The Hot List Strategy[1]

by

Larry Wos with Gail W. Pieper

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

Abstract

Experimentation strongly suggests that, for attacking deep questions and hard problems with the assistance of an automated reasoning program, the more effective paradigms rely on the retention of deduced information. A significant obstacle ordinarily presented by such a paradigm is the deduction and retention of one or more needed conclusions whose complexity sharply delays their consideration. To mitigate the severity of the cited obstacle, I formulated and feature in this article the *hot list strategy.* The hot list strategy asks the researcher to choose, usually from among the input statements characterizing the problem under study, one or more statements that are conjectured to play a key role for assignment completion. The chosen statements—conjectured to merit revisiting, again and again—are placed in an input list of statements, called the *hot list.* When an automated reasoning program has decided to retain a new conclusion C—before any other statement is chosen to initiate conclusion drawing—the presence of a nonempty hot list (with an appropriate assignment of the input parameter known as *heat*) causes each inference rule in use to be applied to C together with the appropriate number of members of the hot list. Members of the hot list are used to *complete* applications of inference rules and not to *initiate* applications. The use of the hot list strategy thus enables an automated reasoning program to briefly consider a newly retained conclusion whose complexity would otherwise prevent its use for perhaps many CPU-hours. To give evidence of the value of the strategy, I focus on four contexts: (1) dramatically reducing the CPU time required to reach a desired goal, (2) finding a proof of a theorem that had previously resisted all but the more inventive automated attempts, (3) discovering a proof that is more elegant than previously known, and (4) answering a question that had steadfastly eluded researchers relying on an automated reasoning program. I also discuss a related strategy, the *dynamic hot list strategy* (formulated by my colleague W. McCune), that enables the program during a run to augment the contents of the hot list. In the Appendix, I give useful input files and interesting proofs. Because of frequent requests to do so, I include challenge problems to consider, commentary on my approach to experimentation and research, and suggestions to guide one in the use of McCune's automated reasoning program OTTER.

Keywords: automated reasoning programs, hot list strategy, OTTER

1 Paradigms and Motivation

Two distinctly different paradigms exist for automating logical reasoning. A key difference between the two paradigms concerns whether to accrue new deduced conclusions. Perhaps

[1]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.

the most effective approaches where *no new information* is retained are based on Prolog technology. Among the more effective approaches in which *new information is accrued* are the computational logic paradigm and the clause language paradigm [14, 15, 18]. The former provides the basis for the Boyer-Moore program [1, 2] (mainly used for program verification), and the latter (in the Argonne variant) provides the basis for McCune's OTTER [7, 8]. Based on my own experiments spread over more than thirty years, and supported by comments from other researchers in automated reasoning, I currently have little doubt that an automated attack on deep questions and hard problems virtually requires the accrual of information—sometimes more than 300,000 deduced facts, relations, and lemmas.

With the retention of so many deduced conclusions, however, the program needs a means for selecting where next to focus its attention. Typically, the researcher instructs the program to use either weighting (conclusion complexity) or level saturation. With level saturation, in the vast majority of cases, the size of the levels of retained conclusions grows so rapidly that the objective remains out of reach. Therefore, of the two cited direction strategies, weighting is ordinarily far more effective. However, with its use, one encounters a significant obstacle: A needed conclusion may have been retained, but, because of its complexity, the program delays focusing on it—sometimes forever.

Why does the complexity of a conclusion have this effect? The answer rests with the fact that (1) the program typically chooses as the focus of attention retained conclusions by *less complex first* and (2) the program typically lacks effective means for looking ahead and identifying good choices from among very complex conclusions.

To address just such an obstacle, I formulated the *hot list strategy.*

1.1 Types of Strategy and the Hot List

For increasing the power of automated reasoning programs, the area of research that offers the greatest promise is strategy. My preference is for strategy that restricts reasoning, but clearly strategy that directs reasoning is also vital. The hot list strategy perhaps belongs in neither category. Its use rearranges the order in which conclusions are drawn.

Specifically, the hot list strategy asks the researcher to choose, from among the statements characterizing the question under attack, those that might be especially useful and merit revisiting again and again. The chosen statements are consulted by the program repeatedly, for *completing* (rather than initiating) applications of inference rules. Indeed, with the hot list strategy, members of the hot list are automatically and immediately considered with each newly retained clause, *before* another conclusion is chosen from list(sos) to be the focus of attention to drive the program's reasoning.

1.2 A Motivating Example

Consider the following theorem from algebra. The theorem asserts that commutativity can be proved in rings in which, for every element x, the cube of x equals x. The mathematical proof begins with $xxx = x$ and substitutes the square of $v + w$ for x; in other words, instantiation is employed, an inference rule that is *not* offered by OTTER. (Instantiation should not be offered by a program, for currently no means is known for wisely applying it; in other words, one encounters an important difference between mathematics and logic on one hand and automated reasoning on the other.) The left side becomes the cube of $v + w$, and the right side is simply $v + w$; call this equation (1). After expanding and simplifying with the hypothesis $xxx = x$, the left side becomes $v+w+vvw+vwv+vww+wvv+wvw+wwv$. Call the resulting equation (2). Equation (3) is obtained from equation (2) by subtracting $v+w$ from both sides. For equation (4), set $v = w$ and simplify with the hypothesis $xxx = x$. One now has the key lemma $0 = 6x$, taken from a proof shown to me by S. Winker.

From the viewpoint of automated reasoning with paramodulation as the inference rule, the corresponding proof (in clause notation) begins by paramodulating (*into* the clause equivalent of) $x(xx) = x$, with the focus on the *into term* xx, *from* the clause equivalent of left distributivity, with the focus on the left-hand argument. After appropriate demodulation, the result is the clause equivalent of equation (2). No clause equivalent of equation (1) is produced, other than the intermediate result obtained by applying the unification to the *into clause* of the preceding paramodulation. Then, to obtain equation (4), the program can first deduce the clause equivalent of equation (3) by various means, for example, a nonstandard use of demodulation; see Section 2.2 showing how the hot list strategy can replace such an approach. At this point, one has an illustration of the obstacle under discussion: The clause equivalent of equation (3) has weight equal to 37, if measured purely in symbol count. Such a "heavy" clause will be delayed from consideration—if ever—for a substantial amount of CPU time, for many, many clauses will almost certainly be retained with weight less than 37. Therefore, the clause equivalent of equation (4), the desired lemma, will not be adjoined to the growing database of deduced conclusions for far, far too long.

This situation is not uncommon. Experiments repeatedly encounter the obstacle of the program's needing to consider some retained conclusion C whose complexity (weight) is so great that far too much CPU time is required before that conclusion is chosen as the focus of attention to initiate applications of some inference rule. When this occurs, the program is prevented from access to those conclusions that would otherwise be deduced.

With the hot list strategy, however, the obstacle is far less formidable. Indeed, in the case under discussion, the obstacle is overcome. Specifically, when I attempted to obtain a proof of the $0 = 6x$ lemma *with* the hot list strategy, only 0.3 CPU-seconds were required, compared with approximately 17 CPU-seconds without the strategy.

1.3 A Brief Glimpse into History

Motivated by my wish to give a reasoning program the power to easily prove the cited lemma, $0 = 6x$, in the context of the theorem that asserts that rings are commutative in the presence of $xxx = x$, I introduced the concept of the hot list in the mid-1980s. My notion focused on paramodulation and no other inference rule. Approximately a decade later (on November 2, 1993), McCune implemented the hot list strategy in OTTER 3.0. Significantly, he generalized my original notion by admitting the use of the hot list for all inference rules. He also generalized the notion of the hot list strategy by formulating a dynamic version to complement my static version; see Section 4.1.

For historical interest, I now correct an error found in some of my earlier writings. Specifically, OTTER was not the first program to offer the hot list strategy. Rather, the strategy was first offered in the program ITP [6] in the mid-1980s. This information escaped me because of my lack of experimentation with ITP, in turn explained (in part) by the program being menu driven rather than file driven; I sharply prefer the latter.

1.4 Areas Benefited and Challenges Presented

To enable researchers to estimate the potential of the hot list strategy, I present (in Section 5) the results of various experiments in lattice ordered groups, in Robbins algebra, and in logic calculi. The evidence (given in Section 5) supporting the value of the hot list strategy focuses on four areas: dramatically reducing the CPU time required to produce proofs, finding proofs that had previously resisted all but the more inventive attacks, discovering proofs more elegant than had been known, and answering a question previously considered intractable for an automated reasoning program.

To encourage researchers—especially mathematicians and logicians—to use the hot list strategy (as well as other strategies), I include (in the Appendix) various input files that are acceptable to the powerful automated reasoning program OTTER, and I include interesting proofs as well. (The input clauses and proofs are also available on the Web; see the URL htpp://www.mcs.anl.gov/home/wos/
hotlist-input.html.) Also, to increase the likelihood of success when using the hot list strategy, I include (in Section 6) diverse hints. Finally, I offer a challenge problem for researchers (see Section 7).

2 Relation to Other Strategies, Procedures, and Inference Rules

The hot list strategy shares several features with other strategies and inference rules implemented in the automated reasoning program OTTER. In this section, I focus first on the possible relevance of the set of support strategy, then (in order) on demodulation, AC-unification, and linked inference rules.

2.1 Set of Support Strategy

Like the hot list, the set of support strategy was motivated by the study of a single (and very simple to prove) theorem: Commutativity can be proved for groups of exponent 2, those in which the square of x (for every element x) is the identity e. My notion (regarding the set of support strategy) was to restrict the applications of the inference rules in use and to force the search to key on information chosen by the researcher. To implement the strategy, the researcher places the *key information* in an input list, the initial set of support. Such is also the case for implementing the hot list strategy: The researcher places what is conjectured to be key information in an input list, the hot list. New conclusions for which the set of support strategy plays a role are recursively traceable to the initial set of support; new conclusions for which the hot list strategy plays a role are recursively traceable to the initial hot list.

For a second way in which the two strategies are related, I have always recommended that the *special hypothesis* (clauses) be included in the initial set of support, and I typically recommend that such information also be placed in the initial hot list. For a third similarity, just as the newly retained conclusions in which the set of support strategy plays a role are added to the set of support list, so also can clauses be added to the hot list if the dynamic version of the hot list strategy, due to W. McCune, is in use (see Section 4.1 for details).

Finally, the conclusions that are retained and that are traceable to the initial set of support can be used, without violating the set of support strategy, to deduce additional conclusions; such is also the case for clauses deduced with the hot list strategy, depending on the value assigned to the *heat* parameter.

Of the cited similarities between the two strategies, perhaps the most important concerns keying the program's search on information selected by the researcher.

As for a key difference, a member from the set of support is chosen to *initiate* an application of an inference rule (when the set of support strategy is in use); in contrast, the members of the hot list come into play only *after* a clause has been chosen as the focus of attention to drive the program's reasoning. Indeed, the members of the hot list are used only to *complete* an application of an inference rule.

For a second important difference, the main object of the set of support strategy is to *restrict* the program's reasoning; on the other hand, the object of the hot list strategy is to *rearrange* the order in which conclusions are drawn.

2.2 Demodulation

Demodulation is used by many programs for simplification and canonicalization. A dramatic improvement in efficiency is often due directly to automatically applying various equalities (demodulators) to each deduced conclusion.

In the mid-1980s, no doubt influenced by the effectiveness of using demodulation, I conjectured that the automatic consideration by paramodulation of each newly retained clause with each member of a chosen set of equalities might also prove to be a powerful move. The chosen set of equalities would be placed on a list to be called the hot list.

Of course, for the hot list strategy, two constraints on the actions of the program must be relaxed: (1) In contrast to demodulation, rather than requiring that no instantiation of variables be permitted in the *into clause*, full two-way unification must be permitted; and (2) in contrast to the usual use of inference rules, rather than having the program wait until the newly retained clause is chosen as the focus of attention, the program must be allowed to immediately use it as one of the parents in the attempt to draw additional conclusions. Indeed, with the hot list strategy, members of the hot list are automatically and immediately considered by paramodulation, if in use, and by any other inference rule in use (as McCune suggested by way of a generalization) with each newly retained clause, *before* another conclusion is chosen from list(sos) to be the focus of attention to drive the program's reasoning.

As noted in Section 1.2, one can also use the hot list strategy to replace certain non-standard uses of demodulation. In particular, the use of demodulation at the literal level can enable a program to apply extended cancellation. Instead, clauses that function as nuclei and that capture various types of cancellation can be placed in the (input) hot list, and hyperresolution can be used as one of the inference rules. Then, when the program decides to retain a new clause, before another clause is chosen as the focus of attention to initiate applications of inference rules, the clause will be processed with cancellation of the type present in the hot list. One might find this alternative more attractive than either (1) waiting for the clause to be chosen as the focus of attention to then be considered with included clauses for cancellation or (2) using demodulation in some usual form or some nonstandard form.

2.3 AC-Unification

In a limited way, the hot list strategy resembles, and hence can be used in place of, associative-commutative unification. For this objective, one begins by including in the hot list a clause for associativity and a clause for commutativity, assigning the heat parameter the value 2, and invokes the use of paramodulation. Then, for each newly retained clause *before* another clause is chosen from list(sos) to be the focus of attention, the program will automatically apply paramodulation to the new clause together with that for associativity and also apply the inference rule to the new clause together with that for commutativity. For each newly retained clause, the two clauses that are so deduced will have heat level 1, and they will be retained depending on the other input parameters and subsumption and such. Then, because of the assignment of the value 2 to the heat parameter, the heat-level-1 clauses (under discussion) that are retained will each immediately be considered by paramodulation with associativity and also with commutativity. In the case under discussion, a limited form of associative-commutative unification is used. Of course, also in use in a limited way in this case is associative unification and commutative unification, at heat level 1 and at heat level 2. Also deduced at heat level 2 are clauses to which associativity has been applied twice and to which commutativity has been applied twice.

Use of the hot list strategy in the described manner can produce clauses early in a run that, because of the reassociation and commuting of terms, admit further canonicalization. By choosing the appropriate assignment of the heat parameter, one has control over the amount of AC-unification that is used. I find this alternative to AC-unification appealing, for I have always been wary of a general and full use of that form of unification. Indeed, a full use of AC-unification can drown a program in unwanted conclusions; in general, for practical considerations, restrictions must be imposed.

If associative unification without commutativity is desired, a clause for associativity is included in the hot list, and no clause for commutativity is included.

