Resolution Theorem Proving, Part 3

Robert Keller
21 March 2005
The Need for “Factoring”

- The simple form of resolution (called “binary resolution”) used so far is not quite enough for full generality.

- Consider these clauses:
  - \( p(X) \lor p(Y) \)
  - \( \neg p(U) \lor \neg p(V) \)

- There are four ways to resolve these two clauses (e.g. \( \{X \leftarrow U\} \)). However, no resolvent introduces anything new. In order to make progress, we need to “factor” the clauses.
Factoring

- If there are two or more literals in the same clause that unify, then the result of reducing the clause after applying the mgu is called a factor of the clause.

- Example:
  - In clause $p(X) \lor p(Y)$, $\{X \leftarrow Y\}$ unifies the two literals.
  - The reduced form is $p(Y)$, which is a factor of the original clause.
  - Evidently this clause could be replaced with its factor. However, this will not always be the case.
Factoring: Another Example

- In clause
  \[ p(X) \lor p(f(Y)) \lor p(f(g(Z))) \lor q(Y) \]
  \{X\leftarrow f(g(Z)), Y\leftarrow g(Z)\} unifies the first three literals.

- The corresponding factor is:
  \[ p(f(g(Z))) \lor q(g(Z)) \]

- The factor is, however, less general, so we cannot replace the original clause with the factor.
General Resolution of Two Clauses

- Two clauses resolve if:
  - They have a binary resolvent (the simplest kind of resolution, without factoring).
  - One clause and a factor of the other have a binary resolvent.
  - There are factors of the two clauses that have a binary resolvent.

- Since a clause is trivially a factor of itself, we could get by with just the third statement above.
Full Resolution Example

- **Clauses:**
  - $p(X, Y) \lor p(Y, X)$
  - $\neg p(U, V) \lor \neg p(V, U)$

- **Factors:**
  - $p(X, X)$
  - $\neg p(U, U)$

- **Resolvent:**
  - $\bot$
Full Resolution Example

• **Clauses:**
  1. \( p(X, Y) \lor q(X, Y) \)
  2. \( p(U, V) \lor \neg q(U, g(W)) \)
  3. \( \neg p(f(R), S) \lor \neg p(f(S), g(T)) \)

• **Resolvents:**
  4. \( p(U, V) \lor p(U, g(W)) \) 1, 2 with \( \{X \leftarrow U, Y \leftarrow g(W)\} \)
  5. \( \bot \) 3, 4 with \( \{U \leftarrow f(g(T)), R \leftarrow g(T), S \leftarrow g(T), V \leftarrow g(T), W \leftarrow T\} \)
Subsumption

- A clause $C$ **subsumes** a clause $D$ if there is a substitution $\theta$ such that $C\theta \subseteq D$, where we interpret the clauses as **sets** of their literals.

- If a clause $D$ in a set of clauses is subsumed by another clause $C$ **within the set**, then we can delete $D$ from the set without affecting the case of whether the empty clause $\bot$ is derivable.
Subsumption Examples

- \( P(X) \) subsumes \( P(X) \lor Q(Y) \) (by the empty substitution \( \{\} \)).

- \( \neg P(X) \lor Q(f(X)) \) subsumes
  \( \neg P(Z) \lor \neg P(h(Y)) \lor Q(f(h(Y))) \)
  (by the substitution \( \{X \leftarrow h(Y), Z \leftarrow h(Y)\} \)).
Answer Extraction

- Resolution is not just for proving theorems anymore.

- Resolution can be used for extracting answers from a database, knowledge base, or reasoning system.
From Yes-No to Answer Terms

- Consider the clause set:
  - \( \neg \text{man}(X) \lor \text{mortal}(X) \)
  - \( \text{man}(\text{socrates}) \)
  - \( \neg \text{mortal}(\text{socrates}) \)

- Obviously this set is unsatisfiable, and we can obtain a proof by resolution.
- But what if we drop the third clause. The first two clauses are satisfiable, and can be thought of as a "knowledge base".
- We can ask a question of the knowledge base:

  Name someone who is mortal.
Asking Questions

- To find an individual for $X$ that satisfies a criterion $p(...X...)$, add to the set of clauses the clause:

  $$\neg p(...X...) \lor \text{answer}(X)$$

- Then conduct resolution as before, but stop when there is a clause containing only answer literals.
Example: Who is mortal?

