The Use of Lemmas in the Model Elimination Procedure

O.L. Astrachan and D.W. Loveland*
Department of Computer Science
Duke University
Durham, NC 27708-0129
919-660-6500
{ola|dwl}@cs.duke.edu

Abstract

When the Model Elimination (ME) procedure was first proposed, a notion of lemma was put forth as a promising augmentation to the basic complete proof procedure. Here the lemmas that are used are also discovered by the procedure in the same proof run. Several implementations of ME now exist but only a 1970's implementation explicitly examined this lemma mechanism, with indifferent results. We report on the successful use of lemmas using the METEOR implementation of ME. Not only does the lemma device permit METEOR to obtain proofs not otherwise obtainable by METEOR, or any other ME prover not using lemmas, but some well-known challenge problems are solved. We discuss several of these more difficult problems, including two challenge problems for uniform general-purpose provers, where METEOR was first in obtaining the proof. The problems are not selected simply to show off the lemma device, but rather to understand it better. Thus, we choose problems with widely different characteristics, including one where very few lemmas are created automatically, the opposite of normal behavior. This selection points out the potential of, and the problems with, lemma use. The biggest problem normally is the selection of appropriate lemmas to retain from the large number generated.

1 Introduction

The Model Elimination (ME) procedure was defined in the 1960's [17, 18] nearly coincident with the Resolution procedure [22]. There was little early

*This work is supported in part by NSF grants CCR-8900383 and CCR-9116203.

experimentation with ME, so the procedure received relatively little attention. (However, it is the basis of the SL-Resolution procedure which played an historical role in the beginning of Logic Programming). A major implementation of ME was completed in the early 1970s by Fleisig *et al.* [14] that did provide a direct comparison with the Unit Preference – Set-of-Support refinement of Resolution (Wos *et al.* [29]), as both procedures were implemented with the same underlying inference tools. Individual wins were found for both methods but no evidence to prefer ME over the known refinements of Resolution was forthcoming. The notion of lemma, introduced in the first papers of ME, was implemented by Fleisig *et al.* with a resulting indifference to their usefulness. The paper included this statement: "The use of lemmas usually was detrimental, although some short refutations were obtained using lemmas. The poor performance is due to the lack of selection rules for lemmas and is aggravated by the depth-first nature of search." ([14], p. 135). This paper will strongly refute the conclusion of that paper regarding the value of lemmas within ME. We report on results that to date are only obtainable within the ME framework by the use of lemmas, and the theorems proven include some not yet provable by Resolution techniques. We now understand that the notion of only direct competition with Resolution is oversimplistic; this procedure tends to do relatively well on non-Horn problems where Resolution methods now employed are not focused. Unlike chess, theorems are a very diverse lot and different proof methods may excel in different areas.

As is well-known to many in this research area, the idea of ME was kept alive by the work of Mark Stickel. Most notably, Stickel exploited the fact that ME is an extension of SLD-Resolution (Prolog) to develop the Prolog Technology Theorem Prover (PTTP) [24, 25]. In the late 1980's almost simultaneously three groups took this one step further, building on the architecture (the Warren Abstract Machine (WAM)) developed for Prolog that the logic programming community had extended to parallel machines. These projects were *PARTHENON* (CMU) [9], *PARTHEO* (Munich) [23], and *METEOR* (Duke) [4, 3]. (The Munich [16] and Duke efforts included sequential provers also.) The work reported here has been implemented on one of the *METEOR* family of ME provers.

What has changed in 20 years that makes the lessons of the Fleisig *et al.* paper invalid in part? A fair number of things: maybe most important is the WAM architecture ideas, but also important are use of iterative deepening (introduced to ME by Stickel), a sophisticated implementation exploiting the WAM ideas, vastly more powerful yet cheaper computers, and careful use of techniques to restrict lemma creation and use.

We look in depth at three examples, theorems proven with use of lemmas that would otherwise not be proved using ME. Two theorems are from the theory of continuous functions and one is a collection of three challenge problem from the 1960's not all solved automatically until now. (Two of the three latter challenge problems were solved in the weak fully automated mode; that is, some initial restructuring of the problem was done to force generation of lemmas. This

was done by splitting the problems into two and six cases respectively, without knowledge of the problem solution. The details are presented later. We mention here that the problem has subsequently been solved in (strong) fully automated mode by the Semantic Hyper-Linking prover of [11]. To our knowledge no other prover has succeeded on this problem in non-interactive mode.) Because the purpose of this paper is to look in-depth at the nature of lemma use, and because other papers [5, 3], contain results of METEOR on standard problems of the Automated Theorem proving (ATP) community, we omit such listings here. For example, the Astrachan and Stickel paper [5] discussing caching in ME includes two tables of results and a brief discussion on lemma use. (Caching is not applicable for the examples we treat here.)

The Model Elimination procedure is a linear input procedure; that is, one parent clause is the preceding clause and the other clause is an input clause. This is the key property of SLD-resolution that allows compilation of Prolog programs, elegantly implemented in the WAM architecture. ME can use the WAM architecture structure and also enjoy a high inference rate. However, like Prolog, ME does suffer high search redundancy due to the depth-first search mode of the WAM architecture. (ME does use iterative deepening rather than pure depth-first search, however.) Lemmas shorten proofs, thus reducing this redundancy. We show by example that the compression of proof achieved with lemma use can be striking, and the gain occurs on "real" (vs. toy) problems. We note that there is a version (restriction) of resolution that is in 1-1 correspondence with ME. The Resolution version is a linear input procedure except for a restricted "ancestor resolution" operation where resolution between two deduced clauses is necessary. (See the TOSS procedure in [19] [1] .) Thus, the ME procedure can be regarded as an encoding of the TOSS Resolution restriction using 2-sorted logic (framed and unframed literals), but it should be noted that ME is not, strictly speaking, a Resolution procedure. This is highlighted by the fact that Linear Input Resolution is complete only for a small extension of Horn clause logic whereas ME is complete for all of first-order logic.

This paper is organized as follows: In Sections 2 and 3 we briefly describe the ME proof procedure and the realization of this procedure in *METEOR*. In Section 4, Section 5, and Section 6 we describe the successful application of lemmas to three theorems none of which can be proved without lemmas using *METEOR* [2]. We conclude with a short summary in Section 7.

2 The Model Elimination Procedure

Like Resolution, Model Elimination is a refutation procedure, with the formula to be refuted presented in conjunctive normal form. Skolem functions are

[1] The TOSS procedure corresponds to a slightly different ME procedure.

[2] None of these problems yields a proof in less than 24 CPU hours using "vanilla" ME without lemmas.

used to eliminate existential quantifiers. (See [10] or [19] for details on formula preparation; [19] contains a full description of the ME procedure.) We view the clauses as sets of literals and the formula as a set of clauses. The corresponding entity to a derived clause in Resolution is the (derived) chain, an ordered list of literals where each literal is of one of two classes, an A-literal or B-literal. ME, as currently defined, has two basic operations[3], extension and reduction. The basic operation of extension is like the Resolution operation with retention of one of the literals resolved upon. The retained literal is promoted to a A-literal. (Intuitively, the A-literal is a type of ancestor literal.) We give a succinct presentation of the ME operations and lemma mechanism here, followed by an example. The appendix gives an expanded presentation of the procedure.

The first chain in a deduction is an ordered input clause with all literals classified as B-literals.

Below and in general we oversimplify the reference to occurrences of literals. In particular, we refer to a literal in successive chains as the same literal when in fact the later literal may be an instantiation of its "parent" literal. This simplification should cause no confusion.

The *extension* operation glues a (shortened) input clause, ordered by user choice, to the left of the current chain if the leftmost B-literal of the chain unifies with the complement of some literal of the input clause. The new chain is the instantiation of the current chain by the unifier with the unifying literal of the input clause dropped, and the other unifying literal (the leftmost literal of the current chain) promoted to A-literal. Newly added literals are B-literals and other literals retain their classification from the current chain[4]. All leftmost A-literals, if any, are removed back to the leftmost B-literal.