2.4 Linked Inference Rules

Use of the hot list strategy can also (in effect) partially substitute for access to linked inference rules [11, 26]. Consider the case in which the clause C is deduced, the decision is to retain C, C has high weight, and, were it not for the fact that a particular term t in C was left associated, paramodulation would apply to C and a clause corresponding to the special hypothesis with the *into term* being the right association of t. With linked paramodulation, one could use associativity as a link to right associate t in C to then permit the result to unify appropriately with the special hypothesis clause. If one was using the hot list strategy with associativity and the special hypothesis clause in the hot list, and if the heat parameter was assigned the value 2, then the decision to retain C would immediately trigger the application of paramodulation to C and the clause for associativity. Then, if the decision was to retain the result, immediately paramodulation would consider the reassociated version of C with the special hypothesis clause.

Linked paramodulation does not work precisely as does the hot list strategy. Among the differences is that concerning so-called intermediate clauses. Specifically, once the program begins an application of a linked inference rule, the weight of a clause that is *temporarily* deduced on the way to the linked conclusion is ignored. Linked paramodulation cannot produce an intermediate clause whose weight prevents the continuation of the application of linked paramodulation. In particular, in the example just discussed, the reassociation of the clause C because of using associativity as a link cannot prevent completion of the application of the linked inference rule because of the weight of the reassociated clause. In contrast, with the hot list strategy, the reassociated C must be retained in order to permit its consideration with the special hypothesis clause. (I am curious about the possible usefulness of considering each clause with members of the hot list *before* the decision concerning retention is made.) A second difference concerns the possible use of demodulation. The intermediate clauses resulting from a partial application of a linked inference rule are *not* subject to demodulation, but their correspondents *are* with the hot list strategy.

3 Intuitive View of the Hot List Strategy

In the following sense, one sees that the use of the hot list strategy enables an automated reasoning program to "look ahead". Assume that paramodulation is the only inference rule in use and that a clause A has just been selected to be considered (by paramodulation) with each of the various clauses that have already been chosen as the focus of attention. Let H be a member of the hot list, and assume that the assignment to the appropriate input parameter (called *heat*) permits consideration of H with each new clause that the program decides to retain. Also assume that the program is choosing where next to focus its attention based purely on symbol count and that the weight (number of symbols) of A is 12. Let C be a clause with weight 25 that is deduced from A and some earlier-considered clause such that the program decides to retain C. Finally, assume that, prior to the decision to retain C, the consideration of A has resulted in the retention of ten new clauses each with weight 15.

Ordinarily, without the intervention of the hot list strategy, the program would not consider applying paramodulation to C and another clause until after focusing first on the ten newly retained clauses each with weight 15. Quite likely the consideration of C would be delayed further, for the focus on the weight 15 clauses would probably result in the retention of additional clauses of weight less than 25.

However—and here is how the use of the hot list strategy enables the program to look ahead—*before* another clause is chosen as the focus of attention, paramodulation is applied to C and H. If the application yields a clause D with weight 10 that is retained, then the program will have almost immediate access to the use of D for initiating applications of inference rules. Otherwise, without the hot list strategy and assuming that H was available to be used, the program would be forced to wait for D to be deduced when C is chosen as the focus of attention—if C is ever chosen. Indeed, as soon as A has completed its role as the focus of attention, D will be chosen to initiate applications of inference rules if all eligible clauses have weight greater than 10. In addition to the deduction of D, other low-weight clauses might be retained whose parents are C and H, and still others from C and some other member of the hot list.

If the size of the hot list is small, the researcher need not in general worry about the program being forced to cope with an avalanche of clauses of the type under discussion. In effect, the program looks ahead to deduce just those immediate descendants of C whose other parent is a member of the hot list, assuming that the heat parameter is assigned the value 1.

Such a deduced clause D has *heat level* equal to 1 (defined formally in Section 4). If the value the researcher assigns to the (input) heat parameter is 2, then *after* the program decides to retain a clause D with heat level 1 but *before* another clause is chosen as the focus of attention (in this case) paramodulation is applied to D and each member of the hot list, *before* that newly retained clause is used in its fullest to initiate applications of inference rules. Any clause that is so deduced has heat level equal to 2. Clauses of heat level 2 are immediate descendants of immediate descendants of a newly retained clause. If the program does retain clauses of heat level 2, then the program is looking even further ahead.

4 Formalism, a Powerul Option, and an Illustration

In this section, I give needed definitions, discuss parameters and options, and employ one or more of the notations acceptable to McCune's program OTTER. I then briefly turn to a discussion of the dynamic hot list strategy, and I close this section with an illustration of the use of the hot list strategy. When I use the phrase "clause or its equivalent", I am *not* restricting the definitions and terminology to OTTER-like programs. Rather, I intend that the appropriate adjustment(s) be made when the program in use requires input in some other form. Often, I use the term "clause" to mean "clause or its equivalent".

Definition, initial and extended hot list. The initial hot list is a (possibly empty) set of clauses (or their equivalent) selected by the researcher and included in the input. Depending on the exercising of appropriate options, the initial hot list can be extended by adjoining new members to it during a run. Each member of the hot list (initial and extended) is eligible for automatic and immediate consideration to *complete* applications of the inference rules in use, with the requirement that the application of one of those inference rules be *initiated* by focusing on a clause (or its equivalent) that the program has decided to retain. In particular, no inference rule is permitted to apply to a set of clauses all of which are members of the hot list.

CHECK THIS **Definition, heat level.** The heat level of a clause is 0 if and only if no clauses of the hot list participate in the application of an inference rule; the heat level of a clause is 1 if and only if (1) clauses from the hot list participate and (2) the heat level of the clause initiating the application of the inference rule is 0; for $n \geq 2$, the heat level of a clause is n if and only if the heat level of the clause that initiates the application of the inference rule is $n-1$.

Regarding the eligibility of the members of the hot list, the (input) parameter known as *heat* must be assigned a value greater than or equal to 1 for members to be eligible for use. In other words, permission must be given to deduce clauses with heat level greater than or equal to 1. To instruct OTTER to attempt to deduce clauses with heat level 1, one adds a single command to the input file. If the value in the command of the following type is equal to 1, *after* OTTER decides to retain a newly generated clause A but *before* another clause is chosen as the focus of attention (to drive the program's reasoning), each inference rule in use is applied to A (as if A were the focus of attention) and the appropriate number of clauses H in the hot list, where that number is determined by the inference rule being applied.

```
assign(heat,1).
```

The default of the parameter heat is 1.

For example, if paramodulation is being applied, then A is considered with each H in the hot list; if hyperresolution is being applied, then, consistent with the requirements of nucleus and satellites, all subsets of clauses from the hot list are considered with A. Any clause B deduced from A and one or more clauses H is treated as all deduced clauses are treated (with regard to subsumption, weighting, demodulation, and the like). If such a clause B is used in a proof, the proof will show for that clause (heat=1), meaning that its heat level is 1.

To enable OTTER to deduce and possibly use clauses whose heat level equals 2, one modifies the preceding command to be the following.

```
assign(heat,2).
```

With this modified command, *after* OTTER decides to retain a newly generated clause B whose heat level equals 1 but *before* another clause is chosen as the focus of attention, each inference rule in use is applied to B (as if B were the focus of attention) and the appropriate number of clauses H in the hot list, where that number is determined by the inference rule being applied. As expected, any clauses that are deduced whose heat level equals 2 are treated as all deduced clauses are treated.

Of course, one can assign to the heat parameter values greater than 2. An assignment of the value 0 instructs the program *not* to consult the hot list.

4.1 Dynamic Hot List Strategy

As a powerful option, OTTER can also be instructed to dynamically adjoin new clauses to the hot list *during the run.* This extension of the hot list, developed by McCune, relies on a command of the following type.

```
assign(dynamic_heat_weight, 20).
```

With this command as part of the input, OTTER will—during a run—adjoin to list(hot) any clause that (1) the program has decided to retain and (2) the program has assigned a weight less than or equal to 20. (Clauses adjoined to list(hot) during a run must have an assigned weight less than or equal to the max_weight currently in use.)

The dynamic hot list strategy shares some similarity with the set of support strategy. Specifically, one expects or intends that clauses dynamically adjoined to list(hot) play a key role in a program's attack on the question or problem under study, just as one wishes, ideally, that clauses dynamically adjoined to list(sos) play a key role. Of course, as experimentation repeatedly shows, the ideal case (for the set of support strategy) is not even approximated: Typically, a few CPU-minutes suffices to produce a large and growing list(sos), many of whose members will *never* be chosen as the focus of attention to drive the program's reasoning. The most common cause for a clause not being chosen as the focus of attention is its high weight or complexity. Perhaps in the future, this deficiency will be sharply reduced by having the program automatically (and possibly self-analytically) move certain clauses from list(sos) to list(usable) *before* they are chosen as the focus of attention; see Section 13.4 of [18]. List(sos) is the name of the list of clauses that have *not* yet been chosen as the focus of attention but are recursively traceable to the initial set of support or were in the initial set of support. List(usable) consists of the input clauses that were part of the problem description but not placed in the initial set of support and also the clauses that were selected from list(sos) to be the focus of attention to drive the program's reasoning. However, an immediate move of a clause to list(usable) will permit its use only for inference rule completion, not for inference rule initiation.

4.2 An Illustration

For an example of a somewhat elaborate use of the hot list strategy, consider the following excerpt from an input file, where "|" denotes logical **or**, "-" denotes logical **not,** the predicate P can be interpreted as "provable", the function i can be interpreted as "implication", and the function n as "negation". (When a line contains a "%", the characters from the first "%" to the end of the line are treated by the program as a comment.)

```
assign(heat,3).
list(hot).
% Following is for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).
% Following is Meredith's single axiom.
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).  % CN-CAM
% Following were proved in temp.otter3.meredith.hot.out
P(i(x,i(y,x))).
P(i(i(i(x,y),z),i(y,z))).
P(i(x,i(n(x),y))).
P(i(n(n(x)),x)).
P(i(n(n(x)),x)).
P(i(i(i(x,y),z),i(n(x),z))).
P(i(x,n(n(x)))).
P(i(x,n(n(x)))).
P(i(i(n(x),x),x)).
P(i(i(x,i(x,y)),i(x,y))).
end_of_list.
```

The theorem under study asserts that Meredith's axiom is a single axiom for two-valued sentential (or propositional) calculus; see [10]. I give Meredith's proof in the Appendix.

As one sees, the inference rule used in the study is condensed detachment, used by Kalman in his landmark study of equivalential calculus [4, 5]; hyperresolution is used. Typically, in such investigations, the only clauses used to initiate applications of an inference rule are unit clauses. Therefore, for the hot list strategy to be usable, one must include at least one nucleus in the hot list; the cited nucleus is the only such clause in the study. After all, as commented earlier, with the hot list strategy, all but the initiating clause for the application of each inference rule must be members of the hot list. Similarly, because the nucleus contains two negative literals, the hot list must also contain at least one other positive clause to permit the use of the hot list strategy to complete applications of condensed detachment through the use of hyperresolution.

In the given example, I included in the hot list Meredith's axiom and various members of known axiom systems, each of which had been proved in a prior experiment. The assignment of the value 3 to the heat parameter instructs OTTER to attempt to deduce clauses with heat level less than or equal to 3. Whether any clauses that are thus deduced are retained depends, of course, on other parameters, such as the max_weight parameter (which places an upper bound on the weight of a retained clause).

5 Experiments

At this point, I present evidence of the power of the hot list strategy by discussing various experiments, comparing results with and without the use of this strategy. The experiments focus on four separate contexts: (1) reducing the amount of CPU time to complete a proof, (2) producing a proof of a theorem previously out of reach without much intervention by the researcher, (3) finding elegant proofs (in terms of length), and (4) answering challenging questions.

5.1 Reducing the CPU Time

In this subsection, I demonstrate that using the hot list strategy can reduce CPU time. However, since no panacea exists, in Section 5.3 one sees that using the hot list strategy can increase CPU time.

I begin with a theorem concerned with lattice ordered groups. The theorem was brought to my attention by I. Dahn at the 1994 QED workshop. The theorem (which I call $LOGT1$ for Lattice Ordered Groups Theorem 1) asks one to prove that, for each element x in a lattice ordered group, x is equal to the product of its *positive part* pp(x) and its *negative part* np(x), where 1 is the identity of the group, pp(x) is the union of x and 1, and np(x) is the intersection of x and 1. The following (in another notation acceptable to OTTER) gives the axioms, needed definitions, and various members of a complete set of reductions for groups. Regarding the notation in the following, i denotes inverse, 1 the group identity, u union, n intersection, pp positive part, and np negative part; `!=` denotes "not equal". (The anomaly of having associativity of both union and intersection expressed as they are results from the manner in which I was given the problem; OTTER, when processing the input, interchanges their respective arguments so that the left-associated argument is on the left.) The significance of the line of dashes in the following will become clear when I focus on the set of support strategy.

```
x = x.
(x*y)*z = x* (y*z).
1*x = x.
x*1 = x.
i(x)*x = 1.
x*i(x) = 1.
i(1) = 1.
```

```
i(i(x)) = x.
i(x*y) = i(y)*i(x).
n(x,x) = x.
u(x,x) = x.
n(x,y) = n(y,x).
u(x,y) = u(y,x).
n(x,n(y,z)) = n(n(x,y),z).
u(x,u(y,z)) = u(u(x,y),z).
-----------------------------------
u(n(x,y),y) = y.
n(u(x,y),y) = y.
x*u(y,z) = u(x*y,x*z).
x*n(y,z) = n(x*y,x*z).
u(y,z)*x = u(y*x,z*x).
n(y,z)*x = n(y*x,z*x).
pp(x) = u(x,1).
np(x) = n(x,1).
pp(a)*np(a) !=  a.
```

Dahn noted that the theorem had been proved in 30 CPU-seconds on a SPARCstation-10 by a program called Discount. (Actually, in addition to the SPARCstation-10, simultaneously two other computers were used, each a SPARCstation-2; after 30 CPU-seconds, Discount announced the theorem provable, but an additional 90 CPU-seconds was required to return the proof.)

For my study of $LOGT1$ with OTTER, my colleague McCune in part paved the way; he chose a Knuth-Bendix approach with the following symbol ordering.

```
lex([1,a,u(_,_),n(_,_),*(_,_),i(_),pp(_),np(_)]).
```

McCune assigned max_weight the value 15 (for a bound on the complexity of retained conclusions, measured in symbol count) and assigned the value 4 to the *pick_given_ratio*. (This latter assignment instructs OTTER to focus on four conclusions based on their complexity, one by first come first serve, then four, then one, and the like.) Finally, McCune placed all clauses in list(sos), in effect instructing OTTER *not* to use the set of support strategy. On a SPARCstation-2, OTTER found a proof (given in the Appendix), of length 33, in approximately 19,280 CPU-seconds.

