Tabled Resolution

CSE 595 – Semantic Web
Instructor: Dr. Paul Fodor
Stony Brook University

http://www3.cs.stonybrook.edu/~pfodor/courses/cse595.html
Recap: OLD Resolution

- Prolog follows OLD resolution = SLD with left-to-right literal selection.
- Prolog searches for OLD proofs by expanding the resolution tree depth first.
  - This depth-first expansion is close to how procedural programs are evaluated:
    - Consider a goal $G_1, G_2, \ldots, G_n$ as a “procedure stack” with $G_1$, the selected literal on top.
    - Call $G_1$.
    - **If** and **when** $G_1$ returns, continue with the rest of the computation: call $G_2$, and upon its return call $G_3$, etc. until nothing is left.
      - Note: $G_2$ is “opened up” only when $G_1$ returns, not after executing only some part of $G_1$. 
OLD Resolution

• Depth-first expansion, however, contributes to the *incompleteness* of Prolog’s evaluation, which may not terminate even when the least model is finite (see the next example!)
Example: Reachability in Directed Graphs

- Determining whether there is a path between two vertices in a directed graph is an important and widespread problem.
- For instance, consider checking whether (or not) a program accesses a shared resource before obtaining a lock:
  - A program itself can be considered as a graph with vertices representing program states!
    - A state may be characterized by the program counter value, and values of variables.
    - There are richer models for representing program evaluation, but a directed graph is most basic.
  - If we can go from state \( s \) by executing one instruction to \( s' \), then we can place an edge from \( s \) to \( s' \).
  - The reachability question may be whether we can reach from the start state to a state accessing a shared resource, without going through a state that obtained a lock.
Graph Reachability as a Logic Program

- A finite directed graph can be represented by a set of binary facts representing an "edge" relation
  - Predicate “q” on right is an example
- Reachability can then be written as a "transitive closure" over the edge relation
  - Observe the predicate “r” defined on right using two clauses:
    - The first clause: there is a path from X to Y if there is an edge from X to Y
    - The second clause: there is a path from X to Y if there is an intermediate vertex Z such that:
      - there is an edge from X to Z, and
      - there is a path from Z to Y

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :- q(X,Z), r(Z,Y).
```
Bottom-Up Evaluation

- Note that the program on the left is a Datalog program: no function symbols
- Its Herbrand Universe is finite, and its least model computation using the bottom up evaluation will terminate:

\[
\begin{align*}
M_0 &= \emptyset \\
M_1 &= T_p(M_0) = M_0 \cup \{q(a, a), q(a, b), q(a, c), q(b, b), q(b, d), q(b, e), q(e, d)\} \\
M_2 &= T_p(M_1) = M_1 \cup \{r(a, a), r(a, b), r(a, c), r(b, b), r(b, d), r(b, e), r(e, d)\} \\
M_3 &= T_p(M_2) = M_2 \cup \{r(a, d), r(a, e)\} \\
M_4 &= T_p(M_3) = M_3
\end{align*}
\]
Bottom-Up Evaluation

\[ M_4 = \{ q(a,a), q(a,b), q(a,c), q(b,b), q(b,d), q(b,e), q(e,d), r(a,a), r(a,b), r(a,c), r(b,b), r(b,d), r(b,e), r(e,d), r(a,d), r(a,e) \} \]

- With care, using bottom-up evaluation all-pairs reachability can be computed in \( O(V \cdot E) \) time for a graph with \( V \) vertices and \( E \) edges.

\begin{align*}
q(a, a) \\
q(a, b) \\
q(a, c) \\
q(b, b) \\
q(b, d) \\
q(b, e) \\
q(e, d) \\
r(a, a) \\
r(a, b) \\
r(a, c) \\
r(b, b) \\
r(b, d) \\
r(b, e) \\
r(e, d) \\
r(a, d) \\
r(a, e) \\
r(X, Y) &\leftarrow q(X,Y). \\
r(X, Y) &\leftarrow q(X,Z), r(Z,Y).
\end{align*}
OLD Resolution with Depth-First Expansion

?- r(a,N).

Consider initial query "?- r(a,N)."
Let’s construct the **SLD tree** for this query