The *reduction* operation removes the leftmost B-literal of the current chain if it can be unified with the complement of an A-literal of the chain. The new chain is the instantiation of the current chain by the unifier with the leftmost B-literal missing. Again, all leftmost A-literals, if any, are removed back to the leftmost B-literal.

The creation of lemmas occurs when the leftmost A-literals are removed, at the end of the extension and reduction operations. Here we use only unit lemmas, although a more general notion of lemma, allowing multiliteral lemmas, is defined in [19]. The lemma chains are created as the complements of the removed A-literals, but only some A-literals can produce a lemma. To create only unit lemmas the eligibility mechanism is simple. In a reduction operation, the A-literal that complements the leftmost B-literal is the reduction A-literal. During a reduction step, all A-literals strictly to the left of the reduction A-literal are marked. Every A-literal in a newly created clause inherits the mark, if any, of its parent A-literal. Any A-literal unmarked at the time of removal

[3]Early papers [17, 18] had three operations.

[4]Traditionally, ME chains grew "to the right" but Prolog conventions of replacing the leftmost literal have influenced recent implementations. We choose to follow the convention used in the *METEOR* implementation.

creates a lemma. Not all generated lemmas are retained; retention depends on a number of criteria discussed later, some under control of the user. If the lemma is retained it acts exactly as if it were a unit input clause regarding extension. (Lemmas are subject to restrictions and modifications not shared by input clauses, however.)

The ME refutation in Figure 1, adapted from [19, 2], illustrates the ME mechanisms.

1.	p(X) -q(Y)	input clause
2.	-p(X) r(Y)	input clause
3.	-p(a) -r(Y)	input clause
4.	q(X) p(a)	input clause (and goal chain)
	begin proof	
5.	p(Y) [q(X)] p(a)	extension with 1
		variable renaming, clause 1
6.	r(Z) [p(Y)] [q(X)] p(a)	extension with 2
		variable renaming, clause 2
7.	-p(a) [r(Z)] [p(Y)] [q(X)] p(a)	extension with 3
8.	[r(Z)] [p(a)] [q(X)] p(a)	*prior to removal of A-literals*
	p(a)	reduction
	unit lemmas formed	
	-p(a)	
	-q(X)	
9.	□	extension with lemma -p(a)
	Proof completion if lemma mechanism is not used:	
9.	-r(Y) [p(a)]	extension with 3
10.	-p(X) [-r(Y)] [p(a)]	extension with 2
11.	□	reduction

Figure 1: ME refutation

In Figure 1 the unit lemmas are created during the final stage of the reduction that yields chain p(a) at step 8. The reduction creates A-literal p(a) from parent p(Y) and the subsequent removal of A-literals p(a) and q(X) create the lemmas. Removal of r(Z) cannot create a unit lemma.

In Figure 2 we present proof trees associated with the example, one proof tree for the proof without a lemma use and one that incorporates the lemma use. In a proof tree, each clause used in an extension is represented, here by its clause number. The goal clause is listed as root of the proof tree. The literals of a clause are represented by the child nodes of the clause, where the clause (actually, the appropriate chain of the clause) that is used in the extension on that literal is named. Reduction is shown by a backwards arrow from a node labeled Rn, for the nth reduction. The arc to an ancestor node of the reduction is labeled by the same Rn to indicate which is the A-literal of the reduction. The proof tree gives sufficient information to reconstruct the proof without backtracking; the instantiations of the variables are determined during

the tracing of the proof using the proof tree.

The proof tree on the right side of Figure 2 is a method of displaying lemma use and also representing a proof tree without lemma use. The leftmost box encloses the subdeduction that defines the lemma and the other occurrences of that box in the deduction indicate positions where the lemma is used. As such, the box would be replaced by a single node. The subdeduction in the box provides a subtree that could occur if the lemma device is not used. (Note that the tree to the left shows that a slightly different deduction actually occurred when the lemma device was not used.) This reporting device is used later in the paper for deductions of considerably larger size, and does give a graphic feeling of the saving in proof size realized by lemma use.

The reduction arrow in the boxes is somewhat confusing in this instance because the reduction is to the A-literal that creates or calls the lemma. For the right side box the reduction arrow refers back to the literal that calls the lemma, for that is the valid A-literal of reduction if the lemma is replaced by its deduction.

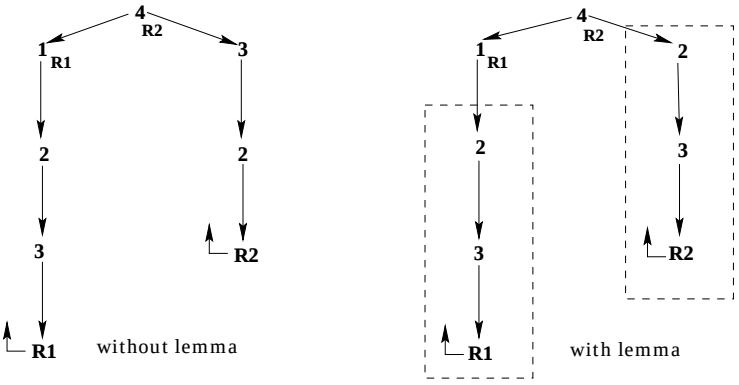

Figure 2: proof trees

3 The METEOR Architecture

We provide here only a brief description of the *METEOR* architecture. A full description can be found in [2]. *METEOR* is written in C and runs on workstations and in parallel and distributed computing environments. The same search engine is used in both the sequential and parallel setting. A discussion of the architecture of *METEOR* with emphasis on the parallel and distributed modes, appears in [3]. Iterative deepening is used to ensure completeness of the search strategy in all cases.

METEOR is designed to conform to the principles of the Warren Abstract Machine (WAM) [1], the *de facto* standard for Prolog implementations. How-

ever, clauses are not compiled into WAM code, but into a data structure that is then interpreted at runtime by either a sequential or parallel search engine.

Several different depth measures can be used in bounding a potential proof during an iterative deepening search. We report on four measures here, these are described in Table 1.

measure	**description**
D_{inf}	inference depth — bound total number of inferences in proof tree
D_{Alit}	A-literal depth — bound depth of proof tree
D_{roll}	rollback-reduction — combination of D_{inf} and D_{Alit} (preference given to reductions)
D_{weight}	weight depth — weight clauses (per branch of proof tree)

Table 1: depth measures employed in *METEOR*

The measures D_{inf} and D_{Alit} are commonly used in ME provers and are the default measures in *PTTP* and *SETHEO* respectively. As a bound on the entire proof tree, the use of D_{inf} ensures that a minimal-length proof (in terms of extensions and reductions) is found. D_{Alit} was also used in the first implementation of ME [14]. At any point in a proof search one can tell the current depth with respect to the D_{Alit} measure by counting the number of A-literals in the current chain, whereas for the D_{inf} measure it is the total number of extensions and reductions needed to derive the current chain. (Of course, one does not include subsearches removed by backtracking.) Note that the depth of a proof tree corresponds to the number of A-literals in the longest chain of the proof; this depth is the minimal D_{Alit} depth for which this proof tree can be found.

For the example deduction at the end of the last section, when no lemma is used the D_{inf} is 7 and D_{Alit} is 3. By this we mean that the proof can be found in the depth stated, and that a depth bound less than that stated would not permit the proof shown to be found. This can be seen from the given derivation or by consulting the appropriate proof tree. Note that for the proof tree the D_{inf} measure is determined by counting the nodes in the proof tree (excluding the root), and the D_{Alit} is determined by counting the depth of the proof tree with the root having depth 0. For the deduction using the lemma the D_{inf} is 5 and the D_{Alit} is 3. For the latter depth we assume that the left branch is searched first; if the right branch were searched first then the D_{Alit} is 2.