Following McCune's lead, I then took up the attack, using a SPARCstation-10 (approximately two times faster than the SPARCstation-2). I report here (not in chronological order) the results of two experiments.

In the first, I chose for the hot list precisely the positive clauses in the initial (input) set of support—the clauses given earlier that follow the line of dashes—assigned the heat parameter the value 1. The experiment produced a proof (given in the Appendix), of length 32, in approximately 3148 CPU-seconds. When the run was terminated, after choosing as the focus of attention 3122 clauses, 12,187 (of which 1714 were hot) were retained and 4,670,281 (of which 61,980 were hot) were generated. This experiment provides the inexperienced researcher with a simple rule to follow for choosing clauses for the hot list and for assigning the heat parameter.

The second experiment provided more dramatic evidence of the power of the hot list in reducing CPU time. The hot list consisted of the members of the set of support used in the just-cited experiment (the positive clauses found after the line of dashes given earlier) augmented by the clauses for commutativity of union and of intersection and the clauses for left and right inverse. I assigned a value of 2 to the heat parameter. The experiment produced a proof, of length 42, in approximately 347 CPU-seconds.

5.2 Bringing a Theorem within Range

In the preceding subsection, I provided evidence of the value of using the hot list strategy to sharply reduce the CPU time required to complete an assignment. Here, my focus is on the use of this strategy to bring within range theorems whose proof resisted various automated attempts. The area is Robbins algebra, an area that I find fascinating mainly for three factors. First, just three axioms suffice to study this algebra, the following expressed in yet one more notation acceptable to OTTER, where one can interpret the function n as complement and the function $+$ as union.

```
EQ(+(x,y),+(y,x)).    % commutativity
EQ(+(+(x,y),z),+(x,+(y,z))).    % associativity
EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).    % Robbins axiom
```

Second, at least on the surface, Robbins algebra is a natural target for automated reasoning; indeed, one can easily study this field by using paramodulation and choosing various options to control this inference rule or by using paramodulation within a Knuth-Bendix approach. Third, and so intriguing, the question of whether every Robbins algebra is a Boolean algebra was open until McCune with his program EQP answered it in the affirmative [9]; in fact, Tarski and his students failed to answer the question. The question is posed in [3].

My focus here is on $RAT5$, a theorem that provides a splendid challenge for automated reasoning programs, especially those that do *not* offer AC-unification or induction. In terms of the "ordering relation" on the elements of Robbins algebra, the theorem says that the existence of two elements c and d with d less than or equal to c together with the Robbins axioms is all that is needed to imply Boolean. The theorem was first proved by Winker using induction [12, 13]; McCune later obtained a proof with AC-unification. My goal, for years, has been to prove the theorem without induction and without AC-unification. In 1996, I made yet another attempt.

I assigned heat the value 1 and placed in the (input) hot list only the clause corresponding to the special hypothesis, $c+d=c$. I assigned max_weight the value 30, the pick_given_ratio the value 3, and max_distinct_vars the value 3 and included clear(eq_units_both_ways). Regarding weight templates, in weight_list
(pick_given), I included two to respectively purge associative variants in four and five variables, one to purge expressions in which $n(n(n(t)))$ terms occur, and the template for the tail strategy. The inclusion of the template for the tail strategy causes OTTER to prefer clauses in the equality predicate whose right-hand argument is short.

Success: In approximately 44,926 CPU-seconds, OTTER produced a proof of length 80 and level 18, with retention of clause (66147).

In fairness, I must admit that I also succeeded *without* the hot list strategy. Indeed, in approximately 9770 CPU-seconds, OTTER produced a proof (given in the Appendix) of length 78 and level 16, with retention of clause (48308). Nevertheless, my delight at finding such a proof—for the first time, with and without the hot list—remains unbounded.

5.3 Finding Elegant Proofs

One measure of the elegance of a proof is its brevity. In this subsection, I show how the hot list strategy has proved useful in the search for elegant proofs, in the context of proof length. Note that no practical algorithm appears to exist for searching for short proofs, and note that numerous obstacles, some of which are indeed subtle, are encountered in such a search; see [18], which takes the form of an experimental notebook. I focus on the formulas known as *XHK* and *XHN*, each of which alone is strong enough to provide a complete axiomatization for equivalential calculus.

```
P(e(x,e(e(y,z),e(e(x,z),y)))).  % XHK
P(e(x,e(e(y,z),e(e(z,x),y)))).  % XHN
```

To prove either of the corresponding theorems (by deducing one of the other known single axioms) provides an excellent test for ideas and for programs. Indeed, whether either is a single axiom was an open question until Winker obtained proofs with excellent insight, many computer runs, much time, and considerable assistance from one of Argonne's automated reasoning programs [24, 25]. As an indication of the difficulty offered by the two benchmark theorems, (not counting the predicate P) Winker's 84-step proof for *XHK* relies on the use of a formula of length 71, and his 159-step proof for *XHN* relies on the use of a formula of length 103.

To enable researchers to conduct similar experiments, here is a complete list of the shortest single axioms for equivalential calculus, each expressed in clause notation.

```
% Following are all of the shortest single axioms
% for equivalential calculus.
P(e(e(x,y),e(e(z,y),e(x,z)))).  % P1_YQL
P(e(e(x,y),e(e(x,z),e(z,y)))).  % P2_YQF
P(e(e(x,y),e(e(z,x),e(y,z)))).  % P3_YQJ
P(e(e(e(x,y),z),e(y,e(z,x)))).  % P4_UM
P(e(x,e(e(y,e(x,z)),e(z,y)))).  % P5_XGF
P(e(e(x,e(y,z)),e(z,e(x,y)))).  % P7_WN
P(e(e(x,y),e(z,e(e(y,z),x)))).  % P8_YRM
P(e(e(x,y),e(z,e(e(z,y),x)))).  % P9_YRO
P(e(e(e(x,e(y,z)),z),e(y,x))).  % PYO
P(e(e(e(x,e(y,z)),y),e(z,x))).  % PYM
P(e(x,e(e(y,e(z,x)),e(z,y)))).  % XGK
P(e(x,e(e(y,z),e(e(x,z),y)))).  % XHK
P(e(x,e(e(y,z),e(e(z,x),y)))).  % XHN
```

From various experiments, I had found a 27-step proof showing that *XHK* is a single axiom and a 24-step proof showing that *XHN* is also a single axiom. I decided next to use the hot list strategy to attempt to find even shorter (more elegant) proofs.

I began with *XHN*, using a level-saturation approach, assigning the value of 36 to max_weight and the value of 2 to each of 24 resonators corresponding to the steps of the 24-step proof I had obtained. I included in list(passive) the negations of each of the other twelve shortest single axioms, expecting a deduction of *UM* only. I assigned the value 1 to the heat parameter and placed in the hot list the clauses corresponding to *XHN* and the condensed detachment nucleus. In approximately 38 CPU-seconds (on the equivalent of a SPARCstation-2), OTTER deduced *UM* with a proof of length 22 and level 11, with retention of clause (864). When I then deleted the use of the resonance strategy [16, 20], in approximately 770 CPU-seconds OTTER deduced *UM* with a proof of length 20 and level 14, with retention of clause (9777).

Why did the hot list strategy succeed? A key rests with the effect the strategy has when level saturation is being used. For an illustration of how this combination causes the program to look ahead, assume the heat parameter is assigned the value 1, that condensed detachment is in use, and that the hot list contains the needed clauses (such as that for condensed detachment and, say, the shortest single axiom candidate under study). When a level-1 clause A is deduced and retained and the hot list strategy is in use, A will immediately be used to initiate applications of condensed detachment with the clauses needed to complete the applications chosen from the hot list. If condensed detachment succeeds, yielding a clause B, B will have level 2 (and heat level 1). If B is retained, it will be placed among the level-1 clauses, even though it has level 2. Keep in mind that, very likely, the program is still generating level-1 clauses and simply paused, because of the use of the hot list strategy. Then, when the program is using the level-1 clauses to deduce those of level 2, B (of level 2) will be used; but its use will generate level-3 clauses, which, if retained will be placed among the level-2 clauses. In addition, because of the use of the hot list strategy, such a level-3 clause C will be used immediately. If clauses are deduced and retained, they will be of level 4, but be placed among those of level 2, just as B was deduced and placed among the level-1 clauses.

Regarding the successful completion of the cited 20-step proof, a glance at the output file shows that the program was deducing and retaining clauses of level 11 when it found and used a level-14 clause to complete the proof. In other words, the program, because of using the hot list strategy in conjunction with level saturation, was able to look ahead into higher levels. One thus sees how the program found a different proof, one of length 20, by traversing a sharply different search path.

To determine the effect on CPU time, on a computer that is perhaps 1.3 times as fast as a SPARCstation-10, I conducted two experiments. Approximately 38 CPU-seconds suffices with level saturation and without the hot list strategy, in contrast to approximately 306 CPU-seconds with the combination of level saturation and the hot list strategy. Again, for part of the explanation, one need only glance at the corresponding output files, finding that level 10 completes with clause (87) when the hot list strategy is not in use, and level 10 completes with clause (1488) when it is in use. These two figures further illustrate how the hot list strategy, when level saturation is in use, causes the program to look ahead into higher levels. The figures also illustrate a disadvantage of using this combination of strategies, for the size of the levels can grow far more rapidly.

One final experiment merits discussion. In the spirit of cursory proof checking (as opposed to rigorous proof checking), both covered in [18], I used as resonators the 20 steps of the just-cited proof, assigned a value of 2 to each, and assigned to the max_weight the value 2. Again, I used the hot list strategy, motivated by a distantly related experiment in another logic calculus, an experiment that yielded under similar conditions an even shorter proof; see [21] and Section 3.4 of the technical report [23] that is a far longer version of this article. I was not rewarded: OTTER merely returned the 20-step proof already discussed. On a whim, I repeated the experiment with one change, that of omitting the use of the hot list strategy. I was more than startled, for OTTER completed a 19-step proof of level 14 (given in the Appendix), showing that a shorter proof can be found with cursory proof checking either by adding the use of the hot list strategy or (in this case) by removing its use. I know of no shorter proof establishing *XHN* to be a single axiom for equivalential calculus, a fact that implicitly poses a possible research question.

Next I turned to a study of *XHK*, applying a similar approach to that which yielded the 20-step proof that *XHN* is indeed a shortest single axiom for equivalential calculus. In one of several experiments, I assigned max_weight the value 48, used ancestor subsumption (and, therefore, used back subsumption), assigned the pick_given_ratio the value 3, reassigned the max_weight to the value 20 after 30 clauses were chosen as the focus of attention, and used the hot list strategy with the heat parameter assigned the value 1. I placed in the hot list the clauses corresponding to *XHK* and the condensed detachment nucleus. I used the pick_and_purge weight_list and included, for the resonance strategy, weight templates corresponding to the steps of the earlier-mentioned 27-step proof, which completed with a deduction of *YRO*. OTTER succeeded in finding a 26-step proof. Four of the steps reflect the use of the hot list strategy, each showing (heat=1).

As with *XHN*, one additional experiment merits citing because of the progress that resulted. However, rather than cursory proof checking providing the key, parameter changes proved to be crucial. Regarding the changes, I assigned the pick_given_ratio the value 2 rather than 3, instructed the program to reduce the max_weight from 48 to 24 after 50 clauses were chosen as the focus of attention, and used for resonators weight templates that correspond to the 26 steps of the just-cited proof. OTTER succeeded in finding a 23-step proof of level 19 (given in the Appendix), but, rather than deducing *YRO*, the proof completed with the deduction of *YQL*. Again, four of the steps reflect the use of the hot list strategy, each showing (heat=1).

5.4 Answering Challenging Questions

In Section 5.1, I focused on the theorem $LOGT1$ to show how the hot list strategy can be used to sharply reduce the time required to find a proof. In this subsection, I focus on another problem in lattice ordered groups, $LOGT2$, which provides evidence of how the hot list strategy can be used to obtain a solution to an interesting question whose answer had eluded researchers. Again, Dahn brought the original theorem to my attention. In this case, however, his program had not been able to obtain a proof.

One is asked in $LOGT2$ to prove a relation among inverse, intersection, and union, a relation whose negation is the following, where i denotes inverse, n denotes intersection, and u denotes union.

```
i(n(a,b)) !=  u(i(a),i(b)).
```

As the problem was proposed to me, one is permitted to use essentially the entire underlying theory. I began by discarding all nonunit clauses and all new (not in the input for $LOGT1$) equalities but two, the following.

```
u(x,n(y,z)) = n(u(x,y),u(x,z)).
n(x,u(y,z)) = u(n(x,y),n(x,z)).
```

I added in list(sos) the two positive equalities to the input for $LOGT1$ (of course, omitting the denial of its conclusion) and added the negative equality to list(passive).

I chose a level saturation approach, using the following command.

```
set(sos_queue).
```

I assigned the value 2 to the heat parameter and used the following hot list.

```
list(hot).
n(x,y) = n(y,x).
u(x,y) = u(y,x).
i(x)*x = 1.
x*i(x) = 1.
u(n(x,y),y) = y.
n(u(x,y),y) = y.
x*u(y,z) = u(x*y,x*z).
x*n(y,z) = n(x*y,x*z).
u(y,z)*x = u(y*x,z*x).
n(y,z)*x = n(y*x,z*x).
u(x,n(y,z)) = n(u(x,y),u(x,z)).
n(x,u(y,z)) = u(n(x,y),n(x,z)).
end_of_list.
```

I also used resonators from an earlier success, each assigned the value 2. With the hot list strategy, OTTER produced a proof in approximately 1826 CPU-seconds with length 37 and level 15, with retention of clause (6698). By comparison, without the hot list strategy, OTTER produced no proof.

The explanation for the success rests with the following. With level saturation and the heat parameter assigned the value 2, when a new clause is retained at, say, level 4, the hot list strategy will first immediately generate clauses of level 5 (and heat level 1) and then, if any of them are retained, use them to immediately generate clauses of level 6 (and heat level 2). So, in one sense, the use of the hot list strategy with a breadth-first search enables a program to look ahead; see Section 6.2 for more discussion.

