Finding Shortest Proofs: An Application of Linked Inference Rules *

Robert Veroff
University of New Mexico, Albuquerque, NM 87131, U.S.A.

Abstract. In this article, we describe a procedure for systematically searching for shortest proofs in logical systems based on the inference rule condensed detachment. The procedure relies on applications of linked UR-resolution and uses demodulation to categorize derivations. Although the procedure is exhaustive in nature—and therefore realistically is applicable only to relatively small problems—it is shown to overcome the obstacles to finding shortest proofs presented by ordinary resolution-style theorem proving.

Keywords: shortest proofs, condensed detachment, linked UR-resolution

1. Introduction

This special issue of the *Journal of Automated Reasoning* is devoted to logical calculi based on the inference rule condensed detachment (CD):

i(s,t)	(major premise)
r	(minor premise)

t_σ, where σ is a most general unifier for terms r and s.

In particular, the focus of the articles is on proving theorems in these logical calculi using resolution-style automated reasoning programs such as OTTER [2]. The questions considered include determining the equivalence of various axiom systems, finding proofs to specific theorems, and finding proofs with certain specified properties. In this article, we consider the problem of finding *shortest* proofs.

CD problems are easily represented for resolution theorem provers. If we let `P(t)` represent the assertion that `t` is a theorem, then applications of condensed detachment can be implemented using hyperresolution and the following clause.

```
-P(i(x,y)) | -P(x) | P(y).
```

A CD derivation in OTTER consists of a sequence of input clauses representing axioms and hyperresolvents representing applications of

* This work was support in part by National Science Foundation grant no. CDA-9503064.

condensed detachment. The usual notion of *length* of such a derivation is the number of hyperresolvents appearing in the sequence (equivalently, the number of applications of CD). We are interested in finding shortest proofs in this sense.

If we let $I(t_1, t_2)$ denote an application of condensed detachment to major premise t_1 and minor premise t_2, then any CD derivation can be represented as an expression in terms of I and the axioms $\{A_1, A_2, ..., A_n\}$ used in the derivation. For example, $I(I(A_3, A_1), A_2)$ denotes the application of CD to axioms A_3 and A_1, followed by the application of CD to the result of the first application and axiom A_2. It is fairly straightforward to see that the length of a derivation is the number of *distinct* I subterms appearing in the corresponding derivation expression, including the expression itself. See Example 1.1.

As we consider the problem of finding shortest proofs, it will be convenient to have the following two additional measures for a given CD derivation.

$$\begin{aligned}
CDcount(A_i) &= 0 \quad 1 \leq i \leq n \\
CDcount(I(d_1, d_2)) &= 1 + CDcount(d_1) + CDcount(d_2) \\
\\
level(A_i) &= 0 \quad 1 \leq i \leq n \\
level(I(d_1, d_2)) &= 1 + max(level(d_1), level(d_2))
\end{aligned}$$

$CDcount$ is the total number of applications of condensed detachment in a derivation, without regard to the use of lemmas. That is, unlike the usual notion of derivation length, $CDcount$ is defined assuming a lemma is rederived each time it is used. The *level* of a CD derivation is the height of the corresponding derivation tree. This agrees with the usual notion of proof level.

Example 1.1. Consider the following two OTTER derivations of the same theorem.

First derivation:

```
1 [] -P(i(x,y)) | -P(x) | P(y).
2 [] P(i(i(x,y),i(i(y,z),i(x,z))))   # label("L1").
3 [] P(i(i(n(x),x),x))   # label("L2").
7 [hyper,1,2,2] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
8 [hyper,1,2,3] P(i(i(x,y),i(i(n(x),x),y))).
29 [hyper,1,7,8]
      P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
```

Second derivation:

```
1 [] -P(i(x,y)) | -P(x) | P(y).
```

```
2 [] P(i(i(x,y),i(i(y,z),i(x,z))))   # label("L1").
3 [] P(i(i(n(x),x),x))   # label("L2").
6 [hyper,1,2,2]
      P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
14 [hyper,1,6,6]
      P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
83 [hyper,1,14,3]
      P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
```

The derivation expressions corresponding to these derivations are, respectively, $I(I(L1,L1),I(L1,L2))$ and $I(I(I(L1,L1),I(L1,L1)),L2)$. The first derivation has length 3, which also is the size of the set

$$\{\ I(L1,L1),\ I(L1,L2),\ I(I(L1,L1),\ I(L1,L2))\ \}$$

of distinct I subterms appearing in the corresponding derivation expression. The *CDcount* of this derivation is 3, and the *level* is 2.

