ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

Experiments Concerning the Automated Search for Elegant Proofs

by

Larry Wos

Mathematics and Computer Science Division

Technical Memorandum No. 221

July 1997

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

Experiments Concerning the Automated Search for Elegant Proofs

Larry Wos

Abstract

The research reported in this technical report was spawned by the request to find an elegant proof (of a theorem from Boolean algebra) to replace the known proof consisting of 816 deduced steps. The request was met by finding a proof consisting of 100 deduced steps. The methodology used to obtain the far shorter proof is presented in detail through a sequence of experiments. Although clearly not an algorithm, the methodology is sufficiently general to enable its use for seeking elegant proofs regardless of the domain of study. The methodology relies heavily on the assistance of McCune's automated reasoning program OTTER. Of the aspects of proof elegance, the main focus here is on proof length, with brief attention paid to the type of term present, the number of variables required, and the complexity of deduced steps. The methodology is iterative, relying heavily on the use of three strategies: the resonance strategy, the hot list strategy, and McCune's ratio strategy. To provide some insight regarding the value of the methodology, I discuss its successful application to other problems from Boolean algebra and to problems from lattice theory. Research suggestions and challenges are also offered.

1. Background, Perspective, and Problem Origin

This technical report features a methodology for automating the search for elegant proofs; for pertinent material, see [Wos95a,Wos96a,Wos96c]. (For the actual proofs, or proofs of a similar flavor, see [McCune96] or either of the following two Web addresses, http://www.mcs.anl.gov/home/mccune/ar/monograph/ or http://www.mcs.anl.gov/people/wos/index.html.) The program that was used is McCune's OTTER [McCune90,McCune91,McCune94]. The methodology is iterative, (usually) beginning with a ''long'' proof and, if the methodology succeeds, culminating with a ''short'' proof. As shown in Section 3, later stages of the methodology use results obtained in earlier stages; for example, the proof steps of a proof completed in Experiment *j* are used to guide the search for a shorter proof in Experiment *k*, where *j* is less than *k*.

For me, the most satisfying success reported here concerns the fulfilling of a request from my colleague McCune to find an elegant proof for a particular theorem from Boolean algebra, a proof to replace the 816-step proof he had found; the theorem is denoted by DUAL-BA-3 in the monograph [McCune96]. (Appropriate equations for studying this theorem are given in Section 2.) The methodology succeeded in finding a 100-step proof.

The search for elegant proofs has continuously played a key role in mathematics and in logic. Such a search can lead to the discovery of new, important relationships, and it can also lead to the formulation of significant concepts. Outside of mathematics and logic, the discovery of an elegant proof can lead to the design of an efficient circuit or the synthesis of an effective algorithm. Such can occur, for example, when the proof is (in effect) constructive, and the researcher extracts from it (perhaps by using an ANSWER literal) the desired circuit or algorithm.

Although what precisely makes a given proof elegant is subject to debate, five properties merit mention: proof length, term structure, variable abundance, deduced-step complexity, and compactness. Here I focus mainly on the first property (proof length), giving brief attention to the second through the fourth properties. As for the fifth property of elegance, compactness, I introduced and studied it in [Wos96a,Wos96c]. (Because of its newness, the concept merits immediate focus. By example, a proof of a theorem of the form *P* implies the **and** of *q*, *r*, and *s* is compact if and only if it is a proof of exactly one of *Q*, *R*, or *S*, implying that the other two proofs are subproofs.) The likelihood of finding a proof that is elegant with respect to one or more of the cited five properties can be sharply increased if an automated reasoning program is heavily relied upon, a program such as OTTER.

Throughout this technical report, the first of the five properties, *proof length*, has a rather precise meaning, namely, the number of deduced steps explicitly present in the proof. The qualifier ''rather'' is present because intermediate steps resulting from the use of demodulation (for canonicalization) are not counted in proof length. Compared with the typical use in mathematics, the treatment of proof length in logic is more likely to be precise. The explanation rests with the frequent practice in mathematics of omitting many obvious steps (for example, those arising from applying symmetry of equality). In fact, on many an occasion, I have heard a logician say that mathematicians only outline proofs. In contrast, in various areas of logic (such as equivalential calculus [Kalman78,Wos95b]), all deduced steps are explicitly presented. Further, various areas of logic require the use of a specific inference rule, a rule such as condensed detachment [Kalman83,Wos95b]. Seldom is a specific inference rule cited in a mathematics paper or book.

The second property of elegance, *term structure*, refers to the type of term present in a proof. For example, does the proof (say, from group theory) contain terms of the form *inverse*(*inverse*(*t*)) for some term *t*, or does the proof (say, from many-valued sentential calculus) contain terms of the form *n*(*n*(*t*)) where *n* denotes negation, or does the proof contain terms in which nested divide instructions occur? Just as a proof may offer added elegance because of being terse, so also may it offer elegance by avoiding some type of term.

The third property of elegance, *variable abundance*, refers to the number of distinct variables required by one or more deduced steps of the proof. For example, the proof might require (for one of its deduced steps) the use of five distinct variables. All things being equal, requiring a smaller number of distinct variables adds to elegance.

The fourth property of elegance, *deduced-step complexity*, is concerned with the length as measured in symbols of the formulas, equations, or clauses present in the proof. (I sometimes use the term ‘‘clause’’ in a less strict fashion to refer to equations and the like written in various notations acceptable to OTTER.) Exclusive of commas or parentheses, a proof may derive some of its elegance from avoiding the use of any deduced steps that are complex (with respect to length). Indeed, when a proof contains formulas or equations that appear to be ‘‘messy’’ in that a lengthy array of symbols is encountered, the scientist often evinces disappointment, and sometimes comments on the lack of ‘‘elegance’’.

Of the four properties of elegance discussed in this technical report, the first (proof length) presents the most difficulty. The concern for proof length presents the type of challenge that is not directly addressable with an automated reasoning program and (apparently) is not effectively addressable with an algorithm. On the other hand, (as will be seen) the concern for the second through the fourth properties of elegance is directly addressable with a program such as OTTER and, at least in theory, can be attacked algorithmically. (The fifth property of elegance, compactness, also presents difficulty, not admitting an algorithmic attack that is practical.)

To attack the tough problem of finding ‘‘short’’ proofs, OTTER offers various means—although far from producing an algorithm; see Section 2. What becomes clear (in Section 3, where the sequence of experiments illustrating the basic methodology is presented) is the narrowness of the path that leads to success. For example, a change from the value 9 to the value 10 in the assignment of but one input parameter results (sometimes) in the completion of *no* proofs, in contrast to the completion of a proof marking significant progress.

2. The Target Theorem and an Arsenal of Weapons

Next in order are two items: a presentation of the specific theorem that serves as the target, and a discussion of the various weapons OTTER offers for attempting to reach the target. The discussion sets the stage for viewing the sequence of experiments (in Section 3) that culminated in the completion of a 100-step proof to replace the 816-step proof serving as the starting point. In addition, in contrast to the indirect attack on the first property of elegance, the discussion shows how OTTER can be used to directly address the second, third, and fourth cited properties of elegance.

The following equations capture the theorem (DUAL-BA-3) to be proved, where *x@* denotes the complement of *x* and where the two inequalities arise from, respectively, negating the theorem to be proved and negating its dual.

```
x = x.
x * (y + z) = (x * y) + (x * z).
x + (y * z) = (x + y) * (x + z).
x + x@ = 1.
x * x@ = 0.
(x * y@) + ((x * x) + (y@ * x)) = x.
(x * x@) + ((x * z) + (x@ * z)) = z.
(x * y@) + ((x * y) + (y@ * y)) = x.
(x + y@) * ((x + x) * (y@ + x)) = x.
```

```
(x + x@) * ((x + z) * (x@ + z)) = z.
(x + y@) * ((x + y) * (y@ + y)) = x.
(A * B) + B != B | $Ans(A2).
(A + B) * B != B | $Ans(A4).
```

The theorem under study asserts that, excluding the two equations in which != occurs, the given set of equations is an axiomatization of Boolean algebra. (Earlier work [McCune96] showed that it is sufficient to derive either of the two absorption laws, whose respective negations are captured by the two equations in which != occurs.) The context of the problem was the search for a Boolean algebra axiomatization consisting of two independent self-dual equations. In fact, the equations lead directly to such an axiomatization by applying a method due to Padmanabhan and Quackenbush [Padmanabhan73]. Each equation has length 1103, measured in symbol count.

McCune had succeeded in obtaining a proof with OTTER, a proof of length 816, counting five steps resulting from ‘‘copy’’ followed either by ‘‘flip’’ (which interchanges the arguments of an equation) or by demodulation, and counting one step resulting from back demodulating an input equation with another input equation. He asked me to find a more elegant proof, adding (in jest) that the goal was a proof of length 100 but, realistically, a 200-step proof would indeed be impressive.

In addition to the incentive of finding a proof that McCune would consider elegant, the theorem offered a challenge rather different from my earlier studies seeking shorter proofs. Specifically, those earlier studies [Wos95a,Wos96a,Wos96c] focused mainly on areas of logic, areas in which the equality relation was totally absent; in contrast, the theorem of interest relies exclusively on the equality relation. (Most of the other theorems used here to test the power of the methodology also rely on the equality relation and on no other.) Therefore, new obstacles no doubt would be encountered, for problems featuring equality behave (in mathematics and especially in automated reasoning) sharply differently from those in which equality is not present or present barely.

Anticipation of new obstacles to overcome naturally suggested I quickly review some of OTTER’s arsenal of weapons that might be useful in seeking a shorter proof for the given theorem. I began with a study of the approach McCune had used to find *any* proof, the approach that culminated in his finding an 816-step proof. He used three familiar weapons: weighting, the ratio strategy, and a Knuth-Bendix approach. Specifically, McCune used a max_weight of 28 to limit the complexity (measured in symbol count) of retained clauses, a limit that I maintained for the first fourteen experiments reported here. Regarding a direct use of strategy, he used his ratio strategy [McCune94,Wos96a], assigning a value of 4 to the pick_given_ratio. With that assignment, OTTER is instructed to choose (for the focus of attention to drive the program’s reasoning) four clauses by weight (complexity, whether determined by weight templates or by symbol count), one by breadth first (first come, first serve), then four, then one, and the like. It seemed obvious to me that, for seeking shorter proofs, both max_weight and the ratio strategy would prove most useful—perhaps even required—no doubt relying on a variety of assignments for their respective values. McCune also used a Knuth-Bendix approach for drawing conclusions and for canonicalization. I adopted this weapon also, although not specifically for proof shortening.

From my earlier studies focusing on finding shorter proofs, I added three weapons to the cited arsenal: the resonance strategy [Wos95a,Wos96a], the hot list strategy [Wos96a,Wos96b], and McCune’s dynamic hot list strategy [Wos95b,Wos96a,Wos96b]. Regarding the *resonance strategy*, its objective is to enable the researcher to suggest equations or formulas, called *resonators*, each of whose symbol pattern (all of whose variables are considered indistinguishable) is conjectured to merit special attention for directing a program’s reasoning. To each resonator one assigns a value reflecting its relative significance; the smaller the value, the greater the significance.

In contrast to directing a program’s reasoning, the objective of the *hot list strategy* is to rearrange the order in which conclusions are drawn. The rearranging is caused by immediately visiting and, depending on the value of the (input) heat parameter, even immediately revisiting a set of input statements chosen by the researcher and placed in an input list, called list(hot). The chosen statements are used to *complete* applications of inference rules rather than to *initiate* them.

The *dynamic hot list strategy* has the same objective as does the hot list strategy. However, whereas the (static) hot list strategy is restricted to using clauses that are present when the run begins, the dynamic hot list strategy has access to members adjoined to the list(hot) *during* the run. The input heat parameter governs the amount of recursion that is permitted for both the hot list strategy and the dynamic hot list strategy. The input dynamic_heat_weight parameter assignment places an upper bound on the pick_given weight of clauses that can be adjoined to the hot list during the run. (A clause technically has two weights: its pick_given weight, which is used in the context of choosing clauses as the focus of attention to drive the program's reasoning, and its purge_gen weight, which is used in the context of clause discarding; often the two weights are the same. If no member of a weight_list matches a clause, where variables are treated as indistinguishable, the clause is given a pick_given weight equal to its symbol count; the same is true for its purge_gen weight.)

All of these strategies, as well as other weapons offered by OTTER, play a vital role in the context of the first property (proof length) of elegance. On the other hand, far fewer weapons are needed when the concern is the second, the third, or the fourth property. Further, whereas a focus on proof length requires the use of a methodology, each of properties 2 through 4 can be directly addressed with one or more features offered by OTTER.