Rather than simply turning to another topic, I mention here one additional set of experiments concerning $LOGT2$. The results of the experiments nicely illustrate how narrow can be the window of opportunity to answer a difficult question and how intertwined various procedures often are. Whereas one of the experiments yielded the shortest proof (given in the Appendix) of $LOGT2$ of which I know—a proof of length 22—the other experiments yielded *no* proof of any type. The 22-step proof was found by dropping the use of level saturation, assigning the value 10 rather than 6 to the pick_given_ratio, assigning the value

3 rather than 2 to the heat parameter, and using a hot list consistent with the recommendation (given in Section 6.1) concerning "short and simple" clauses that occur in the input set of clauses. The hot list consisted of the following ten clauses.

```
1*x = x.
x*1 = x.
i(x)*x = 1.
x*i(x) = 1.
i(1) = 1.
i(i(x)) = x.
n(x,x) = x.
u(x,x) = x.
u(n(x,y),y) = y.
n(u(x,y),y) = y.
```

The other options and assignments were the same as were used in the cited successful level-saturation run for $LOGT2$. Although some of the other experiments from the set failed to yield a proof, they were each most valuable, for their respective failure provides evidence of how narrow is the window of opportunity. In one of the experiments that failed to yield any proof, except for dropping the use of level saturation, the experiment was identical to that which yielded the 37-step proof; in other words, the hot list consisted of the elements used in the level-saturation experiment that succeeded. In another experiment that failed, the hot list consisted of just the following two clauses.

```
i(x)*x = 1.
x*i(x) = 1.
```

In my view, the narrowness of the window of success is not a weakness of the hot list strategy; rather, it simply reflects the depth of mathematics and the fact that no panacea exists.

6 Recommendations and Hints for Using the Hot List Strategy

This section is devoted to guidance and notions about using the hot list strategy most effectively. I begin with a few recommendations, then follow with more specific hints for choosing parameters.

6.1 Recommendations

I recommend that an input clause placed in list(hot) also be placed in some other list. (Among the exceptions was that discussed earlier, at the end of Section 2.2, in the context of cancellation.) For example, if an input clause would ordinarily be included to *complete* the application of an inference rule rather than *initiate* the application, then I recommend that, if the clause is placed in list(hot), it also be placed in list(usable).

As an aside, and independent of the hot list strategy, clauses one suspects are best used to complete rather than initiate inferences belong, in my view, in list(usable). As another aside, I conjecture that the effectiveness of an automated reasoning program would be increased if, when such a (completion) clause is retained, it were immediately placed in list(usable) rather than being placed in list(sos). This option is not offered by OTTER or, for that matter, from what I know by any program, and it might make an interesting research problem.

When one is studying logic calculi (in which condensed detachment is used in the presence of the inference rule hyperresolution), I recommend placing a clause of the following type both in list(usable) and in list(hot).

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

This clause is best used to complete applications of an inference rule, and almost never to initiate them.

On the other hand, if I were using OTTER to apply the hot list strategy to study rings in which the cube of (every element) x is x, I would place the clause equivalent of $xxx = x$ both in list(sos) and in list(hot). I would take this action even though such a clause is best used to initiate applications of an inference rule, rather than complete an application. One might be puzzled by this recommendation, for clauses in list(sos) are best used to initiate applications of an inference rule, while clauses in the hot list are used only to complete applications. Nevertheless, my experience with the hot list strategy suggests that the inclusion (in the hot list) of such clauses adds to the effectiveness of this strategy.

As a global recommendation, I suggest including in the hot list those clauses that correspond to the *special hypothesis* of the proposed theorem under attack. Such clauses also, in my view, are wisely placed in list(sos).

For a related global recommendation, I suggest the hot list consist of those equations from the input set of support having eight or fewer symbols (ignoring parentheses and commas) whose right-hand argument is a single symbol, constant or variable. Of course, I have in mind that predicates, functions, and the like are represented with single letters. The hot list can also be augmented with all similar "short and simple" clauses taken from the usable list.

I recommend using the dynamic hot list strategy when one suspects that some of the clauses adjoined during the run merit repeated visiting as hypotheses for completing applications of an inference rule. In particular, I recommend the assignment of a small value for the dynamic_heat_weight, enough to permit new clauses to be adjoined to the hot list during the run, but not so big as to cause the hot list to become large. Even a small value can drown the hot list, if the weight_list contains templates that both have smaller values assigned to them and are frequently matched during the run. Indeed, one must exercise care when combining the dynamic hot list strategy with the resonance strategy. For example, if one includes resonators corresponding to formulas from equivalential calculus, because many formulas can match a single resonator, havoc may be the result. A clue is provided when one sees that OTTER is spending substantial CPU time on a single clause that is chosen as the focus of attention.

Regarding assignments for the values for the heat and the dynamic_heat_weight parameters—with the exceptions just noted and those discussed in Section 6.2—I can only suggest experimentation. One might profitably glance at some of the experiments I feature in Section 5; see also [17, 18, 19, 20] and especially [21] and [22]. (Of the various references, [21] is the choice for the researcher wishing far more detail concerning tendencies exhibited by the options offered by OTTER.)

Before I turn to hints for using the hot list strategy, the following observation needs utterance. The use of the hot list strategy, as is the case for various options offered by OTTER that affect the search space, can produce unexpected results. For example, an assignment of the value 2 to the heat parameter can yield for a given problem a shorter proof than previously in hand, where an assignment of the value 1 may yield no proof. Indeed, in the latter case, the program might inform the user that the set of support has gone empty. The explanation rests with the reordering of the space of drawn conclusions and canonicalizations that can occur with procedures such as demodulation and subsumption. For a second example, a small hot list may produce no proof, a slightly larger one may produce the best proof one has seen, and an even larger hot list may produce a proof of little interest. See Section 5.4 for examples of the type just discussed. In general, when one takes actions that change the search, one can expect that a longer clause might be needed to get a proof or expect other odd occurrences.

6.2 Hints regarding the Hot List Strategy

To complement the cited general recommendations, I offer the following more specific suggestions and examples. The first bulleted item concerns early experimentation; the remaining items pertain to use of the hot list at any time in one's research.

- Especially in the beginning of one's experiments, I recommend that the heat parameter be assigned the value 1, and I recommend that the (input) hot list consist of the clause or clauses that correspond to the special hypothesis of the theorem under attack. For example, if the theorem concerns groups in which the cube of x is the identity e, then the special hypothesis is the equation $xxx = e$. When studying some logic calculus, for a second recommendation, I suggest the axioms of the theory (if fewer than eight in number) and the nonunit clause (if such is used) corresponding to condensed detachment. For a third recommendation, if the theorem under study offers no special hypothesis, or if the special hypothesis is messy (consisting of several nonunit clauses, for example), then I suggest putting in the hot list the elements of the (input) set of support that take the form of positive unit clauses. On the other hand, especially when no special hypothesis exists (as when one is studying some logic calculus), I repeat my second recommendation.

- A value of 2 or greater for the heat parameter is suggested when one wishes a recursively heavier emphasis on the members of the hot list. The inclusion of an input hot list (and, of course, the use of the hot list strategy) is suggested when one conjectures that certain input clauses have been identified as meriting repeated consideration as hypotheses for drawing conclusions by completing applications of an inference rule.

- By placing in the (input) hot list clauses for associativity and commutativity, one can use the hot list in place of a limited form of AC-unification. The greater the value assigned to the heat parameter, the more AC-unification that occurs. However, effectiveness can be severely impaired with associativity in the hot list if the heat parameter is assigned the value 3 or greater.

- Combining a level saturation search with the hot list strategy often produces impressive results. For a taste, when the program is adjoining clauses at level 4, with the hot list strategy in use and the heat parameter assigned the value 2, the program also is deducing (for possible retention) clauses at level 6. In the obvious sense, the cited combination permits the program to look ahead, and the distance is greater than or equal to the value assigned to the heat parameter. For example, although the heat parameter was assigned the value 1 when studying *XHN*, a proof of level 14 was completed as the program was deducing clauses of level 11.

7 Conclusions and a Challenge

In this article, I have featured the *hot list strategy*, presenting numerous pertinent experiments. I have also discussed briefly the dynamic hot list strategy. The hot list strategy asks the researcher to provide an input list (called the hot list) of statements (clauses) that the automated reasoning program uses to complete, in contrast to initiate, applications of the inference rules in use. Ordinarily, one chooses for the members of the hot list clauses that are conjectured to merit revisiting repeatedly, clauses on which to key the program's attack. The dynamic hot list strategy (formulated by McCune) extends the hot list strategy to permit the program to adjoin members to the hot list during the run.

Both formulations address the inaccessibility of certain retained clauses, for far too long, because of their complexity. The evidence presented in this article shows that the hot list strategy can be used successfully in at least four contexts: reducing CPU time, finding proofs of theorems previously out of reach without much intervention of the researcher, finding more elegant proofs, and answering a question whose answer had steadfastly eluded researchers relying on an automated reasoning program.

Continuing in the tradition begun at Argonne National Laboratory approximately three decades ago (in the early 1960s), I close with a challenging problem for interested researchers.

- Evaluate the effectiveness of the hot list strategy, using as members of the hot list *generated* clauses, rather than *retained* clauses. This incarnation or modification of the hot list strategy would permit the program to draw conclusions that are children of clauses that might be discarded because of being too complex as measured in weight.

Appendix

As promised, here I present input files and proofs. The input files are intended to facilitate the further study by researchers of the areas touched on in this article. They also are intended to serve as templates for research in other areas of mathematics and logic. In some cases, I include lines preceded with "%", which McCune's program OTTER treats as a comment. The input files, as well as the proofs given here, provide the merest taste of what one can do with OTTER; more is found in my new book [21].

One of my main reasons for including specific proofs is my strong conviction that likelihood of experimentation producing valuable results is sharply increased. Indeed, when the objective is the formulation of, say, a new strategy or new inference rule or the testing of a reasoning program, I have always been more than puzzled at the nonchalance of some regarding the value of having in hand a proof of the theorem under attack. Few (if any) means are better for measuring progress than seeing how many proof steps of a given proof have been produced with the new approach or the program under evaluation.

I begin with an input file that can be used to initiate one's attack on finding a means for an automated reasoning program to prove, definitely not in a proof checking mode, that Meredith's single axiom suffices for an axiom system for two-valued sentential calculus. To aid one's research, I also include (essentially) Meredith's proof; it was produced with a cursory proof-checking run, using his steps as resonators and assigning max_weight the value 2.

Input File for Studying Meredith's Single Axiom

```
set(hyper_res).
assign(max_weight, 28).
assign(change_limit_after, 2000).
assign(new_max_weight, 20).
assign(max_proofs, -1).
clear(print_kept).
clear(back_sub).
assign(max_mem, 110000).
assign(report, 1800).
assign(max_distinct_vars, 7).
assign(pick_given_ratio, 3).
assign(heat,1).
set(order_history).
set(input_sos_first).

weight_list(pick_given).
%  The following is Meredith's single axiom.
weight(P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))),2).
% Following are the 17 from the known axiom systems; using resonance.
weight(P(i(i(x,y),i(i(y,z),i(x,z)))),2).
weight(P(i(i(n(x),x),x)),2).
weight(P(i(x,i(n(x),y))),2).
weight(P(i(x,i(y,x))),2).
weight(P(i(i(i(x,y),z),i(y,z))),2).
weight(P(i(i(x,i(y,z)),i(y,i(x,z)))),2).
weight(P(i(i(x,y),i(i(z,x),i(z,y)))),2).
weight(P(i(i(x,i(x,y)),i(x,y))),2).
weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),2).
weight(P(i(i(i(x,y),z),i(n(x),z))),2).
weight(P(i(n(n(x)),x)),2).
weight(P(i(x,n(n(x)))),2).
weight(P(i(i(x,y),i(n(y),n(x)))),2).
weight(P(i(i(n(x),n(y)),i(y,x))),2).
weight(P(i(i(x,y),i(i(n(x),y),y))),2).
weight(P(i(i(n(x),y),i(i(z,y),i(i(x,z),y)))),2).
weight(P(i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))),2).
% Following is for recursive tail strategy.
weight(i($(1),$(2)),1).
end_of_list.

list(usable).
% Following is for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).

%  The following disjunctions are known axiom systems.

-P(i(q,i(p,q))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | -P(i(n(n(p)),p)) |
-P(i(p,n(n(p)))) | -P(i(i(p,q),i(n(q),n(p)))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
```

```
$ANSWER(step_allFrege_18_35_39_40_46_21).  % 21 is dependent.

-P(i(q,i(p,q))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) | -P(i(i(q,r),i(i(p,q),i(p,r)))) |
-P(i(p,i(n(p),q))) | -P(i(i(p,q),i(i(n(p),q),q))) | -P(i(i(p,i(p,q)),i(p,q))) |
$ANSWER(step_allHilbert_18_21_22_3_54_30).  % 30 is dependent.

-P(i(q,i(p,q))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | -P(i(i(n(p),n(q)),i(q,p))) |
$ANSWER(step_allBEH_Church_FL_18_35_49).

-P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) | -P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) |
$ANSWER(step_allLuka_x_19_37_59).

-P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANSWER(step_allWos_x_19_37_60).

-P(i(i(p,q),i(i(q,r),i(p,r)))) | -P(i(i(n(p),p),p)) | -P(i(p,i(n(p),q))) |
$ANSWER(step_allLuka_1_2_3).
end_of_list.

list(sos).
%  The following is Meredith's single axiom.
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).  % CN-CAM
%  The following three are Luka, 1 2 3.
% P(i(i(x,y),i(i(y,z),i(x,z)))).
% P(i(i(n(x),x),x)).
% P(i(x,i(n(x),y))).
end_of_list.

list(passive).
-P(i(i(p,q),i(i(q,r),i(p,r)))) | $ANSWER(step_L1).
-P(i(i(n(p),p),p)) | $ANSWER(step_L2).
-P(i(p,i(n(p),q))) | $ANSWER(step_L3).
-P(i(q,i(p,q))) | $ANSWER(step_18).
-P(i(i(i(p,q),r),i(q,r))) | $ANSWER(step_19).
-P(i(i(p,i(q,r)),i(q,i(p,r)))) | $ANSWER(step_21).
-P(i(i(q,r),i(i(p,q),i(p,r)))) | $ANSWER(step_22).
-P(i(i(p,i(p,q)),i(p,q))) | $ANSWER(step_30).
-P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | $ANSWER(step_35).
-P(i(i(i(p,q),r),i(n(p),r))) | $ANSWER(step_37).
-P(i(n(n(p)),p)) | $ANSWER(step_39).
-P(i(p,n(n(p)))) | $ANSWER(step_40).
-P(i(i(p,q),i(n(q),n(p)))) | $ANSWER(step_46).
-P(i(i(n(p),n(q)),i(q,p))) | $ANSWER(step_49).
-P(i(i(p,q),i(i(n(p),q),q))) | $ANSWER(step_54).
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANSWER(step_59).
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANSWER(step_60).
-P(i(n(n(a)),a)) | $ANSWER(lemma_24).
-P(i(a,n(n(a)))) | $ANSWER(lemma_29).
-P(i(i(a,b),i(i(c,a),i(c,b)))) | $ANSWER(lemma_25).
-P(i(i(a,b),i(n(b),n(a)))) | $ANSWER(lemma_36).
end_of_list.

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

```
list(hot).
% Following is for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).
% Following is Meredith's single axiom.
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).  % CN-CAM
end_of_list.
```

Meredith's Proof

```
-----> EMPTY CLAUSE at   1.35 sec ----> 58 [hyper,6,57,47,38]
$ANSWER(Luka,[1,2,3]).