The second derivation also has length 3, which agrees with the size of the set

$$\{\ I(L1,L1),\ I(I(L1,L1),I(L1,L1)),\ I(I(I(L1,L1),I(L1,L1)),L2)\ \}$$

of distinct I subterms appearing the corresponding derivation expression. We note that subterm $I(L1,L1)$ gets counted only once in the length calculation, even though it appears twice in the derivation expression. The *CDcount* of this derivation is 4, and the *level* is 3.

A derivation expression can be viewed as a strictly binary tree; leaf nodes represent axioms, and nonleaf nodes represent applications of condensed detachment. As such, a CD derivation can be skewed (for example, $I(A,I(A,I(A,A)))$), or full (for example, $I(I(I(A,A),I(A,A)),I(I(A,A),I(A,A)))$). It follows that for any derivation D,

$$level(D)\ \leq\ CDcount(D)\ \leq\ 2^{level(D)}-1.$$

Proof length has the same bounds. That is,

$$level(D)\ \leq\ length(D)\ \leq\ 2^{level(D)}-1.$$

The upper bound for proof length comes from the case in which all of the applications of condensed detachment are distinct. This bound is reachable in theory—subject to successful unifications—if there is a sufficiently large set of input axioms. When there are few axioms, the number of distinct subterms that can appear in a single term is limited, and the upper bound can be tighter. Characterizing this bound—specifically, as a function of both level and number of axioms—is an interesting problem that we would like to study further.

We can use OTTER to construct the derivation expression for any CD derivation simply by adding a history argument to our representation of theorems. For example, if the input clauses of Example 1.1 are represented as

```
% condensed detachment (with derivation history)
-P2(i(x,y),z1) | -P2(x,z2) | P2(y,I(z1,z2)).

% axioms (with labels used for derivation histories)
P2(i(i(x,y),i(i(y,z),i(x,z))),L1).
P2(i(i(n(x),x),x),L2).
```

then any deduced clause will have its derivation expression as its second argument. This feature will be useful when we use OTTER to help categorize derivations.

In the following section, we consider the key obstacles to finding shortest proofs presented by ordinary resolution-style theorem proving. In Section 3, we show how applications of linked UR-resolution [5, 6] can be used to overcome these obstacles, at least in part. In Section 4, we describe how we can use demodulation and a modified representation to categorize proofs and help us identify shortest proofs. In Section 5, we describe, somewhat informally, how a proof found by linked UR-resolution can be converted, for readability, back into an ordinary condensed detachment proof. Section 6 contains a summary and comments about future work.

2. Obstacles to Finding Shortest Proofs

One might expect that ordinary breadth-first search (BFS) would suffice for finding shortest derivations. Unfortunately, the use of subsumption, which typically is required for effective theorem proving, can interfere. With ordinary subsumption, the second derivation of a clause will be forward subsumed by the first derivation, even if the second derivation ultimately leads to a shorter proof. As the following example shows, even the use of ancestor subsumption, in which a clause with a shorter derivation subsumes a logically equivalent clause with a longer derivation, does not suffice. That is, shorter subproofs do not necessary lead to a shorter proof.

Example 2.1. Consider the following two derivations of theorem T_x. Both derivations assume a set of axioms,

$$\{A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4, C_1, C_2, C_3, C_4, C_5, C_6\},$$

and depend on derived lemmas L_A and L_B. Each application of condensed detachment is given in terms of the function symbol I described previously.