Indeed, when the second property, that of *term structure*, is the concern, one can rely either on OTTER's treatment of demodulation or on weighting. Regarding the use of demodulation, consider the case in which one wishes to avoid terms of the form $n(n(t))$ for any term t, and assume that the only predicate present is P. The following set of demodulators will suffice; OTTER discards clauses that are demodulated to $T for **true**. (One can think of the function n as negation and the function i as implication.)

```
list(demodulators).
(n(n(x)) = junk).
(i(junk,x) = junk).
(i(x,junk) = junk).
(n(junk) = junk).
(P(junk) = $T).
end_of_list.
```

A careful analysis of the demodulators in the given list shows that iteration will demodulate (rewrite) a deduced but unwanted conclusion (that takes the form of a unit clause) eventually to $T, standing for **true**.

Regarding the use of weighting, in addition to placing entire equations or formulas in a weight_list (as is the case for resonators), one can place subexpressions in a weight_list. If, for example, the type of elegance that is desired asks for the absence of all terms of the form *inverse*(*inverse*(t)) for terms t (say, in a study of group theory), one can include an appropriate weight template. In the case under discussion, the following weight template can be included in weight_list(purge_gen) or in weight_list(pick_and_purge), with the intention that the program purge deduced conclusions containing the undesired subexpression.

```
weight(inverse(inverse($(1))),1000).
```

Because of the presence of this template, the purge_gen weight of any deduced conclusion containing the cited type of term will be increased by at least 1000, if no other weight template interferes. (The use of $(1) causes the term in the corresponding position, whether complex or not, to be multiplied by 1.) If the max_weight is strictly less than 1000, such a conclusion will be immediately purged, unless its weight is sufficiently reduced because of the actions of yet another template. One thus has a taste of how OTTER offers a means for directly addressing the concern for term structure in the context of elegant proofs.

Because of the way weighting is implemented in OTTER, it can be used to serve another purpose, namely, to purge an entire class of unwanted equations through the use of the *resonance-restriction strategy*. With this strategy, conclusions are purged through the use of resonators, and they are chosen for the focus of attention to direct the reasoning by symbol count. As for the relevance to elegance, one can imagine having identified a class of equations or formulas that are to be avoided, if elegance is to be increased, a class all of whose members have the same functional shape, differing only in the presence of the particular variables. For example, the class to be avoided might contain the following equations.

```
(x* ((y+z)* (y+u)))* ((x*y)+ (x* (z*u))@ ) = (x* (y+y))*1.
(x* ((z+y)* (y+u)))* ((x*y)+ (x* (z*u))@ ) = (x* (y+y))*1.
(x* ((y+z)* (y+u)))* ((y*x)+ (x* (z*u))@ ) = (x* (y+y))*1.
(x* ((z+y)* (y+u)))* ((y*x)+ (x* (z*u))@ ) = (x* (y+y))*1.
```

If the intent is to avoid these equations and others of their class differing only in the particular variables that occur, one need only select an equation of the class and use it as a resonator, assigning to it a value greater than the max_weight. Placing the chosen resonator in weight_list(purge_gen) or in weight_list(pick_and_purge) will, if nothing else interferes, cause the program to purge any deduced conclusion that is in the undesired class. Any member of the undesired class will suffice as the chosen resonator—one of the beautiful aspects of the resonance strategy—for (by definition) all variables in a resonator are treated as indistinguishable. (For more details concerning the resonance-restriction strategy, see Sections 4.2 and 5.2 in [Wos96a].)

Regarding the third cited property of elegance, that focusing on *variable abundance*, OTTER offers precisely what is needed with a command of the following type.

```
assign(max_distinct_vars,5).
```

With the inclusion of this command, the program will purge any deduced clause that contains more than five distinct variables. The use of that feature alone can result in the completion of a proof that is far more attractive than the proof produced without using that feature.

As for the fourth cited property of elegance, *deduced-clause complexity*, the feature that is relevant can be easily guessed, namely, the option to assign a chosen value to max_weight. For example, if no weight templates are included in the input, OTTER will assign as the purge_gen weight to each deduced clause its symbol count, of course ignoring commas and parentheses. Therefore, if one wishes the program to seek a proof in which no deduced step has a length exceeding, say, *k*, one merely assigns to max_weight the value *k*. No problem exists if, at the same time, one wishes to include weight templates to guide the program's search without interfering with the stated intention. In such an event, one places the guiding templates in weight_list(pick_given); no template in that list is consulted when computing the purge_gen weight, the weight used to decide whether to retain a new conclusion in the context of its complexity.

At this point, the pressing questions with regard to *proof length* concern which weapons played the key role, how they were used, and in what order. To answer these questions, I present in some detail in Section 3 a sequence of experiments. To aid the researcher interested in using OTTER for both similar and totally dissimilar objectives, I include some commentary explaining why certain choices were made.

3. A Tortuous Path to Success

Earlier studies seeking shorter proofs [Wos95a,Wos96a,Wos96c] virtually demanded the use of the resonance strategy here. As for which equations to use as resonators, again virtually no thought was required. Indeed, experiments in areas other than Boolean algebra had shown that deduced steps from the proof of a theorem (even if only distantly related to the target theorem) often serve well as resonators, whether the goal is a sharp reduction in CPU time required to complete *any* proof or the goal is to find a *shorter* proof. In fact, steps from the proof of a target theorem often serve well as resonators for improving one's results regarding the target theorem. Therefore, I began (in Experiment 1) by using as resonators the 811 deduced steps of McCune's proof, starting with a clause obtained by back demodulation, but not including the earlier clauses produced by ‘‘copy’’.

Almost all the experiments reported in the remainder of this technical report were conducted on a SPARCstation-10. For brevity, the CPU times are cited without the reminder that they are only approximations. Also, note that all resonators placed in a weight_list *must* be unit clauses, free of ‘‘ | ’’ (the logical **or** symbol). Therefore, when discussing positive deduced steps of a proof to be used as resonators, implicit is the requirement that each be a unit clause.

3.1. Partial Success: Experiment 1

In Experiment 1, each resonator (each of the 811 deduced steps of McCune's proof) was assigned the value 2. All resonators were placed in the pick_and_purge weight_list. This placement instructed OTTER to assign both a pick_given weight and a purge_gen weight of 2 to any deduced clause that matched any of the 811 resonators, where (in the resonator case) two clauses match if they are identical when all variables are treated as indistinguishable from each other.

The steps of McCune's proof that are deduced and retained are thus used to guide OTTER's search for a proof; indeed, because of being assigned the small pick_given weight of 2, they are given preference (over most or all clauses) for being chosen to initiate an application of the inference rule or rules in use. Also (of slightly lesser significance) the deduction of a step of McCune's proof will almost certainly lead to its retention, at least in the context of max_weight, for such a step will be assigned a purge_gen weight of 2. In Experiment 1, the max_weight was assigned the same value McCune used, namely, 28.

Since the pick_given_ratio remained unchanged from that used to obtain the 816-step proof, namely, that of 4, the presence of clauses matching a resonator would only *guide* OTTER's search, for every fifth clause used to initiate an application of an inference rule would be chosen by first come, first serve. Such guiding rather than totally controlling the search gives the program the opportunity of finding a proof different from McCune's and, more to the point, a chance of finding a shorter proof.

In 1613 CPU-seconds on a SPARCstation-10 with retention of clause (15257), OTTER completed a proof of length 680 and level 78; the level of McCune's proof is 89. Five of its steps result from applying copy to an input clause, followed either by flip to interchange arguments or by demodulation, and one step results from back demodulating an input clause with another input clause. (This added information is given in the event that one wishes to measure proof length beginning with the first use of paramodulation. For the remainder of this technical report, the figures for proof length will be taken from an OTTER run; the program counts steps of the cited types in its computation.)

Because McCune was equally interested in a ''short'' proof of the dual, its negation, captured by the following clause, was also included.

(A + B) * B != B.

Even more impressive in the context of proof length reduction, the dual was also proved, the proof having length 628; its level is 67, completing in 1608 CPU-seconds with retention of clause (15031).

Perhaps startling—and certainly charming—except for the clause corresponding to negating the dual, the proof of the dual is a subproof of the proof of the primary conclusion. To remove any doubt concerning technicalities, of course the proof of the dual, being one of contradiction, relies on the negation of the dual, whereas the proof of the primary conclusion relies on its negation; clearly, the negation of the dual is not present in the proof of the primary conclusion, and conversely.

3.2. The Hot List Strategy: Experiment 2

Whether the focus is on the target conclusion or on its dual, this fine start on a path to finding a far shorter proof does not imply that all always went according to plan—the path was unpredictable, as will be discussed in this section. However, with this satisfying reduction in proof length from 816 to 680 or 628 (whichever choice is made), the stage was nicely set for Experiment 2. The only change made from Experiment 1 was that of adding the use of the hot list strategy with the heat parameter assigned the value 1. The object was to gain some insight into the value of using this strategy in the context of seeking a shorter proof for the target theorem. Therefore, the simplest comparison was preferred: Merely make the cited change.

Of course, the addition of the hot list strategy naturally presented the question of which clauses to place in list(hot). Typically, a rule that has served well is to *place in the hot list the clauses corresponding to the special hypothesis*. (The special hypothesis of a theorem of the form **if** *P* **then** *Q* refers to that part of *P*, if such exists, that excludes the underlying axioms and lemmas. For example, in the study of the theory that asserts that rings in which the cube of $x = x$ are commutative, the special hypothesis consists of the equation $xxx = x$.) A less precise rule that often is effective suggests for the members of the hot list those clauses in the initial set of support that are ''simple'', where simplicity usually means expressed in a small number of symbols. The second rule is the one that was used. Therefore, the following two clauses were placed in list(hot).

x + x@ = 1.
x * x@ = 0.

Neither clause is directly relevant to the conclusion of the theorem, namely,

(x*y)+y = y,

nor to its dual, namely,

(x+y)*y = y.

Nevertheless, the presence of the cited two clauses in the (input) hot list, as the following data shows, produced enough of a reduction in proof length to clearly be of interest, and even piquant (to me).

Indeed, Experiment 2 completed a 606-step proof of the dual in 3701 CPU-seconds with retention of clause (23434); the proof has level 78. Then a 607-step proof of the conclusion obtained by McCune was completed in 3702 CPU-seconds with retention of clause (23437); its level is 78. A nearly subproof relation exists.

3.3. Resonance and the Dynamic Hot List Strategy: Experiment 3

The next experiment, Experiment 3, was mainly influenced by a practice reflecting earlier successes (in other areas) in seeking shorter proofs, successes in which the resonance strategy played a key role [Wos95a,Wos96a,Wos96c]. The practice is iterative, using the deduced proof steps from one experiment as resonators in a later experiment.

The first decision I had to make was to choose between the proof steps of the so-called primary conclusion and those of the dual of the conclusion. Although the two proofs completed in Experiment 2 are of almost equal length, 607 (for the primary) and 606 (for the dual), later experiments (as with Experiment 1) might not share this almost-equal-length property. Influencing my decision were two factors. First, aesthetics virtually demanded staying with the conclusion of McCune's 816-step proof; to do otherwise would in a sense be cheating, be loading the dice. Second, science virtually demanded using the primary conclusion, for many situations to which one might wish to apply the methodology might lack a dual. (As an aside, an interesting area for research concerns using as resonators proof steps of the dual in an iterative attack on seeking a shorter proof, either of the primary or of the dual.) Therefore, with few exceptions, the choice of proof steps of the primary conclusion rather than the dual was the rule for most of the sequence of experiments to be discussed, as was the omission of the type of clause obtained with copy. Only when I hoped to extract a few extra droplets of proof-length reduction did I violate this rule.

Therefore, for Experiment 3, 602 resonators were used, each assigned a value of 2. In other words, five of the 607 possible resonators were not included, those in which copy was applied to an input clause.

One other change was made for Experiment 3, namely, the introduction of the dynamic hot list strategy. However, rather than simply adding the use of this strategy, the idea was to imitate a combination strategy that had proved most effective, namely, that of combining the resonance strategy with the dynamic hot list strategy; see [Wos96c]. The combination strategy is, intuitively, an **if-then** type of strategy: **If** a clause is retained that matches a member of a chosen set of resonators, **then** adjoin that clause to the hot list *during* the run. The notion is that the adjunction to the hot list (during the run) of ''short'' clauses might produce an even shorter proof, for the use of short clauses in the input hot list had resulted in progress. Therefore, the decision was to choose from among the 602 resonators to be used those expressed in five or fewer symbols (counting spaces, commas, and parentheses, but not counting the period), assign to each the value 1, and assign to the (input) dynamic_heat_weight the value 1. The consequences of this decision are that (1) any clause that is deduced and that matches a resonator with weight 1 will also be assigned a pick_given weight of 1—because the resonators were placed in the pick_and_purge weight_list—and (2), if such a clause is retained, it will be adjoined to the hot list during the run, because of its weight and because the dynamic_heat_weight has been assigned the value 1. (The value assigned to the dynamic_heat_weight places an upper bound on the pick_given weight of clauses that can be adjoined to the hot list during a run.)