When D_{roll} is employed, depth is calculated as for D_{inf} regarding extension, but a successful reduction operation results in "rolling back" the depth to that depth in force when the A-literal of the reduction was introduced to the chain as a B-literal. The measure was discovered by accident, and has no clear intuition for its definition; we simply noted it worked well in many cases. One observation regarding its success is that it charges for extension by unit clauses, e.g. lemmas, (whereas D_{Alit} has no charge) yet gives added resources to explore the consequences of a reduction as does the D_{Alit} measure, indeed often more

resources than D_{Alit} gives. For the example deductions of the last section the D_{roll} is 4 for both lemma use and no lemma use. None of these measures is uniformly superior to the others nor does it appear that simple syntactic criteria can be used to determine which measure is more appropriate (see [2]).

D_{weight} is used in conjunction with any of these depth measures rather than as a replacement for them. Input clauses are annotated with an integral weight that contributes to the weight of a chain when the clause is used. The weight of a chain is the sum of the weight of all clauses whose literals "appear" in the chain. When D_{weight} is used it is incremented in an iterative manner just as other depth bounds are incremented in an iterative deepening search.

4 Minimum Value Theorem

We first consider a problem where the lemmas are very effective. The theorem we consider is the Minimum Value Theorem (MinVT). This is also known in the ATP literature as AM8 [27].

Theorem: (MinVT) *A continuous function in a real closed interval* $[a, b]$ *attains its minimum in the interval.*

In Table 2 we give a readable rendition of the axioms for MinVT. The numbers on the left refer to the clause numbers for these axioms as they are given in [27] and in the order submitted to *METEOR*.

In Figure 3 we give the axioms as submitted to *METEOR*. The clause numbering relates the input clause to its counterpart in Table 2. The "nocontra" suppresses the contrapositive of the dichotomy axiom (axiom 1) from being generated as it is clearly not needed and is a very costly axiom to include, as is the dichotomy axiom itself. Line 16 is for naming Skolem functions, for which we do not permit self nesting. The semantics of the problem made it likely that no function needed to be so nested.

In Figure 4 and Figure 5 we present in proof tree format the proof found by *METEOR*. The proof tree is given in a coded form with the lemmas labeled but unwound so that the full structure of the tree is apparent.

Figure 4 (on the left) shows the proof actually discovered by *METEOR*. It is a proof involving seven input clauses and four lemmas. Using depth measure D_{inf}, this would be a proof of depth 10. Actually, it took D_{inf} depth 12 (it can be done in D_{inf} depth 11) to discover the proof because some lemmas used in the proof required that depth to be discovered. Figures 4 (on the right) and 5 present the full proof, with each inference labeled by a lemma label or by a clause number. The dotted arrows at the bottom of the proof tree to the right in Figure 4 link to the appropriate subtrees in Figure 5. The node labels beginning with "L" refer to lemmas, with the numbering determined by the proof recovery procedure (not the order of lemma generation); other node labels refer to the input clauses by number.

	the axioms for total order
0.	$x \leq x$
1.	$x \leq y \vee y \leq x$
2.	$x \leq y \wedge y \leq z \Rightarrow x \leq z$
	Roughly, $x = y \Rightarrow f(x) = f(y)$ because of the symmetry of the antecedent
3.	$x \leq y \wedge y \leq x \Rightarrow f(x) \leq f(y)$
	l is a minimum point in [a,l]
4.	$a \leq l$
5.	$l \leq b$
6.	$a \leq x \leq l \Rightarrow f(l) \leq f(x)$
	Any point greater than l has a smaller point with smaller f-value.
7-9.	$a \leq x \leq b \wedge l \leq x \Rightarrow\ a \leq q(x) \leq x\ \wedge f(q(x)) < f(x)$
	For all x, $h(x)$ is the smallest point in [a,b] s.t. $f(h(x)) \leq f(x)$.
10-13.	$a \leq b \wedge a \leq y \leq b \Rightarrow [a \leq h(x) \leq b \wedge f(h(x)) \leq f(x) \wedge (f(y) \leq f(x) \Rightarrow h(x) \leq y)]$
	For all x, there exists a point k(x) in [a,b] such that $f(k(x)) < f(x)$
	This is the negation of the theorem conclusion
14,15,17.	$a \leq x \leq b \Rightarrow\ a \leq k(x) \leq b\ \wedge\ f(k(x)) < f(x)$

Table 2: Axioms for Minimum Value Theorem (MinVT)

We made the point previously that lemmas are often discovered in earlier, "failed" parts of the search tree. That is very evident here from the proof trees. For example, some top lemmas are used only once in the proof and so, to be invoked as lemmas, they must have been established before the proof was found. (In particular, note lemmas L2, L3, and L4, labeled on the trees in Figure 4 with derivations continuing in Figure 5.) Lemmas used in the proof of these lemmas were clearly discovered by the time the more complex lemmas were obtained. Note that this does not mean that they were discovered earlier in the same proof tree iteration because lemmas are retained through successive iterations. Thus information from "later" in the tree can be used on early branches of the succeeding iterate. In general, iterative deepening gives a breadth-first form of search within which the search is depth-first.

The effectiveness of lemmas is dramatically made by noting that the depth of the proof given here, when D_{inf} (the depth measure used here) is employed with no use of lemmas, is over 600! Such search depth is well beyond the capability of any implementation of ME, or any other prover we know.

It should be pointed out that resolution does exactly this kind of compression, and that resolvents are lemmas also, in the sense of retaining intermediate results. Indeed, we noted that unit lemmas are resolvents from tree portions where reduction is not used. The difference, as noted before, is that our lemmas

```
0.  p(X,X)
    nocontra
1.  p(X,Y) | p(Y,X)
2.  -p(X,Y)| -p(Y,Z) | p(X,Z)
3.  -p(X,Y) | -p(Y,X) | p(f(X),f(Y))
4.  p(a,l)
5.  p(l,b)
6.  -p(a,X) | -p(X,l) | p(f(l),f(X))
7.  -p(a,X) | -p(X,b) | p(a,q(X)) | p(X,l)
8.  -p(a,X) | -p(X,b) | -p(f(X),f(q(X))) | p(X,l)
9.  -p(a,X) | -p(X,b) | p(q(X),X) | p(X,l)
10. -p(a,X) | -p(X,b) | p(a,h(X))
11. -p(a,X) | -p(X,b) | p(h(X),b)
12. -p(a,X) | -p(X,b) | p(f(h(X)),f(X))
13. -p(a,X) | -p(X,b) | -p(a,Y) | -p(Y,b) | -p(f(Y),f(X)) | p(h(X),Y)
14. -p(a,X) | -p(X,b) | p(a,k(X))
15. -p(a,X) | -p(X,b) | p(k(X),b)
16. skolem(h(1),q(1),k(1),f(1))
17. -p(f(X),f(k(X)))|-p(a,X)|-p(X,b)
```

Figure 3: Input clauses for Minimum Value Theorem (MinVT)

are optional regarding completeness. That does not mean they are optional regarding realizing an actual proof, as this case demonstrates. However, not being *a priori* concerned about having to retain almost every nonredundant intermediate result we do achieve much more trimming. In the first proof we obtained of MinVT, taking 171 secs., we created and retained just 593 lemmas while making about 393,000 successful inferences. The only constraint we had (besides the unit lemma constraint, which is universal for us but still a constraint) was a term-length stipulation of 10 on the lemmas and a disallowance of iterative nesting of any function. For speed considerations the term-length check is done only on the term replacing a variable at a substitution occurrence; thus the literal may contain many more constant and function letters than the bound indicates. This check can be done on literals in new chains as well as on lemmas, but here was applied only to lemmas. Note, by the way, that the nesting restriction applied here is almost totally safe for a first run as all functions but f are Skolem functions, for which one should never gain by nesting, and nesting f makes no heuristic sense given the problem nature. The sensitivity to term length is quite modest; if term-length is not limited at all, the same number of lemmas are retained. For this problem, the limit on nesting of the same function symbol is necessary in order to constrain the number of lemmas generated. (That is not always the case, however.) Thus we illustrate the point that we can trade off more search with input clauses for less retention