Length of proof is 41.  Level of proof is 30.

---------------- PROOF ----------------

1 [] -P(i(x,y)) | -P(x) |P(y).
6 [] -P(i(i(p,q),i(i(q,r),i(p,r)))) | -P(i(i(n(p),p),p)) | -P(i(p,i(n(p),q))) |
$ANSWER(Luka,[1,2,3]).
7 [] P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).
------------------------------------
8 [hyper,1,7,7] P(i(i(i(i(x,y),i(z,y)),i(y,u)),i(v,i(y,u)))).
9 [hyper,1,7,8] P(i(i(i(x,i(n(y),z)),u),i(y,u))).
10 [hyper,1,7,9] P(i(i(i(x,x),y),i(z,y))).
11 [hyper,1,10,10] P(i(x,i(y,i(z,z)))).
13 [hyper,1,7,11] P(i(i(i(x,i(y,y)),z),i(u,z))).
15 [hyper,1,7,13] P(i(i(i(x,y),z),i(y,z))).
17 [hyper,1,15,7] P(i(x,i(i(x,y),i(z,y)))).
18 [hyper,1,15,17] P(i(x,i(i(i(y,x),z),i(u,z)))).
19 [hyper,1,17,9] P(i(i(i(i(i(x,i(n(y),z)),u),i(y,u)),v),i(w,v))).
20 [hyper,1,7,18] P(i(i(i(i(i(x,i(i(i(y,z),i(n(u),n(v))),u)),w),i(v6,w)),y),i(v,y))).
21 [hyper,1,7,19] P(i(i(i(x,y),i(z,i(n(n(y)),u))),i(v,i(z,i(n(n(y)),u))))).
22 [hyper,1,7,20] P(i(i(i(x,y),i(z,i(i(i(y,u),i(n(v),n(x))),v))),i(w,i(z,i(i(i(y,u),i(n(v),n(x))),v))))).
23 [hyper,1,21,7] P(i(x,i(i(y,z),i(n(n(y)),z)))).
24 [hyper,1,22,17] P(i(x,i(i(i(y,z),u),i(i(i(z,v),i(n(u),n(y))),u)))).
25 [hyper,1,23,23] P(i(i(x,y),i(n(n(x)),y))).
26 [hyper,1,7,23] P(i(i(i(i(x,y),i(n(n(x)),y)),z),i(u,z))).
27 [hyper,1,24,24] P(i(i(i(x,y),z),i(i(i(y,u),i(n(z),n(x))),z))).
29 [hyper,1,10,25] P(i(x,i(n(n(y)),y))).
30 [hyper,1,27,18] P(i(i(i(x,y),i(n(i(i(i(z,i(u,x)),v),i(w,v))),n(u))),i(i(i(z,i(u,x)),v),i(w,v)))).
31 [hyper,1,17,29] P(i(i(i(x,i(n(n(y)),y)),z),i(u,z))).
32 [hyper,1,7,30] P(i(i(i(i(i(x,i(y,i(z,u))),v),i(w,v)),z),i(v6,z))).
33 [hyper,1,7,32] P(i(i(i(x,y),i(z,i(u,i(y,v)))),i(w,i(z,i(u,i(y,v)))))).
34 [hyper,1,33,7] P(i(x,i(i(y,i(y,z)),i(u,i(y,z))))).
35 [hyper,1,34,34] P(i(i(x,i(x,y)),i(z,i(x,y)))).
36 [hyper,1,35,35] P(i(x,i(i(y,i(y,z)),i(y,z)))).
37 [hyper,1,36,36] P(i(i(x,i(x,y)),i(x,y))).
38 [hyper,1,9,37] P(i(x,i(n(x),y))).
39 [hyper,1,37,31] P(i(i(i(x,i(n(n(y)),y)),z),z)).
40 [hyper,1,37,26] P(i(i(i(i(x,y),i(n(n(x)),y)),z),z)).
42 [hyper,1,25,39] P(i(n(n(i(i(x,i(n(n(y)),y)),z))),z)).
43 [hyper,1,7,40] P(i(i(n(x),x),i(y,x))).
46 [hyper,1,18,42] P(i(i(i(x,i(n(n(i(i(y,i(n(n(z)),z)),u))),u)),v),i(w,v))).
47 [hyper,1,37,43] P(i(i(n(x),x),x)).
48 [hyper,1,7,46] P(i(i(i(x,n(i(i(y,i(n(n(z)),z)),n(u)))),v),i(u,v))).
49 [hyper,1,48,47] P(i(x,n(i(i(y,i(n(n(z)),z)),n(x))))).
50 [hyper,1,18,49] P(i(i(i(x,i(y,n(i(i(z,i(n(n(u)),u)),n(y))))),v),i(w,v))).
52 [hyper,1,37,50] P(i(i(i(x,i(y,n(i(i(z,i(n(n(u)),u)),n(y))))),v),v)).
53 [hyper,1,7,52] P(i(i(x,y),i(i(i(z,i(n(n(u)),u)),n(n(x))),y))).
54 [hyper,1,53,53] P(i(i(i(x,i(n(n(y)),y)),n(n(i(z,u)))),i(i(i(v,i(n(n(w)),w)),n(n(z))),u))).
55 [hyper,1,7,54] P(i(i(i(i(i(x,i(n(n(y)),y)),n(n(z))),u),v),i(i(z,u),v))).
57 [hyper,1,55,7] P(i(i(x,y),i(i(y,z),i(x,z)))).
58 [hyper,6,57,47,38] $ANSWER(Luka,[1,2,3]).
```

I now give an input file for proving $LOGT1$. Through appropriate modification, one can use the file to study lattice ordered groups. The 32-step proof I give immediately after the file is obtained by assigning the heat parameter the value 1 (rather than 2) and by commenting out in the hot list the clauses for commutativity of union and of intersection and those for inverse. I then give McCune's 33-step proof of $LOGT1$.

Input File for Studying $LOGT1$

```
set(knuth_bendix).
lex([1,a,u(_,_),n(_,_),*(_,_),i(_),pp(_),np(_)]).
assign(max_weight, 15).
assign(max_proofs, 36).
assign(max_mem, 40000).
assign(pick_given_ratio, 6).
assign(heat, 2).
assign(report, 900).
% set(really_delete_clauses).
clear(print_kept).
clear(print_new_demod).
clear(print_back_demod).

list(usable).
x = x.
(x*y)*z = x* (y*z).
1*x = x.
x*1 = x.
i(x)*x = 1.
x*i(x) = 1.
i(1) = 1.
i(i(x)) = x.
i(x*y) = i(y)*i(x).
n(x,x) = x.
u(x,x) = x.
n(x,y) = n(y,x).
u(x,y) = u(y,x).
n(x,n(y,z)) = n(n(x,y),z).
u(x,u(y,z)) = u(u(x,y),z).
end_of_list.

list(sos).
u(n(x,y),y) = y.
n(u(x,y),y) = y.
x*u(y,z) = u(x*y,x*z).
x*n(y,z) = n(x*y,x*z).
u(y,z)*x = u(y*x,z*x).
n(y,z)*x = n(y*x,z*x).
pp(x) = u(x,1).
np(x) = n(x,1).
pp(a)*np(a) != a.
end_of_list.

list(passive).
pp(a)*np(a) != a | $ANSWER(step_d33).
i(q)*q*r != r | $ANSWER(step_d01).
u(n(q,r),q) != q | $ANSWER(step_d02).
u(q,n(r,q)) != q | $ANSWER(step_d03).
n(u(q,r),q) != q | $ANSWER(step_d04).
u(q,u(r,q)) != u(r,q) | $ANSWER(step_d05).
u(q,n(q,r)) != q | $ANSWER(step_d06).
n(q,n(r,q)) != n(r,q) | $ANSWER(step_d07).
u(q,u(q,r)) != u(q,r) | $ANSWER(step_d08).
u(n(q,r),n(q,n(r,s))) != n(q,r) | $ANSWER(step_d09).
```

```
u(i(u(q,r))*q,i(u(q,r))*r) != 1 | $ANSWER(step_d10).
n(i(n(q,r))*q,i(n(q,r))*r) != 1 | $ANSWER(step_d11).
n(u(x,x*x),u(x,1)) = x | $ANSWER(step_d12).
n(u(x,x*x),u(1,x)) = x | $ANSWER(step_d13).
u(i(u(q,r))*q,1) != 1 | $ANSWER(step_d14).
u(i(u(1,q)),1) != 1 | $ANSWER(step_d15).
u(i(u(q,r))*q*s,s) != s | $ANSWER(step_d16).
n(1,i(u(1,q))) != i(u(1,q)) | $ANSWER(step_d17).
n(i(n(q,r))*r,1) != 1 | $ANSWER(step_d18).
n(i(n(q,1)),1) != 1 | $ANSWER(step_d19).
u(1,i(n(q,1))) != i(n(q,1)) | $ANSWER(step_d20).
n(u(1,x),u(x,x*x)) = x | $ANSWER(step_d21).
u(n(q,1),n(q,i(u(1,r)))) != n(q,1) | $ANSWER(step_d22).
u(i(u(q,r)),i(q)) != i(q) | $ANSWER(step_d23).
u(i(u(i(q),r)),q) != q | $ANSWER(step_d24).
u(i(u(q,i(r))),r) != r | $ANSWER(step_d25).
u(q,i(u(r,i(q)))) != q | $ANSWER(step_d26).
n(q,i(u(r,i(q)))) != i(u(r,i(q))) | $ANSWER(step_d27).
u(i(q),i(n(q,r))) != i(n(q,r)) | $ANSWER(step_d28).
u(n(q,1),i(u(1,i(q)))) != n(q,1) | $ANSWER(step_d29).
i(n(q,1)) != u(1,i(q)) | $ANSWER(step_d30).
n(u(q*r,r),u(r,i(q)*r)) != r | $ANSWER(step_d31).
n(u(1,q),u(q,q*q)) != q | $ANSWER(step_d32).
end_of_list.

list(hot).
% (x*y)*z = x* (y*z).
n(x,y) = n(y,x).
u(x,y) = u(y,x).
% n(n(x,y),z) = n(x,n(y,z)).
% u(u(x,y),z) = u(x,u(y,z)).
pp(x) = u(x,1).
np(x) = n(x,1).
i(x)*x = 1.
x*i(x) = 1.
u(n(x,y),y) = y.
n(u(x,y),y) = y.
x*u(y,z) = u(x*y,x*z).
x*n(y,z) = n(x*y,x*z).
u(y,z)*x = u(y*x,z*x).
n(y,z)*x = n(y*x,z*x).
end_of_list.
```

A Quicker Proof of $LOGT1$

```
----> UNIT CONFLICT at 3148.67 sec ----> 23567 [binary,23565.1,85.1] $F.
Length of proof is 32.  Level of proof is 13.
---------------- PROOF ----------------

36 [] u(n(x,y),y) = y.
37 [] n(u(x,y),y) = y.
40 [] u(y,z)*x = u(y*x,z*x).
41 [] n(y,z)*x = n(y*x,z*x).
44,43 [] (x*y)*z = x*y*z.
46,45 [] 1*x = x.
48,47 [] x*1 = x.
50,49 [] i(x)*x = 1.
51 [] x*i(x) = 1.
54,53 [] i(1) = 1.
56,55 [] i(i(x)) = x.
63 [] n(x,y) = n(y,x).
64 [] u(x,y) = u(y,x).
66,65 [] n(n(x,y),z) = n(x,n(y,z)).
68,67 [] u(u(x,y),z) = u(x,u(y,z)).
```

```
70,69 [] u(n(x,y),y) = y.
71 [] n(u(x,y),y) = y.
74,73 [] x*u(y,z) = u(x*y,x*z).
76,75 [] x*n(y,z) = n(x*y,x*z).
78,77 [] u(x,y)*z = u(x*z,y*z).
82,81 [] pp(x) = u(x,1).
84,83 [] np(x) = n(x,1).
------------------------------------
85 [demod,82,84,76,78,46,48] n(u(a*a,a),u(a,1)) != a.
------------------------------------
88 [para_into,69.1.1.1,63.1.1] u(n(x,y),x) = x.
94 (heat=1) [para_into,88.1.1.1,37.1.1] u(x,u(y,x)) = u(y,x).
100 [para_from,69.1.1,67.1.1.1] u(n(x,y),u(y,z)) = u(y,z).
103,102 (heat=1) [para_into,100.1.1.1,37.1.1] u(x,u(x,y)) = u(x,y).
107 [para_into,71.1.1.1,64.1.1] n(u(x,y),x) = x.
111,110 [para_into,71.1.1,63.1.1] n(x,u(y,x)) = x.
123 [para_from,88.1.1,67.1.1.1] u(n(x,y),u(x,z)) = u(x,z).
141 [para_into,107.1.1,63.1.1] n(x,u(x,y)) = x.
152,151 [para_into,73.1.1,49.1.1] u(i(u(x,y))*x,i(u(x,y))*y) = 1.
154 (heat=1) [para_from,151.1.1,37.1.1.1] n(1,i(u(x,y))*y) = i(u(x,y))*y.
307,306 [para_from,123.1.1,141.1.1.2,demod,66] n(x,n(y,u(x,z))) = n(x,y).
818 [para_into,151.1.1.1.1.1,102.1.1,demod,103,74,152] u(i(u(x,y))*x,1) = 1.
833 (heat=1) [para_from,818.1.1,40.1.1.1,demod,46,44,46] u(i(u(x,y))*x*z,z) = z.
866 [para_into,818.1.1.1,47.1.1] u(i(u(1,x)),1) = 1.
931 [para_from,866.1.1,306.1.1.2.2] n(i(u(1,x)),n(y,1)) = n(i(u(1,x)),y).
957 (heat=1) [para_from,931.1.1,36.1.1.1] u(n(i(u(1,x)),y),n(y,1)) = n(y,1).
1413 [para_into,833.1.1.1.2,51.1.1,demod,48] u(i(u(x,y)),i(x)) = i(x).
1416 [para_into,833.1.1.1.2,49.1.1,demod,48] u(i(u(i(x),y)),x) = x.
1443 [para_into,1416.1.1.1.1,94.1.1] u(i(u(x,i(y))),y) = y.
1460 [para_into,1416.1.1,64.1.1] u(x,i(u(i(x),y))) = x.
1567 [para_from,1443.1.1,141.1.1.2] n(i(u(x,i(y))),y) = i(u(x,i(y))).
1662,1661 [para_into,1413.1.1.1.1,88.1.1] u(i(x),i(n(x,y))) = i(n(x,y)).
1684 [para_from,1413.1.1,141.1.1.2] n(i(u(x,y)),i(x)) = i(u(x,y)).
1705 (heat=1) [para_into,1684.1.1.1.1,36.1.1,demod,70] n(i(x),i(n(y,x))) = i(x).
1910 [para_into,1705.1.1.1,53.1.1,demod,54] n(1,i(n(x,1))) = 1.
1933,1932 (heat=1) [para_from,1910.1.1,36.1.1.1] u(1,i(n(x,1))) = i(n(x,1)).
7237,7236 [para_into,957.1.1.1,1567.1.1] u(i(u(1,i(x))),n(x,1)) = n(x,1).
7252,7251 [para_from,7236.1.1,1460.1.1.2.1,demod,68,1662,1933] i(n(x,1)) = u(1,i(x)).
7302 [para_from,7236.1.1,154.1.1.2.1.1,demod,7252,76,78,46,50,48,307,111,7237,7252,76,78,46,50,48]
     n(u(x,1),u(1,i(x))) = 1.
7338 (heat=1) [para_from,7302.1.1,41.1.1.1,demod,46,78,46,78,46] n(u(x*y,y),u(y,i(x)*y)) = y.
23565 [para_into,7338.1.1.2.2,51.1.1,demod,56,56,56,56] n(u(x*x,x),u(x,1)) = x.
23567 [binary,23565.1,85.1] $F.
```

McCune's 33-Step Proof of $LOGT1$