First derivation:

1.	t_1	$I(A_1, A_2)$
2.	t_2	$I(t_1, A_3)$
3.	L_A	$I(t_2, A_4)$
4.	t_3	$I(B_1, B_2)$
5.	t_4	$I(t_3, B_3)$
6.	L_B	$I(t_4, B_4)$
7.	T_x	$I(L_A, L_B)$

Second derivation:

1.	t_5	$I(C_1, C_2)$
2.	t_6	$I(t_5, C_3)$
3.	t_7	$I(C_6, C_4)$
4.	L_A	$I(t_7, C_5)$
5.	L_B	$I(t_7, C_6)$
6.	T_x	$I(L_A, L_B)$

In the first case, the derivations of lemmas L_A and L_B each have length 3, and the derivation of theorem T_x has a total length of 7. In the second case, the derivations of L_A and L_B each have length 4, but the derivation of T_x has a total length of 6. In the first case, the derivations of the two key lemmas are disjoint, while in the second case, the derivations of the two key lemmas have 3 steps in common. For a shortest proof, we would want the versions of L_A and L_B with the longer derivations to subsume the versions with the shorter derivations, independent of which versions are generated first. Unfortunately, there currently is no known way to effectively choose subsumption direction to optimize proof length.

One obvious solution is simply to turn off subsumption, thereby permitting multiple derivations of the same lemmas (and proper instances of lemmas) to be considered independently. Although this approach works in theory, the cost of retaining and using instances and duplicate copies quickly becomes prohibitive.

Example 2.2. Consider the following axiom system for two-valued sentential calculus.

```
P(i(i(x,y),i(i(y,z),i(x,z))))  # label("L1").
P(i(i(n(x),x),x))              # label("L2").
P(i(x,i(n(x),y)))              # label("L3").
```

After just three levels of BFS, OTTER generates 344 clauses when subsumption is used and 2013 clauses when subsumption is not used. The disparity in the sizes of the respective clause sets grows dramatically at higher levels. We note, however, that the cost of simply generating the duplicates is not nearly as prohibitive as the cost of retaining these duplicates for further use. This observation is key to the approach considered in the next section.

3. A Strategy Using Linked UR-Resolution

Linked UR-resolution [5, 6] can be used to combine several applications of condensed detachment into a single linked inference step. One potential advantage of combining steps is that intermediate results are not ordered by the search strategy; condensed detachment is applied exhaustively within each application of linked UR-resolution, subject to user-specified limits. Such locally exhaustive searching can be particularly valuable when some clause that otherwise would not be considered by the search strategy—for example, because of its complexity—is needed for a key result. This all can be accomplished in OTTER by choosing linked UR-resolution as the inference rule and by including the following clause.

```
$BOTH([3]) | -P(i(x,y)) | -P(x) | P(y).
```

The `$BOTH([3])` literal indicates that the clause (condensed detachment) should serve both as a nucleus for applications of linked UR-resolution and as a linking clause. The `[3]` argument indicates that resolvents must be instances of the third "problem" literal, `P(y)`, which is consistent with the definition of condensed detachment. See [5] for more information about linked inference rules and their implementation in OTTER.

One problem with linking is that the application of linked inference rules is inherently computationally expensive. For example, let $dcount(n,k)$ denote the maximum possible number of CD derivations with $level \leq k$ starting from n axioms. This value can be defined with the following recurrence.

$$\begin{aligned} dcount(n,0) &= n \\ dcount(n,k) &= dcount(n,k-1)^2 + n \qquad (k > 0) \end{aligned}$$

Example 3.1. $dcount(n,k)$

For $n = 1$:

k	$dcount$	derivations
0	1	$\{A\}$
1	2	$\cup\ \{I(A,A)\}$
2	5	$\cup\ \{I(A,I(A,A)), I(I(A,A),A), I(I(A,A),I(A,A))\}$
3	26	$\cup\ ...$
4	677	
5	458,330	

For $n = 3$:

k	$dcount$	derivations
0	3	$\{A,B,C\}$
1	12	$\cup\ \{I(A,A), I(A,B), I(A,C), ..., I(C,C)\}$
2	147	$\cup\ ...$
3	21,612	

It follows that the cost of applying linked inference rules can be prohibitive for large levels. In [5], we discuss the use of user-settable limits to control applications of linked inference rules. The `max_ur_depth` parameter, for example, limits the longest chain of linking clauses that can appear in a single application of linked UR-resolution. Another parameter, `max_ur_deduction_size`, specifies the maximum number of binary resolution steps that can occur in a single application of linked UR-resolution. (For condensed detachment, the deduction size is naturally limited to $2^{n+2} - 2$ when the link depth is limited to n.)