1. $\neg \text{man}(X) \lor \text{mortal}(X)$
2. $\text{man}(\text{socrates})$
3. $\neg \text{mortal}(Y) \lor \text{answer}(Y)$
4. $\text{mortal}(\text{socrates})$ resolution 1, 2
5. $\text{answer}(\text{socrates})$ resolution 2, 3
Example: Who is Caroline’s Grandfather?

1. $\neg\text{father}(X, Y) \lor \text{parent}(X, Y)$
2. $\neg\text{father}(X, Y) \lor \neg\text{parent}(Y, Z) \lor \text{grandfather}(X, Z)$
3. $\text{father}(\text{joe}, \text{john})$
4. $\text{father}(\text{john}, \text{caroline})$
5. $\neg\text{grandfather}(X, \text{caroline}) \lor \text{answer}(X)$
6. $\neg\text{father}(X, Y) \lor \neg\text{parent}(Y, \text{caroline}) \lor \text{answer}(X)$
7. $\neg\text{father}(X, Y) \lor \neg\text{father}(Y, \text{caroline}) \lor \text{answer}(X)$
8. $\neg\text{father}(X, \text{john}) \lor \text{answer}(X)$
9. $\text{answer}(\text{joe})$
Answer Extraction in Otter

- Normally Otter searches for the null clause and stops when and if it has produced it.

- If the special literal

  \$answer(X)

appears in a clause, Otter will stop when it finds a clause containing only literals containing \$answer.
Grandfather Example in Otter

\[-father(x, y) \mid parent(x, y).\]

\[-father(x, y) \mid -parent(y, z) \mid grandfather(x, z).\]

\[father(joe, john).\]

\[father(john, caroline).\]

\[-grandfather(x, caroline) \mid \$answer(x).\]

1  [ ]  \(-father(x,y)\mid parent(x,y).\)
2  [ ]  \(-father(x,y)\mid -parent(y,z)\mid grandfather(x,z).\)
3  [ ]  \(-grandfather(x,caroline)\mid \$answer(x).\)
4  [ ]  father(joe,john).
5  [ ]  father(john,caroline).
7  [hyper,5,1]  parent(john,caroline).
8  [hyper,7,2,4]  grandfather(joe,caroline).
9  [binary,8.1,3.1]  \$answer(joe).
“Logic” Puzzles: Example

% Professors Dodds, Stone, and Thom go to their favorite bars for beer.  
% Each prof prefers a different beer (one of Anchor, Bud, and Miller)  
% and frequents a different bar (one of Alice's, Harry's, or Joe's).  
% Each bar serves a unique beer.

% Professor Stone prefers Bud. (Clue 1)
% Professor Thom doesn't prefer Miller. (Clue 2)
% The prof who prefers Miller frequents Alice's bar. (Clue 3)
% The prof who prefers Anchor does not frequent Joe's. (Clue 4)

% Which bar does each prof frequent and what beer does each prefer?
Solving

- Determine clause form from clues.

- We will use Otter form, so that Otter can try to solve the puzzle.

- Note: This may look really simple, but it is not always easy to get right.
Define clauses for clues

- prefer(Stone, Bud).
  % Clue 1

- prefer(Thom, Miller).
  % Clue 2

- prefer(x, Miller) | frequent(x, Alice).
  % Clue 3

- prefer(x, Anchor) | -frequent(x, Joe).
  % Clue 4
Identity Individuals in Various Categories

- prof(Dodds).
  prof(Stone).
  prof(Thom).

- beer(Anchor).
  beer(Bud).
  beer(Miller).

