Motivation

Alternatives to SLD/OLD Resolution

- Prolog follows OLD resolution (recall: OLD = 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 $A_1, A_2, \ldots, A_n$ as a “procedure stack” with $A_1$, the selected literal on top.
  - Call $A_1$. If and when $A_1$ returns, continue with the rest of the computation (call $A_2$, and upon its return call $A_3$, etc until nothing is left)
  - Note $A_2$ is “opened up” only when $A_1$ returns, not after executing part of $A_1$.
- Depth-first expansion, however, contributes to the incompleteness of Prolog’s evaluation, which may not terminate even when the least model is finite!
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, etc.
  - If we can go from state $s$ by executing one instruction to $s'$, then we can place an edge from $s$ to $s'$.
  - There are richer models for representing program evaluation, but a directed graph is most basic.
  - 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 the edge relation.

- Predicate “q” on left is an example.
- Reachability can then be written as a transitive closure over the edge relation.
  - Observe the predicate “r” defined on left 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 an intermediate vertex $Z$ such that
    - there is an edge from $X$ to $Z$, and
    - there is a path from $Z$ to $Y$.
Motivation

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.

\[ M_0 = \emptyset \]
\[ M_1 = TP(M_0) = M_0 + \{q(a, a), q(a, b), q(a, c), q(b, b), q(b, d), q(b, e), q(e, d)\} \]
\[ M_2 = TP(M_1) = M_1 + \{p(a, a), p(a, b), p(a, c), p(b, b), p(b, d), p(b, e), p(e, d)\} \]
\[ M_3 = TP(M_2) = M_2 + \{p(a, d), p(a, e)\} \]
\[ M_4 = TP(M_3) = M_3 + \emptyset \]
- 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.

Computing with Logic

OLD Resolution with Depth-First Expansion

\[ r(X, Y) := q(X, Y). \]
\[ r(X, Y) := q(X, Z), r(Z, Y). \]
\[ ?- r(a, N) \]
\[ ?- q(a, N) \]
\[ ?- q(a, Z), r(Z, N) \]
\[ N=a \quad N=b \quad N=c \]
\[ ?- r(a, N) \]
\[ ?- q(a, N) \]
\[ ?- q(a, Z), r(Z, N) \]
\[ \infty \]
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 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”.

Breadth-First Expansion of the OLD tree

- Breadth-First expansion does have the *completeness* property:
  - Every OLD derivation will be eventually constructed.
- But we may not be able to conclude negative information.
  - If something is a logical consequence, we will eventually confirm it.
  - If something is not a logical consequence, we may never be able to identify it.
- 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 in L?
% Assume X is not in L.
p(L, X, Y) :-
    q(X, Y), not member(Y, L).
p(L, X, Y) :-
    q(X, Z), not 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.
- We are assured termination for reachability queries.
- Still, this is inefficient: may take exponential time.

Depth-First Expansion of OLD tree with a little twist

Stop if a goal has been seen before.
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