```
?-(a,N).
```

- **q(a, a).**
- **q(a, b).**
- **q(a, c).**
- **q(b, b).**
- **q(b, d).**
- **q(b, e).**
- **q(e, d).**

```
r(X,Y) :- q(X,Y).
r(X,Y) :- q(X,Z), r(Z,Y).
```
OLD Resolution with Depth-First Expansion

?- r(a,N).

?- q(a,N).

Resolving this goal with the first clause of \( r \), we get a new goal "?- q(a,N)."
OLD Resolution with Depth-First Expansion

?- r(a,N).

?- q(a,N).

\{N=a\} \{N=b\} \{N=c\}

Resolving "?- q(a,N)." results in the empty goal, under three answer substitutions: \(a\), \(b\), and \(c\). There are no more ways to resolve "?- q(a,N)."

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d). \\
r(X,Y) &:- q(X,Y). \\
r(X,Y) &:- q(X,Z), r(Z,Y).
\end{align*}
\]
OLD Resolution with Depth-First Expansion

(?- r(a,N).

?- q(a,N).  ?- q(a,Z), r(Z,N).

\{N=a\} \{N=b\} \{N=c\}

Resolving "?- r(a,N)." with the second clause of r results in goal "?- q(a,Z), r(Z,N)."

Note: We will use same variable names as in the program clause when possible, instead of mechanically inventing new variable names in every step.
The selected literal is \( q(a, Z) \) which unifies with fact \( q(a, a) \) with \( Z=a \). Thus we get the goal "?\(-r(a, N)\)."

Note: This is the same goal that we had at the beginning.
OLD Resolution with Depth-First Expansion

Resolving "?- r(a,N)" leads to "?- q(a,N)" (one of two options).

?- q(a,N).

?- q(a,N).  ?- q(a,Z), r(Z,N).

?- q(a,N).  ?- q(a,Z), r(Z,N).

?- r(a,N).

{N=a} {N=b} {N=c}  ?- r(a,N).

? q(a,N).

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).

r(X,Y) :-
          q(X,Z), r(Z,Y).
OLD Resolution with Depth-First Expansion

?- r(a,N).

?- q(a,N).  
?- q(a,Z), r(Z,N).

{N=a} {N=b} {N=c}  
?- r(a,N).  

?- q(a,N).

{N=a} {N=b} {N=c}  

"?- q(a,N) . ", in turn, leads to empty goal with answers N=a, b, and c.

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d). \\
r(x, y) :- q(x, y). \\
r(x, y) :- q(x, z), r(z, y).
\end{align*}
\]
OLD Resolution with Depth-First Expansion

\[ \text{?- } r(a, N). \]

The other way to resolve "?-r(a, N)." is …

You get the drift: we are repeating work done before, and there is an infinite branch here.

\[ q(a, a). \]
\[ q(a, b). \]
\[ q(a, c). \]
\[ q(b, b). \]
\[ q(b, d). \]
\[ q(b, e). \]
\[ q(e, d). \]

\[ r(X,Y) :- q(X,Y). \]
\[ r(X,Y) :- q(X,Z), r(Z,Y). \]
OLD Resolution with Depth-First Expansion

?- r(a,N).

?-q(a,N).  
?-q(a,Z), r(Z,N).

{N=a} {N=b} {N=c}

?-q(a,N).  
?-q(a,Z), r(Z,N).

{N=a} {N=b} {N=c}

?-r(a,N).

We never get to visit the other ways of resolving "?-q(a,Z), r(Z,N)."
Depth-First Expansion of the OLD tree

- If the underlying graph is acyclic, all branches in the OLD tree will be finite.
- If the graph is cyclic, nothing to the right of an infinite branch is expanded.
- This renders the evaluation **incomplete**: goals for which there are OLD derivations, but they are not found.
- Moreover, the same answer may be returned multiple times (even **infinitely**)!
  - Even if the underlying graph is acyclic, this evaluation is not efficient.
    - For query of the form \( r(a, N) \) we will return \( N=b \) for each path from “a” to “b”.
Depth-First Expansion of the OLD tree

- **Breadth-First** expansion does have the completeness property:
  - Every OLD derivation will be eventually constructed.
    - If something is a logical consequence, we will eventually confirm it in a finite number of levels.
  - But we may not be able to conclude *negative information*.
    - If something is not a logical consequence, we may never be able to identify it because we don't know when to stop?
Depth-First Expansion of the OLD tree

- Moreover, Breadth-First expansion does not give a natural operational understanding:
  - If we view predicates as being defined by “procedures”, then breadth first expansion steps through a procedure’s evaluation, switching contexts at the end of each step.
  - As in procedural programming, context switching is expensive (in this case, we’ve to switch substitutions)
Programming our way around the problem

- Is there a path from $X$ to $Y$ that does not visit any vertex already seen in $L$?

  $$p(L, X, Y) : -
  q(X, Y),
  not \text{member}(Y, L).$$

  $$p(L, X, Y) : -
  q(X, Z),
  not \text{member}(Z, L),
  p([Z|L], Z, Y).$$

- Now, start from $L=[]$ to look for reachable vertices

  $$r(X, Y) : -
  p([], X, Y).$$

- In $L$ we remember the path so far, and use this to avoid loops
Programming our way around the problem

• We are assured termination for reachability queries
  • We stop if a node has been seen before on the same branch.
• Still, this is inefficient: may take exponential time
  • We re-execute queries on different branches of the SLD/OLD tree
What is Tabled Resolution?

• Memoize calls and results to avoid repeated subcomputations.
• Termination: Avoid performing computations that repeat infinitely often.
• Complete for Datalog programs
• Efficiency: Dynamically share common subexpressions.
Depth-First Expansion of OLD tree with a little twist: **Stop if a goal has been seen before.**

