Automating the Search for Elegant Proofs*

Larry Wos

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4801

Abstract

The research reported in this article was spawned by a colleague's request to find an elegant proof (of a theorem from Boolean algebra) to replace his 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. In addition to (usually) being more elegant, shorter proofs often provide the needed path to constructing a more efficient circuit, a more effective algorithm, and the like. 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. These strategies, as well as other features on which the methodology relies, do exhibit tendencies that can be exploited in the search for shorter proofs and for other objectives. 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. Problem Origin

Two independent themes are central to this article. One is concerned with evidence of advances made in automated reasoning, and the other is concerned with a methodology for attacking the general problem of finding shorter proofs. Though independent, both themes are addressed by a single success with a specific problem posed by W. McCune.

Using his program OTTER [McCune90,McCune91,McCune94], McCune had obtained an 816-step proof of a theorem in Boolean algebra; he calls the theorem DUAL-BA-3 in a recent monograph by him and his colleague R. Padmanabhan [McCune96]. The problem he posed to me was to find a far more elegant proof than his 816-step proof, with the focus strictly on proof length. This article shows how the request was met by finding a proof of length 100, details the methodology that was used, and demonstrates that automated reasoning has met an important test measuring its advance.

1.1. Measuring Progress

The early years of a field are quite often occupied with developing the theory, formulating necessary components, and—where a computer is involved—designing and implementing programs. Eventually, however, the field must be evaluated. In the case of automated reasoning, crucial to an evaluation is the testing of its programs on problems supplied by others. By solving a problem supplied by another researcher (in this case, McCune), I provide yet more evidence of the value of automated reasoning. The successful gathering of the evidence is especially appealing—even piquant—because McCune also designed and implemented the very program, OTTER, I used to solve the problem he posed.

---

*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.

1.2. A Methodology or an Algorithm?

Although the problem that is solved concerns the finding of an elegant proof for a specific theorem, where elegance focuses solely on proof length, the research that produced the desired solution also resulted in the formulation of a general methodology for automating the search for elegant proofs, where elegance is not confined to proof length. Where proof length is the concern, the methodology is iterative, usually beginning with a ''long'' proof and, if the methodology succeeds, culminating with a ''short'' proof. As shown in Section 4, 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*.

Ideally, the world wishes access to an algorithm. Perhaps disappointing to some, the approach presented here is indeed not an algorithm. In fact, my experiments suggest that an algorithm for producing shorter proofs is in the main impractical. Nevertheless, certain program options and parameters are beneficial, and, to aid the researcher who wishes to use OTTER, in Section 3 I discuss tendencies concerning their use. For example, quite often the setting of a smaller (rather than a larger) maximum (max_weight) on the length of retained conclusions promotes the completion of shorter proofs. This phenomenon is by no means dependent on the use of OTTER; indeed, with his program EQP, McCune was able to answer the long-standing open question on Robbins algebra by producing a 194-step proof when the max_weight was assigned the value 60, and he later obtained an 86-step proof when the max_weight was assigned the value 50 (with no other changes made). An intuitive explanation asserts that the presence in a proof of a complicated formula or equation forces the inclusion of numerous steps used to extract from the complicated item what is needed. Where I have some notion of the type just given, I supply it. Perhaps such hints will enable a researcher to explain which tendencies hold in which cases and why.

Although I do not share in the view, I have colleagues who suspect that a practical algorithm can be found. I offer this as a topic for research, if one is so disposed. Indeed, just as the experiments of many years provided the basis for McCune's formulation of an autonomous mode for OTTER to apply—a mode that automatically sets options and makes choices that the researcher would otherwise make—perhaps the experiments covered in this article will provide a basis for a far more automated approach for finding shorter proofs. If such were to occur, the practical consequences would be significant, for mathematics, circuit design, program synthesis, and the like.

1.3. Perspective

To put in perspective the research reported here and to provide some insight into the difficulty of pursuing the topic of a more fully automated search for elegant proofs, two items merit mention. First, I have studied such an automated approach for some years now; see [Wos96a]. However, my in-depth experiments focused mainly on problems in which equality played little or no role. The presence of equality presents many subtle obstacles, some of which concern the use of demodulation. For example, even a simple experiment whose object is to cursorily check a given proof by supplying as weight templates the proof steps, each assigned a weight of say *k*, coupled with an assignment of *k* to the max_weight can result in OTTER completing *no proof* at all.