- bar(Alice).
  bar(Harry).
  bar(Joe).
Distribution Requirements

- % Every bar is frequented by some prof.
  - \(-\text{bar}(y) \mid \text{frequent(Dodds, } y) \mid \text{frequent(Stone, } y) \mid \text{frequent(Thom, } y)\).

- % Every beer is preferred by some prof.
  - \(-\text{beer}(y) \mid \text{prefer(Dodds, } y) \mid \text{prefer(Stone, } y) \mid \text{prefer(Thom, } y)\).
Uniqueness Requirements

- % Each bar serves a unique beer.
  - serves(x, y) | serves(x, z) | y = z.

- % Each prof prefers a unique beer.
  - prefer(x, y) | prefer(x, z) | y = z.

- % Each prof frequents a unique bar.
  - frequent(x, y) | frequent(x, z) | y = z.
Answer Clause

Which bars are frequented, and which beers preferred, by which professors?

- frequent( Dodds, x) | -frequent(Stone, y) | -frequent(Thom, z)
| -prefer( Dodds, u)    |   - prefer(Stone, v)   |   -prefer(Thom, w)
| $answer( [Dodds, x, u], [Stone, y, v], [Thom, z, w] ).
Otter Solution

$\text{answer}([\text{Dodds,Alice,Miller}],
[\text{Stone,Joe,Bud}],
[\text{Thom,Harry,Anchor}]).$
What if Solution not Unique?

Try removing one or more clues.

Note that Otter will give a *disjunctive* solution.

\[
\text{\$answer([Dodds,Harry,Anchor],}
\text{[Stone,Joe,Bud],}
\text{[Thom,Alice,Miller])} \\
\text{\mid \$answer([Dodds,Alice,Miller],}
\text{[Stone,Joe,Bud],}
\text{[Thom,Harry,Anchor])}.
\]
What if No Solution?

If there is no refutation, Otter will run out of clauses to create, or run forever.
Set of Support (sos) Strategy

- A typical clause set to be refuted will involve:
  - A set of clauses known, or thought to be mutually consistent (satisfiable), e.g. derived from axioms.
  - A single clause which is derived from the negation of the “theorem” to be proved.

- The sos strategy entails always picking one clause for resolution from the sos, and others from outside the sos.

- Resolvents are added to the sos.

- This is a complete strategy and is the one used by Otter.
Motion Puzzles and Games

- Moves in a motion puzzle or game can often be encoded as logic.

- Resolution can be used to find a solving or winning sequence of moves.
Example: Linear Peg Solitaire
Linear Peg Solitaire Explanation

• Pegs of two colors are shown in their home positions at the top.
• The objective is to completely reverse the pegs, so that each peg’s original home is occupied by a peg of the opposite color.
• Allowable actions:
  • Move: A peg can be moved toward the opposite side by moving into an adjacent empty hole.
  • Jump: A peg can jump toward the opposite side over a peg of the opposite color, provided that there is a hole to receive the jumping peg.
General Form of the Puzzle

• Versions of the puzzle exists for $2n$ pegs ($n$ of each color) and $2n+1$ holes.

• Ideally, each version can be solved.
Peg Game Formulation

- Represent the **state** of the game with two terms.
- Say the pegs are **w** for white, **r** for red.
- Represents the pegs **away from the hole** in either direction as a composition of function symbols.
- The initial state shown is:
  \[ s(w(w(w(w(c)))))), r(r(r(r(c)))) \]
- The second state shown is:
  \[ s(r(w(r(w(w(c)))))), w(r(r(c))) \]
- **c** is a dummy constant symbol
Formulating Moves

- **Simple moves (non-jump):**
  - move(s(w(X), Y), s(X, w(Y))) (wm)
  - move(s(X, r(Y)), s(r(X), Y)) (rm)

- **Jump moves:**
  - move(s(r(w(X)), Y), s(X, r(w(Y)))) (wj)
  - move(s(X), w(r(Y))), s(w(r(X)), Y)) (rj)
Formulating Reachability

• Initial state:
  reachable\(w(w(w(w(c))))\), \(r(r(r(r(c))))\))