```
?- r(a, N)
```

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

?- r(a, N)

?- q(a, N)
N=a  N=b  N=c

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[ q(a, a). \]
\[ q(a, b). \]
\[ q(a, c). \]
\[ q(b, b). \]
\[ q(b, d). \]
\[ q(b, e). \]
\[ q(e, d). \]

\[ r(X,Y) :- q(X,Y). \]
\[ r(X,Y) :- q(X,Z), r(Z,Y). \]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[
\begin{array}{c}
q(a, a) . \\
q(a, b) . \\
q(a, c) . \\
q(b, b) . \\
q(b, d) . \\
q(b, e) . \\
q(e, d) . \\
\hline
r(X, Y) :- q(X, Y) . \\
r(X, Y) :- \\
\qquad q(X, Z), r(Z, Y) .
\end{array}
\]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

?- r(a, N)
?- q(a, N)
?- q(a, Z), r(Z, N)
?- N=a
?- N=b
?- N=c
?- r(a, N)
?- r(b, N)

STOP
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[\begin{array}{c}
\text{?- } r(a, N) \\
\text{?- } q(a, N) \\
\text{?- } q(a, Z), r(Z, N) \\
\text{N=a} \quad \text{N=b} \quad \text{N=c} \\
\text{?- } r(a, N) \\
\text{?- } r(b, N) \\
\text{STOP} \\
\text{?- } q(b, N) \\
\text{N=b} \quad \text{N=d} \quad \text{N=e} \\
\end{array}\]

\text{q(a, a)}.
\text{q(a, b)}.
\text{q(a, c)}.
\text{q(b, b)}.
\text{q(b, d)}.
\text{q(b, e)}.
\text{q(e, d)}.

\text{r(X,Y) :- q(X,Y).}
\text{r(X,Y) :- q(X,Z), r(Z,Y).}
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