Second, none of my experiments had ever begun with a proof of such magnitude, in this case, a proof of length 816. Perhaps the longest proof that initiated an experiment in my earlier studies was one of length 63.

Because of these two factors, the culmination of my research in producing a 100-step proof was most startling.

1.4. Elegance—Importance and Properties

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. In addition to the aesthetic aspects of elegance of concern to mathematics and logic, practical aspects also exist, easily illustrated when the focus is on proof length. Indeed, the methodology here may also serve well in the context of constructing more efficient circuits, synthesizing more effective algorithms, and the like. Often a ''short'' proof (when it is, in effect,

constructive) can be used to provide the needed path, for example, by using an answer literal to extract from the proof the sought-after object. The desired object might be a circuit relying on few AND gates or an algorithm in which divide rarely occurs.

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. Each of the cited properties has aesthetic appeal, but each is also relevant to practical considerations, for example, in circuit design. Here I focus mainly on the first property (proof length), giving brief attention to the second through the fourth properties. (Should one happen to wonder whether proof length is indeed relevant to elegance, one need only examine the history of mathematics, for example, that focusing on the various proofs of the Church-Rosser theorem.) 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 article, 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. In the vast majority of cases, 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 article, 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. To dispel the thought that the use of a (frequently impractical) breadth-first (or level-saturation) approach suffices, consider the case in which such an approach yields at level 3 a proof of length 7 and at level 4 a proof of length 4; see Chapter 3 of [Wos96a] for a more detailed discussion. 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.)

1.5. A Narrow Window

To attack the tough problem of finding ''short'' proofs, OTTER offers various means—although far from producing an algorithm; see Section 2. (Regarding the other aspects of elegance discussed in this article, one learns that OTTER offers, in most cases, just what is needed.) What becomes clear (in Section 4, where an abbreviated sequence of experiments illustrating the basic methodology is presented) is the narrowness of the path that leads to success. (A far more detailed treatment of the crucial experiments is given in [Wos97].) 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.

Nevertheless, when the goal is proof-length reduction, certain tendencies seem to hold regarding option choices and parameter assignments. In that regard, Section 7 presents a shortcut to moving toward the objective of finding a ''short'' proof, an approach that might be termed a pseudo-algorithm, and Section 3 discusses tendencies exhibited by various parameters and strategies.

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. In particular, I address three pressing questions with regard to *proof length*, namely, which weapons played the key role, how they were used, and in what order. The discussion sets the stage for viewing (in Section 4) some of the crucial sequence of experiments that answer these questions; the experiments 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. To aid the researcher interested in using OTTER for both similar and totally dissimilar objectives, the experiments are accompanied by some commentary explaining why certain choices were made.

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; both McCune and I placed both equations in the passive list for our respective studies. (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; because of duality, a proof of one of the negated equations provides a proof of the other.) The context of the problem was the search for a Boolean algebra axiomatization consisting of an independent self-dual set of two equations. (A set is self-dual if and only if the dual of each equation is also in the set.) In fact, the ten equations (given earlier) lead directly to such an axiomatization by applying a method due to Padmanabhan and Quackenbush [Padmanabhan73]. Each equation of the pair

has length 1103, measured in symbol count. This theorem DUAL-BA-3 was used to find the first known independent self-dual 2-basis for Boolean algebra. The theorem is significant in that, from preliminary results, deriving either absorption law showed that the set of ten equations is a basis for Boolean algebra. By using Pixley reduction, the set of ten equations can be reduced to an equivalent self-dual set of two equations, which is easily shown to be independent.

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. (The proof length is increased by the use of ''flip'' if and only if the equation that is flipped is also present in the proof; both forms of an equation are retained when eq_units_both_ways is set; the use of Kunth Bendix automatically sets this option.) 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. For the actual proofs produced by the methodology, 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.) 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.