If no `max_ur_depth` or `max_ur_deduction_size` limits are specified, every CD derivation can be implemented as a single application of linked UR-resolution. This can be prohibitively expensive, of course, but it does provide a way to search for shortest proofs without the interference of subsumption and without the overhead of retaining and using all of the intermediate clauses. As we will describe in the next section, we can use OTTER to both generate and help us identify shortest proofs.

In addition to the potentially large number of derivations that must be generated, there is another substantial cost associated with the search for these derivations. Under the current implementation of linked UR-resolution, the same *derivation* may be generated many times. In particular, when a clause is chosen by the search strategy as the focus of attention (the "given" clause), OTTER searches for all possible applications of linked UR-resolution using the given clause and all other available clauses (the "usable" clauses), subject to the user-specified limits placed on the search. Under the current implementation, a derivation will be generated once for every occurrence of the given clause in the derivation. This redundancy, which can be substantial, is a simple consequence of the algorithm used for generating linked UR-resolvents.

There is no known way to effectively remove this redundancy in the general case, that is, using the usual given/usable paradigm for generating lemmas in search of a proof. There is, however, a way to completely eliminate the redundancy in the special case of trying to generate an entire proof as a single linked inference step. The key is to initiate the search for derivations from the nucleus, which appears uniquely in any given derivation, and to avoid initiating the search from a linking clause, which may appear in several places in a derivation.

Example 3.2. Consider proofs of a theorem t in the system described in Example 2.2. The usual way to represent this problem to OTTER for proof with linked UR-resolution, is as follows.

```
list(usable).

   $BOTH([3]) | -P(i(x,y)) | -P(x) | P(y).

end_of_list.

list(sos).

   P(i(i(x,y),i(i(y,z),i(x,z))))  # label("L1").
   P(i(i(n(x),x),x))              # label("L2").
   P(i(x,i(n(x),y)))              # label("L3").

end_of_list.

list(passive)
```

```
   % negation of theorem t
   -P(t').

end_of_list.
```

The following alternate representation effectively eliminates the redundancy of generating identical derivations of t multiple times.

```
list(usable).

   $NUCLEUS([3]) | -go | -P(t') | PROOF.
   $LINK([]) | -P(i(x,y)) | -P(x) | P(y).

   $SATELLITE([])
      | P(i(i(x,y),i(i(y,z),i(x,z)))) # label("L1").
   $SATELLITE([])
      | P(i(i(n(x),x),x))             # label("L2").
   $SATELLITE([])
      | P(i(x,i(n(x),y)))             # label("L3").

end_of_list.

list(sos).

   % initiate the deduction
   go.

end_of_list.

list(passive)

   % proof found
   -PROOF.

end_of_list.
```

The nucleus is a strictly syntactic device for initiating and defining the termination of an application of linked UR-resolution. Condensed detachment appears as a linking clause only. The given clause, `go`, always resolves with a literal in the nucleus, thereby ensuring that there will be at most one orientation for the application. If we eliminate the `-PROOF` clause (or don't stop searching when a proof is found), then all possible derivations of t will be generated, subject to the `max_ur_depth` and `max_ur_deduction_size` limits that are in effect.

The previous example easily generalizes to arbitrary axiom systems and goal theorems. Unfortunately, there currently is no simple way to generalize this optimization for searches that deduce and use intermediate lemmas. In particular, there is no natural way, under the current implementation, to make new lemmas available for future applications of the linked inference rule while still preventing the type of redundancy being discussed. This is a topic for future work.

A direct correlation exists between the sizes associated with a CD derivation and the values used to characterize an application of linked UR-resolution. With the usual formulation—that is, without the special nucleus—the deduction size of a linked inference step equals 2 times the *CDcount* of the derivation, and the link depth of the deduction is one less than the *level* of the derivation. If we use the formulation with the special nucleus, the deduction size of the linked inference step equals two times the *CDcount* of the derivation plus two, and the link depth of the deduction equals the *level* of the deduction. It is useful to know these relationships when setting the `max_ur_depth` and `max_ur_deduction_size` limits in an OTTER input file.