```
----> UNIT CONFLICT at 19280.59 sec ----> 34171 [binary,34169.1,1674.1] $F.
Length of proof is 33.  Level of proof is 10.
---------------- PROOF ----------------

3,2 [] (x*y)*z = x*y*z.
5,4 [] 1*x = x.
7,6 [] x*1 = x.
8 [] i(x)*x = 1.
10 [] x*i(x) = 1.
15,14 [] i(i(x)) = x.
20 [] u(x,x) = x.
22 [] n(x,y) = n(y,x).
23 [] u(x,y) = u(y,x).
24 [] n(n(x,y),z) = n(x,n(y,z)).
27,26 [] u(u(x,y),z) = u(x,u(y,z)).
28 [] u(n(x,y),y) = y.
30 [] n(u(x,y),y) = y.
33,32 [] x*u(y,z) = u(x*y,x*z).
```

```
35,34 [] x*n(y,z) = n(x*y,x*z).
37,36 [] u(x,y)*z = u(x*z,y*z).
39,38 [] n(x,y)*z = n(x*z,y*z).
41,40 [] pp(x) = u(x,1).
43,42 [] np(x) = n(x,1).
------------------------------------
44 [demod,41,43,35,37,5,7] n(u(a*a,a),u(a,1)) != a.
------------------------------------
46,45 [para_from,8.1.1,2.1.1.1,demod,5] i(x)*x*y = y.
60 [para_into,28.1.1.1,22.1.1] u(n(x,y),x) = x.
62 [para_into,28.1.1.1,23.1.1] u(x,n(y,x)) = x.
64 [para_into,30.1.1.1,23.1.1] n(u(x,y),x) = x.
70 [para_into,60.1.1.1,30.1.1] u(x,u(y,x)) = u(y,x).
74 [para_into,60.1.1,23.1.1] u(x,n(x,y)) = x.
79,78 [para_from,62.1.1,30.1.1.1] n(x,n(y,x)) = n(y,x).
88,87 [para_into,26.1.1.1,20.1.1] u(x,u(x,y)) = u(x,y).
107 [para_into,74.1.1.2,24.1.1] u(n(x,y),n(x,n(y,z))) = n(x,y).
120,119 [para_into,32.1.1,8.1.1] u(i(u(x,y))*x,i(u(x,y))*y) = 1.
134,133 [para_into,34.1.1,8.1.1] n(i(n(x,y))*x,i(n(x,y))*y) = 1.
211 [para_into,44.1.1.1,23.1.1] n(u(a,a*a),u(a,1)) != a.
555 [para_into,211.1.1.2,23.1.1] n(u(a,a*a),u(1,a)) != a.
637 [para_into,119.1.1.1.1.1,87.1.1,demod,88,33,120] u(i(u(x,y))*x,1) = 1.
671 [para_into,637.1.1.1,6.1.1] u(i(u(1,x)),1) = 1.
683 [para_from,637.1.1,36.1.1.1,demod,5,3,5] u(i(u(x,y))*x*z,z) = z.
725 [para_from,671.1.1,64.1.1.1] n(1,i(u(1,x))) = i(u(1,x)).
934 [para_into,133.1.1.1.1.1,78.1.1,demod,79,35,134] n(i(n(x,y))*y,1) = 1.
1117 [para_into,934.1.1.1,6.1.1] n(i(n(x,1)),1) = 1.
1169,1168 [para_from,1117.1.1,60.1.1.1] u(1,i(n(x,1))) = i(n(x,1)).
1674 [para_into,555.1.1,22.1.1] n(u(1,a),u(a,a*a)) != a.
1783 [para_from,725.1.1,107.1.1.2.2] u(n(x,1),n(x,i(u(1,y)))) = n(x,1).
2464 [para_into,683.1.1.1.2,10.1.1,demod,7] u(i(u(x,y)),i(x)) = i(x).
2466 [para_into,683.1.1.1.2,8.1.1,demod,7] u(i(u(i(x),y)),x) = x.
2490 [para_into,2466.1.1.1.1,70.1.1] u(i(u(x,i(y))),y) = y.
2567 [para_into,2490.1.1,23.1.1] u(x,i(u(y,i(x)))) = x.
2599 [para_from,2490.1.1,64.1.1.1] n(x,i(u(y,i(x)))) = i(u(y,i(x))).
2649,2648 [para_into,2464.1.1.1.1,60.1.1] u(i(x),i(n(x,y))) = i(n(x,y)).
19584 [para_into,1783.1.1.2,2599.1.1] u(n(x,1),i(u(1,i(x)))) = n(x,1).
19808 [para_from,19584.1.1,2567.1.1.2.1,demod,27,2649,1169] i(n(x,1)) = u(1,i(x)).
20017 [para_from,19808.1.1,45.1.1.1,demod,39,5,35,37,5,46,37,5] n(u(x*y,y),u(y,i(x)*y)) = y.
34169 [para_into,20017.1.1.1.1,8.1.1,demod,15] n(u(1,x),u(x,x*x)) = x.
34171 [binary,34169.1,1674.1] $F.
```

The following input file can be used, with suitable modifications, to study Robbins algebra. It was used to prove $RAT5$. The commented-out weight templates illustrate, if comments are removed, how one can used the resonance strategy by keying on the positive proof steps from a related theorem.

Input File for Studying Robbins Algebra

```
set(knuth_bendix).
clear(eq_units_both_ways).
set(index_for_back_demod).
set(dynamic_demod_lex_dep).
% set(lex_rpo).
set(process_input).
% set(display_terms).
set(input_sos_first).
clear(print_kept).
clear(print_new_demod).
clear(print_back_demod).
assign(max_proofs, 2).
assign(report, 1800).
```

```
assign(max_weight, 30).
assign(pick_given_ratio, 3).
assign(max_mem, 80000).
% assign(heat,1).
% assign(dynamic_heat_weight, 2).
assign(max_distinct_vars, 3).
lex([a, b, c, d, e, f, g(x), n(x), +(x,x)]).
% lrpo_lr_status([+(x,x)]).

weight_list(pick_given).
% Following is hypothesis.
% weight(EQ(+(c,d),c),2).
% Following are positive steps from a 31-step proof of +(c,c) = c,
% the union of c and c = c, modified to not mention constants.
% weight(EQ(n(+(n(+(x,y)),n(+(y,n(x))))),y),2).
% weight(EQ(n(+(n(+(x,y)),n(+(n(y),x)))),x),2).
% weight(EQ(n(+(n(+(x,+(y,z))),n(+(z,n(+(x,y)))))),z),2).
% weight(EQ(n(+(n(+(x,y)),n(+(n(x),y)))),y),2).
% weight(EQ(n(+(n(+(x,n(y))),n(+(y,x)))),x),2).
% weight(EQ(n(+(n(+(n(x),y)),n(+(y,x)))),y),2).
% weight(EQ(n(+(n(+(x,+(y,z))),n(+(y,n(+(x,z)))))),y),2).
% weight(EQ(n(+(n(+(x,+(y,z))),n(+(x,n(+(z,y)))))),x),2).
% weight(EQ(n(+(n(+(x,+(y,z))),n(+(n(+(x,z)),y)))),y),2).
% weight(EQ(n(+(n($(1)),n(+($(1),n($(1)))))),$(1)),-2).
% weight(EQ(+($(1),+($(1),x)),+($(1),x)),-1).
% weight(EQ(n(+($(1),n(+(n($(1)),+($(1),n($(1))))))),n($(1))),-3).
% weight(EQ(n(+(n(+($(1),x)),n(+(n($(1)),+(x,n(+($(1),n($(1))))))))),x),-2).
% weight(EQ(+($(1),+(x,$(1))),+($(1),x)),-1).
% weight(EQ(n(+(n($(!)),n(+(n($(1)),+($(1),n($(1))))))),$(1)),-3).
% weight(EQ(n(+(n($(1)),+($(1),n($(1))))),n(+($(1),n($(1))))),-3).
% weight(EQ(n(+($(1),n(+($(1),n($(1)))))),n($(1))),-2).
% weight(EQ(n(+(n($(1)),n(+(n($(1)),+($(1),n(+($(1),n($(1))))))))),$(1)),-4).
% weight(EQ(n(+(n(+($(1),x)),n(+(n($(1)),+(x,$(1)))))),+(x,$(1))),-2).
% weight(EQ(+($(1),+(x,+($(1),y))),+($(1),+(x,y))),-1).
% weight(EQ(+($(1),n(+($(1),n($(1))))),$(1)),-2).
% weight(EQ(+($(1),+(x,n(+($(1),n($(1)))))),+($(1),x)),-2).
% weight(EQ(+(x,n(+($(1),n($(1))))),x),0).
% weight(EQ(+(n(+($(1),n($(1)))),x),x),0).
% weight(EQ(n(+(n(x),n(+($(1),+(x,n($(1))))))),x),0).
% weight(EQ(n(+(n(x),n(n(x)))),n(+($(1),n($(1))))),0).
% weight(EQ(n(+(x,n(x))),n(+($(1),n($(1))))),0).
% weight(EQ(n(n(+(x,n(n(x))))),x),2).
% weight(EQ(n(n(x)),x),2).
% weight(EQ(+(n(+(y,x)),n(+(n(y),x))),n(x)),2).
% Following are to pitch associative variants.
weight(EQ(+(x,+(x,+(x,x))),+(x,+(x,+(x,x)))), 500).
weight(EQ(+(x,+(x,+(x,+(x,x)))),+(x,+(x,+(x,+(x,x))))), 500).
% Following is for tail strategy.
weight(EQ($(1),$(2)), 1).
% Following is for discarding triple n.
weight(n(n(n(1))), 500).
end_of_list.

list(usable).
EQ(x,x).
EQ(+(x,y),+(y,x)).
EQ(+(+(x,y),z),+(x,+(y,z))).
end_of_list.

list(sos).
EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).    % Robbins axiom
EQ(+(c,d),c).   % hypothesis
-EQ(+(n(+(a,n(b))),n(+(n(a),n(b)))),b).  % denial of Huntington axiom
end_of_list.
```

```
list(passive).
-EQ(+(x,x),x) | $ANS(step_thm).
-EQ(n(n(n(n(a)+b)+n(a+b)+e)+n(e+b)),e) | $ANS(step_winker01).
-EQ(n(n(n(a+b)+n(a)+b)+b),n(a+b)) | $ANS(step_winker02).
-EQ(n(n(c+c+n(d+n(a))+n(d+a))+n(d+n(c))),c+c) | $ANS(step_winker03).
-EQ(n(n(n(a+b)+n(a)+b+b)+n(a+b)),b) | $ANS(step_winker04).
-EQ((n(n(c)+n(d+n(c)))),d) | $ANS(step_m01).
-EQ((n(n(c+a+b)+n(d+n(c+a)+b))),d+b) | $ANS(step_m02).
-EQ((n(d+n(c+n(d+n(c))))),n(d+n(c))) | $ANS(step_m03).
-EQ((n(n(n(n(a)+b)+n(a+b)+e)+n(b+e))),e) | $ANS(step_m04).
-EQ((n(n(n(n(a)+b)+a+b)+b)),n(n(a)+b)) | $ANS(step_m05).
-EQ((n(n(c)+n(d+n(c+n(a))+n(c+a)))),d) | $ANS(step_m06).
-EQ((n(n(d+n(c+n(d+n(c)))+a)+n(n(d+n(c))+a))),a) | $ANS(step_m07).
-EQ((n(n(c+n(d+n(c)))+n(d+n(c)))),d) | $ANS(step_m08).
-EQ((n(n(c+n(c+n(d+n(c))))+n(c+n(d+n(c))))),c) | $ANS(step_m09).
-EQ((n(d+n(d+n(c)+n(c+n(d+n(c)))))),n(c+n(d+n(c)))) | $ANS(step_m10).
-EQ((n(d+n(n(c)+n(n(d+n(c))+n(a))+n(n(d+n(c))+a)))),n(c)) | $ANS(step_m11).
-EQ((n(n(n(n(n(a)+b)+a+b)+b+e)+n(n(n(a)+b)+e))),e) | $ANS(step_m12).
-EQ((n(c+n(d+n(c)))),n(c)) | $ANS(step_m13).
-EQ((n(n(c)+n(c+n(c)))),c) | $ANS(step_m14).
-EQ((n(n(c+a)+n(n(c)+n(c+n(c))+a))),a) | $ANS(step_m15).
-EQ((n(n(c+c+n(c+n(c)))+n(c+n(c)))),c) | $ANS(step_m16).
-EQ((n(c+n(c+n(c)+n(c+c+n(c+n(c)))))),n(c+c+n(c+n(c)))) | $ANS(step_m17).
-EQ((n(c+c+n(c+n(c)))),n(c)) | $ANS(step_m18).
-EQ((d+n(c+n(c))),d) | $ANS(step_m19).
-EQ((c+n(c+n(c))),c) | $ANS(step_m20).
-EQ((n(c+n(c))+a),a) | $ANS(step_m21).
end_of_list.