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

Start with query “p(a, N)”
Expand tree as usual
Note that the right-most branch has ever-growing goals.
```
OLD Resolution with Tabling (OLDT)

- Selected literal at a step in a derivation is known as a **call**.
- Maintain a table of calls (initially empty).
- With *each call*, maintain 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: First 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>

Start with empty tables
Pick selected literal. Is it in call table?
Add to call table
Do OLDS resolution with program clauses
Add computed answer to table (if not already there)
Continue with OLD resolution
Yes, resolve with answers in 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 \(\theta\) 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

- 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
Complexity of OLDT Resolution— I

Let \( q/2 \) encode the edge relation in a directed graph.

<table>
<thead>
<tr>
<th>%Alternative formulation</th>
<th>%ORIGINAL formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of reachability: Note</td>
<td>% of reachability: Note</td>
</tr>
<tr>
<td>% use of LEFT recursion</td>
<td>% use of RIGHT recursion</td>
</tr>
<tr>
<td>( p(X,Y) :- q(X,Y) ).</td>
<td>( r(X,Y) :- q(X,Y) ).</td>
</tr>
<tr>
<td>( p(X,Y) :- )</td>
<td>( r(X,Y) :- )</td>
</tr>
<tr>
<td>( p(X,Z), q(Z,Y) ).</td>
<td>( q(X,Z), r(Z,Y) ).</td>
</tr>
</tbody>
</table>

- Let there be \( E \) edges and \( V \) vertices in the graph.
- For original formulation (right):
  - For each call of the form \( r(s,N) \) there are as many as \( O(E) \) consumers (due to right recursion).
  - For each consumer, there are \( O(V) \) answers.
  - In total, we spend \( O(E \cdot V) \) time to answer query of the form \( r(s,N) \).

Complexity of OLDT Resolution

Let \( q/2 \) be the edge relation of a graph with \( E \) edges and \( V \) vertices.

<table>
<thead>
<tr>
<th>%Alternative formulation</th>
<th>%ORIGINAL formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of reachability: Note</td>
<td>% of reachability: Note</td>
</tr>
<tr>
<td>% use of LEFT recursion</td>
<td>% use of RIGHT recursion</td>
</tr>
<tr>
<td>( p(X,Y) :- q(X,Y) ).</td>
<td>( r(X,Y) :- q(X,Y) ).</td>
</tr>
<tr>
<td>( p(X,Y) :- )</td>
<td>( r(X,Y) :- )</td>
</tr>
<tr>
<td>( p(X,Z), q(Z,Y) ).</td>
<td>( q(X,Z), r(Z,Y) ).</td>
</tr>
</tbody>
</table>

- For the alternative formulation (left):
  - For each call of the form \( p(s,N) \) there is exactly one consumer call (due to left recursion).
  - There are \( O(V) \) possible answers to this call.
  - For each answer \( Z \), we call \( q(Z,Y) \), and may get up to \( O(V) \) bindings for \( Y \), giving \( O(V^2) \) pairs of bindings to \( (Z,Y) \).
  - But with only \( O(E) \) edges, the number of bindings is limited to \( O(E) \).
  - In total, we spend \( O(E) \) time to answer query of the form \( p(s,N) \).
Parsing using OLDT— I

% DCG Form
s --> s, [a], s, [b].
s --> [].

% Prolog expansion:
s(V,Z) :- s(V,W), W=[a|X], s(X,Y), Y=[b|Z]. s(V,V).

- For a string of length $N$, there are $\Theta(N^2)$ generator calls.
- Each generator has one consumer of the form “s(V,W)”
  ... with possibly $\Theta(N)$ answers.
- Each answer to “W” gives one binding to “X”
  ... one additional consumer of the form “s(X,Y)”.
- Total number of calls = $\Theta(N^3)$.
- Each call takes $\Theta(N)$ time to look up (since binding of “V” and “Z”
  may be $\Theta(N)$ lists).
- This can be made constant time by representing the string as a set of
  facts (symbol/position pairs).

Parsing using OLDT— II

- With OLDT, we get a universal parser for all context-free languages.
- Preceding complexity analysis was a loose estimate.
  - Parsing in unambiguous grammars can be done in $\Theta(N^2)$ time.
  - Parsing in ambiguous grammars can be done in $\Theta(N^3)$ time.
- The famous Cocke-Kasami-Younger dynamic programming algorithm
  has complexity of $\Theta(N^3)$ for all grammars.
- With OLDT we get the same running time as the more complex
  algorithm of Earley’s parser.
Solutions to other problems using OLDT

- **Language Intersection:** Do the languages of a given finite automaton and a given grammar have an empty intersection: \( O(G \cdot N^3) \), where \( G \) is grammar size and \( N \) is automaton size.
  - Used in verification of programs (program traces represented by a grammar; property represented by a finite automaton).

- **Program Analysis:** E.g. Anderson’s points-to analysis: \( O(S \cdot V^2) \) for program with \( S \) statements and \( V \) variables.

- **Model checking:** \( O(S \cdot F) \) for systems of size \( S \) and CTL (temporal logic) formulas of size \( F \).

Further on OLDT

- For all the above problems, 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.
- Inferring negative information from logic programs is a complex topic that will be studied separately.