OTTER and the Moufang Axiom Problem*

Larry Wos

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

Abstract

This article provides additional evidence of the value of using an automated reasoning program as a research assistant. Featured is the use of Bill McCune's program OTTER to find proofs of theorems taken from the study of Moufang loops, but not just *any* proofs. Specifically, the proofs satisfy the property of *purity*. In particular, when given, say, three equivalent identities (which is the case in this article), one is asked to prove the second identity from the first, the third from the second, and the first from the third. If the proof that 1 implies 2 does not rely on 3, then by definition the proof is pure with respect to 3, or simply the *proof is pure*. If for the three identities one finds three pure proofs showing that 1 implies 2, 2 implies 3, and 3 implies 1, then by definition one has found a *circle of pure proofs*. This article shows that there does exist a circle of pure proofs for the three equivalent identities for Moufang loops. In addition—in part to promote the use of automated reasoning programs and to answer questions concerning the choice of options—featured here is the methodology that was employed and a discussion of some of the obstacles, some of which are subtle. The approach relies on paramodulation (which generalizes equality substitution), on demodulation, and—so crucial for attacking deep questions and hard problems—on various strategies, most important of which are McCune's ratio strategy and the hot list strategy. To permit verification of the results presented here, extension of them, and application of the methodology to other unrelated fields, a sample input file and three proofs (of which a circle of pure proofs consists) are included. Research topics and challenges are offered at the close of this article.

1. The Problem, the History, and the Objectives

Automated reasoning programs are now (in 1995) used to answer open questions and prove interesting theorems from a wide variety of disciplines. Two programs that have proved most useful in this context and that are readily available to researchers are Bill McCune's OTTER [McCune94] and Robert Boyer and J Moore's Nqthm [Boyer88]. The fields in which successes have occurred include universal algebra, algebraic geometry, logic calculi, program verification, and chip design. At least in mathematics and in logic—with added insight regarding the use of OTTER—far more is within reach.

One of the objectives of this article is to add to that needed insight by illustrating various features of OTTER; see the sample input file and three proofs given in the Appendix. In that context, some light will be shed on making wise choices from among the numerous options offered by McCune's program. By exploring some of OTTER's options, this article will acquaint the researcher with various techniques for using this program as an assistant. Additional knowledge and understanding regarding OTTER and, more generally, regarding automated reasoning can be gleaned from consulting three books [Wos87,Wos92],Wos95a]. The second of the three books provides a thorough introduction to automated reasoning and, therefore, serves well for learning more about various inference rules, strategies, and procedures cited in this article, for example, *paramodulation*, the *set of support strategy*, and *demodulation*. The first book offers research problems in automated reasoning, and its fourth chapter provides a rather extensive review of the field. The third book (to be published shortly) contains a diskette of OTTER, demonstrates in detail how research in various fields was conducted with this

*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.

program, and (in Chapter 8) gives guidelines for option choosing.

This article has for a second objective the demonstration of the usefulness of OTTER to answer questions that have remained unanswered for many years; see the cited history given shortly. Such demonstrations may encourage researchers totally unfamiliar with automated reasoning to experiment with a reasoning program.

The final and most pressing objective is to present the specific problem under study, the methodology for attacking the problem, and the resulting successes. With the objectives stated, next in order is the promised history.

In the mid-1960s, I first heard (from Wayne Cowell, a colleague at Argonne National Laboratory) about Moufang loops [Moufang35], formally defined in Section 2. Informally, a Moufang loop is an algebraic structure in which multiplication is the operation; however, one's usual intuition is interfered with because certain expected laws do not hold. Indeed, reasoning about such structures is made more difficult because associativity is replaced by an identity of the following type, where multiplication is the implicit operator.

Axiom, Moufang 1:
(xy)(zx) = (x(yz))x

I was informed that, in the presence of the axioms for a loop (given in Section 2), Moufang 1 is equivalent to Moufang 2, which is in turn equivalent to Moufang 3.

Axiom, Moufang 2:
((xy)z)y = x(y(zy))
Axiom, Moufang 3:
x(y(xz)) = ((xy)x)z

Such in fact is the case.

When presented with *n* equivalent identities (such as the three Moufang identities), properties, or definitions, mathematicians and logicians sometimes seek an aesthetic property that can be termed *purity of proof* [Wos95d]. Intuitively (with the formal definitions given in Section 2), purity asks for *n* proofs showing that identity 1 implies 2, 2 implies 3, ..., identity *n* implies identity 1 such that, other than the hypothesis and conclusion, no identity among the *n* is present in a proof. For the three Moufang identities, three proofs would be required. However, such was not available. Instead, the proof of the equivalence of the three identities first showed that 1 implies 2, next that 2 implies 3, then—and here is the so-called flaw in the context of purity—that 3 implies 2, and finally that 2 implies 1. (To be faithful to the proof given by Bruck [Bruck71], he proves Moufang 1 if and only if Moufang 2, then Moufang 2 if and only if Moufang 3; hence, purity is absent. See also [Chein78].) In other words, the proof consisted of four, rather than three, subproofs.