The correlation between derivation length and the linking values follows from the property that $level(D) \leq length(D) \leq 2^{level(D)} - 1$ for any derivation D. Assuming the formulation with the special nucleus, if we want all possible proofs of $length \leq n$, we must set `max_ur_depth` to at least n, and `max_ur_deduction_size` must be set to at least

$$\begin{aligned} 2 * CDcount(n) + 2 \\ &= 2 * (2^n - 1) + 2 \\ &= 2^{n+1}. \end{aligned}$$

In summary, using the alternate representation described in this section, and setting parameters `max_ur_depth` and `max_ur_deduction_size` to values n and 2^{n+1} respectively, OTTER will find all possible CD proofs with $length \leq n$ of a given theorem. Hence, in order to find a shortest proof, it suffices to choose a value of n that yields at least one proof. Similarly, if no proof is found for some value of n, then the shortest proof, if there is one at all, necessarily has $length > n$.[1]

[1] Both of these criteria depend, of course, on the correctness of the implementation of linked UR-resolution, which is fairly complex. Although it is straightforward

In order to keep costs down—the cost of executing a search for large values of n can be prohibitive—it suffices to start with a reasonably small estimate for n and increase this value as necessary (that is, when no proof is found). Further, if no proof is found for some value of n, it suffices to find *any* proof of length $n+1$. This gives us some flexibility in searching at `max_ur_depth` $n+1$, for example, by using values smaller than 2^{n+2} for `max_ur_deduction_size`, which can reduce significantly the running time of an OTTER experiment.

4. Using Otter to Identify Shortest Proofs

In the preceding section, a procedure is suggested for systematically searching for shortest proofs. In this section we show how OTTER can be used to categorize derivations in order to help identify shortest proofs. Specifically, we show how a modified representation and a set of input demodulators can be used effectively to show for any derived theorem the derivation expression; the *level*, *CDcount*, and *length* of the derivation; and the set of distinct subterms appearing in the derivation. Because demodulation is destructive—terms are rewritten in place—some care is taken to avoid having to derive the list of subterms a second time for the proof length computation.

This all can be accomplished with the following new version of the linked UR-resolution nucleus,

```
% -P(t') represents the negation of the goal theorem
$NUCLEUS([3]) | -go
   | -P(t')
   | PDT(xd,Level(xd),CDcount(xd),Summary(Lemmas(xd))).
```

and the following list of input demodulators.

```
list(demodulators).

   % -----------------------------------------
   % Identify the axioms.
   % -----------------------------------------

   % example:  axioms L1, L2, and L3
   Axiom(L1) = $T.
   Axiom(L2) = $T.
```

to get an independent confirmation of a found derivation, it is not so easy to get an independent confirmation of the *absence* of a derivation.

```
Axiom(L3) = $T.

% -----------------------------------------
% Compute the level of a CD derivation tree.
% -----------------------------------------

Axiom(x) -> Level(x) = 0.
Level(I(x,y)) = $SUM(MAX(Level(x),Level(y)),1).

% not using $IF to ensure application to numbers only
$GE(x,y) -> MAX(x,y) = x.
$GE(y,x) -> MAX(x,y) = y.

% ----------------------------------------------
% Compute the total number of applications of CD.
% ----------------------------------------------

Axiom(x) -> CDcount(x) = 0.
CDcount(I(x,y)) = $SUM($SUM(CDcount(x),CDcount(y)),1).

% --------------------------------------------
% Extract a list of distinct lemmas from a CD
% derivation.
%
% This relies on an underlying lexical ordering
% of terms.
% --------------------------------------------

Axiom(x) -> Lemmas(x) = x.
Lemmas(I(x,y)) = l(I(x,y),l(Lemmas(x),Lemmas(y))).

% eliminate duplicates
l(x,x) = x.
l(x,l(x,y)) = l(x,y).

% bring duplicates together
l(l(x,y),z) = l(x,l(y,z)).
l(x,l(y,z)) = l(y,l(x,z)).
l(x,y) = l(y,x).

% ---------------------------------------------
% Count the distinct lemmas (i.e., proof length)
% ---------------------------------------------
```

```
    Summary(x) = [x,PL(PFlen(x))].

    Axiom(x) -> PFlen(x) = 0.
    PFlen(I(x,y)) = 1.
    PFlen(l(x,y)) = $SUM(PFlen(x),PFlen(y)).

end_of_list.
```