2.1. Attacking Proof Length

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 (maximum weight) of 28 to limit the complexity (measured in symbol count) of retained clauses, a limit that I maintained for the first fourteen experiments, some of which are reported here. To direct the program's reasoning, he used his ratio strategy [McCune94,Wos96a], assigning a value of 4 to the pick_given_ratio. That assignment instructs OTTER to choose as the focus of attention 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.) Each of the cited three strategies, as well as other weapons offered by OTTER, plays a vital role in the context of the first property (proof length) of elegance.

2.2. Attacking Other Properties 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. For pertinent material, see [Wos95a,Wos96a,Wos96c].

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 that match a resonator are purged; of the conclusions that are retained, the choice for the focus of attention to direct the reasoning is based on 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.

3. Making Choices and Identifying Tendencies

Consistent with my promise regarding tendencies, this section offers notions gathered from experimentation. Indeed, as one learns here, studies with OTTER suggest that some types of parameter assignment tend to be more effective than others. This observation holds whether the objective is that of finding shorter proofs (featured in this article) or the objective is that of finding any proof. The tendencies are discussed in lieu of a practical algorithm. (Of course, the word *tendency* is the correct term, for exceptions abound no matter which option, parameter, strategy, and the like is the focus.) That no such algorithm exists (which is my position) might be less surprising if one pauses to note that a mathematician who is asked for an algorithm for finding any proof would, at best, be amused. (The experiments that are cited in this section are discussed in Section 4 and, more fully, in [Wos97].)

3.1. Max_Weight

One of the more innocent-appearing parameters that governs the actions of OTTER is that which places a maximum on the weight (complexity) of retained information, namely, max_weight.

Tendency 1. Assignment of a smaller max_weight rather than a larger tends to produce shorter proofs, tends to promote finding any proof, and tends to reduce the CPU time required to

complete the designated task.

Intuitively, my guess is that long and messy formulas or equations force the program to include numerous steps to unpack the information that they contain. Perhaps the situation is like the gift that is in a box, which is in a box, which is in yet another box, and the like, with the actual object of delight buried deeply within the nested boxes.

To immediately answer a natural question, I note that I typically choose to assign max_weight (in the beginning of an attack) a value between 20 and 30.

3.2. The Hot List Strategy

Just as the values assigned to parameters can be crucial, so also can the choice of strategy or strategies.

Tendency 2. Use of the hot list strategy tends to promote the finding of shorter proofs, tends to promote the finding of any proof, and tends to reduce the CPU time that is required to complete the assigned task.

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. (Nevertheless, this tendency 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.)

Tendency 3. Use of clauses in the hot list that are representable in a few (perhaps less than 10) symbols—with the added property that the right-hand argument is a single symbol, variable or constant—tends to promote reaching the desired objective.

Tendency 4. Inclusion of a clause in the hot list that corresponds to the added hypothesis of a theorem—for example, when studying ring theory, the assumption that the cube of *x* is *x* for all *x*—tends to promote program effectiveness.

The analogue of this tendency is found in many mathematics books and papers where a proof continually leans on the added hypothesis of the theorem in focus to derive one step after another.

Tendency 5. Use of clauses in the hot list corresponding to associativity or commutativity tends to sharply decrease the effectiveness of a reasoning program.

If a clause, such as that for commutativity, easily matches terms within each newly retained conclusion, then the program can become preoccupied over and over, sharply reducing the number of clauses chosen from the set of support to initiate inference rule application.

As the cited tendencies suggest, the content of the (input) hot list can be indeed critical. For example, in Experiment 26 if one uses for the input hot list that used throughout many of the experiments reported in Section 3 and in [Wos97] (just two clauses), no proofs are returned; in contrast, with the following four clauses in the hot list, the desired 100-step proof is found. (As one clearly sees, the clauses do not adhere to Tendency 3; but the discussion focuses on tendencies, not on rigorous rules.)

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

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.

Tendency 6. Adjoining clauses frequently to the hot list during a run, in the context of the dynamic hot list strategy, tends to sharply decrease program effectiveness.

The performance tends to increase if clause adjunction during a run consists of the type of clause classed as useful in the context of the static hot list strategy.

Tendency 7. Assignment of a value of 1 to the input heat parameter, for both the static and the dynamic hot list strategy, tends to promote the finding of shorter proofs and tends to promote the completion of any proof.