% list(demodulators).
% EQ(+(x,y),+(y,x)).
% EQ(+(+(x,y),z),+(x,+(y,z))).
% EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).    % Robbins axiom
% end_of_list.

% list(hot).
% EQ(+(c,d),c).   % hypothesis
% EQ(n(+(n(+(x,y)),n(+(x,n(y))))),x).    % Robbins axiom
% EQ(+(x,y),+(y,x)).
% EQ(+(+(x,y),z),+(x,+(y,z))).
% end_of_list.
```

In view of the historical significance of finding a proof of $RAT5$ *without* induction and *without* AC-unification, I include the following proof.

A Historically Significant Proof of $RAT5$

```
----> UNIT CONFLICT at 9770.64 sec ----> 48310 [binary,48308.1,1.1] $ANS(step_thm).
Length of proof is 78.  Level of proof is 16.
---------------- PROOF ----------------

1 [] -EQ(x+x,x) | $ANS(step_thm).
29,28 [] EQ(x+y,y+x).
31,30 [] EQ((x+y)+z,x+y+z).
32 [] EQ(n(n(x+y)+n(x+n(y))),x).
35,34 [] EQ(c+d,c).
------------------------------------
37 [para_into,32.1.1.1.1.1,30.1.1,demod,31] EQ(n(n(x+y+z)+n(x+y+n(z))),x+y).
39 [para_into,32.1.1.1.1.1,28.1.1] EQ(n(n(x+y)+n(y+n(x))),y).
41 [para_into,32.1.1.1.1,32.1.1] EQ(n(x+n(n(x+y)+n(n(x+n(y))))),n(x+y)).
43 [para_into,32.1.1.1.2.1.2,32.1.1,demod,29] EQ(n(n(x+y)+n(x+n(y+z)+n(y+n(z)))),x).
45 [para_into,32.1.1.1.2.1,28.1.1] EQ(n(n(x+y)+n(n(y)+x)),x).
```

```
52,51 [para_from,34.1.1,30.1.1.1] EQ(c+d+x,c+x).
61 [para_into,37.1.1.1.1.1.2,28.1.1] EQ(n(n(x+y+z)+n(x+z+n(y))),x+z).
63 [para_into,37.1.1.1.1.1.1,28.1.1,demod,31] EQ(n(n(x+y+z)+n(z+x+n(y))),z+x).
71 [para_from,37.1.1,32.1.1.1.2,demod,29,31] EQ(n(x+y+n(n(x+y+z)+x+y+n(z))),n(x+y+z)).
75 [para_into,51.1.1.2,28.1.1] EQ(c+x+d,c+x).
79 [para_from,51.1.1,32.1.1.1.1.1] EQ(n(n(c+x)+n(c+n(d+x))),c).
81 [para_into,75.1.1.2,30.1.1] EQ(c+x+y+d,c+x+y).
90,89 [para_from,75.1.1,30.1.1.1,demod,31,31] EQ(c+x+d+y,c+x+y).
95 [para_into,39.1.1.1.1.1.1,34.1.1] EQ(n(n(c)+n(d+n(c))),d).
97 [para_into,39.1.1.1.1.1.1,30.1.1] EQ(n(n(x+y+z)+n(z+n(x+y))),z).
101 [para_into,39.1.1.1.1,39.1.1] EQ(n(x+n(n(x+n(y))+n(n(y+x)))),n(x+n(y))).
115 [para_into,39.1.1.1.1.2.1,28.1.1] EQ(n(n(x+y)+n(n(x)+y)),y).
119 [para_into,39.1.1.1.1,28.1.1] EQ(n(n(x+n(y))+n(y+x)),x).
126,125 [para_from,39.1.1,32.1.1.1.1] EQ(n(x+n(n(y+x)+n(n(x+n(y))))),n(y+x)).
131 [para_into,41.1.1.1.2.1.1.1,28.1.1,demod,126] EQ(n(x+y),n(y+x)).
166,165 [para_into,131.1.1.1,30.1.1] EQ(n(x+y+z),n(z+x+y)).
173 [para_from,131.1.1,39.1.1.1.2.1.2,demod,31] EQ(n(n(x+y+z)+n(z+n(y+x))),z).
195 [para_into,45.1.1.1.2.1.1,39.1.1] EQ(n(n(x+n(y+z)+n(z+n(y)))+n(z+x)),x).
199 [para_into,45.1.1.1.2,95.1.1,demod,29,29] EQ(n(d+n(c+n(d+n(c)))),n(d+n(c))).
203 [para_into,45.1.1.1.2,45.1.1,demod,166,29,29] EQ(n(x+n(y+x+n(n(y)+x))),n(n(y)+x)).
209 [para_into,45.1.1.1,28.1.1] EQ(n(n(n(x)+y)+n(y+x)),y).
229 [para_into,43.1.1.1.1,95.1.1] EQ(n(d+n(n(c)+n(n(d+n(c))+x)+n(n(d+n(c))+n(x)))),n(c)).
267 [para_into,115.1.1.1.1.1,30.1.1] EQ(n(n(x+y+z)+n(n(x+y)+z)),z).
325 [para_into,119.1.1.1.1.1,30.1.1] EQ(n(n(x+y+n(z))+n(z+x+y)),x+y).
329 [para_from,119.1.1,43.1.1.1.2.1.2.2,demod,166,29,29] EQ(n(n(x+n(y+n(z)))+n(x+y+n(y+z
     +n(y+n(z))))),x).
361 [para_into,209.1.1.1.2.1,30.1.1] EQ(n(n(n(x)+y+z)+n(y+z+x)),y+z).
411 [para_into,79.1.1.1.1.1,28.1.1] EQ(n(n(x+c)+n(c+n(d+x))),c).
551 [para_into,411.1.1.1,28.1.1] EQ(n(n(c+n(d+x))+n(x+c)),c).
739 [para_into,81.1.1.2,28.1.1,demod,31,90] EQ(c+x+y,c+y+x).
755 [para_into,739.1.1.2,739.1.1,demod,29] EQ(c+c+x+y,c+c+y+x).
862,861 [para_from,61.1.1,32.1.1.1.2,demod,29,31] EQ(n(x+y+n(n(x+z+y)+x+y+n(z))),n(x+z+y)).
889 [para_into,97.1.1.1.1.1.2,75.1.1,demod,31] EQ(n(n(x+c+y)+n(y+d+n(x+c))),y+d).
903 [para_into,97.1.1.1.1.1.1,28.1.1,demod,31] EQ(n(n(x+y+z)+n(y+n(z+x))),y).
1000,999 [para_into,63.1.1.1.1.1.2,739.1.1,demod,31,31] EQ(n(n(x+c+y+z)+n(z+y+x+n(c))),z+y+x).
1047 [para_into,63.1.1.1.1,28.1.1] EQ(n(n(x+y+n(z))+n(y+z+x)),x+y).
1829 [para_into,71.1.1.1.2.2.1.1.1.2,28.1.1,demod,862] EQ(n(x+y+z),n(x+z+y)).
1997 [para_into,173.1.1.1.2.1.2.1,75.1.1,demod,31] EQ(n(n(x+d+c+y)+n(y+n(c+x))),y).
2087,2086 [para_from,173.1.1,119.1.1.1.2,demod,29] EQ(n(x+n(n(x+n(y+z))+n(n(z+y+x)))),
     n(x+n(y+z))).
2410 [para_into,267.1.1.1.1.1.2,28.1.1] EQ(n(n(x+y+z)+n(n(x+z)+y)),y).
2416 [para_into,267.1.1.1.1.1.1,28.1.1,demod,31] EQ(n(n(x+y+z)+n(n(z+x)+y)),y).
3178 [para_into,903.1.1.1.2.1.2.1,51.1.1,demod,31] EQ(n(n(d+x+y+c)+n(y+n(c+x))),y).
3294 [para_into,101.1.1.1.2.1.1.1.2,131.1.1,demod,31,2087] EQ(n(x+n(y+z)),n(x+n(z+y))).
4032 [para_into,2410.1.1.1.2.1.1,32.1.1,demod,29] EQ(n(n(x+y)+n(n(x+z)+y+n(x+n(z)))),y).
4064 [para_into,2416.1.1.1.2.1.1.1,75.1.1,demod,31] EQ(n(n(x+d+y+c)+n(n(c+x)+y)),y).
4887,4886 [para_into,165.1.1.1.2,28.1.1] EQ(n(x+y+z),n(y+x+z)).
5674 [para_into,1829.1.1.1,28.1.1,demod,31] EQ(n(x+y+z),n(z+y+x)).
7392 [para_into,1997.1.1.1.2.1.2.1,34.1.1] EQ(n(n(d+d+c+x)+n(x+n(c))),x).
7562 [para_into,7392.1.1.1.1.2.2,28.1.1] EQ(n(n(d+d+x+c)+n(x+n(c))),x).
7710 [para_into,7562.1.1.1,28.1.1] EQ(n(n(x+n(c))+n(d+d+x+c)),x).
9589,9588 [para_into,195.1.1.1.2.1,34.1.1,demod,29] EQ(n(n(c)+n(d+n(x+c)+n(c+n(x)))),d).
9834 [para_into,4064.1.1.1.1.1,28.1.1,demod,31,31] EQ(n(n(d+x+c+y)+n(n(c+y)+x)),x).
9998 [para_from,199.1.1,3178.1.1.1.2,demod,29,35,29,4887,52,29] EQ(n(n(d+n(c))+n(c+n(d+n(c)))),d).
10020 [para_from,199.1.1,551.1.1.1.1.1.2,demod,29] EQ(n(n(c+n(d+n(c)))+n(c+n(c+n(d+n(c))))),c).
10049,10048 [para_from,9998.1.1,2416.1.1.1.2,demod,29,4887,29] EQ(n(d+n(d+n(c)+n(c+n(d+n(c))))),
     n(c+n(d+n(c)))).
12883,12882 [para_into,229.1.1.1.2.1.2.2,7710.1.1,demod,29,35,29,35,29,35,29,29,4887,10049]
     EQ(n(c+n(d+n(c))),n(c)).
12889,12888 [back_demod,10020,demod,12883,12883] EQ(n(n(c)+n(c+n(c))),c).
12954 [para_from,12888.1.1,203.1.1.1.2.1.2.2,demod,29,12889] EQ(n(n(c+n(c))+n(c+c+n(c+n(c)))),c).
16947,16946 [para_from,12954.1.1,9834.1.1.1.2,demod,29,31,4887,52,29] EQ(n(c+n(c+n(c)+
     n(c+c+n(c+n(c))))),n(c+c+n(c+n(c)))).
19738 [para_into,325.1.1.1.1,5674.1.1] EQ(n(n(n(x)+y+z)+n(x+z+y)),z+y).
```

```
19912 [para_into,329.1.1.1.1,12888.1.1,demod,4887,16947] EQ(n(c+c+n(c+n(c))),n(c)).
22688 [para_into,361.1.1.1.2.1,28.1.1,demod,31] EQ(n(n(n(x)+y+z)+n(z+x+y)),y+z).
23276 [para_from,755.1.1,63.1.1.1.1.1,demod,31,1000,29] EQ(x+y+c,c+x+y).
24330 [para_from,23276.1.1,30.1.1.1,demod,31,31,31] EQ(c+x+y+z,x+y+c+z).
24334 [para_from,23276.1.1,28.1.1,demod,31] EQ(c+x+y,y+c+x).
24970 [para_into,24334.1.1.2,28.1.1] EQ(c+x+y,x+c+y).
30025,30024 [para_from,3294.1.1,131.1.1] EQ(n(x+n(y+z)),n(n(z+y)+x)).
41187,41186 [para_into,22688.1.1.1.2.1.2,24970.1.1,demod,31,31] EQ(n(n(n(c)+x+y+z)
     +n(z+x+c+y)),x+y+z).
43403,43402 [para_from,24330.1.1,19738.1.1.1.2.1,demod,31,41187] EQ(x+y+z,z+x+y).
43621,43620 [para_into,43402.1.1.2,28.1.1] EQ(x+y+z,y+x+z).
43961,43960 [para_into,43620.1.1,43402.1.1] EQ(x+y+z,z+y+x).
48174 [para_into,889.1.1.1.1,19912.1.1,demod,43961,43621,9589,29] EQ(d+n(c+n(c)),d).
48245,48244 [para_from,48174.1.1,4032.1.1.1.2.1.2.2.1,demod,43621,52,43403]
     EQ(n(n(d+x)+n(n(d)+n(c+n(c))+x)),x).
48308 [para_from,48174.1.1,1047.1.1.1.2.1.2,demod,43403,30025,48245] EQ(n(c+n(c))+x,x).
48310 [binary,48308.1,1.1] $ANS(step_thm).
```

Next, for those interested in equivalential calculus, I give two short proofs. The first is the shortest of which I know that the formula *XHN* implies the formula *UM*, and the second is the shortest of which I know that the formula *XHK* implies the formula *YQL*.

A Short Proof for *XHN* Implies *UM*

```
----> UNIT CONFLICT at   0.30 sec ----> 38 [binary,37.1,6.1] $ANSWER(P4_UM).
Length of proof is 19.  Level of proof is 14.
---------------- PROOF ----------------

