unification Problem:
\[
p(X, X)\{X \leftarrow f(Y), Y \leftarrow f(Y)\} = p(f(Y), f(Y))
p(Y, f(Y))\{X \leftarrow f(Y), Y \leftarrow f(Y)\} = p(f(Y), f(f(Y)) )
\]

Before assigning a variable to an expression, first check that the variable does not occur within that expression.

This is called, oddly enough, the *occur check* test.

Prolog does not do the occur check (and is proud of it).
General Goals and Rules
Procedure without variables uses *equality* tests.

\[
p(a,b) \\
p(b,c) \\
s(a,c) :- p(a,b) \& p(b,c)
\]

\[s(a,c)\]?

Procedure with variables uses *unification*.

\[
p(a,b) \\
p(b,c) \\
s(X,Z) :- p(X,Y) \& p(Y,Z)
\]

\[s(a,c)\]?
Given an atom with a base relation and a substitution:

(a) Compare the goal to each factoid in our dataset.

(b) If there is an extension of the given substitution that unifies the goal and the factoid, add to our list of answers.

(c) Once all relevant factoids examined, return answers.
Example 1 - Atoms with Base Relations

Goal: $p(X, Y)$
Substitution: $\{X\leftarrow a\}$
Dataset: $\{p(a, b), p(a, c), p(b, c)\}$

Result: $[\{X\leftarrow a, Y\leftarrow b\}, \{X\leftarrow a, Y\leftarrow c\}]$
Given a negation and a substitution:

(a) Execute the procedure on the argument of the negation and the given substitution.

(b) If the result is empty, return a singleton list containing the given substitution, indicating success.

(c) Otherwise, return the empty list, indicating failure.
Example 2 - Negations

Goal: \( \sim p(X,Y) \)
Substitution: \( \{X \leftarrow a, Y \leftarrow d\} \)
Dataset: \( \{p(a,b), p(a,c), p(b,c)\} \)
Result: \( [{\{X \leftarrow a, Y \leftarrow d\}}] \)

Goal: \( \sim p(X,Y) \)
Substitution: \( \{X \leftarrow a, Y \leftarrow c\} \)
Dataset: \( \{p(a,b), p(a,c), p(b,c)\} \)
Result: \( [] \)
Step 3 - Conjunctions

Given a negation and a substitution:

(a) Execute our procedure on the first conjunct and the given substitution to get a list of answers.

(b) Iterate through the list of substitutions, calling the procedure recursively on the remaining conjuncts with each substitution in turn.

(c) Collect the answers from recursive calls and return.
Example 3 - Conjunctions

Goal: \( p(X, Y) \ & \ p(Y, Z) \)
Substitution: \( \{X \leftarrow a\} \)
Dataset: \( \{p(a,b), p(a,c), p(b,c)\} \)

Call: \( p(X, Y), \{X \leftarrow a\} \)
Result: \( \{\{X \leftarrow a, Y \leftarrow b\},\{X \leftarrow a, Y \leftarrow c\}\} \)

Call: \( p(Y, Z), \{X \leftarrow a, Y \leftarrow b\} \)
Result: \( \{\{X \leftarrow a, Y \leftarrow b, Z \leftarrow c\}\} \)

Call: \( p(Y, Z), \{X \leftarrow a, Y \leftarrow c\} \)
Result: \( p(Y, Z) : [] \)

Overall Result: \( \{\{X \leftarrow a, Y \leftarrow b, Z \leftarrow c\}\} \)
Given atom with view relation and a substitution:
(a) Iterate through the rules in our program.
(b) Copy each rule, replacing variables with new variables.
(c) Try to unify the given goal and the new rule head.
(d) Call the procedure recursively on the body of the rule.
(e) Return substitutions from all successful cases.
Example 4 - Atoms with View Relations

Goal: $q(X, Y)$
Substitution: $\{X \leftarrow a\}$
Rule: $q(X, Z) :- p(X, Y) \& p(Y, Z)$
Dataset: $\{p(a, b), p(a, c), p(b, c)\}$

Copy of rule: $q(U, W) :- p(U, V) \& p(V, W)$
Unification: $q(U, W) \leftarrow q(X, Y) \leftarrow \{X \leftarrow a\}$
Result: $\{U \leftarrow a, W \leftarrow Y, X \leftarrow a\}$

New Goal: $\{p(U, V) \& p(V, W)\}$
New Substitution: $\{U \leftarrow a, W \leftarrow Y, X \leftarrow a\}$

Result: $[\{U \leftarrow a, W \leftarrow c, X \leftarrow a, Y \leftarrow c\}]$
Facts and Rules

\[ \begin{align*}
  p(a,b) \\
  p(b,c) \\
  s(X,Z) :&= p(X,Y) & p(Y,Z) 
\end{align*} \]

Trace

Call: \( s(X,Z) \)
| \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} Call: \ p(X,Y) \\
| \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} Exit: \ p(a,b) \\
| \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} Call: \ p(b,Z) \\
| \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} Exit: \ p(b,c) \\
Exit: \ s(a,c)
Facts and Rules
   p(a,b)
   p(b,c)
   s(X,Z) :- p(X,Y) & p(Y,Z)

Trace
Call: s(X,Z)          Redo: s(X,Z)
   | Call: p(X,Y)          | Redo: p(b,Z)
   | Exit: p(a,b)          | Fail: p(b,Z)
   | Call: p(b,Z)          | Redo: p(X,Y)
   | Exit: p(b,c)          | Exit: p(b,c)
Exit: s(a,c)          Exit: s(a,c)
   | Fail: p(c,Z)          | Fail: p(c,Z)
   | Fail: s(X,Z)          | Fail: s(X,Z)
Summary
Comparison of Evaluation Strategies

**Bottom-Up Evaluation**
- Easy to understand
- Computes all results
- Computes subresults just once
- Wasteful when want just one or a few answers, not all
- Problematic on logic programs with infinite models

**Top-Down Evaluation**
- Less waste when want one or a few answers
- More complicated
- Subqueries evaluated multiple times
- Possibility of infinite loops on programs with finite models
Bottom-Up Evaluation
Can be focussed using Magic Sets

Top-Down Evaluation
Top-Down can avoid duplication through caching
Infinite Loops can be avoided using iterative deepening

*The arms race continues.*