Higher values tend to becalm the program, forcing it to remain in one place for a long time before choosing a new initiating clause from the set of support. Other tendencies regarding the use of McCune's dynamic hot list strategy are more difficult to identify at this time, for its use has received relatively small experimental attention so far.

The answers to other questions about the hot list strategy—the value assigned to the input heat parameter or the content of the (input) hot list, for example—rest with the properties of the correspondingly perturbed space of retained conclusions. Currently, the only means for making the appropriate decisions is through experimentation.

Indeed, as the following shows, 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 yielded no proof of either the target conclusion or of the dual. (I ignored the case for the value 2 because its use in Experiment 25 did not produce the desired 100-step proof.) 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.

3.3. The Resonance Strategy

The resonance strategy 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.

Tendency 8. Resonators that correspond to proof steps of a closely related theorem aid the program, whether the objective is proof-length reduction or simply proof finding (of any proof).

In the limiting case for the resonance strategy, steps from a proof of a theorem obtained in an earlier experiment, if used as resonators, tend to be useful for finding an even shorter proof in a later experiment. An explanation for this phenomenon asserts that such steps (of course, depending on the other option and parameter choices) can tightly direct a program's search, but not restrict the program to finding the already-completed proof. Note that, in contrast to lemmas that should be true when included in an attack, resonators have no truth value. Indeed, when a resonator is used that corresponds to a step of an earlier proof of the theorem under study or corresponds to a proof step of another theorem, its inclusion does *not* play the role of an included lemma. For a fuller discussion of the relation of resonators to lemmas, see [Wos95a].

The added needed latitude to enable a new proof to be completed is derived from such elements as a max_weight whose value is perhaps ten times the value assigned to each of the resonators, usually a small value such as 2. (However, if some of the resonators match many retained clauses, their effectiveness can be totally destroyed and, worse, no proof is found because the program becomes

mired in focusing on a large number of similar clauses.)

3.4. The Ratio Strategy

Only relatively recently did I and my colleagues at Argonne find useful a level-saturation or breadth-first search; see [Wos92] for relevant data. Formerly, our preference was for a search based on the complexity of clauses that might initiate applications of inference rules. Currently, McCune and I typically rely on the ratio strategy, which combines both direction strategies.

Tendency 9. Regarding assignments to the pick_given_ratio, progress toward finding a shorter proof (or any proof, for that matter) appears to be correlated with small values, but not the smallest. Tendency 9. Assignment of small values, but not the smallest, to the pick_given_ratio tends to promote the finding of a shorter proof and tends to promote the completion of any proof.

When the smallest possible value, 1, is assigned to the pick_given_ratio parameter, the program is instructed to place equal weight (for choosing clauses to initiate applications of the inference rules in use) on first-come first-serve information and on information of least complexity. If misfortune smiles, first-come first-serve information (that saved as the program's attack begins) will include statements whose complexity equals the maximum allowed in the experiment.

As conjectured, the use of messy formulas or equations can interfere with the seeking of shorter proofs. On the other hand, if one assigns the pick_given_ratio a large value (such as 20), then the program will likely consider few complicated conclusions to initiate applications of an inference rule. More than rarely, such a conclusion proves to be the key to unlocking the puzzle, to finding a shorter proof or to finding any proof. In fact, the cited phenomenon is precisely why McCune formulated the ratio strategy.

To immediately answer a natural question, I note that I typically favor an assignment of either the value 3 or the value 4 to the pick_given_ratio.

3.5. Ancestor Subsumption

The only means offered by OTTER for directly attacking the problem of finding shorter proofs is by using ancestor subsumption (which, for practical reasons, requires the use of back subsumption). This procedure 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.

Tendency 10. Use of ancestor subsumption tends to promote the finding of shorter proofs; however, use of ancestor subsumption tends to sharply increase the required CPU time if the goal is simply finding *a* proof.

For an example of the fact that shorter subproofs do not necessarily lead to a shorter total proof, see Chapter 3 of {Wos96a}.