To show how the implied open question was answered—that focusing on the possible existence of three pure proofs for the equivalence of the Moufang identities—is central to this article. Without access to OTTER, such proofs might have stayed out of reach. Indeed, although I made some effort in the mid-1960s to use an automated reasoning program to study Moufang loops, too little power was at that time offered by such programs.

Clearly, as one learns here, the situation has changed, in part because of the significant advances in the design of such programs—OTTER is a fine example—and in part because of the formulation of powerful strategies. Of the strategies that did not exist in the mid-1960s, the ones that play the more important role in this article are McCune's *ratio strategy* [McCune94,Wos95a} and the *hot list strategy* [Wos95a,Wos95b,Wos95d]. Both for the ordering 1, 2, 3 (of the Moufang identities) and the ordering 1, 3, 2, this article shows how the needed proofs were obtained.

Before turning to the formal elements of this article, one might wish the answers to three questions. How does the methodology used here relate to that used in the study of pure proofs for the thirteen shortest single axioms for equivalential calculus [Wos95d]? Why are Moufang loops of interest? What makes the property of purity significant?

Regarding the first question, in [Wos95d], the object is to produce (if possible) a circle of pure proofs for some ordering of all thirteen shortest single axioms for equivalential calculus. Where that area of logic is not concerned with equality, the study of Moufang loops is. Naturally, therefore, a different inference rule was used; instead of hyperresolution, paramodulation (which generalizes equality substitution) was used. Regarding strategy, the ratio strategy and the hot list strategy played a key role here and in the study of equivalential calculus, but level saturation and the dynamic hot list strategy were not used here. For this study of Moufang loops, the methodology did not rely on a phase designed to see how many of the sought-after conclusions could be deduced given an equation as hypothesis. Such a phase was included in the equivalential-calculus study because many possible orderings of the thirteen shortest axioms exist, and the idea was to partially determine which were more likely to succeed in yielding a circle of pure proofs.

As for the second question (for whose answer I thank Ken Kunen), Moufang loops are of interest in part because of their relation to group theory, which itself is of substantial interest to mathematicians and physicists. For example, every subloop generated by two elements is in fact a group. For a second example, every subloop generated by three elements such that $(ab)c = a(bc)$ is a group. Because of having so many subloops that are groups, much nontrivial structure is present in a Moufang loop. Chein found all 159 non-group Moufang loops of order less than 64. An attempt at finding all such non-group Moufang loops of order 64 may be computationally difficult indeed. Various open questions are of interest. For example, does every finite Moufang loop contain Sylow P-subgroups for all primes p that divide its order?

Regarding the third question, purity of proof is related to proof elegance, as are proof length and proof structure (with respect to the type of term that is present or absent). An interest in purity is also distantly related to an interest in the independence of axioms that characterize a set of structures. For example, in group theory, one can dispense with the axioms of right inverse and right identity, for they are provable from the remaining axioms. Then, to perhaps add to one's intuition concerning the significance of purity of proof, one might keep in mind the sometimes-expressed preference for proofs that avoid the use of certain lemmas (such as the inverse of the inverse of x equals x) or avoid the use of a law such as commutativity, and the seeking of a proof avoiding some type of term (such as $n(n(t))$ for any term t, where n denotes negation). Finally, though only distantly related, an interest in purity (to me) is somewhat reminiscent of the seeking of single axioms for some variety, for example, the seeking of a single axiom for all groups such that (for all elements x) the 17th power of x equals the identity e.

2. Definitions and Notation

A *quasigroup* is a set in which multiplication is defined and in which unique left and right solution exist. For example, regarding right solution, for all x and all y, there exists a unique z such that $xz = y$. A *loop* is a quasigroup in which an identity exists, a 1 with $1x = x1 = x$. A *Moufang loop* is a loop satisfying any one of the three equivalent Moufang identities given in Section 1; of course, satisfaction of one of the three means satisfaction of all of the three, the following. (When a line contains a ‘‘%’’, the characters from the first ‘‘%’’ to the end of the line are treated by the program OTTER as a comment.)

```
% Moufang 1:
(x * y) * (z * x) = (x * (y * z)) * x.
% Moufang 2:
((x * y) * z) * y = x * (y * (z * y)).
% Moufang 3:
x * (y * (x * z)) = ((x * y) * x) * z.
```

The Moufang identities just given are expressed in one of the notations acceptable to OTTER, McCune's automated reasoning program used for the experiments and successes reported here.