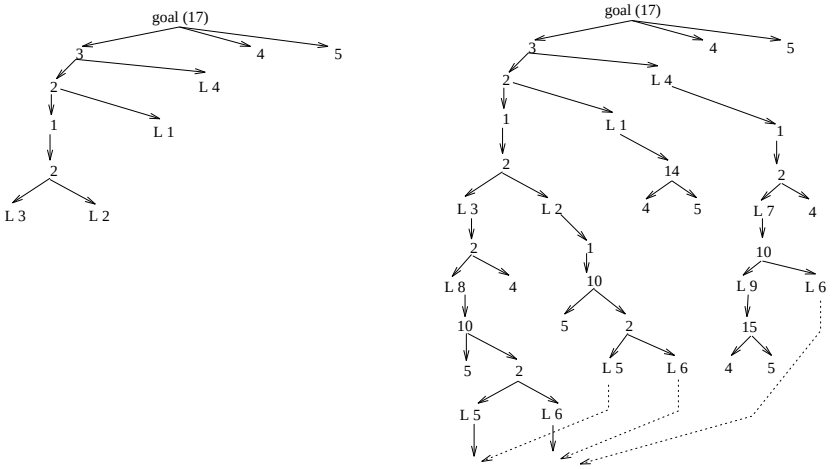

Figure 4: top-level proof of AM8 and partial unrolling of proof

of intermediate results than many strategies of resolution employ.

5 Wang Challenge Problem

It is the usual case that we generate too many lemmas and seek ways of trimming lemmas without removing those helpful to the proof. Therefore, it is an interesting situation to encounter a problem where too few lemmas are produced. This situation arose when we pursued a challenge problem set that had not been fully proven by an automated theorem prover, to our knowledge. In 1965, Hao Wang put forth several challenge problems for automated theorem provers, problems beyond the capability of theorem provers at that time [26]. Moreover, they proved difficult for automated provers for some time. The problem set we consider were labeled ExQ1, ExQ2, and ExQ3 by Wang, and were treated in the section "Examples from Quantification Theory". Although Wang suggests that for a more intuitive understanding of these examples, we should think of F as the membership relation ϵ, this suggestion did not achieve its purpose for us.

We state the three problems, which are formulas to be refuted in Table 3, followed, in Figure 6, by one set of axioms used by METEOR. In Figure 6, $r(x, y)$ denotes $x = y$ and b replaces constant n of the axioms (the latter for purely historical reasons).

Problem ExQ1 has been solved by a number of systems, we suspect, because it is also known as "Wos 31", in [28]. As such it has been solved by *OTTER* [20], but because this is one of numerous problems that circulate under labels as meaningful as "Wos 31", it is not easily determined what provers have solved

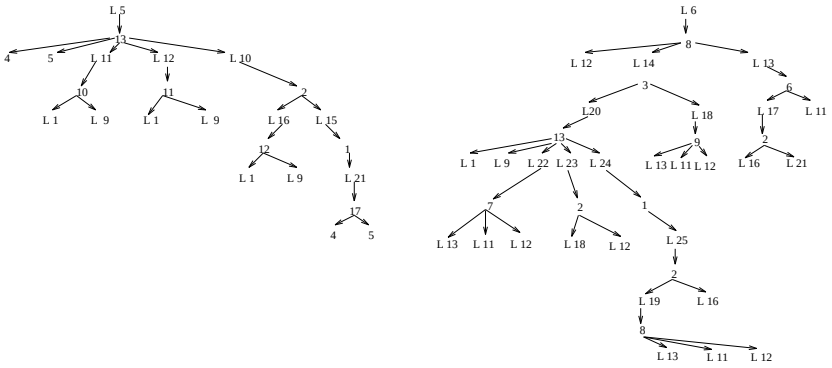

Figure 5: proof of AM8, lemmas unrolled

ExQ1: The conjunction of the following formulas is unsatisfiable.
1. $m \neq n$
2. $n \neq k$
3. $k \neq m$
4. $y = m \vee [Fym \equiv (\exists z)(z \neq m \wedge z \neq y \wedge Fyz \wedge Fzy)]$
5. $y = n \vee [\neg Fyn \equiv (\exists z)(z \neq n \wedge z \neq y \wedge Fyz \wedge Fzy)]$
6. $y = k \vee [Fyk \equiv (y = m \vee y = n)]$
ExQ2: Replace (2) and (3) above by
1. $(n = k \vee k = m)$
2. $y = j \vee (Fyj \equiv y = k)$
ExQ3: Remove (1) from ExQ2.

Table 3: Axioms for ExQ problems

this first problem. *OTTER* has not solved ExQ2 or ExQ3 although ITP, in some sense an ancestor of *OTTER*, did solve ExQ2.

We have solved it with some preconditioning on the manner of presentation of the problem; we broke it into cases so as to force generation of lemmas. As previously mentioned, the semantic hyper-linking theorem prover of Chu and Plaisted subsequently has provided a fully automated proof, the first to our knowledge [11]. (This problem is particularly well suited for their prover because it is highly non-Horn, like a logic puzzle. That is why it turns out to be poorly structured for us; the high number of reductions blocks the production of unit lemmas.)

A proof of Wos31 (ExQ1) was obtained using D_{Alit} without breaking the theorem into cases; using D_{weight} reduces the time to one-third of that when no weighting is employed. Note that our ExQ3 solution contains the ExQ1 case, in terms of the inequalities for the three basic constants. As for ExQ1 we again use D_{Alit} for case 6 of ExQ3; for all other cases we use D_{roll} . It is interesting

```
0   -r(m,b)                                                    13  -p(Y,k),r(Y,k),r(Y,m),r(Y,b)
1   r(b,k)                                                     14  -r(Y,m),p(Y,k),r(Y,k)
2   r(Y,j),-r(Y,k),p(Y,j)                                      15  -r(Y,b),p(Y,k),r(Y,k)
3   r(Y,j),r(Y,k),-p(Y,j)                                      16   r(X,X)
4   -p(Y,m),-r(f(Y),m),r(Y,m)                                  17  -r(X,Y),r(Y,X)
5   -p(Y,m),-r(f(Y),Y),r(Y,m)                                  18  -r(X,Y),-r(Y,Z),r(X,Z)
6   -p(Y,m),p(Y,f(Y)),r(Y,m)                                   19  -r(X,Y),-p(X,Z),p(Y,Z)
7   -p(Y,m),p(f(Y),Y),r(Y,m)                                   20  -r(X,Y),-p(Z,X),p(Z,Y)
8   -p(Y,V),-p(V,Y),r(Y,m),p(Y,m),r(V,m),r(V,Y)                21  -r(X,Y),r(f(X),f(Y))
9   -r(g(Y),b),r(Y,b),p(Y,b)                                   22  -r(X,Y),r(g(X),g(Y))
10  -r(g(Y),Y),r(Y,b),p(Y,b)                                   23  lex(b,m,k,j,f(X),g(X))
11  p(Y,g(Y)),r(Y,b),p(Y,b)                                    24  skolem(f(1),g(1))
12  p(g(Y),Y),r(Y,b),p(Y,b)                                    25  r(Y,b),-p(Y,b),r(V,b),
                                                                   r(V,Y),-p(Y,V),-p(V,Y)
```

Figure 6: Input clauses for ExQ2, case 1

that all other cases are solved using D_{roll} and not easily solved (in some cases) using D_{Alit}.

When we first ran ExQ2, very few lemmas were generated and no proof was obtained. Indeed, an overnight run (14 hours) yielded only 6 lemmas. In contrast, for ExQ1 87 lemmas were created in one successful run. Our experience with the success of lemmas made us consider immediately how more lemmas could be generated. Because unit lemma generation is inhibited by reductions we decided to introduce unit clauses to allow more frequent proof branch termination by unit extension rather than reduction. Although we did not understand the proof of the theorem (and still don't) it did seem apparent that ExQ1 and ExQ2 were in effect dealing with cases of equality and inequality of three constants; k,m,n. We broke up the problem into the cases listed in Table 4 and were able to prove each theorem. Note that the case definition for ExQ2 follows immediately from the axioms added to define ExQ2. Statistics for these cases are provided in Table 4.