4. A Tortuous Path to Success

At this point, the focus is on various experiments whose objective was to find a proof of DUAL-BA-3, a proof far shorter than McCune's 816-step proof. Only barely was I seriously seeking a proof of length 100, McCune's original request (though said in jest). I conducted more than one hundred experiments, the result of which was indeed a 100-step proof and, perhaps more important, a general methodology for seeking ''short'' proofs. Immediately preceding Section 5, a table of twenty-six of the more interesting experiments is given, the details of which are covered in [Wos97]. In this article, I focus on far fewer experiments, noting that setbacks did occur; my objective is mainly to illustrate the methodology, to show how it was developed and used, and to provide data supporting the tendencies regarding option choices and parameter assignments.

Earlier studies seeking shorter proofs [Wos95a,Wos96a,Wos96c] virtually demanded the use of the resonance strategy, if the goal of finding a 100-step proof (or, for that matter, a 400-step proof) were to be reached. 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. Note that, typically, Experiment k+1 uses as resonators proof steps of a proof obtained in Experiment k. 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 article 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 I discuss positive deduced steps of a proof to be used as resonators, implicit is the requirement that each be a unit clause.

4.1. A Fine Start: 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 influence 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. (The two levels are given to provide yet one more measure of the difference between the two proofs; level will not be cited for the remaining experiments, but it is given in Table 1.) 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 article, 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.

4.2. 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 shown in Table 1. 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 (coupled with the resonance strategy) in the context of seeking a shorter proof for the target theorem.

Of course, the addition of the hot list strategy naturally presented the question of which clauses to place in list(hot). Typically, a practice 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, for every *x*, 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).

A 607-step proof of the conclusion obtained by McCune was completed in 3702 CPU-seconds with retention of clause (23437); a 606-step proof of the dual was also completed in the experiment.

4.3. 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. Data regarding the dual will not come into play or be presented until Experiment 18, where its use became crucial; [Wos97] covers that data.

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, ignoring 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 length 344 was completed. Thus the iterative approach was working well, having cut the required number of steps to prove the primary conclusion more than in half.

4.4. Layering Resonators: Experiment 4 through 7

As one sees in the preceding section, the use of a set of resonators does not require that each member of the set be assigned the same value. Indeed, a subset of the resonators used in Experiment 3 was assigned the value 1 (with the remaining assigned the value 2) with the intention that any retained clause matching a resonator whose assigned value is 1 be added to the hot list. Experiment 3 (as noted) employed the dynamic hot list strategy. Although I dropped the use of the dynamic hot list strategy for the next four experiments (as well as for all experiments other than 3, 14, and 23), for Experiments 4 through 7 I still used a layered set of resonators, meaning that not all of the resonators were assigned the same value.

In particular, for Experiment 4, resonators represented in seven or fewer symbols were assigned the value 1, and the remaining were assigned the value 2. As expected, all of the resonators correspond to proof steps from the proof completed in Experiment 3. 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.

As one might expect because of the iterative nature of the methodology, for Experiment 5 new resonators were used, those corresponding to the proof steps of the proof obtained in Experiment 4. The same type of layering of resonators was employed. Experiment 5 produced another significant reduction in proof length, completing a 287-step proof with retention of clause (9400) in 164 CPU-seconds.

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 1 sometimes 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 and layering 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 Experiment 6 was conducted.

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), requiring 80 CPU-seconds.

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, 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.

4.5. Perturbing the Search Space: Experiment 15

Experiments 8 through 14 (as shown in Table 1) produced slow but steady progress. Experiment 15 (of the 26 detailed in [Wos97]) reflected some of the difficulty that was being encountered in attempting to find a proof of length strictly less than 207, the length of the proof completed in Experiment 14. 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, which was still in use and with an assignment of the value 9 to the pick_given_ratio, permits a program to focus on complex clauses, especially those retained early in the run. Also, consistent with the discussion of tendencies given in Section 3, the conjecture was that a smaller max_weight might enable OTTER to find a shorter proof. 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 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.

4.6. Focusing on the Dual: Experiment 18

At this point, I encountered an impasse. Various moves produced no progress. Therefore, in desperation—or from the belief that far more was possible—I made 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 still based far more on intuition than on analysis, the idea also 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 the primary conclusion in 16 CPU-seconds. Although still not the main goal, the dual was proved, with a 145-step proof, 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.