The `xd` argument in a `PDT` resolvent contains the relevant derivation expression. The `Level` and `CDcount` arguments demodulate, respectively, to the corresponding *Level* and *CDcount* values. The `Lemmas` subterm demodulates to the list of distinct derivation subterms appearing in the derivation expression. The resulting `Summary` subterm then demodulates to a pair consisting of the list itself along with the size of the list, which is the derivation length.

Example 4.1. The problem is to find the shortest proof that the formula $M5$,

$$i(i(i(v1,v2),v3),i(i(v2,i(v3,v5)),i(v4,i(v2,v5)))),$$

can be derived from the single axiom $M4$,

$$i(i(v2,i(v3,v5)),i(i(i(v1,v2),v3),i(v4,i(v2,v5)))),$$

for the implicational fragment of intuitionist logic [3, 4]. Using `max_ur_depth` 6 and `max_ur_deduction_size` 128, OTTER finds seven proofs of *length* 6:

```
% Length = 6, Level = 6, CDcount = 14
I(I(I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),M4),M4),
I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),M4)

% Length = 6, Level = 6, CDcount = 16
I(I(I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),I(M4,M4)),M4),
I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),I(M4,M4))

% Length = 6, Level = 6, CDcount = 20
I(I(I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),I(I(M4,M4),
I(M4,M4))),M4),I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),
I(I(M4,M4),I(M4,M4)))

% Length = 6, Level = 6, CDcount = 24
I(I(I(I(I(M4,M4),I(I(M4,M4),I(M4,M4))),I(I(M4,M4),
I(I(M4,M4),I(M4,M4)))),M4),I(I(I(M4,M4),I(I(M4,M4),
```

```
I(M4,M4))),I(I(M4,M4),I(I(M4,M4),I(M4,M4))))

% Length = 6, Level = 6, CDcount = 18
I(I(I(I(I(I(M4,M4),I(M4,M4)),M4),I(I(M4,M4),
I(M4,M4))),M4),I(I(I(I(M4,M4),I(M4,M4)),M4),
I(I(M4,M4),I(M4,M4)))

% Length = 6, Level = 6, CDcount = 20
I(I(I(I(I(I(M4,M4),I(M4,M4)),I(M4,M4)),I(I(M4,M4),
I(M4,M4))),M4),I(I(I(I(M4,M4),I(M4,M4)),I(M4,M4)),
I(I(M4,M4),I(M4,M4)))

% Length = 6, Level = 6, CDcount = 24
I(I(I(I(I(I(M4,M4),I(M4,M4)),I(I(M4,M4),I(M4,M4))),
I(I(M4,M4),I(M4,M4))),M4),I(I(I(I(M4,M4),I(M4,M4)),
I(I(M4,M4),I(M4,M4))),I(I( M4,M4),I(M4,M4)))
```

In addition to identifying properties of a derivation, we can automatically restrict attention to proofs having $length \leq k$ for some value k by including the conditional demodulator,

```
% --------------------------------------------
% Restrict to proofs with specified max length.
% --------------------------------------------

% example:  restrict to length <= 6
$GT(x,6) -> PL(x) = PL(TooLong).
```

and the following weight template for purging generated clauses.

```
assign(max_weight,99).

weight_list(purge_gen).

   weight(TooLong,999).

end_of_list.
```