ExQ2 cases	
case 1	$n = k$
case 2	$k = m$
ExQ3 cases	
case 1	$m \neq n, n = k, k \neq m, m \neq j$
case 2	$m \neq n, n = k, k \neq m, m = j$
case 3	$m \neq n, n = k, k = m, m \neq j$
case 4	$m \neq n, n = k, k = m, m = j$
case 5	$m \neq n, n \neq k, k = m$
case 6	$m \neq n, n \neq k, k \neq m$

problem	**depth measure**	**time (secs)**	**# inferences**	**# lemmas**
ExQ1	D_{Alit}	389	670,707	49
ExQ2 (case 1)	$D_{\text{roll}} D_{\text{weight}}$	4,562	14,462,880	47
ExQ2 (case 2)	D_{roll}	4.6	12,683	26
ExQ3 (case 1)	D_{roll}	108	170,185	67
ExQ3 (case 2)	D_{roll}	14	19,916	104
ExQ3 (case 3)	D_{roll}	0.14	240	12
ExQ3 (case 4)	D_{roll}	0.19	275	15
ExQ3 (case 5)	D_{roll}	9	12,741	26
ExQ3 (case 6)	D_{Alit}	408	683,226	49

Table 4: cases and statistics for ExQ1, ExQ2, and ExQ3

As reported in Table 4 for ExQ2(case 1), we used D_{weight} , although the same result was obtained by D_{roll} alone in twice the time (9784 sec.) and double the inference total (29,886,276 inferences). The weights are the entries seen at the end of each input clause. It is interesting to view a case of use of weights; these weights were entered once prior to any weighted run, using an "algorithm" that accounts for clause length primarily, but with high penalty for known problem axioms such as the equality axioms. There is a small adjustment in general for the number of free variables in a clause. The major gain in using the weights here is the control of the equality axioms.

We comment on other aspects of the METEOR listing of the axiom set in Figure 6. The lex predicate is used to order constant symbols for use in demodulation, and the skolem predicate lists the Skolem functions as mentioned in the previous section. Neither restriction was used in the runs reported in this section; demodulation was not used at all in these reported runs. (We obtained so few lemmas that use of demodulation was not a useful option, but the problem was initially specified so that demodulation could be used.) The input axioms are seen to differ in small ways from the natural translation of the Wang problem specification. Besides the replacement of b for n, there is the appearance everywhere of $m = k$ instead of $k = m$, for example. We had done our experiments with the axiom set obtained from [27] before tracing the origin of the problem in preparation for this paper. We have chosen to accept the variance from the original problem statement since [27] has become a standard source for problems, including this problem set.

We now feel that we have discovered a new technique for ME, that of breaking problems into cases to generate useful lemmas. It may even pay to do this when many lemmas are generated, by severely limiting the natural lemmas being produced if they do not yield a proof, and trying to generate new lemmas by the case method. It is clear that this is not (yet) an automatable technique, as other tries at this have not been useful. That is, we tried to see if the discovery of a proof for ExQ1 could be speeded up by introduction of cases. However, all runs of ExQ1 using added cases resulted in increased time to proof. But we have seen that the technique of forcing lemma creation is fruitful. How widely applicable the idea is awaits further experience.

6 Bledsoe Challenge Problem

The third problem we consider is actually a collection of variants on a problem investigated by Bledsoe starting in the early 1970's. Using a natural deduction style theorem prover with some special heuristics for limit theorems, Bledsoe and Boyer [7] proved that the sum of two continuous functions at a limit point is the limit of the sums at that point. They assigned the label LIM+ to this problem. The theorem has since been proven on the *STR+VE* prover of Hines and Bledsoe [8, 15], a general theorem prover with some special devices for

handling inequalities on the real line. Bledsoe recognized the difficulty of the theorem for the uniform provers and issued a challenge set of problems based on the LIM+ problem [6]. There are five different first-order axiom sets designated as the challenge set; two make explicit use of the equality predicate with one formulation requiring use of paramodulation or another rule that builds-in equality.

Using *METEOR* we have been able to prove the first three formulations of the challenge problem. The lack of a full built-in equality mechanism (no mechanism such as paramodulation) precludes an attempt on the fifth formulation of the problem. The fourth formulation, using equality without any strong equality restriction or simplification mechanism stronger than the unadorned demodulation mechanism for lemmas that we do have, is too much for *METEOR* at this point. The first three formulations use the single predicate $\leq$ (also used in the MinVT theorem discussed in Section 4). Several uniform provers have now obtained the third formulation of the challenge problem but *METEOR* was the first to achieve success at this level to our knowledge. (See Digricoli [13, 12] regarding proofs of the fourth and fifth formulations done in a partially interactive mode.)

Our purpose here is to report on the experience of *METEOR* with lemma use on this problem set. The third formulation is the hardest problem where *METEOR* has succeeded, where by "hardest" we mean that the strongest set of restrictions we have ever used were needed to obtain a proof. One proof requires over 20 hours on a SUN SparcStation 2; shorter proofs required some insight as to the proof so do not count as legitimate datapoints regarding the capability of *METEOR* as a fully automated theorem prover. Proof attempts using proof knowledge are used to test various constraints in shorter time periods. We will make clear which experiments use proof insight when such results are presented.

In Figure 7 we give the clauses used in the three formulations, in roughly the order that *METEOR* took them as input (read from left-to-right and top-to-bottom). The clause numbers are those of the original presentation by Bledsoe in [6]. *METEOR* normally re-orders clauses by number of literals (with preference for unit clauses), with attention to degree of generality also considered. The item `\n` at the end of each clause assigns a clause weight `n`, as discussed later.

We do not give the original non-clausal formulation of the clause set because the meaning of most clauses is self-evident, once the constants are interpreted. The predicate lt denotes $\leq$, the function ab denotes absolute value, $d1$ and and $d2$ are Skolem functions for delta regions (see below), ha is one-half, pl is $+$, ng is negative, xs is a Skolem function for exception points in the delta regions, and min is, of course, minimum. Clauses 3 and 4 give the continuity condition for function f and g respectively. For example, clause 3 is a direct translation of Equation 1

$$\forall \epsilon > 0,\ \exists \delta > 0 \text{ s.t. } \epsilon > 0 \wedge \mid x - a \mid \leq \delta \ \Rightarrow \mid f(x) - f(a) \mid \leq \epsilon \qquad (1)$$

```
%clause 5                                % clauses 1,2
-lt(e0,0)\1                              lt(E,0),-lt(d1(E),0)\3
                                         lt(E,0),-lt(d2(E),0)\3

% clause 12                             % clause 8
lt(X,0),-lt(ha(X),0)\3                   lt(ab(pl(X,Y)),pl(ab(X),ab(Y)))\2
                                         % clause 8.1
                                        -lt(pl(ab(X),ab(Y)),Z), lt(ab(pl(X,Y)),Z)\2
% clauses 3,4
lt(E,0),-lt(ab(pl(Z,ng(a))),d1(E)), lt(ab(pl(f(Z),ng(f(a)))),E)\8
lt(E,0),-lt(ab(pl(Z,ng(a))),d2(E)), lt(ab(pl(g(Z),ng(g(a)))),E)\8

% clause 6
lt(ab(pl(xs(D),ng(a))),D),lt(D,0)\2

% clause 9.1                             % clause 10.1
lt(min(X,Y),X)\6                         lt(min(X,Y),Y)\6
% clause 9.2                             % clause 10.2
-lt(X,Y),lt(X,min(X,Y))\8                -lt(Y,X),lt(Y,min(X,Y))\8

% clause 9.11                            % clause 10.11
-lt(Z,min(X,Y)),lt(Z,X)\8                -lt(Z,min(X,Y)),lt(Z,Y)\8

% clause 10.3                            % clause 11.3
lt(X,0),lt(Y,0),-lt(min(X,Y),0)\6        -lt(X,ha(Z)), -lt(Y,ha(Z)), lt(pl(X,Y),Z)\6

%  clause 14                             % clause 15
 lt(X,Y),lt(Y,X)\12                      -lt(X,Y),-lt(Y,Z),lt(X,Z)\12

 % clause 7.1
 -lt(ab(pl(pl(f(xs(D)),ng(f(a))),pl(g(xs(D)),ng(g(a)))),e0), lt(D,0)\1
 % clause 7.2
 -lt(pl(ab(pl(f(xs(D)),ng(f(a)))),ab(pl(g(xs(D)),ng(g(a)))),e0), lt(D,0)\1
```