4.7. Victory: Experiments 25 and 26

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 dynamically or input, 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, in 8 CPU-seconds. For the record, 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, I assigned heat the value 3 rather than 2 (or 1, as was the case for so many experiments). For resonators, I chose the proof steps of the proof completed in Experiment 25, that of the primary target and not the dual. The return to focusing on proof steps of the primary conclusion as 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. The assignment of values to the resonators was that used in so many of the cited experiments, namely, 1, 2, and 4. I assigned max_weight the value 4 rather than 14. 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.

With Experiment 26, my 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. Because the level of a proof gives yet one more measure of its difficulty, the table also gives that information. (By definition, the level of input clauses is 0; the level of a deduced clause is one greater than the maximum of the levels of its parents.)

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

center; lcccc lrrrr. **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

5. Obstacles to Finding Short Proofs

To provide additional insight into why the path that began with a proof of length 816 and ended with a proof of length 100 was of necessity tortuous and, at the same time, to show why the various parameters and options exhibit tendencies only, the following general but brief discussion of obstacles is in order. (Chapter 3 of [Wos96a] focuses in detail on the obstacles to finding shorter proofs and the means to overcome them.)

My purpose is twofold: first, by discussing various obstacles, to give some feeling for 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.

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 (when equality is relied upon heavily), 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. For the theorem (DUAL-BA-3) featured in Sections 2 and 4, demodulation was indispensable, whether the goal was that of finding a shorter proof or, for that matter, finding any proof.

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. Regarding the tendency of smaller max_weight to promote the completion of a shorter proof, one thus sees that an obstacle does indeed exist and that the word *tendency* is appropriate. One also has a glimpse of why the path culminating in the discovery of a 100-step proof was of necessity tortuous.

For a subtler 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. Again one sees how a pick_given_ratio of 6 rather than 7 having the (possible) tendency of promoting the completion of a shorter proof might be in conflict with another obstacle. Also, the fact that the search is so touchy (as illustrated) suggests that the path from the 816-step proof to the 100-step proof was understandably tortuous.

Compared with the preceding two examples, regarding the seeking of shorter proofs, a direct attack based on using ancestor subsumption may yield far more sinister and unintuitive results. 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 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*). Nevertheless, the use of ancestor subsumption does tend to result in the finding of a shorter proof, but, as the example shows, only tends to.

Finally, although one is free to use or ignore ancestor subsumption, the situation regarding forward subsumption is radically different: The use of the latter is virtually required, if one is to avoid being buried in useless information. Nevertheless, trouble can occur with the use of forward subsumption, somewhat reminiscent of that discussed for ancestor subsumption; see [Wos97] for some detail.

6. 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 specific theorem DUAL-BA-3. In the following, I present evidence that refutes this conjecture. The emphasis is still on theorems in which equality is the dominant relation. Each of the theorems discussed in this section was studied by McCune and Padmanabhan and proved by OTTER; each answers a question that was open before OTTER's attack.