Of the 602 resonators to be used, 19 were assigned the value 1; they were placed in the pick_and_purge weight_list before the 602 resonators. Therefore, if one ignores the assigned value, 19 weight templates appeared twice. The earliest weight template that matches a clause is that which is used to assign the weight to the clause; so the second copy of a template with assigned value of 2 did not interfere with the assignment of 1 as the weight of a clause matching one of the 19 used in conjunction with the dynamic hot list strategy. Summarizing, **if** a ''short'' clause (as just defined) was retained that matched a ''short'' clause of the proof completed in Experiment 2, **then** that clause was adjoined to the hot list during the run.

Experiment 3 was markedly successful. In 321 CPU-seconds with retention of clause (9789), a proof (of the primary conclusion) of level 53 and length 344 was completed. As for the dual, 317 CPU-seconds was sufficient to complete a proof, with retention of clause (9709), a proof of level 50 and length 300. (The possible subproof relation is absent.) Thus the iterative approach was working well, having cut the required number of steps to prove the primary conclusion more than in half, from 816 to 344.

3.4. Momentary Setback

At this point in the experimentation, the first of many failures to come occurred, a failure that heralded the difficulty and delicacy of what would be needed to finally complete the promised far shorter proof, namely, a proof of length 100. The experiment that failed—unnumbered so that progress is captured by a sequence of experiments that are numbered consecutively—relied on the type of proof step that the first three experiments do. With respect to Experiment 3, two deliberate changes were made: Rather than focusing on the proof steps from Experiment 2, the focus was on those from Experiment 3; rather than focusing on steps expressible in five or fewer symbols, the focus was on those expressible in seven or fewer, 28 of them.

However, in keeping with an accurate account of history, one additional and unintentioned change was made, one that—as further experimentation verified—was crucial. Unintentionally, two extra resonators were included, making a total of 341 rather than 339, corresponding to a copy application to an input clause and corresponding to an input clause. (Such an inclusion is in no way a form of cheating, as in the case where one already knows the answer, for example; it merely does not rigorously follow the cited rule I planned to use for choosing resonators. Note that resonators are used to guide a program’s search and have no true or false value as lemmas do.)

Were it not for this accident, perhaps the 100-step proof would never have been found, for Experiment 4 would have relied on a smaller set of resonators. Sometimes the gods are so kind, even protecting one against oneself, bringing good fortune despite what was obviously an error. (As another aside, perhaps all would have gone even better if for resonators all clauses in a proof were used, including those from the input.) Although a proof was completed in 126 CPU-seconds—certainly not much computer time—the experiment was deemed a failure, for the proof has length 392, a rather impressive setback when compared with the 344-step proof obtained with Experiment 3.

3.5. Back on Track: Experiments 4 and 5

Nevertheless, because of a conviction about the value of this type of iteration (based on experimentation in other areas in which equality plays no role), a fourth experiment was conducted, one very closely related to that which failed. Specifically, the single change from the failed experiment was that of removing the use of the dynamic hot list strategy, correctly implying that the two unintended resonators remained in use. Experiment 4 produced more progress, completing a proof of the desired theorem in 111 CPU-seconds with retention of clause (5871), a proof of length 325 and level 52. The dual was also proved, in 112 CPU-seconds with retention of clause (5977), a proof also of length 325 and level 52.

Experiment 5 was quite similar to Experiment 4. The only change was in the context of the choice of resonators, switching the focus from the proof steps (including the cited two extra ones) of Experiment 3 (that were used in 4) to the deduced steps of Experiment 4 (to be used in Experiment 5). Experiment 5 produced another significant reduction in proof length, completing a 287-step proof of level 44 with retention of clause (9400) in 164 CPU-seconds. Then, less than 1 CPU-second later, the dual was proved, with a proof of length 256 and level 43, with retention of clause (9409).

Experiment 5 was followed by numerous failures in the context of finding shorter proofs. For a taste of how much ground could be lost with ineffective choices of the parameters and the strategies, one of the unsuccessful experiments completed a proof of length 514, virtually proving that an algorithm is not being discussed, merely a methodology. Also strongly and correctly suggested are the subtlety, intricacy, and complexity of seeking shorter proofs; indeed, the process is quite delicate in the sense that the slightest change in a parameter assignment can result either in finding *no* proofs or in finding a rather shorter proof. Regarding what was tried, the value of the heat parameter was assigned sometimes 1 and sometimes 2, and sometimes the hot list strategy was not used. The max_weight was assigned various values, from 12 to 36. Although the usual assignment for the pick_given_ratio was 4, sometimes 3 was chosen. Diverse moves were made in the context of the choice of resonators. One learns that the methodology is experimental, often requiring playing with various combinations of parameter assignments and strategy choices whose effects are indeed unpredictable. In short, not much worked—until what is called Experiment 6 was conducted.

3.6. A New Tack: Experiments 6 and 7

Experiment 6 differed from Experiment 5 in two ways. First, as expected, the resonators were the correspondents of the proof steps of the proof (of the primary conclusion) obtained in Experiment 5, in place of those from Experiment 4. Second, the resonators that correspond to a step obtained with back demodulation were given a weight of 4 rather than 2, to delay their consideration for initiating applications of an inference rule. (This added action resulted directly from a remark my colleague McCune made concerning the possible value of being able to apply all relevant demodulators at once in sequence, rather than by means of a set of sequences, each producing another step in the proof.)

The ideal case explains the assignment of higher weight to correspondents of proof steps obtained with back demodulation. Imagine that the proof under consideration (whose steps are to serve as resonators) contains a chain of clauses, *A*, *B*, *C*, *D*, *E*, obtained by applying back demodulation: back demodulation applied to *A* yields *B*, back demodulation applied to *B* yields *C*, similarly *D* from *C*, and *E* from *D*. In the ideal case, when the program (relying on the resonators from the just-mentioned proof) was seeking a shorter proof, the deduction of *A* would find already present and in the needed order the demodulators that had been used to respectively deduce *B* through *E*. In that event, *A* would be successively demodulated to yield *B* through *E*, but—and here is the point—*A*, *B*, *C*, and *D* would be intermediate steps, and they would therefore not be explicitly present in the proof to add to its length.

Thus, by causing the program to delay considering for the focus of attention clauses that (in the proof whose steps are used as resonators) were obtained by back demodulation, perhaps such clauses would not be needed to complete a proof, preferably a shorter proof. Nevertheless, because the ideal case seldom occurs (regardless of the context), assigning such a high weight that the type of clause under discussion is in fact purged asks for serious trouble. After all, a clause that was obtained with back demodulation might have been a parent (when paramodulation was used) of a crucial clause; indeed, its absence might prevent the program from completing *any* proof, much less a shorter one.

Of course, as one might predict from the preceding, Experiment 6 succeeded, completing a 260-step proof with retention of clause (5977), a proof of level 45, requiring 80 CPU-seconds. Almost immediately, the dual was proved, with a proof of length 284 and level 45. No subproof relationship holds between the two proofs.

After additional unsuccessful attempts to find a proof of length strictly less than 260, the conditions for a successful experiment were determined. For this experiment, Experiment 7, two changes were made from Experiment 6. First, as expected with this iterative methodology that relies on the results of earlier experiments to be used for later experiments, the resonators used in Experiment 7 were the proof steps of the proof completed in Experiment 6. The assignment of their values followed the pattern of Experiment 6. Second, because of some of the data gathered in the unsuccessful runs, the pick_given_ratio was assigned the value 7, rather than 4. The intention was to use complex clauses retained early in the run less often than was the case in Experiment 6.

Another significant reduction was obtained in that OTTER completed a 236-step proof of level 43, requiring 63 CPU-seconds and the retention of clause (5092). For but one data point that might be of interest, 30 steps of the 236-step proof are not among the 260-step proof obtained in the preceding experiment. The dual was proved, with a 237-step proof of level 43, in 64 CPU-seconds with retention of clause (5156).

3.7. Small Advances: Experiments 8-13

Regarding the next few experiments in the pursuit of a shorter proof, advances (when they occurred) were small. (For the discussion of the next few experiments, no comments concerning the dual will be included, nor will there be commentary in detail concerning failures.) Experiment 8 (closely imitating Experiment 7) yielded a 235-step proof of level 43 in 51 CPU seconds, an experiment that relied for resonators on the proof from Experiment 7 and on an assignment of 10 to the pick_given ratio. Experiment 9 (closely imitating Experiment 8) yielded a 233-step proof of level 43 in 51 CPU seconds, an experiment that relied for resonators on the proof from Experiment 8 and on an assignment of 10 to the pick_given ratio. Experiment 10 (closely imitating Experiment 9) yielded a 229-step proof of level 42 in 52 CPU seconds, an experiment that relied for resonators on the proof from Experiment 9 and on an assignment of 7 to the pick_given ratio rather than an assignment of 10 (believing that more participation of complex clauses retained early would be beneficial). However, no resonator was assigned the value 4 to delay matching clauses from being chosen as the focus of attention to drive the program's reasoning. Just as a reminder, the hot list strategy was still in use, with the (input) heat parameter still assigned the value 1.

Experiment 11 was identical to Experiment 10, except for assigning the value 10 to the pick_given_ratio, an action taken just to be sure that a good bet was not missed. The safety play succeeded: a 225-step proof of level 42 was completed in 51 CPU-seconds. The continued small CPU times to find shorter and still shorter proofs was indeed encouraging. Because the methodology was succeeding, Experiment 12 mirrored Experiment 11, of course except for the replacement of resonators. Rather than the proof steps from Experiment 9, those of Experiment 11 were used. A 224-step proof of level 41 was completed, again in 51 CPU-seconds.

Next, Experiment 13 was conducted, using as resonators the proof steps from Experiment 12. One additional change was made: The set of resonators assigned the value 1 was expanded to include those expressed in 10 or fewer symbols (including spaces, commas, and parentheses, but not including the period). The experiment completed a 219-step proof of level 39 in 47 CPU-seconds.

By glancing at the experiments discussed so far, one sees how the methodology borrows from earlier experiments, not just proof steps to be used as resonators, but also combinations of option choices. Of course, tinkering with the assignments and choices is required, arriving at what works only through more experimentation. The claim that one learns what to do in later experiments from what occurs in earlier experiments is clearly an overstatement; rather, one learns about promising combinations of options and possibly effective parameter assignments.

3.8. The Dynamic Hot List: Experiment 14

For Experiment 14, a feature used earlier (in Experiment 3) in the sequence was added, that of the dynamic hot list strategy. The motivation was a sequence of unsuccessful attempts to find a proof shorter than that produced in Experiment 13, a sequence that suggested that an impasse might have been encountered. The thought was that, if the hot list was proving useful—which it seemed to be—clauses retained *during* the run and that were similar to those included in the input hot list would make profitable additions.

Therefore, from among the resonators that were used (those corresponding to the proof steps from Experiment 13), eight were selected and assigned a weight of 0. The dynamic_heat_weight was also assigned the value 0. When and if OTTER deduced and retained a clause matching any of the chosen eight resonators, it would be assigned a weight of 0 and, because of the value assigned to the (input) dynamic_heat_weight, the matching clause would be adjoined to the hot list during the run. Also, based on the unsuccessful runs, the pick_given_ratio was assigned the value 9.

Six clauses were adjoined to the hot list during the run while Experiment 14 was completing a 207-step proof of level 44 in 50 CPU-seconds; in other words, McCune's possibly realistic goal was near at hand. For but one indication of how narrow is the path to progress, an assignment of 10 (rather than 9) to the pick_given_ratio had yielded a 213-step proof of the primary target and had yielded a 208-step proof of the dual.

3.9. Perturbing the Search Space: Experiments 15-17

Experiment 15 reflected some of the difficulty that was being encountered in the context of attempting to find a proof of length strictly less than 207. For that experiment, use of the dynamic hot list strategy was dropped, and the max_weight was reduced from an assignment of 28 to one of 20; 28 had been the assigned value until this experiment. The reduction was designed to perturb the search space by avoiding the retention of numerous complex clauses; recall that the ratio strategy permits a program to focus on complex clauses, especially those retained early in the run. Of course, as expected, the resonators were the correspondents of the proof steps of the proof completed in Experiment 14. However, as yet another example of returning to what worked earlier, those corresponding to a step obtained with back demodulation were assigned a weight of 4 rather than assigned a weight of 2 (as was the usual case in this sequence of experiments).

Finally, for the first time, a direct attack on proof length was initiated, by adding the use of *ancestor subsumption* and (as almost always then required) also adding the use of back subsumption. With the use of ancestor subsumption, when a conclusion is drawn more than once, the lengths of the corresponding derivation path lengths are compared; the program retains the copy reached by the shorter path.