```
?- r(a, N)
?- q(a, N)
?- q(a, Z), r(Z, N)
?- r(a, N)
?- r(b, N)
?- q(b, N)
?- q(b, Z), r(Z, N)
?- r(b, N)
N=a N=b N=c
?- r(a, N)
N=b N=d N=e
?- r(b, N)
```
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

?- r(a, N)
?- q(a, N)
?- q(a, Z), r(Z, N)

N=a N=b N=c
?- r(a, N)
?- r(b, N)

STOP

?- q(b, N)
?- q(b, Z), r(Z, N)

N=b N=d N=e
?- r(b, N)

STOP

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[
\begin{align*}
q(a, a) & . \\
q(a, b) & . \\
q(a, c) & . \\
q(b, b) & . \\
q(b, d) & . \\
q(b, e) & . \\
q(e, d) & . \\
\end{align*}
\]

\[
\begin{align*}
r(X, Y) & : q(X, Y) . \\
r(X, Y) & : \\
& q(X, Z), r(Z, Y) . \\
\end{align*}
\]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d). \\
r(X,Y) & : - q(X,Y). \\
r(X,Y) & : - q(X,Z), r(Z,Y). \\
\end{align*}
\]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[q(a, a).\]
\[q(a, b).\]
\[q(a, c).\]
\[q(b, b).\]
\[q(b, d).\]
\[q(b, e).\]
\[q(e, d).\]

\[r(X, Y) :- q(X, Y).\]
\[r(X, Y) :- q(X, Z), r(Z, Y).\]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
  q(X,Z), r(Z,Y).
```

(c) Paul Fodor (CS Stony Brook) and Elsevier
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d). \\
\end{align*}
\]

\[
\begin{align*}
r(X,Y) & :- q(X,Y). \\
r(X,Y) & :- q(X,Z), r(Z,Y). \\
\end{align*}
\]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d). \\
r(X,Y) :- q(X,Y). \\
r(X,Y) :- q(X,Z), r(Z,Y).
\end{align*}
\]
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

?- r(a, N)
?- q(a, N)
?- q(a, Z), r(Z, N)
N=a, N=b, N=c
?- r(a, N)
?- r(b, N)
?- r(c, N)

?- q(b, N)
?- q(b, Z), r(Z, N)
N=b, N=d, N=e
?- r(b, N)
?- r(d, N)

STOP

?- q(d, N)
?- q(d, Z), r(Z, N)
STOP

?- q(e, N)
?- q(e, Z), r(Z, N)
N=d

STOP

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
Rationale for goal-based stopping

- The OLD tree is a representation of search for successful derivations
  - which are finite sequences of goals terminating in an empty goal.
- If there is a successful derivation, then there is an equivalent one that does not repeat the same goal (compare to reachability via loop-free paths in a graph).
- Hence ignoring paths with repeated goals is **sound**: the derivations pruned away by stopping have equivalent ones that will not be ignored.
- Unfortunately, this scheme still does not fix the problem of infinite derivations.


Infinite Derivations Despite Stopping Condition

\begin{center}
\begin{tikzpicture}
  \node (a) at (0,0) {a};
  \node (b) at (-2,-2) {b};
  \node (c) at (2,-2) {c};
  \node (d) at (-4,-4) {d};
  \node (e) at (4,-4) {e};

  \draw (a) edge (b);
  \draw (a) edge (c);
  \draw (b) edge (d);
  \draw (b) edge (e);
  \draw (c) edge (e);

\end{tikzpicture}
\end{center}

?- p(a, N)

\begin{verbatim}
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
  p(X,Z), q(Z,Y).
\end{verbatim}

(c) Paul Fodor (CS Stony Brook) and Elsevier
Expand tree as usual

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
Expand tree as usual

\[
\begin{align*}
q(a, a). \\
qu(a, b). \\
qu(a, c). \\
qu(b, b). \\
qu(b, d). \\
qu(b, e). \\
qu(e, d). \\
\end{align*}
\]

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
\[
\begin{align*}
p(X, Y) & :\; q(X, Y). \\
p(X, Y) & :\; \\
& \quad p(X, Z), \; q(Z, Y). \\
\end{align*}
\]
Infinite Derivations Despite Stopping Condition

Note that the right-most branch has ever-growing goals.