6.1. Testing in the Same Field

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.

```
24       122       51       9 25       102       44       9 26       100       44       6
     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, which explains the presence of the

last of the given clauses. In particular, the clause is obtained by negating the **and** of the three conclusions.

As in the theorem DUAL-BA-3, 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 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. 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. Then, following the iterative aspects of the approach, the next experiment mirrored the preceding, except that I used as resonators those corresponding to 87 positive deduced steps of the 91-step proof. In 192 CPU-seconds with retention of clause (22114), OTTER completed a 63-step proof, a reduction in proof length that was (for me) indeed unexpected. (As I recall, further resonator replacement, unfortunately, led nowhere.) Then, as called for if one is curious about how well the cited tendencies hold, 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.

The last two cited results together provide a nice illustration of why a practical algorithm for finding shorter proofs would be difficult to produce—perhaps even out of reach—and they also provide some insight into why the methodology presented in this article 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. Again one sees that strategies and parameters exhibit tendencies only, rather than obey hard and fast rules.

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: With shell scripts, 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) 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 cited result provides yet one more data point supporting the fact that elements such as max_weight obey tendencies only.

6.2. Further Testing

The success with DUAL-BA-3 (when studied with the methodology central to this article) 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. 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 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.

By following the type of iteration just illustrated a 77-step proof was completed, 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.

6.3. Broadening the Test

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

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.

7. 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.

For the first step of the methodological shortcut, one 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 article). If successful, the program will find a shorter proof and will find it in far less CPU time. If the first step succeeds, for the second step, one then 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; see Section 4. 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. 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; 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 of 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 most fascinated. 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 6). Therefore, as if nothing untoward had occurred, 159 resonators were adjoined, and the third experiment was conducted.

In 36 CPU-seconds, a proof of 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 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 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 (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 length 120.

Regarding further evidence, two additional theorems were studied with the two-step shortcut. Each is found in [McCune96], and by proving each, an open question was answered. 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. 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 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 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. After all, as noted in Section 3.1 with the discussion of Tendency 1, the use of a smaller max_weight tends to promote the finding of a shorter proof. In 1 CPU-second, after changing the max_weight from 19 to 15, OTTER completed a proof of 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 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 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. 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. 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, completing a proof of length 18. 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 it has a different level, 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. Such an action is in the spirit of Tendency 5 of Section 3.2. In 55 CPU-seconds, OTTER produced a 19-step proof. 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. 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.

8. Research, Review, and Remarks

The focus in this article has been on automating the search for elegant proofs. Although a precise definition of elegance is debatable, four properties are studied: proof length, term structure, variable abundance, and deduced-step complexity. The first of these four is featured.

8.1. Proof Length—Hazy and Complicated

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 normally 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, for mathematics or for OTTER. Proof length in this article 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.

Proof length—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. Still, as discussed in Section 3,

tendencies do exist regarding the values assigned to parameters and the choice and use of strategies.

One might wonder why the task of seeking shorter proofs is so complicated. At least for problems in which equality is featured, the answer rests in part with demodulators (rewrite rules). Indeed, even a small change in the order in which demodulators are found and applied can dramatically change the program's performance, the paths pursued, and the results 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, an iterative methodology (featured in this article) now exists for automating the search for shorter proofs. Although the methodology 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 (of the theorem DUAL-BA-3) 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 4, the key experiments from among the 26 fully covered in [Wos97] are detailed with commentary; they are summarized in Table 1 at the end of that section.

8.2. Research Problems

To add to the interest in the study of automating the search for elegant proofs, I offer here several challenging research problems. Solutions to these problems will increase the power of automated reasoning programs.

Research Problem 1. My main research, in the context of elegance, has focused on theorems in which equality was the only or the dominant relation and on theorems featuring the use of condensed detachment. The research problem to solve (and one that is related to the material in this article) asks for techniques for 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, this 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.

Research Problem 2. One of the most difficult problems to solve concerns correctly identifying the key ingredient for finding elegant proofs, for proving a new theorem, and for completing a difficult assignment when one is relying on the assistance of an automated reasoning program. As one expects, my answer is access to a variety of strategies, some to restrict the reasoning and some to direct it. For evidence, the following experiment suffices.

Again, the theorem in focus is DUAL-BA-5, but with three changes from the earlier discussion. The restriction on paramodulation that blocks paramodulation *into* nonunit clauses was removed, the use of what is called *basic paramodulation* [Bachmair92] was added, and the focus was changed to a proof of distributivity only. Using one of his automated reasoning programs, McCune 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. Finally, I then used super-loop with McCune's approach (of relying on hints), and OTTER found a 16-step proof of DUAL-BA-5; OTTER also found a 13-step proof of distributivity.

Research Problem 3. Regarding research of a more general nature, (in the context of automated reasoning) clearly merited is solving the problem of formulating 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 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 and 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.

8.3. Three Key Themes

Three themes are central to the research presented in this article. First, without the use of a program such as OTTER, many elegant proofs would remain undiscovered. Second, 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 study of mathematics or 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.

Acknowledgments

I thank Ross Overbeek for his invaluable suggestions regarding this article, especially those concerning the inclusion of material on *tendencies*. I also thank Robert Boyer and Robert Veroff for their excellent guidance. Finally, I thank Gail Pieper for her significant insights.

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.

[Wos92] Wos, L., and McCune, W., ''The application of automated reasoning to questions in mathematics and logic'', *Annals of Mathematics and AI*, **5,** 321-370 (1992).

[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., ''Experiments Concerning the Automated Search for Elegant Proofs'', Tech. Memo ANL/MCS-TM-226, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1997.