Figure 7: clauses for Bledsoe challenge problems

The goal clauses, 7.1 and 7.2, give the negation of the theorem we assert (in conjunction with clause 6). For example, clause 7.1 states

$$\delta > 0 \;\Rightarrow |\;\; [f(X_\delta) - f(a)] \;+\; [g(X_\delta - g(a)] \;\; | > \epsilon_0 \qquad (2)$$

Clause 7.1 is a more difficult goal to achieve because of the need to use the triangle inequality. However, that change and the related need for clause 8.1, are not significant changes. Formulation 2 is not much harder than formulation 1. The introduction of the dichotomy axiom (axiom 14) and especially the transitivity axiom (axiom 15) along with axiom changes for *min* makes formulation 3 a considerably harder problem.

We list the three formulations by clause set, but will here only study the third formulation. The first two problem variations *METEOR* could handle relatively

straightforwardly. The clauses for all three formulations are:

formulation 1 clauses 1, 2, 3, 4, 5, 6, 7.2, 9.11, 10.11, 10.3, 11.3, 12. The goal is clause 7.2.

formulation 2 clauses 1, 2, 3, 4, 5, 6, 7.1, 8.1, 9.11, 10.11, 10.3, 11.3, 12. The goal is clause 7.1.

formulation 3 clauses 1, 2, 3, 4, 5, 6, 7.1, 8, 9.1, 9.2, 10.1, 10.2, 11.3, 12, 14, 15. The goal is clause 7.1.

We include some data on runs for the first two formulations with little discussion; see Table 5. In the lemmaizing run, only ground lemmas are stored and only ground subgoals are extended by lemmas. For reasons not important here these combined restrictions are labeled "ground generalizations only".

thm	**search parameters** time (*secs*) and number of inferences					
	D_{Alit}		D_{weight}†		lemmaizing‡	
one	1222.7	2,854,399	3.94	14,682	26.37	36,232
two	3859.8	12,294,133	9.76	43,980	10242.9	11,682,410

†using D_{Alit} **and** D_{weight}
‡ground generalizations only

Table 5: time and inferences for formulations one and two

We also give the proof tree for the second and third formulations in Figure 8 with the proof of the second formulation on the left. The dotted boxes surrounding subtrees denote lemma occurrences as discussed below.

6.1 The Third Formulation

We now list the proof devices and restrictions used to tackle the third formulation. Many have been used here only, so our experience with them is limited. Others are more common, and we can comment on our more general experience with them.

1. We use a combination of depth measures, D_{Alit} and D_{weight}. This has been tried often, usually obtaining considerably improved performance. This is one of the few times that the proof has not been obtained without this device. (This is an oversimplification in that a proof has been obtained using a combination of D_{roll} and D_{weight} but only under optimal limits on other parameters. It was considerably more time-consuming than with D_{Alit} and D_{weight}, so more general attempts were not tried.)

2. Only ground lemmas are stored, and these are only used when no instantiation of the calling goal occurs. This trims the branching factor to one at this node.

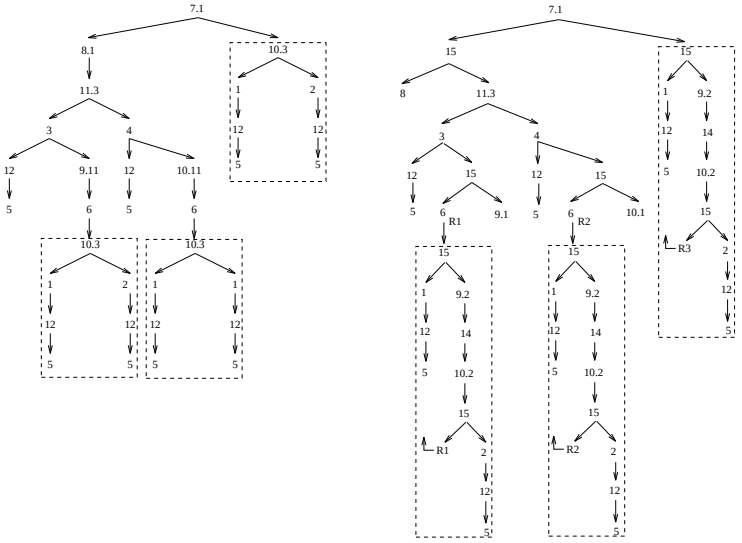

Figure 8: proof trees for second and third formulations

3. Lemma literals are limited in number of symbols allowed.

4. The number of occurrences of any specific input clause allowed in a proof is controlled. *METEOR* is very sensitive to this parameter, so we could not set it by uninformed guess and realize a proof. For our no-proof-knowledge proofs *METEOR* increments the upper bound on the clause occurrence count as a function of current depth according to a ratio set by the user. The ratio needed is moderately insensitive to user setting in terms of obtaining a proof at all, but the time needed to realize the proof is strongly sensitive to that ratio.

5. Literals containing terms of certain forms are not retained. The exclusions range from obvious redundancies to possibly strong exclusions. We have results for several combinations of exclusions.

6. Reduction is restricted to B-literals whose source is the transitivity axiom. This restriction is more natural than it sounds, for reasons stated below.

7. The goal clause is reordered from the usual rule of ordering. There are only two literals so there is a binary choice, but normally we would expand the more constrained literal first. This is a significant deviation, because we rarely had to have to rotate through the literals of the goal clause, yet in this case no proof has been obtained using the usual literal as the literal first expanded.

We now expand on some of the above.

The depth measure D_{weight} utilizes the clause weight that appears appended at the end of each input clause. The depth measure functions as does D_{Alit} except the clause weight is used to increment the count instead of the unit increase used by D_{Alit}. This depth measure may be used in conjunction with each of the other measures; the proof development terminates appropriately when any depth count is exceeded, so one gets the conjunction of the depth criteria.

The weights used in the clauses were guessed once and not changed, following a prescribed pattern. Some adjustment from a strict formula occurred. For example, the min axioms constitute one-third of the axioms, but are not likely to constitute one-third of the clauses used in a proof, so are weighted higher. The initial "guess" did not use proof knowledge, so is considered a non-sensitive constraint.

Controlling the number of occurrences of each input clause was necessary for the any proof to succeed. In actuality it is control of dichotomy and transitivity that is crucial; experiments show that we can make quite safe guesses about upper bounds for all other axioms if the transitivity and dichotomy axioms are limited to the optimal count. We have solved the problem with iterative incrementing on all clauses, but this is effectively equivalent to binding only dichotomy and transitivity (clauses 14 and 15) as shown in Line 6 of Table 7 which indicates limits on clause use of optimal plus twenty except for dichotomy and transitivity. When iterative deepening on clause limits is employed the user still must pick a ratio of clause-limit increment to depth bound increment, but this is less sensitive than guessing the clause count. Data on Line 7 of Table 7 increases the count by one for every two levels of increase in the D_{Alit} count.

Literals can be excluded from retention when certain terms are created upon unification. This is an action at the chain level, not just at the lemma level. Thus this can effect completeness "in the small". This avenue is taken because of the large number of proof branches explored that seem to expand from along very questionable routes, dealing with terms that are either unlikely to occur in the proof or are redundant with other terms. The terms excluded range from clearly safe for exclusion to perhaps quite risky. We stress that we do obtain proofs of the theorem excluding only the very safe exclusions, but performance is definitely enhanced with more risky exclusions. In Table 6 we list the exclusions used.