1 [] -P(e(x,y)) | -P(x) | P(y).
2 [] P(e(x,e(e(y,z),e(e(z,x),y)))).
6 [] -P(e(e(e(a,b),c),e(b,e(c,a)))) | $ANSWER(P4_UM).
------------------------------------
18 [hyper,1,2,2] P(e(e(x,y),e(e(y,e(z,e(e(u,v),e(e(v,z),u)))),x))).
19 [hyper,1,18,18] P(e(e(e(e(x,e(y,e(e(z,u),e(e(u,y),z)))),v),e(w,e(e(v6,v7),e(e(v7,w),v6)))),e(v,x))).
20 [hyper,1,18,2] P(e(e(e(e(x,y),e(e(y,z),x)),e(u,e(e(v,w),e(e(w,u),v)))),z)).
21 [hyper,1,19,20] P(e(e(x,e(e(y,z),e(e(z,x),y))),e(e(e(u,v),e(e(v,e(w,e(v6,e(e(v7,v8),
     e(e(v8,v6),v7))))),u)),w))).
22 [hyper,1,18,20] P(e(e(x,e(y,e(e(z,u),e(e(u,y),z)))),e(e(e(v,w),e(e(w,x),v)),e(v6,e(e(v7,v8),
     e(e(v8,v6),v7)))))).
23 [hyper,1,21,2] P(e(e(e(x,y),e(e(y,e(z,e(u,e(e(v,w),e(e(w,u),v))))),x)),z)).
24 [hyper,1,2,23] P(e(e(x,y),e(e(y,e(e(e(z,u),e(e(u,e(v,e(w,e(e(v6,v7),e(e(v7,w),v6))))),z)),v)),x))).
25 [hyper,1,24,2] P(e(e(e(e(x,y),e(e(y,z),x)),e(e(e(u,v),e(e(v,e(w,e(v6,e(e(v7,v8),
     e(e(v8,v6),v7))))),u)),w)),z)).
26 [hyper,1,20,25] P(e(e(x,e(e(x,y),e(z,e(e(u,v),e(e(v,z),u))))),e(y,e(w,e(e(v6,v7),e(e(v7,w),v6)))))).
27 [hyper,1,26,22] P(e(e(e(e(x,e(e(y,z),e(e(z,x),y))),u),u),e(v,e(e(w,v6),e(e(v6,v),w))))).
28 [hyper,1,22,27] P(e(e(e(x,y),e(e(y,e(e(e(z,e(e(u,v),e(e(v,z),u))),w),w)),x)),
     e(v6,e(e(v7,v8),e(e(v8,v6),v7))))).
30 [hyper,1,20,28] P(e(e(e(x,e(e(y,z),e(e(z,x),y))),u),u)).
31 [hyper,1,19,28] P(e(e(e(e(x,e(e(y,z),e(e(z,x),y))),e(e(e(u,e(e(v,w),e(e(w,u),v))),v6),v6)),v7),v7)).
32 [hyper,1,25,30] P(e(e(e(e(x,y),z),e(y,e(u,e(e(v,w),e(e(w,u),v))))),e(z,x))).
33 [hyper,1,30,31] P(e(e(x,y),e(e(y,e(e(z,e(e(u,v),e(e(v,z),u))),e(e(e(w,e(e(v6,v7),
     e(e(v7,w),v6))),v8),v8))),x))).
34 [hyper,1,20,32] P(e(e(x,e(e(y,z),e(e(z,e(u,x)),y))),u)).
35 [hyper,1,32,33] P(e(e(e(e(e(x,e(e(y,z),e(e(z,x),y))),u),u),v),e(e(w,v6),e(e(v6,v),w)))).
36 [hyper,1,32,35] P(e(x,e(e(y,e(e(z,u),e(e(u,y),z))),e(e(e(v,w),e(e(w,e(v6,x)),v)),v6)))).
37 [hyper,1,34,36] P(e(e(e(x,y),z),e(y,e(z,x)))).
38 [binary,37.1,6.1] $ANSWER(P4_UM).
```

A Short Proof for *XHK* Implies *YQL*

```
----> UNIT CONFLICT at 380.90 sec ----> 4789 [binary,4788.1,3.1] $ANSWER(P1_YQL).
Length of proof is 23.  Level of proof is 19.
---------------- PROOF ----------------

1 [] -P(e(x,y)) | -P(x) | P(y).
2 [] P(e(x,e(e(y,z),e(e(x,z),y)))).
3 [] -P(e(e(a,b),e(e(c,b),e(a,c)))) | $ANSWER(P1_YQL).
16 [] -P(e(x,y)) | -P(x) | P(y).
17 [] P(e(x,e(e(y,z),e(e(x,z),y)))).
-------------------------------------
18 [hyper,1,2,2] P(e(e(x,y),e(e(e(z,e(e(u,v),e(e(z,v),u))),y),x))).
19 (heat=1) [hyper,16,17,18] P(e(e(x,y),e(e(e(e(z,u),e(e(e(v,e(e(w,v6),e(e(v,v6),w))),u),z)),y),x))).
20 (heat=1) [hyper,16,18,17] P(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(e(u,v),e(e(w,v),u))),w)).
21 [hyper,1,18,18] P(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(e(e(u,e(e(v,w),e(e(u,w),v))),v6),v7)),e(v7,v6))).
27 [hyper,1,19,18] P(e(e(e(e(x,y),e(e(e(z,e(e(u,v),e(e(z,v),u))),y),x)),e(e(e(w,e(e(v6,v7),
     e(e(w,v7),v6))),v8),v9)),e(v9,v8))).
35 (heat=1) [hyper,16,27,17] P(e(e(e(e(e(x,y),e(e(e(z,e(e(u,v),e(e(z,v),u))),y),x)),w),
     e(v6,e(e(v7,v8),e(e(v6,v8),v7)))),w)).
47 [hyper,1,21,20] P(e(e(e(x,y),e(e(e(e(z,e(e(u,v),e(e(z,v),u))),w),y),x)),w)).
58 [hyper,1,35,47] P(e(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(u,e(e(v,w),e(e(u,w),v)))),e(e(e(v6,e(e(v7,v8),
     e(e(v6,v8),v7))),v9),v10)),e(v10,v9))).
77 [hyper,1,58,21] P(e(x,e(e(e(e(y,e(e(z,u),e(e(y,u),z))),x),e(e(v,w),e(e(v6,w),v))),v6))).
81 [hyper,1,77,47] P(e(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(e(e(u,v),e(e(e(e(w,e(e(v6,v7),
     e(e(w,v7),v6))),v8),v),u)),v8)),e(e(v9,v10),e(e(v11,v10),v9))),v11)).
91 [hyper,1,20,81] P(e(e(e(x,y),e(e(z,e(e(u,v),e(e(z,v),u))),x)),y)).
93 [hyper,1,77,91] P(e(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(e(e(u,v),e(e(w,e(e(v6,v7),
     e(e(w,v7),v6))),u)),v)),e(e(v8,v9),e(e(v10,v9),v8))),v10)).
97 [hyper,1,91,47] P(e(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(e(u,e(e(v,w),e(e(u,w),v))),e(v6,v7))),v7),v6)).
121 [hyper,1,20,93] P(e(e(e(x,e(e(y,z),e(e(u,z),y))),u),x)).
133 [hyper,1,121,121] P(e(x,e(e(y,z),e(e(e(e(u,v),e(e(x,v),u)),z),y)))).
154 [hyper,1,18,133] P(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(e(u,v),e(e(e(e(w,v6),e(e(v7,v6),w)),v),u))),v7)).
177 [hyper,1,97,154] P(e(e(e(x,y),e(e(z,y),x)),e(e(u,v),e(e(z,v),u)))).
181 [hyper,1,121,177] P(e(e(e(e(e(x,y),e(e(z,y),x)),u),z),u)).
197 [hyper,1,121,181] P(e(e(e(x,y),e(e(e(e(z,u),e(e(v,u),z)),y),x)),v)).
219 [hyper,1,181,197] P(e(e(e(e(x,y),e(e(z,y),x)),e(e(z,u),v)),e(v,u))).
240 [hyper,1,18,219] P(e(e(e(x,e(e(y,z),e(e(x,z),y))),e(u,v)),e(e(e(w,v6),e(e(v7,v6),w)),e(e(v7,v),u)))).
4776 [hyper,1,219,240] P(e(e(e(x,y),e(e(z,u),e(e(v,w),e(e(y,w),v)))),e(e(x,u),z))).
4788 (heat=1) [hyper,16,4776,17] P(e(e(x,y),e(e(z,y),e(x,z)))).
4789 [binary,4788.1,3.1] $ANSWER(P1_YQL).
```

To close this article, I give the shortest proof of which I know for *LOGT*2.

A Short Proof of *LOGT*2

```
----> UNIT CONFLICT at 3566.71 sec ----> 19786 [binary,19784.1,1.1] $ANSWER(step_thm).
Length of proof is 22.  Level of proof is 10.
---------------- PROOF ----------------

1 [] i(n(a,b)) != u(i(a),i(b))|$ANSWER(step_thm).
4 [] i(x)*x = 1.
5 [] x*i(x) = 1.
10 [] u(n(x,y),y) = y.
11 [] n(u(x,y),y) = y.
13 [] x*y*z = (x*y)*z.
15,14 [copy,13,flip.1] (x*y)*z = x*y*z.
17,16 [] 1*x = x.
19,18 [] x*1 = x.
20 [] i(x)*x = 1.
34 [] n(x,y) = n(y,x).
35 [] u(x,y) = u(y,x).
39 [] u(x,u(y,z)) = u(u(x,y),z).
```

```
40 [copy,39,flip.1] u(u(x,y),z) = u(x,u(y,z)).
46 [] u(n(x,y),y) = y.
51,50 [] x*u(y,z) = u(x*y,x*z).
54 [] u(x,y)*z = u(x*z,y*z).
61 [] n(x,u(y,z)) = u(n(x,y),n(x,z)).
-----------------------------------
65 [para_into,46.1.1.1,34.1.1] u(n(x,y),x) = x.
70,69 (heat=1) [para_into,65.1.1.1,11.1.1] u(x,u(y,x)) = u(y,x).
73 [para_from,46.1.1,40.1.1.1,flip.1] u(n(x,y),u(y,z)) = u(y,z).
76,75 (heat=1) [para_into,73.1.1.1,11.1.1] u(x,u(x,y)) = u(x,y).
151,150 [para_into,50.1.1,20.1.1,flip.1] u(i(u(x,y))*x,i(u(x,y))*y) = 1.
164 [para_into,150.1.1.1.1.1,75.1.1,demod,76,51,151] u(i(u(x,y))*x,1) = 1.
166 [para_into,150.1.1.1.1.1,69.1.1,demod,70,51,151] u(i(u(x,y))*y,1) = 1.
342 [para_into,54.1.1.1,166.1.1,demod,17,15,17,flip.1] u(i(u(x,y))*y*z,z) = z.
344 [para_into,54.1.1.1,164.1.1,demod,17,15,17,flip.1] u(i(u(x,y))*x*z,z) = z.
354 (heat=1) [para_into,342.1.1.1.2,4.1.1,demod,19] u(i(u(x,i(y))),y) = y.
356 (heat=1) [para_into,344.1.1.1.2,5.1.1,demod,19] u(i(u(x,y)),i(x)) = i(x).
371,370 (heat=2) [para_into,356.1.1.1.1,10.1.1] u(i(x),i(n(y,x))) = i(n(y,x)).
514 [para_into,344.1.1,35.1.1] u(x,i(u(y,z))*y*x) = x.
518 (heat=1) [para_into,514.1.1.2.2,4.1.1,demod,19] u(x,i(u(i(x),y))) = x.
526 (heat=2) [para_from,518.1.1,11.1.1.1] n(x,i(u(i(x),y))) = i(u(i(x),y)).
599,598 [para_into,356.1.1.1.1,65.1.1] u(i(x),i(n(x,y))) = i(n(x,y)).
1009 [para_from,354.1.1,61.1.1.2,flip.1] u(n(x,i(u(y,i(z)))),n(x,z)) = n(x,z).
1033 [para_into,518.1.1,40.1.1] u(x,u(y,i(u(i(u(x,y)),z)))) = u(x,y).
13284 [para_into,1009.1.1.1,526.1.1] u(i(u(i(x),i(y))),n(x,y)) = n(x,y).
19784 [para_from,13284.1.1,1033.1.1.2.2.1,demod,371,599] i(n(x,y)) = u(i(x),i(y)).
19786 [binary,19784.1,1.1] $ANSWER(step_thm).
```

References

[1] Boyer, R., and Moore, J, *A Computational Logic.* New York: Academic Press, 1979.

[2] Boyer, R. S., and Moore, J S., *A Computational Logic Handbook,* 2nd ed., New York: Academic Press, 1998 (also Web information ftp://ftp.cs.utexas.edu/pub/boyer/nqthm/index.html).

[3] Henkin, L., Monk, J., and Tarski, A., *Cylindric Algebras, Part I*, North-Holland, Amsterdam, 1971.

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

[5] Kalman, J., "Condensed detachment as a rule of inference", *Studia Logica* **42,** 443–451 (1983).

[6] Lusk, E., and Overbeek, R., *The Automated Reasoning System ITP,* Technical Report ANL-84-27, Argonne National Laboratory, Argonne, Illinois, 1984.

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

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

[9] McCune, W., "Solution of the Robbins problem", *J. Automated Reasoning,* accepted for publication, 1997.

[10] Meredith, C. A., "Single axioms for the systems (C,N), (C,0), and (A,N) of the two-valued propositional calculus", *J. Computing Systems* **1,** 155–164 (1953).

[11] Veroff, R., and Wos, L., "The linked inference principle, I: The formal treatment", *J. Automated Reasoning* **8**, no. 2, 213–274 (April 1992).

[12] Winker, S., "Robbins algebra: Conditions that make a near-Boolean algebra Boolean", *J. Automated Reasoning* **6**, no. 4, 465–489 (December 1990).

[13] Winker, S., "Absorption and idempotency criteria for a problem in near-Boolean algebras", *J. Algebra* **153**, no. 1, 414–423 (December 1992).

[14] Wos, L., *Automated Reasoning: 33 Basic Research Problems.* Englewood Cliffs, N.J.: Prentice-Hall, 1987.

[15] Wos, L., Overbeek, R., Lusk, E., and Boyle, J. *Automated Reasoning: Introduction and Applications*, 2nd ed., New York: McGraw-Hill, 1992.

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

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

[18] 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).

[19] Wos, L., "OTTER and the Moufang identity problem", *J. Automated Reasoning* **17**, no. 2, 215–257 (1996).

[20] Wos, L., "The power of combining resonance with heat", *J. Automated Reasoning* **17**, no. 2, 23–81 (1996).

[21] Wos, L., "Automating the search for elegant proofs", *J. Automated Reasoning*, accepted for publication, 1997.

[22] Wos, L., "Experiments concerning the Automated Search for Elegant Proofs", Technical Memorandum ANL/MCS-TM-221, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1997.

[23] Wos, L., "Experiments with the Hot List Strategy", Technical Memorandum ANL/MCS-TM-232, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1997.

[24] Wos, L., Winker, S., Veroff, R., Smith, B., and Henschen, L., "Questions concerning possible shortest single axioms in equivalential calculus: An application of automated theorem proving to infinite domains", *Notre Dame J. Formal Logic* **24,** 205–223 (1983).

[25] Wos, L., Winker, S., Veroff, R., Smith, B., and Henschen, L., "A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains", *Artificial Intelligence* **22,** 303–356 (1984).

[26] Wos, L., Veroff, R., Smith, B., and McCune, W., "The Linked Inference Principle, II: The User's Viewpoint", in *Lecture Notes in Computer Science,* Vol. 170, ed. R. E. Shostak, Springer-Verlag, New York, pp. 316–332, 1984.