The approach just described is designed to search for shortest proofs of a single theorem. Sometimes we are interested in finding the shortest (combined) proof of a set of theorems—for example, the derivation of one axiom system from another. We easily can extend our approach to account for the combined evaluation of a set of proofs. For example, let `-P(t1,xd1)` and `-P(t2,xd2)` be literals representing the negations of two theorems. The linked UR-resolution nucleus

```
$NUCLEUS([4]) | -go | -P(t1,xd1) | -P(t2,xd2)
      | PDT(M(xd1,xd2),
            Level(M(xd1,xd2)),
            CDcount(M(xd1,xd2)),
            Summary(Lemmas(M(xd1,xd2)))).
```

can be used to derive proofs for the two theorems and to combine the respective derivation expressions, `xd1` and `xd2`, into a single expression using the function symbol `M`. The demodulators

```
Level(M(x,y)) = MAX(Level(x),Level(y)).
CDcount(M(x,y)) = $SUM(CDcount(x),CDcount(y)).
Lemmas(M(x,y)) = l(Lemmas(x),Lemmas(y)).
```

then are used to evaluate the combined derivations.

5. Reproducing Linked Proofs As Ordinary CD Proofs

Once we have a linked derivation, it often is desirable to reproduce the proof as an ordinary CD derivation, that is, an ordinary hyperresolution derivation that explicitly includes the intermediate steps as lemmas. This can be accomplished in two steps as follows.

1. Extract from the derivation expression a list of demodulators that can be used to identify the desired applications of condensed detachment. This is easily done with a simple script or program.

2. Run the problem using ordinary hyperresolution and a representation that includes a history argument. Keep only the resolvents that are fully demodulated by the demodulators identified in Step 1. This is easily done with OTTER's weighting procedures.

Example 5.1. Let's say we want to reproduce derivation

$$I(I(I(L1,L1),I(L1,L1)),L2)$$

from Example 1.1. We can do this with the following formulation.

```
list(usable).

   % condensed detachment (with derivation history)
   -P2(i(x,y),z1) | -P2(x,z2) | P2(y,I(z1,z2)).
```

```
end_of_list.

list(sos).

   % axioms (with names used in derivation histories)
   P2(i(i(x,y),i(i(y,z),i(x,z))),L1).
   P2(i(i(n(x),x),x),L2).

end_of_list.

list(demodulators).

   % specify the desired derivation
   I(L1,L1) = c01.
   I(c01,c01) = c02.
   I(c02,L2) = c03.

end_of_list.

assign(max_weight,99).

weight_list(purge_gen).

   % delete any theorem with an undemodulated I term
   weight(I($(1),$(1)),999).

end_of_list.
```

The three demodulators "accept" derivations $I(L1, L1)$, $I(I(L1, L1), I(L1, L1))$, and $I(I(I(L1, L1), I(L1, L1)), L2)$ respectively. All other derivations will exceed the max_weight threshold and will be purged.

Example 5.2. Consider the following axiom system due to Frege (pp. 490–491 in [1]).

```
%  Frege's axioms A1, A2, and A3
P(i(x,i(y,x)))                         # label("A1").
P(i(i(x,i(y,z)),i(i(x,y),i(x,z))))     # label("A2").
P(i(i(x,i(y,z)),i(y,i(x,z))))          # label("A3").
```

The problem is to show that axiom A3 follows from axioms A1 and A2. Using a representation that includes a history argument, we can state this problem with the following clauses.

```
%  Frege's axioms A1 and A2
P2(i(x,i(y,x)),A1).
P2(i(i(x,i(y,z)),i(i(x,y),i(x,z))),A2).

% negation of axiom A3
-P2(i(i(a,i(b,c)),i(b,i(a,c))),xd).
```

The history argument in the negation of the theorem is changed to a universally quantified variable to ensure that any derivation of the theorem will suffice.

The procedure for finding shortest proofs yields a number of proofs of *length* 8, including one with the following derivation expression.

$$I(I(A2, I(A1, I(I(A2, A2), I(A1, A1)))), I(I(A2, I(A1, A1)), A2))$$

This derivation can be specified with the following set of demodulators.

```
list(demodulators).

   I(A2,A2) = C2.
   I(A1,A1) = C3.
   I(C2,C3) = C4.
   I(A1,C4) = C5.
   I(A2,C5) = C6.
   I(A2,C3) = C7.
   I(C7,A2) = C8.
   I(C6,C8) = C9.

end_of_list.
```