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
OLD Resolution with Tabling (OLDT)

- The selected literal at a step in a derivation is known as a call.
- OLDT maintains a table of calls (initially empty).
- With each call, it maintains a table of computed answers (initially empty).
- Start resolution as in OLD.
- When a literal is selected, check the call table.
  - If the literal is in the table, resolve it with its answers in its answer table.
  - If the literal is not in the table, resolve with program clauses (as in OLD), and add computed answers to its answer table.
OLDT Example

?- p(a, N)

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d).
\end{align*}
\]

%Alternative formulation
%of reachability: Note
%use of LEFT recursion
p(X, Y) :- q(X, Y).
p(X, Y) :-
    p(X, Z), q(Z, Y).

Start with empty tables
OLDT Example

?- p(a, N)

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d).
\end{align*}
\]

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pick selected literal. Is it in call table?
OLDT Example

?- p(a, N)

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td></td>
</tr>
</tbody>
</table>

Add to call table
OLDT Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td></td>
</tr>
</tbody>
</table>

Do OLD resolution with program clauses
OLDT Example

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td>{p(a,a)}</td>
</tr>
</tbody>
</table>

Add computed answer to table (if not already there)
OLDT Example

\[
\begin{array}{c}
\text{Calls} \\
p(a, W) \\
\{ p(a,a), p(a,b), p(a,c) \}
\end{array}
\]

Add computed answer to table (if not already there)
OLDT Example

\[
\begin{aligned}
&\text{\begin{tabular}{c}
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
\end{tabular}}
\end{aligned}
\]

\begin{equation*}
\begin{cases}
\text{N=a} \\
\text{N=b} \\
\text{N=c}
\end{cases}
\end{equation*}

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td>{p(a,a), p(a,b), p(a,c)}</td>
</tr>
</tbody>
</table>

Continue with OLD resolution

(c) Paul Fodor (CS Stony Brook) and Elsevier
OLDT Example

\[
\begin{array}{c}
\text{?- } p(a, N) \\
\text{?- } q(a, N) \\
\text{?- } p(a, Z), q(Z, N) \\
\text{N=a} \quad \text{N=b} \quad \text{N=c}
\end{array}
\]

\[
\begin{array}{c}
\text{q(a, a).} \\
\text{q(a, b).} \\
\text{q(a, c).} \\
\text{q(b, b).} \\
\text{q(b, d).} \\
\text{q(b, e).} \\
\text{q(e, d).}
\end{array}
\]

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
\quad p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td>{p(a,a), p(a,b), p(a,c)}</td>
</tr>
</tbody>
</table>

Pick selected literal. Is it in call table?
OLDT Example

\[
\begin{align*}
&\text{q(a, a).} \\
&\text{q(a, b).} \\
&\text{q(a, c).} \\
&\text{q(b, b).} \\
&\text{q(b, d).} \\
&\text{q(b, e).} \\
&\text{q(e, d).} \\
\end{align*}
\]

\[
\begin{align*}
\text{?- q(a, N)} \\
\text{N=a} \\
\text{N=b} \\
\text{N=c} \\
\end{align*}
\]

\[
\begin{align*}
\text{?- p(a, N)} \\
\text{?- p(a, Z), q(Z, N)}
\end{align*}
\]

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :- p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td>{p(a,a), p(a,b), p(a,c)}</td>
</tr>
</tbody>
</table>

Yes, resolve with answers in table
OLDT Example

\[
\begin{align*}
\text{q(a, a)}.
\text{q(a, b)}.
\text{q(a, c)}.
\text{q(b, b)}.
\text{q(b, d)}.
\text{q(b, e)}.
\text{q(e, d)}.
\end{align*}
\]

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
\[
\begin{align*}
p(X,Y) & :- q(X,Y). \\
p(X,Y) & :- p(X,Z), q(Z,Y).
\end{align*}
\]

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td>{p(a,a), p(a,b), p(a,c)}</td>
</tr>
</tbody>
</table>

Add computed answer to table (if not already there)
OLDT Example

\[
\begin{aligned}
\text{q(a, a).} \\
\text{q(a, b).} \\
\text{q(a, c).} \\
\text{q(b, b).} \\
\text{q(b, d).} \\
\text{q(b, e).} \\
\text{q(e, d).} \\
\end{aligned}
\]

% Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
  p(X,Z), q(Z,Y).