The experiment succeeded nobly, completing a 185-step proof of level 41 in 28 CPU-seconds. Not only was the realistic goal reached—a goal that McCune had classed as impressive—but the result provided incentive to seek the goal he had suggested in jest, namely, a proof of length 100.

In view of the encouraging result, a natural experiment (Experiment 16) was to keep all as in Experiment 15, but reduce the max_weight to a smaller value; 14 was the choice. The move was a good one, leading to a proof of length 176 and level 47 in 31 CPU-seconds. Even more impressive, the dual was proved, with a proof of length 159 and level 42, in 30 CPU-seconds.

Immediately, Experiment 17 was in order, similar to Experiment 16 but with the pick_given_ratio assigned the value 10 and the max_weight assigned the value 12. The notion was that such small changes might be what was needed to reduce the proof length of the primary conclusion to 159 or less. The motivation was that of continuing to emphasize for the source of resonators proof steps of the primary conclusion, that which McCune had chosen as the goal or conclusion to be proved. Where the dual was proved, with a proof of length 158 and level 48, in 27 CPU-seconds, the primary conclusion required approximately 69 CPU-seconds to prove and, unfortunately, the proof has length 211 and level 67.

3.10. Focusing on the Dual: Experiments 18-21

At this point, what amounted to an impasse was encountered. Various moves produced no progress. Therefore, desperation—or the belief that far more was possible—motivated an additional modification to the methodology.

For Experiment 18, rather than choosing for resonators proof steps of the primary conclusion, the dual was used, from Experiment 17. A significant change was made regarding the values assigned to the resonators. Although those with ten or fewer symbols in them were still assigned the value 1, those used as demodulators and that showed back_demod in their history were assigned the value 2, with the remaining assigned the value 4. The notion was that progress might occur if clauses used as demodulators that in addition were obtained with back demodulation were given preference for being chosen as the focus of attention to drive the program's reasoning. Although based far more on intuition than on analysis, the idea was to have certain clauses used as demodulators available earlier than they might otherwise be, in turn perhaps reducing the need for the presence (in a completed proof) of clauses obtained from back demodulation. The pick_given ratio was assigned the value 9, and the max_weight was assigned the value 14.

Experiment 18 completed a 153-step proof of level 51 of the primary conclusion in 16 CPU-seconds. Although still not the main goal, the dual was proved, with a 145-step proof of level 46, in 14 CPU-seconds. Not only was the proof length decreasing, so also was the required CPU time. In case one is tempted to assert that such is expected, various experiments in diverse fields do not support such an assertion. Nevertheless, when less and less CPU time is required, the methodology under study at least gives the appearance of power.

Before continuing the account of the results of the sequence of experiments, I give some commentary concerning the role of the resonance strategy. Of the 153 steps of the proof of the primary conclusion obtained in Experiment 18, 24 of them are *not* among the resonators that were used in the experiment. Recall that the resonators correspond to the proof steps of the dual proved in Experiment 17. Thus—precisely in the spirit of the resonance strategy—one has a nice illustration of the value of using as resonators the proof steps of a related theorem when attacking the main target theorem. Perhaps even more persuasive, 21 steps of the 153 do not match at the resonator level any of the resonators that were used, where (as a reminder) matching in this context means treating all variables as indistinguishable from each other. In the given example, the resonance strategy can be viewed as having supplied a detailed outline of a proof, with OTTER finding the remaining needed steps.

Because focusing on the dual in regard to choosing resonators had enabled the program to cope with the potential impasse, the decision was to continue the practice for Experiment 19. In 10 CPU-seconds, a 128-step proof of the dual was completed, a proof of level 46. In 12 CPU-seconds, a 136-step proof of the primary target was completed, a proof of level 51.

Experiment 20 (like Experiment 19, except for assigning the value 12 to max_weight and using as resonators proof steps from Experiment 19 rather than from Experiment 18) produced a 124-step proof of the dual of level 45 in 8 CPU-seconds, and it produced a 172-step proof of the primary target of level 64 in approximately 38 CPU-seconds. How intriguing it is to see that the proof length of the dual again is reduced, in contrast to completing a 172-step proof of the target conclusion—indeed, negative progress!

Still in the spirit of the last few experiments, Experiment 21 was conducted. However, in addition to using (as expected) resonators from the proof obtained with Experiment 20, the max_weight was assigned the value 14 rather than 12, and the following clause was added to the input hot list.

```
x + (y * z) = (x + y) * (x + z).
```

As a reminder, the only other clauses that were members of the (input) list(hot) for this experiment and its predecessors are the following two.

```
x + x@ = 1.
x * x@ = 0.
```

The addition of the cited clause to the hot list was prompted by the perceived need to rely more heavily on the hot list strategy, not in the context of recursion (where one increases the heat parameter), but in the context of giving the program access to immediately visiting some powerful law as each new conclusion is retained.

In 9 CPU-seconds, a 119-step proof of the dual was found of level 42, and (as in Experiment 19) a 136-step proof of the primary target of level 43 was found in approximately 13 CPU-seconds.

3.11. Seesawing to Victory: Experiments 22-26

Because of the progress, Experiment 22 was conducted with just the expected change, the replacement of resonators to enable the proof steps (of the dual) obtained in Experiment 21 to be used to guide the search for a shorter proof. In 8 CPU-seconds, a 111-step proof of level 44 of the dual was completed, and in approximately 11 CPU-seconds a 126-step proof of level 45 of the primary conclusion was completed.

For Experiment 23, in addition to using resonators from the proof of the dual completed in Experiment 22, the dynamic hot list strategy was brought back into the picture, but this time coupled with the resonance strategy in the following manner. One resonator was included with an assigned value of 0, the following, and the dynamic_heat_weight was assigned 0.

```
weight(((x*y)+x)* ((x*y)+z)=x* (y+z),0).
```

The cited resonator was chosen because it appeared to play an important role in the proof completed in Experiment 22. The notion was that if the corresponding clause were retained, then each new clause as it was retained should be immediately considered with the designated resonator (by applying paramodulation to the pair) *before* any other clause was chosen as the focus of attention.

Also, the pick_given_ratio was assigned the value 12 (rather than 9) to place less emphasis on complex clauses retained early. Finally, no resonator was assigned the value 1, as was the case in the preceding experiment for those requiring 10 or fewer symbols to express. The cited three changes correctly suggest that some analysis occurred, some examination of the output file corresponding to Experiment 22. Approximately 8 CPU-seconds was all that was needed to prove the dual with a 108-step proof of level 45, and 13 CPU-seconds produced a 131-step proof of level 49 of the primary target.

For Experiment 24, the dynamic hot list strategy was omitted, the extra clause present in the input hot list was removed, and those resonators expressed in ten or fewer symbols were again assigned the value 1. A 106-step proof (of the dual) of level 45 was found in 6 CPU-seconds, and a 122-step proof of level 51 of the main target was found in 9 CPU-seconds.

Except for changes regarding the use of the hot list strategy, Experiment 25 was indeed similar to Experiment 24, of course using for resonators the proof steps of the completed proof (from the dual still) from Experiment 24. The (input) heat parameter was assigned the value 2 rather than 1 to permit recursion, and the following four clauses were placed in the input hot list.

```
x * (y + z) = (x * y) + (x * z).
x + (y * z) = (x + y) * (x + z).
x + x@ = 1.
x * x@ = 0.
```

When the heat parameter is assigned the value 1, conclusions that result from consulting the hot list (whether one is using the hot list strategy or the dynamic hot list strategy) have *heat level* 1. If the program decides to retain a conclusion of heat level 1 and if the (input) heat parameter is assigned the value 2, then *before* another conclusion is chosen as the focus of attention, the heat-level-1 conclusion is used to initiate the search for conclusions of heat level 2 (with the hypotheses needed to complete the corresponding application of the inference rule chosen from the hot list).

Assigning a value of 2 to the heat parameter certainly has great potential for perturbing the search space, and, at this stage of the search for a ‘‘short’’ proof, such perturbation seemed imperative. Placing additional clauses in the hot list, whether before the run starts or during the run, also sharply increases the likelihood that the search will be perturbed. Indeed, the hot list strategy was formulated to enable a program to draw conclusions far sooner than it would otherwise; with the use of the strategy, the order in which conclusions are drawn is usually dramatically rearranged.

The first proof completed in Experiment 25 was that of the dual, a proof of length 108 and level 46, in 8 CPU-seconds. In other words, in the context of the dual, ground finally was lost. However, in 9 CPU-seconds, OTTER completed a 102-step proof of level 44 of the target conclusion. Almost as rewarding as finding such a short proof was the fact that the (primary) target had been proved in fewer steps than had the dual, which was not the case for so many of the experiments reported here.

The result led to the conjecture that, with a few experiments based on fiddling with the parameters and option choices, a singular event would occur: OTTER would complete a 100-step proof of the target. (That numbers such as 100 would have appeal for a mathematician must amuse many.)

For Experiment 26, max_weight was assigned the value 4 rather than 14, heat was assigned the value 3 rather than 2 (or 1, as was the case for so many experiments), and for resonators the choice was the proof steps of the proof completed in Experiment 25, that of the primary target and not the dual. The assignment of values to the resonators is that used in so many of the cited experiments, 1, 2, and 4 (as already discussed). The assignment of the value 4 to max_weight merits a short explanation. The notion was that of preventing the program from retaining *any* clause that does not match (at the resonator level) one of the resonators, match in the sense that all variables are considered indistinguishable. The return to focusing of proof steps of the primary conclusion for the source for resonators was in part because of the length of its proof (102) in Experiment 25 and in part because it is the target of the study.

With Experiment 26, the not-so-well hidden and actual goal was reached: OTTER completed a 100-step proof of level 44 of the target of the study in 6 CPU-seconds. As for the dual, its proof has length 99 and level 44, also requiring 6 CPU-seconds.

The results of each experiment are summarized in Table 1 for the target study.

Table 1: The Road from an 816-Step Proof to a 100-Step Proof

Experiment	Length	Level	Time (sec)
1	680	78	1613
2	606	78	3701
3	344	53	321
4	325	52	111
5	287	44	164
6	260	45	80
7	236	43	63
8	235	43	51
9	233	43	51
10	229	42	52
11	225	42	51
12	224	41	51
13	219	39	47
14	207	44	50
15	185	41	28
16	176	47	31
17	211	67	69
18	153	51	16
19	136	51	12
20	172	64	38
21	136	43	13
22	126	45	11
23	131	49	13
24	122	51	9
25	102	44	9
26	100	44	6

4. Obstacles to Finding Short Proofs

Because one might naturally question the tortuousness of the path that began with a proof of length 816 and ended with a proof of length 100, the following commentary is in order. Its purpose is twofold: first, by discussing various obstacles, to give some insight into the difficulty of finding such a short proof; and, second, by touching on the consequences of different choices, to provide some understanding of the complexity of option choosing in this context. (Chapter 3 of [Wos96a] focuses in detail on the obstacles to finding shorter proofs and the means to overcome them.)

To begin with, the presence of the equality relation introduces an element often not present, namely, the need for demodulation (the use of rewrite rules for simplification and canonicalization). Indeed, without demodulation, in the vast majority of cases the program will drown in redundant information (the same fact expressed in too many tightly coupled ways); see [Wos91b]. For but one example, the program might deduce that $a+b = c$, that $0+(a+b) = c$, that $a+b = c+0$, and the like.

When demodulators are present, the order in which demodulators are applied can be crucial, sometimes producing the needed equation to complete the assignment, but sometimes preventing its discovery. To complicate matters further, without an arduous analysis, one must simply wait and see which demodulator is applied before which, for example, when one chooses to use a Knuth-Bendix approach (in which demodulators are adjoined during the run). The situation is rather like creating a grammar, a set of rules (the analogue of demodulators) that affect sentence structure; when the order of rule application changes, the sentence structure changes, not always in a manner that best fits the objective.

The discovery and addition of demodulators are of course dependent on the choice of options and the assignment of values to the parameters. For a simple example, whereas a max_weight of 20 might enable a program to deduce and retain a demodulator (rewrite rule) expressed in 20 symbols, a max_weight of 16 might prevent its deduction, much less its retention. For a more complicated example, assume that the max_weight is well chosen in the sense that it itself presents no obstacle, and consider two cases: an assignment of the value 6 to the pick_given_ratio, and an assignment of the value 7. On the surface, one not familiar with the ratio strategy might hastily assert that this small difference (6 versus 7) could be of little import. Actually, as shown by some of the experiments not detailed here, quite the opposite is the case. Indeed, in some experiments a change of 1 in the value assigned to the pick_given_ratio resulted in the program completing *no* proofs, whereas before the change a shorter proof was found.