• State change:
  \(\neg \text{reachable}(X) \lor \neg \text{move}(X, Y) \lor \text{reachable}(Y)\)

• Final state:
  \(\neg \text{reachable}(r(r(r(r(c))))), w(w(w(w(c))))))\)
Otter Formulation

\[
\text{move}(s(w(x), y), s(x, w(y))).
\]

\[
\text{move}(s(x, r(y)), s(r(x), y)).
\]

\[
\text{move}(s(r(w(x)), y), s(x, r(w(y)))).
\]

\[
\text{move}(s(x, w(r(y))), s(w(r(x)), y)).
\]

\[
\text{reachable}(s(w(w(w(w(c)))), r(r(r(r(c)))))).
\]

\[
-\text{reachable}(x) \mid -\text{move}(x, y) \mid \text{reachable}(y).
\]

\[
-\text{reachable}(s(r(r(r(r(c)))), w(w(w(w(c)))))).
\]
Otter proof for 2 pegs of each color

1 [] -reachable(x) | -move(x,y) | reachable(y).
2 [] -reachable(s(r(r(c)),w(w(c)))).
3 [] move(s(w(x),y),s(x,w(y))).
4 [] move(s(x,r(y)),s(r(x),y)).
5 [] move(s(r(w(x)),y),s(x,r(w(y)))).
6 [] move(s(x,w(r(y))),s(w(r(x)),y)).
7 [] reachable(s(w(w(c)),r(r(c)))).
10 [hyper,4,1,7] reachable(s(r(w(w(c))),r(c))).
11 [hyper,10,1,5] reachable(s(w(c),r(w(r(c))))).
14 [hyper,11,1,3] reachable(s(c,w(r(w(r(c))))).
18 [hyper,14,1,6] reachable(s(w(r(c)),w(r(c)))).
22 [hyper,18,1,6] reachable(s(w(r(w(r(c)))),c)).
24 [hyper,22,1,3] reachable(s(r(w(r(c))),w(c))).
26 [hyper,24,1,5] reachable(s(r(c),r(w(w(c))))).
28 [hyper,26,1,4] reachable(s(r(r(c)),w(w(c)))).
29 [binary,28.1,2.1] $\text{F}$.
Otter Proof for 3 pegs of each color

7  []  reachable(s(w(w(w(c)))),r(r(r(c)))).
8  [hyper,7,1,4]  reachable(s(r(w(w(w(c)))),r(r(c)))).
12 [hyper,5,1,8]  reachable(s(w(w(c))),r(w(r(r(c))))).
16 [hyper,12,1,3]  reachable(s(w(c)),w(r(w(r(r(c)))))).
20 [hyper,16,1,6]  reachable(s(w(r(w(c))),w(r(r(c))))).
27 [hyper,20,1,6]  reachable(s(w(r(w(r(w(c))))),r(c))).
34 [hyper,27,1,4]  reachable(s(r(w(r(w(r(c))))),c)).
39 [hyper,34,1,5]  reachable(s(r(w(r(w(c)))),r(w(c)))).
44 [hyper,39,1,5]  reachable(s(r(w(c)),r(w(r(r(c)))))).
51 [hyper,44,1,5]  reachable(s(c,r(w(r(w(r(w(c))))))).
57 [hyper,51,1,4]  reachable(s(r(c),w(r(w(r(w(c))))))).
63 [hyper,57,1,6]  reachable(s(w(r(r(c))),w(r(w(c))))).
69 [hyper,63,1,6]  reachable(s(w(r(w(r(r(c))))),w(c))).
72 [hyper,69,1,3]  reachable(s(r(w(r(r(c)))),w(w(c)))).
75 [hyper,72,1,5]  reachable(s(r(r(c)),r(w(w(w(c)))))).
77 [hyper,75,1,4]  reachable(s(r(r(r(c))),w(w(w(c))))).
78 [binary,77.1,2.1]  $F$. 

### Pegs vs. Proof Length (# of Moves)

<table>
<thead>
<tr>
<th>Pegs of Each Color</th>
<th>Proof Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>( n )</td>
<td>( n^2 + 2n )</td>
</tr>
</tbody>
</table>
Determining the Move Sequence

- The previous proofs only showed that the puzzle could be solved for those variations.