\[
\begin{array}{|c|c|}
\hline
\text{Calls} & \text{Answers} \\
\hline
p(a, W) & \{p(a,a), p(a,b), p(a,c)\} \\
\hline
\end{array}
\]

Continue resolving with answers in table
OLDT Example

\[
\begin{array}{c}
\text{q(a, a).} \\
\text{q(a, b).} \\
\text{q(a, c).} \\
\text{q(b, b).} \\
\text{q(b, d).} \\
\text{q(b, e).} \\
\text{q(e, d).} \\
\end{array}
\]

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
p(X,Z), q(Z,Y).

\[
\begin{array}{|c|c|}
\hline
\text{Calls} & \text{Answers} \\
\hline
p(a, W) & \{p(a,a), p(a,b), p(a,c) \\
& \quad p(a,d), p(a,e) \\
\hline
\end{array}
\]

Add computed answer to table (if not already there)
OLDT Example

%Alternative formulation
%of reachability: Note
%use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
p(X,Z), q(Z,Y).

<table>
<thead>
<tr>
<th>Calls</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, W)</td>
<td>{p(a,a), p(a,b), p(a,c), p(a,d), p(a,e)}</td>
</tr>
</tbody>
</table>

Complete when all answers have been considered for resolution
OLDT Forest

- When we get new answer, we will have to return to previous queries to continue their execution.
- When a literal is selected, mark it as a *consumer*.
- Check the call table.
  - If the literal is not in the table, *start a new tree for that literal* with its root marked as *generator*.
  - *Resolve generator with program clauses* (as in OLD), and add computed answers to its answer table.
- Resolve consumer with answers in its generator's table.
Calls and answers in tables

- Calls in table are standardized apart: i.e. their variables are renamed so that they are not identical to any other variable.
- Answers in a call's computed answer table share variables with their call. Any other variables are standardized apart.
- When checking if a literal is in call table:
  - We can check for variance: for a call \( c \) that is identical to the given literal \( l \), modulo names of variables.
    - All answers to \( c \) are answers to \( l \), and vice versa.
  - We can check for subsumption: for a call \( c \) that is more general than a given literal \( l \), i.e. if there is a substitution \( \theta \) such that \( c\theta = l \).
    - Not all answers to \( c \) may be answers to \( l \), but every answer to \( l \) is an answer to \( c \).
Notes on OLDT

- We can selectively mark which predicates we want to maintain tables for. (e.g. "p" in the previous example).
  - In general, no need to maintain tables for predicates defined solely by facts (i.e. clauses with empty bodies).

- For a Datalog program, there can be only finitely many distinct calls and answers.
  - So the size of tables is bounded.

- The number of literals in each goal is limited by the largest clause in the program (or original goal).

- Hence for Datalog, the OLDT forest as well as table sizes are bounded
OLDT: Second Example

?- r(a, N)

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

- Construct a forest: one tree for each call
- Root of each tree (blue) is a generator
- Selected literal that matches a tabled call (green) is a consumer

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X, Y) :- q(X, Y).
r(X, Y) :-
    q(X, Z), r(Z, Y).

(c) Paul Fodor (CS Stony Brook) and Elsevier
OLDT: Second Example

\[
\begin{array}{c}
\text{a} \\
\text{b} \quad \text{c} \\
\text{d} \quad \text{e}
\end{array}
\]

\[
\begin{array}{c}
\text{?- } r(a, N) \\
\?	ext{- q(a, N)} \\
\text{N=a} \quad \text{N=b} \quad \text{N=c}
\end{array}
\]

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d).
\end{align*}
\]

% ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
q(X,Z), r(Z,Y).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

?- q(a, N).
?- q(a, Z), r(Z, N).
?- r(a, N).
N=a  N=b  N=c
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
\[ r(X,Y) :\ q(X,Y) . \]
\[ r(X,Y) :\ q(X,Z) , r(Z,Y) . \]
OLDT: Second Example

?- r(a, N)
?- q(a, N)
?- q(a, Z), r(Z, N)

N=a  N=b  N=c

?- r(a, N)

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
q(X,Z), r(Z,Y).