To gain some insight into how such a situation can occur, first recall that an assignment of the value 6 to the pick_given_ratio causes the program to choose (for the focus of attention to drive the reasoning) six clauses by weight (whether determined by weight templates or by symbol count), then one clause by first come first serve, then six, then one, and the like. What can occur is that the first clause A chosen as the focus of attention by first come first serve is used to deduce and retain a key demodulator D. With a value of 7 assigned to the pick_given_ratio, however, the program would delay choosing the cited clause A until one additional clause B had been chosen (as the focus of attention) by weight. Consideration of B might lead to the deduction and retention of a demodulator E such that, when demodulation is applied to clauses retained later, E is applied before D is. Were such to occur, one can imagine that the space of retained clauses could be wildly different with the value 7 from that with the value 6 (assigned to the pick_given_ratio). In one case, the program might be prevented from finding *any* proofs, while in the other it might find a charmingly short proof. Without experimentation or brilliant insight, nothing suggests in general whether the smaller or the greater value is a wise assignment for the pick_given_ratio.

One thus has at least a glimpse of how the order in which demodulators are retained and the order in which they are applied can be crucial. In turn, the choice of parameters (such as that limiting the number of distinct variables in a retained clause or that governing the amount of recursion when the hot list strategy is in use) that affect the adjunction of demodulators (during the run)—as with the use of Knuth-Bendix, for example—can have rather startling effects. Clearly, as the discussion shows, the smallest change in parameter assignments can have significant changes in the results.

Just as the values assigned to parameters can be crucial, so also can the choice of strategy or strategies. For but one example, the presence or absence of the hot list strategy can produce sharply contrasting results. Whereas its use in Experiment 26 enabled OTTER to complete the desired 100-step proof, the corresponding experiment differing only in the omission of the hot list strategy completed *no* proofs. Such contrasting results are explained by the rearranging of the order in which conclusions are drawn. Indeed, without the hot list strategy, the retention of a new clause does not cause the program to immediately choose that new clause to initiate an application of the inference rules in use, as is the case when the hot list strategy is in use. This observation in no way implies that such immediate initiation is necessarily an advantage. In particular, the new demodulators that might be retained (with the use of the hot list strategy) might get in the way, might rewrite some clause into a form that blocks finding a shorter proof or, for that matter, finding any proof.

Still in the context of using the hot list strategy, the value assigned to the (input) heat parameter can save or lose the day. Whereas the value 3 in Experiment 26 worked well, the value 1 yields no proof of either the target conclusion or of the dual. In the cited case, the smaller value did not permit the program to deduce clauses of heat level greater than 1, and at least one such clause must have played a significant role.

The content of the (input) hot list can be equally critical. For example, in Experiment 26 if one uses for the input hot list that used throughout many of the experiments reported here (just two clauses), no proofs are returned; in contrast, with the four clauses (cited earlier) in the hot list, the already-discussed 100-step proof is found. Obviously, at least one of the two additional clauses (in the hot list) enables the immediate deduction and retention of one or more needed clauses. (As a reminder, the clauses in the hot list are used to *complete* application of inference rules.) However, as experimentation shows, sometimes a greater value assigned to the heat parameter harms rather than helps, and sometimes additional members in the hot list impede rather than aid progress. The answers to the questions of how and when to use the hot list strategy rest with the properties of the correspondingly perturbed space of retained conclusions; currently, the only means for making the appropriate decisions is through experimentation.

As for the resonance strategy, its use is designed to cope with obstacles such as cul de sacs or dead ends. Indeed, often experiments lead to a sequence of shorter and shorter proofs that reach a point conjectured to be far from what is possible—and yet no further progress occurs. To escape such a cul de sac, the resonance strategy is recommended, using proof steps from the last ‘‘good’’ proof or from some earlier or related proof. Although only distantly related to coping with dead ends, one sees the value of the resonance strategy by revisiting Experiment 1. That experiment differed from the approach taken by McCune (when he obtained his 816-step proof) only in the use of resonators, 811 that correspond to proof steps from McCune’s proof. The result (as cited) was the completion of a 680-step proof. Whereas McCune’s proof was found in 8185 CPU-seconds, that of Experiment 1 was found in approximately 1613 CPU-seconds. Of the 1613 CPU-seconds, 930 were spent in weighing clauses, no doubt mainly because of the use of 811 weight templates (corresponding to the resonators).

Not unexpected, the specific resonators that are used can make all the difference. However, rather than an algorithm for choosing effective resonators, only guidelines can be given, such as ‘‘use proof steps from the proofs of related theorems’’, ‘‘use (as resonators) correspondents to members of the initial set of support’’, and ‘‘use proof steps from earlier proofs (if a sequence of experiments is being conducted)’’. Experimentation in this context is the best teacher and, quite often, the only teacher.

Still in the context of the resonance strategy and in the spirit of option choosing, a review of Experiments 1 through 26 correctly suggests that the values assigned to the resonators can be crucial. Indeed, separating the resonators into classes by giving some weight 1, some weight 2, and some weight 4 (for example) was the key to progressing toward the final success, the 100-step proof. One of the obstacles encountered rests with the interconnection of the choices made for parameter assignments and strategy choices, an obstacle that is most likely inescapable in view of the nature of reasoning (whether automated or not).

Compared with the preceding items, far more sinister and unintuitive regarding the seeking of shorter proofs is that which can occur with a direct attack, an attack based on using ancestor subsumption. As a reminder, ancestor subsumption (which, for practical reasons, requires the use of back subsumption) compares the derivation path lengths when presented with two copies of the same deduced conclusion and retains the copy reached by the strictly shorter path. On the surface, one might surmise that ancestor subsumption is precisely what is needed to find shorter proofs and, perhaps further, that its use is guaranteed to find the shortest proof. Such a surmise is so far from the truth that the term ‘‘sinister’’ is in fact justified. Actually, the truth is captured by the following aphorism. ‘‘*Shorter subproofs do not necessarily a shorter total proof make.*’’

For a glimpse of what can go wrong when using ancestor subsumption (with finer detail given in Chapter 3 of [Wos96a]), begin by assuming *A* is a clause that is present in all proofs of the theorem under study. (The assumption, though not necessary, permits focusing on a simple example.) Second, assume that *A* occurs well before the end of any proof (also not a needed assumption). Next, assume that there exist two subproofs of *A* of respective lengths 5 and 3, where length is measured solely in terms of deduced clauses. Then assume that, other than *A*, the two subproofs share no steps in common. If two proofs of the theorem under consideration are completed respectively based on the two subproofs such that, ignoring *A*, all of the steps of the first subproof are present in the second total proof and such that none of the steps of the second subproof are present in the first total proof, then quite often the first total proof is shorter than the second (despite the presence of a longer subproof of *A*).

One can extrapolate from the preceding and see how a disaster in fact can occur. The program begins its search for a proof of the desired theorem and finds a subproof of *A* of length *k*. Then, before the proof is completed, a strictly shorter subproof of *A* of length *j* is found; through ancestor subsumption the second copy of *a* is retained and, in effect, the shorter subproof is preferred. Experimentation shows that the shorter subproof sometimes is of less value, perhaps far less, to the total proof than is the longer subproof, for the shorter subproof can rely on steps present only because of deducing *A*, steps used nowhere else in the total proof. Disaster can occur—although topologically beautiful in the obvious sense—when the program finds a number of such short subproofs such that they share few steps in common. Indeed, the program might be on a path to producing what amounts to a set of disjoint subproofs whose union is the total proof, where the total proof has a length equal to the sum of the lengths of the subproofs. In contrast, there may exist a proof whose so-to-speak subproofs are not nearly so short, sharing many steps in common, such that the resulting total proof (of the theorem in question) is far shorter than that just described. Summarizing, although ancestor subsumption (coupled with back subsumption) can provide the program with a powerful means for completing shorter proofs, its use can lead to a sinister outcome.

As further evidence of how complicated the search for shorter proofs is—in contrast to the preceding case—even if one decides not to use ancestor subsumption but to rely on forward subsumption to block the retention of many copies of the same conclusion, again trouble can easily be encountered. In particular, the program might complete a long proof of some needed intermediate conclusion, a proof many of whose steps are of use nowhere else, and then derive the same needed conclusion by pursuing a much shorter path. The second copy will not be retained because of the use of forward subsumption, which may in turn cause the program to complete a proof (of the target theorem) that is far longer than would otherwise be completed if the shorter subproof were used. Just for the record, if one is tempted to suggest that a feasible approach rests with the avoidance of forward subsumption, experience with automated reasoning programs shows that such an approach will drown in redundant information; see [Wos91b].

5. Applying the Iterative Methodology

At this point, one might naturally conjecture that the effectiveness of the methodology is derived (perhaps inadvertently) in some way from the theorem (DUAL-BA-3) that is the main focus of the preceding sections. In the following, I present evidence that refutes this conjecture. The emphasis is still on theorems in which equality is the dominant relation.

The first bit of evidence still concerns Boolean algebra, specifically, a theorem in which equality is the only relation. In [McCune96], the theorem is denoted by DUAL-BA-5; with its proof, an open question was answered. The following clauses capture the theorem to be proved. (When a line contains a ‘‘%’’, the characters from the first ‘‘%’’ to the end of the line are treated by OTTER as a comment.)

```
x = x.
y + (x * (y * z)) = y.          % L1
((x * y) + (y * z)) + y = y.    % L3
(x + y) * (x + y@) = x.          % B1
y * (x + (y + z)) = y.          % L2
((x + y) * (y + z)) * y = y.    % L4
(x * y) + (x * y@) = x.          % B2
x + y = y + x.
x * y = y * x.
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
(A * B) + (A * C) != A * (B + C) | (A + B) * B != B | B + B@ != A + A@.
```

The theorem asserts that the equations L1-L4, B1, and B2 are a basis for Boolean algebra. To prove the theorem, one is asked to show that three conclusions hold, respectively corresponding to a distributive law, an absorption law, and the fact that x+x@ is a constant.

As in the theorem DUAL-BA-3, discussed earlier, a starting point was supplied by McCune; he had obtained with OTTER a proof of length 119 and, as with DUAL-BA-3, was interested in a more elegant proof. The goal was to find a proof of length 100 or less. My approach was iterative, in the sense detailed earlier. The following summarizes the highlights.

For the first experiment of significance, the pick_given_ratio was assigned the value 4, max_weight the value 23, and heat the value 1. The following two clauses were placed in the (input) hot list.

```
x+y=y+x.
x*y=y*x.
```

The assigned values were those used by McCune; the choice of clauses for list(hot) was based on the usual and perhaps uninspired rule of including simple clauses. The resonators were the correspondents of the deduced steps of McCune's 119-step proof.

In 115 CPU-seconds with retention of clause (12600), OTTER completed a proof of length 103 and level 21. As is so typical in view of my interest in the hot list strategy, the experiment was then repeated with but one change: The heat parameter was assigned the value 2 rather than 1. In 206 CPU-seconds with retention of clause (18556), a 91-step proof was completed, one of level 26. Then, following the iterative aspects of the approach, the next experiment mirrored the preceding with one exception: New resonators were used, those corresponding to 87 positive deduced steps of the 91-step proof produced during the preceding experiment. In 192 CPU-seconds with retention of clause (22114), OTTER completed a 63-step proof of level 12, a reduction in proof length that was (for me) indeed unexpected. Then, a slight variant of the experiment was conducted, in which the value 1 (instead of 2) was assigned to the heat parameter. In 48 CPU-seconds with retention of clause (9397), a 118-step proof was found, one of level 27.

The last two cited results together provide a nice illustration of why an algorithm for finding shorter proofs would be difficult to produce, and they also provide some insight into why the methodology presented in this technical report was formulated. Specifically, although many experiments suggest that assigning the value 1 to the (input) heat parameter (when the hot list strategy is in use) is the most effective assignment, here one witnesses a far better result obtained with an assignment of the value 2.

At the same time the first of the two items (the 63-step proof) suggests that an even more elegant proof might be obtainable if one were to conduct a quite extensive set of experiments. Such a set of experiments would involve assigning a small time limit for each run and simply trying numerous combinations of option choices and parameter assignments, still emphasizing the methodology that proved so successful with the theorem DUAL-BA-3. One can easily feel impatience at the prospect of having to set up one experiment after another, even though the use of OTTER is often so rewarding and even though the modifications in each case take little actual time. McCune again came to the rescue: He wrote a program, super-loop, that systematically and sequentially sets up and runs experiments that focus on combinations of a variety of user-chosen options and assignments. To use super-loop, one attaches to the end of the designated input file a set of commands of the following type.

```
% end of fixed part

meta_hot_set(2,3,4).
x+y=y+x.
x*y=y*x.
(x + y) + z = x + (y + z).
(x * y) * z = x * (y * z).
end_of_list.

assign(heat,2,1,0,3).
assign(pick_given_ratio,4,5,6,7,8,9,10,12).
assign(max_weight,5,10,15,20,25).
flag(ancestor_subsume,set,clear).
```

For example, using the remainder of the specified input file, in one experiment OTTER will assign max_weight the value 15 and the pick_given ratio the value 7, will set ancestor_subsume, and will use three of the given four clauses in the context of the hot list strategy.