Using the strategy described at the beginning of this section, OTTER produces the following proof, which is converted to a pure CD proof simply by ignoring the history argument and demodulators.

```
1 [] -P2(i(x,y),xd)| -P2(x,yd)|P2(y,I(xd,yd)).
2 [] P2(i(x,i(y,x)),A1).
3 [] P2(i(i(x,i(y,z)),i(i(x,y),i(x,z))),A2).
4 [] -P2(i(i(a,i(b,c)),i(b,i(a,c))),xd).
5 [] I(A2,A2)=C2.
6 [] I(A1,A1)=C3.
7 [] I(C2,C3)=C4.
8 [] I(A1,C4)=C5.
9 [] I(A2,C5)=C6.
10 [] I(A2,C3)=C7.
11 [] I(C7,A2)=C8.
```

```
12 [] I(C6,C8)=C9.
13 [hyper,2,1,2,demod,6] P2(i(x,i(y,i(z,y))),C3).
14 [hyper,3,1,3,demod,5]
      P2(i(i(i(x,i(y,z)),i(x,y)),
           i(i(x,i(y,z)),i(x,z))),C2).
15 [hyper,3,1,13,demod,10]
      P2(i(i(x,y),i(x,i(z,y))),C7).
16 [hyper,15,1,3,demod,11]
      P2(i(i(x,i(y,z)),i(u,i(i(x,y),i(x,z)))),C8).
17 [hyper,14,1,13,demod,7]
      P2(i(i(x,i(i(y,x),z)),i(x,z)),C4).
18 [hyper,17,1,2,demod,8]
      P2(i(x,i(i(y,i(i(z,y),u)),i(y,u))),C5).
19 [hyper,18,1,3,demod,9]
      P2(i(i(x,i(y,i(i(z,y),u))),i(x,i(y,u))),C6).
20 [hyper,19,1,16,demod,12]
      P2(i(i(x,i(y,z)),i(y,i(x,z))),C9).
21 [binary,20.1,4.1] $F.
```

We note that the clause numbers are in sequence—that is, with no gaps. The procedure for mapping a linked UR-resolution proof to an ordinary hyperresolution proof ensures that only clauses participating in the proof are retained (and assigned clause numbers).

In some cases, because of the length of a derivation, it may be unwieldy to create an entire derivation expression explicitly as a single term. This is especially true for proofs that consist of several linked UR-resolution steps. In these cases, extracting an appropriate set of demodulators from a proof is more complicated, but it still can be done.

6. Summary

In this article, we presented a strategy for systematically searching for shortest proofs relying on the inference rule condensed detachment. The strategy, which is based on an application of linked UR-resolution, avoids many of the obstacles to finding shortest proofs presented by ordinary resolution-style theorem proving. We now are able to find shortest proofs we previously had been unable to find when relying on more traditional resolution strategies. The article also includes procedures for categorizing proofs and for mapping linked resolution proofs to ordinary hyperresolution.

We currently are attempting to generalize this work to be applicable to inference rules other than condensed detachment, including, for example, inference rules based on equality substitution. We also plan on looking more closely at redundancy—that is, coming up with an effective way to permit the use of lemmas in the search for a shortest proof, while avoiding the redundancy inherent in the current implementation of linked UR-resolution. Finally, we would like to derive tighter bounds for the possible length of a derivation as a function of derivation depth and the number of input axioms.

References

1. Kneale, W. and Kneale, M., *The Development of Logic*, Oxford: Clarendon Press, 1962.
2. McCune, W., OTTER 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, 1994.
3. Meredith, C., "A Single Axiom of Positive Logic", *Journal of Computing Systems* **1**, 1953, 169–170.
4. Ulrich, D., "New Single Axioms for Positive Implication", *Bulletin of the Section of Logic,* to appear.
5. Veroff, R., and Wos, L., "The Linked Inference Principle, I: The Formal Treatment", *J. Automated Reasoning* **8**(2), 1992, 213–274.
6. Wos, L., Veroff, R., Smith, B., and McCune, W., "The Linked Inference Principle, II: The User's Viewpoint", in R. Shostak (ed.), *Proceedings of the 7th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 170, Springer-Verlag, 1984, 316–332.