(c) Paul Fodor (CS Stony Brook) and Elsevier
OLDT: Second Example

\[ \text{q(a, a).} \]
\[ \text{q(a, b).} \]
\[ \text{q(a, c).} \]
\[ \text{q(b, b).} \]
\[ \text{q(b, d).} \]
\[ \text{q(b, e).} \]
\[ \text{q(e, d).} \]

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
        q(X,Z), r(Z,Y).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
\[ r(X,Y) :- q(X,Z), r(Z,Y). \]

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
OLDT: Second Example

\[
\begin{align*}
q(a, a) &. \\
q(a, b) &. \\
q(a, c) &. \\
q(b, b) &. \\
q(b, d) &. \\
q(b, e) &. \\
q(e, d) &. \\
\end{align*}
\]

% ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X, Y) :- q(X, Y).
r(X, Y) :-
q(X, Z), r(Z, Y).
OLDT: Second Example

\[
\begin{array}{c}
\text{q(a, a).} \\
\text{q(a, b).} \\
\text{q(a, c).} \\
\text{q(b, b).} \\
\text{q(b, d).} \\
\text{q(b, e).} \\
\text{q(e, d).} \\
\end{array}
\]

%ORIgINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

\[ \begin{align*}
q(a, a). & \\
q(a, b). & \\
q(a, c). & \\
q(b, b). & \\
q(b, d). & \\
q(b, e). & \\
q(e, d). & \\
\end{align*} \]

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
q(X,Z), r(Z,Y).

(c) Paul Fodor (CS Stony Brook) and Elsevier
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X, Y) :- q(X, Y).
r(X, Y) :-
    q(X, Z), r(Z, Y).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X, Y) :- q(X, Y).
r(X, Y) :-
  q(X, Z), r(Z, Y).

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

?- r(a, N)
?- q(a, N), r(z, N)
?- q(b, N)
?- r(c, N)

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X, Y) :- q(X, Y).
r(X, Y) :-
    q(X, Z), r(Z, Y).
OLDT: Second Example

?- r(a, N)
  ?- q(a, N)
  ?- q(a, Z), r(Z, N)
  N=a N=b N=c
  ?- r(a, N)
  ?- r(b, N)
  ?- r(c, N)
  N=a N=b N=d N=e N=b N=e

?- r(b, N)
  ?- q(b, N)
  ?- q(b, Z), r(Z, N)
  N=b N=d N=e
  ?- r(b, N)

?- r(c, N)
  ?- q(c, N)
  ?- q(c, Z), r(Z, N)
  ff ff

?- r(c, N)

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
  q(X,Z), r(Z,Y).
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
OLDT: Second Example

\[
\begin{align*}
\text{q(a, a).} \\
\text{q(a, b).} \\
\text{q(a, c).} \\
\text{q(b, b).} \\
\text{q(b, d).} \\
\text{q(b, e).} \\
\text{q(e, d).}
\end{align*}
\]

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X, Y) :- q(X, Y).
r(X, Y) :-
    q(X, Z), r(Z, Y).

(c) Paul Fodor (CS Stony Brook) and Elsevier
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example_tree}
\caption{Example tree for OLDT.}
\end{figure}

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
\begin{verbatim}
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
\end{verbatim}
OLDT: Second Example

\[ q(a, a). q(a, b). q(a, c). q(b, b). q(b, d). q(b, e). q(e, d). \]

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
  q(X,Z), r(Z,Y).
OLDT: Second Example

q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAl formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
OLDT: Second Example

\begin{verbatim}
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
\end{verbatim}
OLDT: Second Example

\[
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d). \\
\end{align*}
\]

\%ORIGINAL formulation
\% of reachability: Note
\% use of RIGHT recursion
\r(X, Y) :- q(X, Y).
\r(X, Y) :-
  q(X, Z), r(Z, Y).
OLDT: Second Example

\[
\text{q(a, a).} \\
\text{q(a, b).} \\
\text{q(a, c).} \\
\text{q(b, b).} \\
\text{q(b, d).} \\
\text{q(b, e).} \\
\text{q(e, d).}
\]

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
\[
\begin{align*}
\text{r(X,Y)} & : - \text{q(X,Y).} \\
\text{r(X,Y)} & : - \text{q(X,Z), r(Z,Y).}
\end{align*}
\]
OLDT: Second Example

\[
\text{\texttt{\textbackslash{}nl\textbackslash{}nl}}
\begin{align*}
q(a, a). \\
q(a, b). \\
q(a, c). \\
q(b, b). \\
q(b, d). \\
q(b, e). \\
q(e, d).
\end{align*}
\]

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
q(X,Z), r(Z,Y).