- The actual move sequence would have to be dug out from the proof steps.

- We can modify the rules so that the move sequence is obtained as a byproduct.
Determining the Move Sequence

- Use function composition to represent accumulated move sequence.
- Revised rules (4-pegs, where specific):
  - move(s(w(x), y), s(x, w(y)), z, wm(z)).
  - move(s(x, r(y)), s(r(x), y), z, rm(z)).
  - move(s(r(w(x)), y), s(x, r(w(y))), z, wj(z)).
  - move(s(x, w(r(y))), s(w(r(x)), y), z, rj(z)).
  - reachable(s(w(w(w(w(c)))), r(r(r(r(c))))), d).
  - -reachable(x, z) | -move(x, y, z, zz) | reachable(y, zz).
  - -reachable(s(r(r(r(r(c)))), w(w(w(w(c))))), z) | $answer(z).
The Move Sequence is Read Inside-Out:

• For 4 pegs of each color:
  \[\text{answer(rm}(wj(wm(rj(rj(rm(wj(wj(wj(wj(wm(rj(rj(rj(rj(wm(wj(wj(wj(rm(d))))))))))))))))))))\).

• The sequence is:
  \[\text{rm wj wm rj rj rm wj wj wj wm rj rj}
  \text{rj rj wm wj wj rm rj rj wm wj rm}\]

• (For this puzzle, the move sequence is coincidentally a palindrome.)
Other Notes on Otter

- Otter can preprocess formulas into clauses for you:
  - Quantifiers as input
  - Automatic prenexing and skolemization

- Otter can automatically determine \textit{an sos.}
Example from the Mid-Term

- Given:
  - \( \forall x \exists y \ R(x, y) \)
  - \( \forall x \ \forall y \ \forall z \ ((R(x, y) \land R(y, z)) \rightarrow R(x, z)) \)
  - \( \forall x \ \forall y \ (R(x, y) \rightarrow R(y, x)) \)

To derive:
- \( \forall x \ R(x, x) \)
Otter Input

formula_list(usable). % use formula_list rather than list
all x (exists y r(x, y)).
all x all y all z ((r(x, y) & r(y, z)) -> r(x, z)).
all x all y (r(x, y) -> r(y, x)).
-(all x r(x, x)).
end_of_list.
Result of Pre-Processing by Otter

\[ r(x, f1(x)). \] %Auto-identified as sos by Otter
\[ -r(x,y) \lor -r(y,z) \lor r(x, z). \]
\[ -r(x,y) \lor r(y,x). \]
\[ -r(c1,c1). \]

Note that $ is Otter’s way of specifying generated Skolem functions and constants.
The original list was:

all x (exists y r(x, y)).

all x all y all z ((r(x, y) & r(y, z)) -> r(x, z)).

all x all y (r(x, y) -> r(y, x)).

-(all x r(x, x)).
Otter’s Proof of the Midterm Problem

1 [ ] \(-r(x,y) | -r(y,z) | r(x,z)\).
2 [ ] \(-r(x,y) | r(y,x)\).
3 [ ] \(-r(c1,c1)\).
4 [ ] \(r(x,f1(x))\).
5 [hyper,4,2] \(r(f1(x),x)\).
8 [hyper,5,1,4] \(r(x,x)\).
9 [binary,8.1,3.1] $F$. 
Otter Proof Rules and Nomenclature

- **binary**: means binary resolution
- **factoring**: is indicated when used
- **hyper**: means hyper-resolution: resolving multiple clauses in one step.
- **paramodulation**: a rule for handling equality
- **demodulation**: use of user-specified equalities
- **Knuth-Bendix**: a system for pre-processing equality rules
Otter is Not Prolog

- The syntax is different, although Otter has a “Prolog variables” mode.

- Otter has genuine negation.

- Prolog only has “negation as failure”.

- Prolog relies on the “closed world assumption”.