With access to super-loop and with a quick review of the methodology used to yield the 100-step proof of DUAL-BA-3, I decided to run the sequence of experiments determined by the just-cited set of commands, with one addition, namely, the use of the resonance strategy. I chose for the resonators the steps of McCune's proof of length 119, assigning to each the value 2. Except for the cited variations from experiment to experiment, the approach was reminiscent of that used to obtain the 103-step proof discussed earlier. I set the time limit to 120 CPU-seconds.

In the 209th experiment (directed by super-loop), a 30-step proof (of DUAL-BA-5) of level 6 was completed in 15 CPU-seconds. The crucial option choices and parameter assignments that were used were an assignment of the value 8 to the pick_given_ratio and the value 25 to the max_weight and the setting of ancestor subsumption; the hot list strategy was not used.

The success with DUAL-BA-3 (when studied with the methodology central to this technical report) coupled with the just-cited success with DUAL-BA-5 motivated the study of DUAL-BA-2; see [McCune96]. The theorem asserts that the following seven clauses axiomatize Boolean algebra.

```
(x + y) * y = y.
x * (y + z) = (x * y) + (x * z).
x + x@ = 1.
p(x,y,z) = (x*y@) + ((x*z) + (y@ * z)).
p(x,x,y) = y.
p(x,y,y) = x.
p(x,y,x) = x.
```

For the study, the following clause was also included because of the use of paramodulation.

```
x = x.
```

The theorem asks one to prove three properties of Boolean algebra, the same three properties proved in DUAL-BA-5, namely, a distributive law, an absorption law, and the fact that x+x@ is a constant. If one denies the **and** of the three properties, one obtains the following clause.

```
A + (B * C) != (A + B) * (A + C) | (A * B) + B != B | B * B@ != A * A@.
```

To initiate the study (with the goal of fulfilling McCune's request for a more elegant proof), I used his 135-step proof. The iterative approach led to the completion of a 93-step proof. Then, by using super-loop focusing on 89 deduced steps of the 93-step proof as resonators, an 86-step proof was completed. The proof was found in the 87th experiment directed by super-loop. Regarding options and values, the pick_given_ratio was assigned the value 5, the max_weight was assigned the value 8, ancestor subsumption was used, and the hot list strategy was not used; 6 CPU-seconds sufficed to produce the 86-step proof of level 22. Super-loop was again used, focusing on 82 deduced steps from the 86-step proof as resonators. In the third experiment, in 3 CPU-seconds with the value 12 assigned to the pick_given_ratio, the value 8 assigned to max_weight, and the use of ancestor subsumption, an 83-step proof of level 23 was completed.

I then had an inspiration, most likely resulting from some earlier experiments with Robbins algebra. In the next use of super-loop, I merely cleared eq_units_both_ways. (If this flag is set, then unit equality clauses, positive and negative, can be stored in both orientations; see [McCune94].) In the seventh experiment, in 4 CPU-seconds, with the pick_given_ratio assigned the value 12, the max_weight assigned the value 16, and the use of ancestor subsumption, OTTER found an 81-step proof of level 28.

By following the type of iteration just illustrated, a 77-step proof was completed, one of level 27, in 3 CPU-seconds. The experiment required assigning the value 12 to the pick_given_ratio, assigning the value 8 to the max_weight, assigning the value 1 to the heat parameter, using ancestor subsumption, and setting the flag eq_units_both_ways; only 3 CPU-seconds were required. The hot list contained the following four clauses.

```
x + x@ = 1.
p(x,x,y) = y.
p(x,y,y) = x.
p(x,y,x) = x.
```

The result was obtained in the 221st experiment directed by super-loop.

To test the iterative methodology further, I shifted focus from Boolean algebra to quasilattices, focusing on the theorem denoted by QLT-3. McCune and his colleague Padmanabhan [McCune96] had obtained with OTTER the first known equational proof that the following self-dual equation can be used to specify distributivity.

(((x ˆ y) v z) ˆ y) v (z ˆ x) = (((x v y) ˆ z) v y) ˆ (z v x)

(All earlier proofs are model-theoretic.) The proof I was asked to replace with a more elegant proof has length 183.

As with DUAL-BA-3, this theorem taxed the iterative methodology greatly, in the sense that many experiments were required to reach the best result, a proof of length 108 and level 14, completed in 166 CPU-seconds. Among the points of interest are the fact that McCune's proof (used as the starting point) has level 24 and the fact that, in contrast to many of the so-called final experiments in a study, this one required a few CPU-minutes. Also of interest, perhaps suggesting the difficulty of obtaining such a short proof and even suggesting the charm of the proof, the use of super-loop produced no further progress.

For the researcher intrigued by the challenge of finding a shorter proof, the following clauses can be used to attack the problem.

```
x = x.
x ˆ x = x.
x ˆ y = y ˆ x.
(x ˆ y) ˆ z = x ˆ (y ˆ z).
x v x = x.
x v y = y v x.
(x v y) v z = x v (y v z).
(x ˆ (y v z)) v (x ˆ y) = x ˆ (y v z).
(x v (y ˆ z)) ˆ (x v y) = x v (y ˆ z).
(((x ˆ y) v z) ˆ y) v (z ˆ x) = (((x v y) ˆ z) v y) ˆ (z v x).
A ˆ (B v C) != (A ˆ B) v (A ˆ C).
```

Then, I used the methodology to attack another theorem chosen by McCune from quasilattices. The theorem, QLT-5 [McCune96], asserts that the following self-dual equation can be used to express modularity in quasilattices.

(x ˆ y) v (z ˆ (x v y)) = (x v y) ˆ (z v (x ˆ y))

As in the preceding problem, the object was to find a proof even more elegant than that first obtained by McCune using OTTER, the first known equational proof. The so-called target proof has length 125 and level 19. To consider the problem, I used the following clauses.

```
x = x.
x ˆ x = x.
x ˆ y = y ˆ x.
(x ˆ y) ˆ z = x ˆ (y ˆ z).
x v x = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x ˆ (x v y) = x.
x v (x ˆ y) = x.
x * x = x.
x * y = y * x.
(x * y) * z = x * (y * z).
x * (x v y) = x.
x v (x * y) = x.
A ˆ B != A * B.
```

Before the use of super-loop was invoked, the basic methodology eventually led to a 95-step proof. Then, for some obscure reason, I initiated the attack with super-loop by using as resonators 101 deduced steps of a proof of length 106, obtained early in the use of the basic methodology. Perhaps the reason was that use of the 95-step proof would lead no further, that it would place the program in a cul de sac, whereas starting a bit further back might avoid the dead end. As in some of these experiments, the approach was to use super-loop to find (if successful) a shorter proof, then use the positive deduced steps of the shorter proof for the next use of super-loop.

In Experiment 3745 of the last use of super-loop, with the pick_given_ratio assigned the value 9, the max_weight assigned the value 15, the heat parameter assigned the value 1, and the use of ancestor subsumption, 7 CPU-seconds sufficed to yield a proof of length 30 and level 13. Two clauses, the following, were present in list(hot).

```
x v x=x.
x v y=y v x.
```

This result was singularly satisfying, and it demonstrated splendidly (because of occurring in Experiment 3745) the value of using super-loop.

6. A Promising Shortcut

With little doubt, one interested in finding a more elegant proof than that in hand might wish access to an approach that is far less complex, an approach that requires far fewer iterations and judgments to be made. The following two-step approach is easy to apply and often does yield impressive results.

The first step of the approach (or methodological shortcut) merely takes the input file that produced the target proof and adds the use of the hot list strategy. For the (input) hot list, one chooses for its members so-called simple or short clauses (as was the case with DUAL-BA-3, the theorem that is the focus for much of this technical report). If successful, the program will find a shorter proof and will find it in far less CPU time. If the first step succeeds, the second step merely takes the input file used in the first step and adds the use of the resonance strategy. The resonators are the correspondents of the positive deduced steps (required to be unit clauses) of the proof obtained in the first step of the abridged approach. The results of applying the just-cited two-step approach to various theorems are next in order.

Quite naturally, the first theorem that was attacked was DUAL-BA-3, that studied in detail in the 26 cited experiments; see Section 3. Three experiments were conducted. The first was designed to find (if possible) a proof without the use of the hot list strategy or the resonance strategy. The second experiment, as dictated by the two-step approach, differed from the first only in its use of the hot list strategy. The members of the hot list were the following.

```
x + x@ = 1.
x * x@ = 0.
```

The third, again dictated by the approach, added the use of the resonance strategy, where the resonators were the correspondents of the positive deduced steps yielded by the second experiment. Of course, I was somewhat prepared for the possibility that any of the three experiments might fail to complete a proof.

Fortune smiled: Each of the three experiments produced a proof. In order (on the equivalent of a SPARCstation-10), the CPU times in seconds are 6681, 585, and 242. In order, the levels are 96, 75, and 64. The respective lengths are 803, 515, and 475. The ensemble of the three experiments suggests that the shortcut is promising, having reduced the proof length from 803 to 475. (Regarding the proof length of 803, in contrast to the expected 816, apparently the precise input file used by McCune is lost to antiquity.)

Next, three corresponding experiments were conducted for the theorem DUAL-BA-5. As expected, the first experiment produced a proof of length 119; its level is 16, and it was completed in 52 CPU-seconds. Two clauses, the following, were then placed in the hot list.

```
x+y=y+x.
x*y=y*x.
```

The second experiment required 100 CPU-seconds to complete a proof, with level 38 and length 160. These results were a surprise, for adding the use of the hot list strategy so often reduces the CPU time and yields a proof of smaller length. I was faced with a dilemma: (1) truncate the study of DUAL-BA-5, concluding that the shortcut was a failure for this theorem; (2) intervene by making some modification to the shortcut; or (3) continue the study as if nothing unusual had occurred.

The third possibility was (to me) clearly the preferred, for one should apply a suggested approach (that is being tested for value) even in the face of intermediate disappointment. Also, I was most intrigued to determine what would happen if I used so many resonators, those corresponding to the positive deduced steps of the 160-step proof (only 101 were used in the first visit to this theorem, as discussed in Section 5). Therefore, as if nothing untoward had occurred, 159 resonators were adjoined, and the third experiment was conducted.

In 36 CPU-seconds, a proof of level 12 and length 47 was found. The result, which vindicated the shortcut, is indeed satisfying, if one takes into account that the best proof obtained with the basic methodology followed by the use of super-loop has level 6 and length 30.

The next target was DUAL-BA-2. The first experiment behaved as expected in regard to proof length, completing a proof of length 135 and level 32 in 55 CPU-seconds. The following clauses were then placed in the hot list for the second experiment.

```
x + x@ = 1.
p(x,x,y) = y.
p(x,y,y) = x.
p(x,y,x) = x.
```

Reminiscent of the preceding study, the second experiment required 229 CPU-seconds to complete a proof, one of level 37 and (not enticing) length 150. When 144 resonators were then adjoined, corresponding to positive deduced steps of the 150-step proof, 36 CPU-seconds were sufficient to find a proof of level 27 and length 120.

Regarding further evidence, two additional theorems were studied with the two-step shortcut. The first, QLT-1, is from quasilattices. The following clauses were used to attack this theorem.

```
x = x.
x ^ x = x.
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).
x v x = x.
x v y = y v x.
(x v y) v z = x v (y v z).
(x ^ (y v z)) v (x ^ y) = x ^ (y v z).
(x v (y ^ z)) ^ (x v y) = x v (y ^ z).
x ^ (y v (x ^ z)) = x ^ (y v z).
A ^ (B v C) != (A ^ B) v (A ^ C).
```

The first nine clauses axiomatize quasilattices. The theorem asserts that the tenth clause is a new way to express distributivity in quasilattices. The eleventh clause is the denial of distributivity. McCune had obtained a 47-step proof. In the three experiments with the shortcut methodology, I obtained proofs of length 47, 46, and 61, respectively. The time required was 3, 6, and 5 CPU-seconds, and—of some small disappointment—the levels are 10, 15, and 18, respectively. The conclusion is that the shortcut was far from effective for this theorem.

However, the results naturally led to an experiment in which the hot list was trimmed from the following four members to just two, the first and the third.

```
x ^ x = x.
x ^ y = y ^ x.
x v x = x.
x v y = y v x.
```

This move led to a proof of level 8 and length 19, completed in 5 CPU-seconds. Then when resonators were adjoined that correspond to the positive deduced steps of the 19-step proof (to satisfy the natural curiousity), in 3 CPU-seconds, a proof of level 5 and length 21 was found.