1. d1(d2(X))	2. d2(d1(X))	3. d1(d1(X))	4. d2(d2(X))
5. d1(pl(X,Y))	6. d2(pl(X,Y))	7. min(min(X,Y),Z)	8. min(X,min(Y,Z))
9. d1(ab(X))	10. d2(ab(X))	11. ha(pl(X,Y))	12. min(X,ab(Y))
13. min(ab(X),Y)	14. min(pl(X,Y),Z)	15. min(X,pl(Y,Z))	16. min(X,X)
17. ha(ab(X))	18. min(0,X)	19. min(X,0)	

Table 6: terms excluded in Bledsoe challenge problems

We used four classes of risk in the exclusion heuristic: very safe, safe, some risk, more risk. In the very safe category are the nested Skolem functions, terms such as $d1(d2(X))$. Also included is the term $min(X, X)$. Still considered safe is the term $ha(ab(X))$, because $ab(ha(X))$ is not excluded. Likewise for $ha(pl(X, Y))$, because $pl(ha(X), ha(Y))$ is still included. That is slightly riskier perhaps because the symbol count has increased and there is a limit on the total number of symbols allowed per literal. (The limit used was either 11 or 15, about equally used, but 20 symbols still allowed a proof.) Others in Table 6 are riskier; we discuss this further in the next paragraph. Line 3 of Table 7 gives data for a run using very safe exclusions only. (The phrase "ex. 1-4, 16" means that terms 1-4 and 16 are excluded terms for that run.) Lines 4 and 5 provide data using other subsets of the terms in Table 6. The column labeled *att.infs* gives attempted inferences, which are potential inferences identified prior to the unification attempt.

run	**statistics**					
	description of run	time (sec)	infs. ($\times 10^3$)	att.infs. ($\times 10^3$)	# of lemmas	lem.infs. ($\times 10^3$)
1.	optimal (termsize=15)	27858	23,146	66,485	115	590
2.	optimal (termsize=20)	23095	23,052	66,346	153	601
3.†	very safe (ex. 1-4,16)	66609	65,871	196,968	186	1,136
4.†	ex. 1-4,7-10,12,14,16	49206	47,861	143,409	133	1,037
5.†	ex. 1-4,9,10,16	64368	61,499	186,207	180	1,124
6.	limits + 20 (not 14,15)	75066	75,243	198,660	131	1,291
7.	iter. deep.‡	152077	129,419	355,250	181	2,443

†termsize limit of 15 symbols
‡Sparc ELC, increase all limits by 1 every other stage

Table 7: altering limits on extensions for third Bledsoe problem

We now briefly consider some of the riskier exclusions we considered. Excluding $min(min(X, Y), Z)$ and similar terms may seem very risky, but we noticed a proliferation of such terms and it led to asking if they were needed. Note that terms of this type could still arise through instantiation of variables. Here exclusion means that unification cannot instantiate a variable with an excluded term. To illustrate this, the goal $-lt(min(e0, e0), min(X, e0))$ can be refuted with the exclusion of all terms in Table 6. In obtaining the proof the term $min(min(e0, e0), e0)$ is generated when X in the goal is instantiated to $min(e0, e0)$. Thus, an excluded term's appearance is not forbidden, just the use of a term that embeds the excluded term as a substitution instance. The reader can see this by first extending with clause 9.2 followed by extension with clause 9.1. Even though completeness is endangered this is a reasonable action to take and this type of step may be needed to get very difficult theorems. We are adverse to this not because it endangers completeness if the exclusions are risky, but because we have been developing the thesis that basic ME should be

hidden and the user should play with the lemmas. We give in Table 8 data on runs with exclusions, and note again that the exclusions are not needed for the proof.

run	**statistics**						
	description of run	time (sec)	infs. ($\times 10^3$)	att.infs. ($\times 10^3$)	# of lemmas	lem.infs. ($\times 10^3$)	% succ. lem. infs
1.	all limits	23324	23,144	66,994	65	590	33
2.	D_{roll}, all limits	38663	37,928	116,069	71	1,081	33
3.	ex. 1-17	38144	38,450	116,421	71	987	20
4.	ex. 1-10,12-16,18	23533	23,709	68,196	65	590	33
5.	ex. 1-8,18	43228	38,601	110,147	75	664	22
6.	ex. 1-4,7,8,16-18	42101	37,633	107,665	75	659	22
7.	ex. min terms	26529	31,290	89,898	65	661	31
8.†	ex. 1-4,7-10,12,14,16	49206	47,861	143,409	133	1,037	21
9.†	ex. 1-4,9,10,16	64368	61,499	186,207	180	1,124	12
10.†	ex. 1-4,16	66609	65,871	196,968	186	1,136	12
14.‡	trm.size=15	27858	23,146	66,485	115	590	48
15.‡	trm.size=20	23095	23,052	66,346	153	601	48

†term-size limit = 15
‡otherwise as in line 1

Table 8: performance characteristics for the third formulation

That reductions are restricted to B-literals from the transitivity axiom actually came from noting that the simpler proofs for formulations 1 and 2 did not use reductions. That puts this in the category of "cheating" by our standards. Even though it was not based on knowing the proof of formulation three, it is not likely in the real world that simpler versions of the problem are around to break in our heuristics. Even though that is how we arrived at this guess, there is reason to guess that way again in similar situations. That is, it is a guess we would make in other similar situations. The reason is that we have seen many proofs having some non-Horn clauses where nevertheless reduction is not needed. So, after our initial tries that tells us the problem is hard, we would first try without reduction. Then we would try as here, allowing it on transitivity, because they often do use it. This is the first time that a problem has strongly benefited from this restriction; we have simply made the observation about when reduction occurs in other problems. More experience is needed to tell whether the pattern we just have suggested really holds.

The interchange of literal order within the goal is the hardest to justify on uniform grounds. It works this way and not the other way. Usually, starting with the more instantiated literal first is the winning strategy. The normal ordering works in the first two formulations; the proof can be obtained without lemma use starting with the more instantiated literal. The proof is impossible to find without lemmas when the literal $lt(D, 0)$ is expanded first. The situation reverses even for these easier formulations when lemmas are tried. The reason

is clear from the proof tree. In Figure 8 we see that if one follows the second literal (the right branch) then the key lemma, in the dotted line box, is reached in D_{Alit} depth 8 while pursuit down the other branch requires depth 13. (Recall that D_{Alit} is proof tree depth, so the reader can check this directly by counting branch lengths in the proof trees of Figure 8).

We note that the variable D in the goal is instantiated by the key lemma as follows: $-lt(min(d1(ha(e0)), d2(ha(E0))), 0)$. This instantiation gives the delta found in the typical proof found in analysis texts: for a given epsilon (ϵ_0) the corresponding delta is the minimum of the deltas (δ_1 and δ_2) corresponding to half of the given epsilon. The min axioms are necessary for discovering this instantiation. However (and somewhat surprisingly), if the min axioms are removed from the input clause set a proof is obtained in less than 600 seconds with no limits on clause use. However, in this case "lt(d1(ha(e0)),0) or lt(d2(ha(e0)),0)" is deduced, giving less information than is provided when the min axioms are included, but in significantly less time.

7 Summary

Plaisted [21] has shown that ME with unit lemmas and caching is one of the few goal-sensitive procedures that has a polynomial search time within the propositional Horn clause setting in a suitable sense. The result assumes that ME is used with iterative deepening search, the normal mode for METEOR. Elsewhere (see Astrachan and Stickel [5]) data is given on use of unit lemmas and caching in the Horn clause domain. Here we have presented examples in the non-Horn setting (where caching is not implemented nor easily designed) but where we have demonstrated that unit lemmas are still very effective. We have considered examples illustrating very different situations: the nearly ideal, the insufficient lemma supply situation (where case analysis provided the needed lemmas), and a barely adequate lemma supply (where only one deep lemma is used and not even created until the correct proof is under development). This wide scope of situations we believe demonstrates the usefulness of unit lemma addition to the Model Elimination procedure. We hope to achieve enough knowledge about lemma use that henceforth we will view the basic ME procedure as a black box and the collection of lemmas as the data the user views and manipulates.