(c) Paul Fodor (CS Stony Brook) and Elsevier
OLDT: Second Example

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, c).
q(e, d).
q(e, e).

%ORIGIN formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
   q(X,Z), r(Z,Y).
```
OLD Resolution with Tabling (OLDT)

- OLDT evaluation can be used to infer **negative answers**: e.g. a vertex is not reachable from another.

- Note that Breadth-First evaluation, or even the evaluation with goal-based stopping condition cannot do this.
Tabled Resolution in XSB

\[ \text{reach}(X,Y) : \neg \text{edge}(X,Y). \]
\[ \text{reach}(X,Y) : \neg \text{reach}(X,Z), \text{edge}(Z,Y). \]
\[ \text{edge}(a,a). \]
\[ \text{edge}(a,b). \]
\[ \text{edge}(b,c). \]
\[ : \neg \text{table(reach}/2). \]
\[ \%\text{OR} : \neg \text{auto_table}. \]

- Call:

\[ ?- \text{reach}(a,V). \]
Tabled Resolution

\[ \text{reach}(X,Y) : - \text{edge}(X,Y). \]
\[ \text{reach}(X,Y) : - \text{reach}(X,Z), \text{edge}(Z,Y). \]
\[ \text{edge}(a,a). \]
\[ \text{edge}(a,b). \]
\[ \text{edge}(b,c). \]

- Calls

?- \text{reach}(a,V).

Answers
Tabled Resolution

\[ \text{reach}(X, Y) :\neg \text{edge}(X, Y). \]
\[ \text{reach}(X, Y) :\neg \text{reach}(X, Z), \text{edge}(Z, Y). \]
\[ \text{edge}(a, a). \]
\[ \text{edge}(a, b). \]
\[ \text{edge}(b, c). \]

- Calls

\[ ?- \text{reach}(a, V). \]

**Answers**

\[ V = a \]
Tabled Resolution

\[ \text{reach}(X,Y) :\neg \text{edge}(X,Y). \]
\[ \text{reach}(X,Y) :\neg \text{reach}(X,Z), \text{edge}(Z,Y). \]
\[ \text{edge}(a,a). \]
\[ \text{edge}(a,b). \]
\[ \text{edge}(b,c). \]

- Calls

?− \text{reach}(a,V).

Answers

\[ V = a \]
\[ V = b \]
Tabled Resolution

reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).

edge(a,a).
edge(a,b).
edge(b,c).

- Calls
?- reach(a,V).

Answers
V = a
V = b
Tabled Resolution

\[ \text{reach}(X, Y) \text{ :- } \text{edge}(X, Y). \]
\[ \text{reach}(X, Y) \text{ :- } \text{reach}(X, Z), \text{ edge}(Z, Y). \]

\text{edge}(a, a).
\text{edge}(a, b).
\text{edge}(b, c).

- Calls

? - \text{reach}(a, V).

Answers

- V = a
- V = b
Tabled Resolution

\[
\text{reach}(X, Y) :\neg \text{edge}(X, Y).
\]

\[
\text{reach}(X, Y) :\neg \text{reach}(X, Z), \text{ edge}(Z, Y).
\]

\[
\text{edge}(a, a).
\]

\[
\text{edge}(a, b).
\]

\[
\text{edge}(b, c).
\]

- Calls

\[- \text{reach}(a, V) \]

Answers

\[
V = a
\]

\[
V = b
\]

\[
V = c
\]
Tabled Resolution

reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).

• Calls

?- reach(a,V).

Answers
V = a
V = b
V = c

Answer completion!