Finally, I thought one additional experiment must be run. The reasoning was that if progress was made by trimming the size of the hot list, and if some of that progress was then lost, perhaps a reduction in max_weight was in order with all other items left unchanged. In 1 CPU-second, after the max_weight was changed from 19 to 15, OTTER completed a proof of level 4 and length 8. (For the researcher who wonders how wild is the space of proofs, one of the experiments focusing on this theorem yielded a proof of level 70 and length 386, providing an impressive contrast to the proof of length 8.) One perhaps unfortunate conclusion from this ensemble of experiments focusing on QLT-1 is that even the shortcut methodology benefits from tinkering. Yet, if one steps back from the study a bit, tinkering should be expected, for mathematics and logic so seldom admit an algorithmic approach when the prize is an elegant proof.

Before turning to the last theorem on which the promising shortcut was tested at this time, I answer the following natural question that an experienced researcher might ask. What results are obtained if one conducts three experiments using the effective max_weight (15) rather than that which was originally used (19) and using the smaller hot list? If, in addition to testing the shortcut, one has the goal of finding an elegant proof, the new set of three experiments begins rather explosively, completing in 5 CPU-seconds a proof of level 8 and length 25. The second experiment brings the excitement and the attempt to a halt, producing no proof when the smaller hot list is then added. Other paths are left as research topics and challenges, especially where the goal is to find a proof of length strictly less than 8.

The final theorem that was studied in the context of the promising shortcut is denoted by McCune as LT-8. This theorem from lattice theory says that the meet (intersection) operation is unique in the sense that, given a set *L* of elements that have the same join (union) operation and two possibly different meet operations, the two meet operations are the same. The following clauses were used to study this theorem, again suggested by McCune.

```
x = x.
%  (v,^) is a lattice.
x ^ x = x.
x ^ y = y ^ x.
(x ^ y) ^ z = x ^ (y ^ z).
x v x = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x ^ (x v y) = x.
x v (x ^ y) = x.
%  equations to make (v,*) a lattice.
x * x = x.
x * y = y * x.
(x * y) * z = x * (y * z).
x * (x v y) = x.
x v (x * y) = x.
A ^ B != A * B.
```

McCune’s proof, used as a target, has length 19 and level 6. In the first experiment (in which neither the hot list strategy nor the resonance strategy was used), 50 CPU-seconds were required to produce a proof of length 19 and level 6. For the second experiment, use of the hot list strategy was added with the following clauses in the hot list.

```
x ^ x = x.
x v x = x.
x ^ y = y ^ x.
x v y = y v x.
```

The required time increased to 119 CPU-seconds, with a proof of length 18 and level 6. For the third experiment, 18 resonators were adjoined, the correspondents of the deduced steps of the just-cited proof. Although but 1 CPU-second sufficed, the resulting proof still has length 18; in that its level is 9, the proof is different from the 18-step proof yielded by the second experiment. In the second 18-step proof, commutativity of join was not required. Of the deduced steps, five are not shared by the two 18-step proofs.

Annoyed by the insignificant decrease in proof length, influenced by what occurred in the study of QLT-1, and motivated by the objective of presenting McCune with a more elegant proof and the objective of offering an interesting challenge, I conducted two tightly coupled additional experiments. First, I removed from the hot list the two clauses that encode commutativity. In 55 CPU-seconds, OTTER produced a 19-step proof of level 6. The proof contains the same deduced clauses as the 19-step proof obtained when neither the hot list strategy nor the resonance strategy was used. Then, I used the 19 deduced steps as resonators, noting that three of them are not among the 18 that were used in the third experiment (that which yielded an 18-step proof). Success occurred: In 1 CPU-second, a proof was completed, one of length 14 and level 5. Immediately, this advance motivated me to tinker just a bit more, seeking an extra drop (or perhaps more) of satisfaction. When the pick_given_ratio (which had been assigned the value 4) was commented out, causing OTTER to choose as the focus of attention clauses by weight only, 1 CPU-second sufficed to complete a 13-step proof of level 6.

7. Research, Review, and Remarks

In this closing section, research topics are briefly discussed, a review of the content of this technical report is given, and various remarks of a somewhat disjointed nature are made. Of the various alternatives, motivated by my typical intertwined list of objectives, I have chosen to present the material of this section as if the three areas (research, review, and remarks) are tightly coupled.

The focus in this technical report is on automating the search for elegant proofs. (For the actual proofs, or proofs of a similar flavor, see [McCune96] or either of the following two Web addresses, http://www.mcs.anl.gov/home/mccune/ar/monograph/ or http://www.mcs.anl.gov/people/wos/index.html.) Although a precise definition of elegance is debatable, four properties are studied: proof length, term structure, variable abundance, and deduced-step complexity. The first (proof length) of these four is featured.

Clearly, proof length is somewhat hazy. For example, frequently in a mathematics book, one finds a proof in which symmetry and transitivity of equality are used implicitly only. For OTTER, their use would be explicit and, if used, add to the proof length. On the other hand, the use of demodulators (rewrite rules, for example, in the context of canonicalization) does not ordinarily contribute to proof length. Proof length in this technical report and for OTTER measures the number of specific steps that are present but that are not part of the input. (Sometimes, the cited length of a proof is not what is expected, for proof length does not measure, for example, the number of equality substitutions; in particular, as noted, the substitutions resulting from the use of demodulators do not contribute to proof length. If one wishes to see explicitly all substitutions and their effect, as one might with a preference for a different definition of proof length, OTTER offers the needed feature, namely, build_proof_object, which gives all of the gory details.) Therefore, here most or all ambiguity regarding this aspect of elegance is removed.

In contrast to merely touching on means offered by OTTER for addressing the second through the fourth aspects of elegance, this technical report features a methodology for automating the search for shorter proofs; in Section 6, a shortcut is studied. A review of the details of the methodology shows how complex such a search can be. Although the approach on which this technical report is based is clearly not an algorithm—a practical algorithm appears to be out of reach—the evidence given here suggests that the methodology can be indeed effective. One of the more graphic illustrations concerns starting with a proof of length 816 and, by applying the approach, eventually obtaining a proof of length 100. The approach relies heavily on OTTER, McCune's automated reasoning program. In Section 3, the key experiments, 26 of them, are detailed with commentary; they are summarized in Table 1 at the end of that section.

To dispel the thought that the space of proofs, in the context of proof length, does *not* behave wildly, one merely considers the theorem QLT-1; see Section 6. The best proof that was obtained has length 8. In contrast, in the search for a shorter proof, the program also completed a proof of length 386. The longer proof was found in 56 CPU-seconds with the heat parameter assigned the value 1, the pick_given_ratio assigned the value 6, max_weight assigned the value 27, and without the use of ancestor subsumption but with the use of eq_units_both_ways. Two clauses were present in the hot list, that for idempotence of join (union) and that for commutativity of join. The proof was found when the program super-loop was used (in job 4735), a program that permits the researcher to run a sequence of OTTER experiments.

The shorter proof (that of length 8) was completed with assigning the heat parameter the value 1, the pick_given_ratio the value 4, max_weight the value 11, and with the use of both ancestor subsumption and eq_units_both_ways. The hot list differed from that used to find the 386-step proof; it consisted of idempotence of join and of meet. The shorter proof was found with the first job directed by super-loop, in the same run in which the longer proof was completed.

In contrast to the first aspect of elegance studied here (that of proof length), the second through the fourth aspects of elegance admit a quite straightforward attack. The second property of elegance, term structure, can be directly addressed by relying either on weighting or on demodulation. The third property, variable abundance, is directly addressable by assigning the desired value to max_distinct_vars. For the fourth property, deduced-clause complexity, the value assigned to max_weight is the key. (As an aside, a tempting research problem focuses on adding the feature of being able to assign a value to max_level, instructing the program to retain no clauses whose level exceeds the chosen value; access to such a feature might give the researcher and the program another weapon for seeking more elegant proofs.)

Proof length (in contrast to the other three discussed properties of elegance)—especially when the goal is to find a ‘‘short’’ proof—naturally brings one face to face with cul de sacs and, most piquant, the need to follow a tortuous path that is often indeed narrow. For example, some experiments yield no proof at all unless the pick_given_ratio is assigned the value 11; all other values yield nothing. For a second example—especially because of the appeal of proving lemmas early—intuitively, one might conjecture that it is always profitable to have the program focus on the clauses in the initial set of support before focusing on any deduced clause (to drive the reasoning); in fact, such a move sometimes leads to finding no proofs at all.

One might immediately wonder why the task of seeking shorter proofs is so complicated, why the path to success can be so convoluted. At least for problems in which equality is featured, the answer rests in part with the need to find and then use appropriate demodulators (rewrite rules). Indeed, even a small change in the order in which demodulators are found and applied can produce dramatic changes in the program’s performance, in the paths that are pursued, and in the results that are obtained. Nevertheless, such diverse behavior is typically what is required, if the objective is to find ‘‘short’’ proofs. Often, the objective is reached only by radically perturbing the search space. Fortunately, OTTER offers various means in this regard, not the least of which is use of the hot list strategy. As detailed, to find the proofs cited with the methodology presented in this technical report required basing later runs on earlier runs or on combinations of earlier runs. At this time, I can offer no rules for deciding which early runs to key on. For the researcher desirous of a ‘‘quick fix’’, the shortcut presented in Section 6, however, is often effective to a significant extent.

That use of an automated reasoning program is necessary if one is to obtain so-called best results seems evident to me. For a supporting view from a researcher not in this field of automated reasoning, I need only turn to Dana Scott and his reaction to a success concerning two-valued sentential (or propositional) calculus [Wos91a]. I had sent Scott a new axiom system for this area of logic and the corresponding 8-step proof; he returned to me a 7-step proof [Scott90], suggesting that the shortest in that regard had been found. Shortly thereafter, I used OTTER with a different set of options and, most unexpectedly, found a 4-step proof, one that appeared to me to indeed be elegant. When I notified Scott of this 4-step proof, his reaction, given by e-mail, was that it might indeed be ‘‘a very neat proof that would not be obvious to a human investigator’’. I offer this anecdote as but one indication of why the assistance of a program such as OTTER might be just what is needed. Indeed, this program often attacks a theorem in a manner that a mathematician or logician would simply avoid; examining in detail a huge space of conclusions is simply untenable.

In part as a challenge and in part to add to one’s appreciation of the power of some automated reasoning programs, using the following clauses (pertinent to the study of two-valued sentential calculus and the communication with Scott), one might attempt to find a 4-step proof (by contradiction) of the type just discussed, with or without a program.

```
-P(i(x,y)) | -P(x) | P(y).
P(i(i(i(x,y),z),i(y,z))).
P(i(i(i(x,y),z),i(n(x),z))).
P(i(i(u,i(n(x),z)),i(u,i(i(y,z),i(i(x,y),z))))).
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANS(step_th_59).
```

The first of the five clauses when considered with hyperresolution encodes condensed detachment, the inference rule often used to study various areas of logic. The second through the fourth are what Lukasiewicz calls theses 19, 37, and 60, respectively, the axiom system I discovered with OTTER’s assistance. The fifth clause is the negation of thesis 59; one of the Lukasiewicz axioms systems for this area of logic consists of theses 19, 37, and 59.

In contrast to the study of a property of elegance in isolation, OTTER can also be used to study combinations of properties simultaneously. For an example (studied in [Wos96a]) that concerns two aspects of elegance simultaneously, proof length and term structure, I focus on many-valued sentential calculus (which is weaker than two-valued sentential calculus). The theorem asserts that the fifth of the following five formulas is deducible from the first four; again the inference rule is condensed detachment.

(*MV*1) i(x,i(y,x))
(*MV*2) i(i(x,y),i(i(y,z),i(x,z)))
(*MV*3) i(i(i(x,y),y),i(i(y,x),x))
(*MV*4) i(i(n(x),n(y)),i(y,x))
(*MV*5) i(i(i(x,y),i(y,x)),i(y,x))

Lukasiewicz conjectured that the five formulas axiomatize many-valued sentential calculus; the function *i* can be interpreted as *implication* and the function *n* as *negation*. The (primary) challenge was met by Wajsberg (page 145 in [Lukasiewicz70]) by proving that the five given formulas *do* provide an axiomatization of the calculus. When years later Meredith proved that *MV*5 is dependent on *MV*1 through *MV*4 [Lukasiewicz70], the related challenge (for automated reasoning) was born: With an automated reasoning program unaided by a researcher's guidance, obtain a proof of that fact. Although the challenge was met in Chapter 5 of [Wos96a] by relying on a not-very-often used strategy, one might find an independent study of the theorem more than intriguing.

In the context of elegance (rather than simply finding *any* proof), the challenge was to find an elegant proof simultaneously with respect to proof length and term structure. Regarding term structure, the goal was to find a proof that avoids the use of terms of the form *n*(*n*(*t*)) for any term *t*. At least one of my colleagues thought the goal unreachable, especially if coupled with finding a ‘‘short’’ proof. As it turns out, with invaluable assistance from OTTER, I found a 34-step proof of the theorem and a proof avoiding the *n*(*n*(*t*)) terms; see Chapter 11 of [Wos96a]. For the researcher who enjoys a substantial challenge, whether one uses OTTER or not, finding such a proof will prove difficult—or worse.