8 Appendix

We give a more formal definition of the Model Elimination(ME) procedure as used by METEOR. A fuller treatment of the ME procedure(s) is given in [19].

A *chain* is a sequence of literals, each literal an A-literal or a B-literal. (That is, a chain is an ordered clause with two types of literals.)

An *input chain* is a chain of B-literals, obtained by placing an ordering

on a clause from a conjunctive normal form (CNF) formula to be tested for unsatisfiability. Clauses from the CNF formula are called *input clauses*. A input clause of n literals from the CNF formula must be represented by at least n chains, where each literal of the input clause is leftmost in at least one input chain.

The empty chain is denoted by $\Box$.

If l is the leftmost literal in chain C we write $(C - \{l\})$ to represent the chain C with the leftmost occurrence of l removed. There will be no occasion to remove a literal other than the leftmost literal, or a sequence of literals leftmost. In the latter case we regard the literals as being removed one at a time as the literal becomes leftmost in an intermediate chain.

A chain is *preadmissible* iff

1. complementary literals are separated by an A-literal;
2. no B-literal is to the left of an identical A-literal;
3. no A-literal is identical or complementary to another A-literal.

A chain is *admissible* iff it is preadmissible and the leftmost literal is a B-literal. In practice we will sometimes not enforce condition (2); that can be a user option.

Let C_1 be an admissible chain and let C_2 be a variable-disjoint input chain. If there exists a most-general-unifier(mgu) σ of the leftmost literal l_1 of C_1 and the complement of the leftmost literal l_2 of C_2, then the *extension* operation extends C_1 by C_2 to form chain C_3 by promoting the leftmost literal of $C_1\sigma$ to an A-literal and placing $(C_2 - \{l_2\})\sigma$ to the left, the literals of $(C_2 - \{l_2\})\sigma$ arranged in any order the user chooses. All literals in $C_3\sigma$ inherit the type of their parent literal in C_1 and C_2 with the noted exception of the promoted A-literal. If the chain that results is not preadmissible then the extension is not defined. If the chain that results is a preadmissible chain with an A-literal leftmost (i.e., nonadmissible) then the leftmost A-literals are removed back to the leftmost B-literal, which yields an admissible chain.

Let C be an admissible chain. If there exists a mgu σ of the leftmost literal l of C and an A-literal of C, then the *reduction* operation yields chain $(C - \{l\})\sigma$. If the chain that results is not preadmissible then the reduction operation is not defined. If the chain that results is a nonadmissible preadmissible chain then the A-literals to the left of the leftmost B-literal are removed.

An ME derivation is a sequence of chains composed of an input chain as first (goal) chain, followed by derived chains, each either an extension of the preceding chain by some input chain or a reduction of the preceding chain.

An ME refutation is an ME deduction of the empty chain.

Lemmas, an optional property of ME, are created when A-literals are removed in the transformation from preadmissible to admissible chain. A lemma is an informal label for either a lemma clause that results from the process of

lemma creation, or the chains that are created from the clause. Here we consider only unit lemmas so the distinction is pedantic. From the lemma clause only one chain is created, consisting of the single B-literal and regarded as an ordered clause. A lemma chain is treated as an input chain after generation (if it passes filters that may eliminate it; these are generally user-controlled options).

The lemma clause is the complement of the A-literal removed when the A-literal is eligible to create a lemma. The eligibility criterion is exactly as stated in the body of the paper but repeated here for completeness. This definition is only correct for unit lemmas. In a reduction operation, the A-literal that complements the leftmost B-literal is the reduction A-literal. During a reduction step, all A-literals strictly to the left of the reduction A-literal are marked. Every A-literal in a newly created clause inherits the mark, if any, of its parent A-literal. Any A-literal unmarked at the time of removal creates a lemma clause.

References

[1] H. Aït-Kaci. *Warren's Abstract Machine A Tutorial Reconstruction.* MIT Press, 1991.

[2] O.L. Astrachan. *Investigations in Model Elimination Based Theorem Proving.* PhD thesis, Duke University, 1992. (also technical report CS-1992-21).

[3] O.L. Astrachan. METEOR: Exploring model elimination theorem proving. *Journal of Automated Reasoning*, 13(2):283–296, 1994.

[4] O.L. Astrachan and D.W. Loveland. METEORs: High performance theorem provers using model elimination. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe.* Kluwer Academic Publishers, 1991.

[5] O.L. Astrachan and M.E. Stickel. Caching and lemmaizing in model elimination theorem provers. In Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction.* Springer Verlag, 1992.

[6] W.W. Bledsoe. Challenge problems in elementary calculus. *Journal of Automated Reasoning*, 6(3):341–359, 1990.

[7] W.W. Bledsoe, R. Boyer, and W. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3:27–60, 1972.

[8] W.W. Bledsoe and L. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. In *Proceedings of the Fifth Conference on Automated Deduction*, pages 281–292. Springer-Verlag, 1980.

[9] S. Bose, E. Clarke, D.E. Long, and S. Michaylov. Parthenon: A parallel theorem prover for non-Horn clauses. *Journal of Automated Reasoning*, 8, 1992.

[10] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[11] H. Chu and D. A. Plaisted. Semantically guided first-order theorem proving using Hyper-linking. In A. Bundy, editor, *Proceedings of the Twelfth Conference on Automated Deduct ion*, Lecture Notes in Artificial Intelligence 814, pages 192–238. Springer-Verlag, Berlin, June 1994.

[12] V. J. Digricoli. The Rue theorem-proving system: the complete set of LIM+challenge problems. *Journal of Automated Reasoning*, 12:241–264, 1994.

[13] V.J. Digricoli and E. Kochendorfer. Lim+ challenge problems by rue hyper-resolution. In Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, pages 239–252. Springer Verlag, 1992.

[14] S. Fleisig, D. Loveland, A. Smiley, and D. Yarmash. An implementation of the model elimination proof procedure. *Journal of the Association for Computing Machinery*, 21:124–139, January 1974.

[15] L.M. Hines. The central variable strategy of str$\dot{+}$ve. In Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction.* Springer Verlag, 1992.

[16] R. Letz, S. Bayerl, J. Schumann, and W. Bibel. SETHEO—a high-performance theorem prover. *Journal of Automated Reasoning*, 8:183–212, 1992.

[17] D.W. Loveland. Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery*, 15(2):236–251, April 1968.

[18] D.W. Loveland. A simplified format for the model elimination procedure. *Journal of the Association for Computing Machinery*, 16(3):349–363, July 1969.

[19] D.W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, 1978.

[20] W. McCune. Otter 2.0. In Mark Stickel, editor, *Proceedings of the Tenth International Conference on Automated Deduction*, pages 663–664. Springer Verlag, 1990.

[21] D.A. Plaisted. The search efficiency of theorem proving strategies. In Alan Bundy, editor, *Proceedings of the Twelfth International Conference on Automated Deduction*. Springer Verlag, 1994.

[22] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.

[23] J. Schumann and R. Letz. PARTHEO: A high performance parallel theorem prover. In *Proceedings of the Tenth International Conference on Automated Deduction*, pages 40–56, 1990.

[24] M.E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. In *Proceedings of the Eighth International Conference on Automated Deduction*, pages 573–587. Springer-Verlag, 1986.

[25] M.E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:343–380, 1988.

[26] H. Wang. Formalization and automatic theorem-proving. In *Proceedings of IFIP Congress 65*, pages 51–58, 1965. Washington, D.C.

[27] T.C. Wang and W.W. Bledsoe. Hierarchical deduction. *Journal of Automated Reasoning*, 3:35–77, 1987.

[28] G.A. Wilson and J. Minker. Resolution, refinements, and search strategies: A comparative study. *IEEE Transactions on Computers*, C-25(8):782–801, August 1976.

[29] L. Wos, G.A. Robinson, and D. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12, 1965.