Regarding a comparison of a short proof (one of length 100) that completes a study with the long proof (one of length 816, serving as a target) that often initiates a study, the theorem DUAL-BA-3 provides an interesting example. Of the deduced steps of the (short) 100-step proof, 75 are not present among the deduced steps of the (long) 816-step proof obtained by McCune. To be candid, I was overwhelmed by this comparison. However, the result is indeed consistent with what I have learned from more than thirty years of using an automated reasoning program. Specifically, in contrast to what I would have conjectured as a graduate student in mathematics at the University of Chicago, the number of widely differing proofs for a given theorem is often spectacularly abundant. I suspect that mathematicians and logicians who have little or no experience with an automated reasoning program will find it a delight to examine the myriad of diverse but interesting proofs that can be completed.

Of a different nature, and surprising even to the experienced user of an automated reasoning program, is the failure to return a given proof all of whose steps are being used to guide the search. In fact, sometimes the use of proof steps as resonators coupled with a max_weight no larger than the value assigned to the resonators leads to a shorter proof, as can occur when, for example, the hot list strategy is used; see Chapter 13, Section 5 of [Wos96a].

Especially for the researcher new to the use of OTTER, one feature is of particular interest, namely, the autonomous mode. The choice of this option removes for the user the need to make other option choices, for the program sets them based on various syntactic criteria. If one is familiar with the options, but not sure which values to assign, then one can rely on super-loop; see Section 5. As for what to vary when using super-loop, I recommend max_weight, the pick_given_ratio, the heat parameter, the number of elements in the hot list, the setting and clearing of ancestor_subsume, and (if appropriate) the setting and clearing of eq_units_both_ways. By way of a piquant phenomenon relevant to the use of ancestor subsumption, although the presence of back subsumption is virtually required in most cases, to my surprise its absence (clearing it) can lead to an interesting proof.

From a general perspective, one finds that the iterative methodology (clearly not an algorithm) often returns shorter proofs and in rather small CPU times. Immediately several challenges arise, challenges of finding proofs shorter than those cited in this technical report. One can rely on the approach taken here, or one can devise a radically different approach. If the former, one must expect to (at least briefly) examine results obtained early to decide on which combination of options and assigned values to use in later runs. One must also expect to make choices regarding the members of the initial hot list, when the hot list strategy is in use. Fortunately, a good rule to follow suggests the members be short clauses, which (to me) seems natural and, as indicated, can lead to significant progress. To begin applying the approach, one can either initiate the attack with a known proof or attempt to obtain a proof with OTTER. As expected (in view of the nature of the problem of seeking elegant proofs), one will often be forced to backtrack or to combine option settings from various runs.

In addition to the just-given rule for clause placement in list(hot), one might wish some guidance regarding the placement of the clause or clauses that result from assuming that the theorem under study is false. When the denial of the theorem is encoded with a negative unit clause, my preference is to place that clause in list(passive); the elements of that list are used to detect unit conflict (as a test for proof completion) and used also for forward subsumption. My preference is in part based on the desire to seek forward proofs, rather than, say, bidirectional proofs. However, because clauses in list(passive) are not subject to demodulation, one may be forced to place such a denial clause in, for example, list(sos). Indeed, the search for a proof may require the demodulation of the denial clause, or the search may require paramodulation into the denial clause, neither permitted with members of the passive list.

Many of the obstacles discussed in this technical report (see, for example, Section 4) are not inherent in automated reasoning; instead, they are actually present in the search for a proof in general (whether by program or by person). Those familiar with the literature of mathematics and of logic can most likely cite various instances of theorems whose proof eluded the researcher for years. Proof finding, especially if one is seeking an elegant proof, is indeed challenging, often complicated by the narrowness and tortuousness of the path that must be followed if one is to succeed. For example, unaided, the researcher can be misled into believing that some particular property of the underlying theory merits emphasis. For a second example, when aided by the use of an automated reasoning program, one sometimes finds that the decrease by 1 in the assignment of the value to a single parameter proves to be the crucial move.

Regarding the first example, I can offer little other than the accrual of experience and the guidance of other researchers. As for the second example, fortunately, one can rely on super-loop or on the autonomous mode. The use of either markedly reduces the need (for the user of a program such as OTTER) to master the fine points of automated reasoning. Even the researcher familiar with OTTER will find super-loop and the autonomous mode useful, as suggested by the fact that some of the successes reported here relied on assigning the value 4 to max_weight where others required an assignment of 40.

My main research, in the context of elegance, has focused on problems in which equality was the only or the dominant relation and on problems featuring the use of condensed detachment. Quite different from such problems are problems focusing on non-Horn sets. Related to non-Horn problems is the problem of finding elegant proofs when the equality relation is heavily mixed with relations not concerned with equality, as occurs, for example, in the study of set theory. Just as reported here, either area of research most likely will meet with cul de sacs and occasionally require reliance on a proof far from the best in hand. Such is the nature of searching for elegant proofs.

At the other end of the spectrum, in the context of automated reasoning, more than occasionally the simplest approach to proof checking some given proof can fail, the approach in which the proof steps are placed in a weight_list with an assignment to each of, say, the value 2 and with an assignment of 2 to max_weight. Failures of the cited type are especially likely when equality, and hence demodulation, are in focus. Even more complicated than one might expect, the use of proof steps as resonators in a second run, where the resonators are the correspondents of the deduced steps of a proof obtained in a first run, and where all other conditions are unchanged, often does not result in the program simply returning the proof found in the first run.

Of course, perhaps the most pressing question concerns identifying the key ingredient for finding elegant proofs, for proving a new theorem, for completing a difficult assignment, and for succeeding in research when one is relying on the assistance of an automated reasoning program. Predictably, my answer is the one I give when asked about the key ingredient for sharply increasing the power of such a program, namely, access to a variety of strategies, some to restrict the reasoning and some to direct it. For evidence that supplements the results presented here regarding the importance of strategy, the following experiment suffices.

Again, the theorem in focus is DUAL-BA-5, whose best proof known to me had length 30, that is, until a different strategic approach was taken by my colleague McCune. He removed the restriction on paramodulation that blocks paramodulation *into* nonunit clauses, and he added the use of what is called *basic paramodulation* [Bachmair92]. Briefly, the cited strategy prevents paramodulation *into* terms that arise by instantiation of variables in ancestor clauses; in particular, nonvariable terms in input clauses are marked as admissible *into terms*, and terms in paramodulants that correspond to marked terms in the parents are marked as admissible. He also decided to focus on a proof of distributivity only. To my amazement, the automated reasoning program he used produced an 18-step proof. He then used Veroff's *hints strategy* [Veroff96], using as hints the steps from the

18-step proof, and obtained a 17-step proof of DUAL-BA-5. I then used super-loop with McCune's approach, and OTTER found a 16-step proof of DUAL-BA-5; OTTER also found a 13-step proof of distributivity.

Finding the 16-step proof demonstrates the value of using strategy, but it also suggests what may exist in the context of elegant proofs. After all, the study of DUAL-BA-5 began with a proof of length 119. Neither I nor McCune would have believed possible the completion of a proof of length 16. Therefore, research designed to find proofs dramatically shorter than those cited in this technical report could prove fascinating and, perhaps, markedly rewarding.

Regarding research of a more general nature, (in the context of automated reasoning) clearly merited are strategies aimed at dramatically reducing the CPU time required to complete proofs. Admittedly, such reductions do not impress all researchers. However, often overlooked is the fact that theorems and, even more significant, open questions that were out of reach will become within reach of such a program as the required CPU time decreases sharply. Rather than with the reduction in time—which one might suggest is obtainable merely by using a more powerful computer—the significance rests with the cause of the reduction. Specifically, if the saving in time results from sharply reducing the number of paths to be traversed—which is not addressable with a faster computer—progress is occurring. Indeed, as Bledsoe has remarked more than once, the fastest computer is not the solution, for the space of possible conclusions to be deduced is indescribably large. Were it not for access to strategy, (it seems clear to me) an attack on deep questions or hard problems would almost always fail, for there exist far too many ways to get lost. Despite years of experience and the accrued intuition, even the talented mathematician or logician can get lost, often pursuing one false lead after another. The use of strategy, such as the set of support strategy, decreases the likelihood of the program getting lost, tends to focus its attention on more profitable paths of inquiry. Ideally, additional strategies offering power somewhat comparable to that offered by the set of support strategy would mark a most significant advance for automated reasoning.

The following three themes are central to the research presented in this technical report. First, without the use of a program such as OTTER, many elegant proofs would remain undiscovered. Second, as expected by those familiar with my research, I am certain that the greatest advances in automated reasoning when measured by the power to prove a theorem will result from the formulation of yet more strategies, some to restrict the reasoning and some to direct it. Third, OTTER functions as an invaluable research assistant for the pursuit of knowledge of various types when the study focuses on mathematics or on logic.

I can say with conviction that if one learns how to use OTTER, one will in the long run save many more hours than one spends in the learning, especially if one's interest is an area of abstract algebra.

Remarks. Portions of this technical report have been expanded, refined, replaced, or modified in a manuscript entitled ''Automating the Search for Elegant Proofs'' [Wos97], which is to appear in the *Journal of Automated Reasoning.*

References

[Bachmair92] Bachmair, L., Ganzinger, H., Lunch, C., Snyder, W., ''Basic paramodulation and superposition'', pp. 462-476 in *Proc. CADE-11, Lecture Notes in Artificial Intelligence,* Vol. 607, ed. D. Kapur, Springer-Verlag: Berlin, 1992.

[Kalman78]. Kalman, J., ''A shortest single axiom for the classical equivalential calculus'', *Notre Dame J. Formal Logic* **19,** 141-144 (1978).

[Kalman83] Kalman, J., ''Condensed detachment as a rule of inference'', *Studia Logica* **42,** 443-451 (1983).

[Lukasiewicz70] Lukasiewicz, J., ''The equivalential calculus'', pp. 250-277 in *Jan Lukasiewicz: Selected Works,* ed. L. Borkowski, North-Holland: Amsterdam, 1970.

[McCune90] McCune, W., *OTTER 2.0 Users Guide,* Technical Report ANL-90/9, Argonne National Laboratory, Argonne, Illinois, 1990.

[McCune91] McCune, W., *What's New in OTTER 2.2,* Technical Memorandum ANL/MCS-TM-153, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1991.

[McCune94] McCune, W., *OTTER 3.0 Reference Manual and Guide,* Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, 1994.

[McCune96] McCune, W., and Padmanabhan, R., *Automated Deduction in Equational Logic and Cubic Curves,* Lecture Notes in Computer Science, Vol. 1095, Springer-Verlag: Heidelberg, 1996. See http://www.mcs.anl.gov/home/mccune/ar/monograph/ for additional information.

[Padmanabhan73] Padmanabhan, R., and Quackenbush, R. W., ''Equational theories of algebras with distributive congruence'', *Proc. of AMS* **41**, no. 2, 373-377 (1973).

[Scott90] Scott, D., Private communication, 1990.

[Veroff96] Veroff, R. ''Using hints to increase the effectiveness of an automated reasoning program: Case studies'', *J. Automated Reasoning*, **16,** no. 3, 223-239 (1996).

[Wos91a] Wos, L., ''Automated reasoning and Bledsoe's dream for the field'', pp. 297-345 in *Automated Reasoning: Essays in Honor of Woody Bledsoe,* ed. R. S. Boyer, Kluwer Academic Publishers: Dordrecht, 1991.

[Wos91b] Wos, L., Overbeek, R., and Lusk, E., ''Subsumption, a sometimes undervalued procedure'', pp. 3-40 in *Festschrift for J. A. Robinson,* ed. J.-L. Lassez and G. Plotkin, MIT Press: Cambridge, Mass., 1991.

[Wos95a] Wos, L., ''The resonance strategy'', *Computers and Mathematics with Applications* (special issue on automated reasoning) **29**, no. 2, 133-178 (February 1995).

[Wos95b] Wos, L., ''Searching for circles of pure proofs'', *J. Automated Reasoning* **15**, no. 3, 279-315 (1995).

[Wos96a] Wos, L., *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial,* Academic Press: New York, 1996. See http://www.mcs.anl.gov/people/wos/index.html for input files and information on shorter proofs.

[Wos96b] Wos, L., ''OTTER and the Moufang identity problem'', *J. Automated Reasoning,* **17,** no. 2, 259-289, 1996.

[Wos96c] Wos, L., ''The power of combining resonance with heat'', *J. Automated Reasoning* **17,** no. 1, 23-81, 1996.

[Wos97] Wos, L., ''Automating the search for elegant proofs'', *J. Automated Reasoning* (to appear).