Although such expressions, in the strict sense that I normally use the term, are *not* clauses (see Wos87,Wos92), throughout this article, I shall be cavalier and use the term clause loosely to include such expressions. Also, I am somewhat cavalier when referring to a Moufang identity, sometimes meaning the equation and sometimes meaning its encoding for OTTER. If Moufang 1, 2, and 3 were expressed in clause form (in the notation I often use), they would appear in the following manner.

```
% Axiom, Moufang 1:
```

```
EQUAL(prod(prod(x,y),prod(z,x)),prod(prod(x,prod(y,z)),x)).
%  Axiom, Moufang 2:
EQUAL(prod(prod(prod(x,y),z),y),prod(x,prod(y,prod(z,y)))).
%  Axiom, Moufang 3:
EQUAL(prod(x,prod(y,prod(x,z))),prod(prod(prod(x,y),x),z)).
```

When equality is present in a problem, typically, I choose as the inference rule paramodulation [Wos87,Wos92]. If that is the choice, I prefer a set of *unit clauses*, clauses (or their equivalent) free of the (logical) **or** symbol, in OTTER the symbol ‘‘|’’. Logical **not** is denoted in OTTER by ‘‘-’’.

The following units can be used to study Moufang loops, using any one of (or all three) Moufang identities.

```
x = x.
x * rs(x,y) = y.  %  right solvable
rs(x, x * y) = y.  %  right solution is unique (implies left cancellation)
ls(x,y) * y = x.  %  left solvable
ls(x * y , y) = x.  %  left solution is unique (implies right cancellation)
1 * x = x.  %  left identity
x * 1 = x.  %  right identity
(x * y) * (z * x) = (x * (y * z)) * x.  %  Moufang 1
((x * y) * z) * y = x * (y * (z * y)).  %  Moufang 2
x * (y * (x * z)) = ((x * y) * x) * z.  %  Moufang 3
```

The study reported here relied on the use of the first seven cited units and, taken one at a time, a Moufang identity.

The researcher interested in studying Moufang loops with the aid of OTTER might begin by proving the following useful property of inverses. For all x, there exist a left and a right inverse of x, and, further, they are equal. Next, one might prove that cancellation, left and right, follows from the given units; see the sample input file given in the Appendix for the two laws. Finally, one might explore the use of cancellation in place of the uniqueness units. If the cancellation laws are employed, which take the form of nonunit clauses, then the inference rule *UR-resolution* [Wos87,Wos92] is used in addition to paramodulation. Different from the approach taken in this article, Kunen studied Moufang loops briefly, using unit clauses for left and right inverse (not assuming the two are equal), for left and right identity, and (one at a time) for a Moufang identity, and two nonunit clauses for the cancellation laws.

Regarding the presence of the unit $x = x$ (reflexivity of equality), standard practice strongly recommends its inclusion when paramodulation is in use (explicitly or implicitly), for proofs often terminate with the deduction of a statement of the form a != a for some a.

Next, needed are the definitions of *circle of proofs* and *pure proof*, for the problem to be solved asks one to find, if such exists, a circle of pure proofs for the three Moufang identities.

Definition, circle of proofs. For a set of k equivalent elements—formulas, equations, properties, conditions, or definitions—a circle of proofs is a sequence of proofs such that the first proof shows that the first element implies the second, the second proof shows that the second element implies the third, ..., and the k-th proof shows that the k-th element implies the first.

Definition, pure proof with respect to a set of elements. For a set of elements—formulas, equations, properties, conditions, or definitions—a proof of element j from element i is pure with respect to the set of elements if and only if it does *not* rely on the use of any of the elements but the j-th and the i-th. The presence of a proper instance of an element other than the j-th or i-th does not render the proof impure. If such instances are absent, we say the proof is *instance pure*. Further, if none of the deduced steps contains as a proper subterm an instance of an unwanted element, we say the proof is *subterm pure*.

3. Obstacles and Subtleties

Ordinarily my approach to attacking a new problem is to make one or more runs and see what I can learn by examining the corresponding output file(s). For but one example, I might discover that a type of term is present in many, many clauses and conjecture that such terms, and hence the clauses containing them, are best purged immediately upon generation. In this study, however, I first discussed the problem with my colleague McCune, for I thought it likely that various obstacles could be recognized and circumvented early. Indeed, our discussions did help to identify certain obstacles and to shape the approach (presented in this article) that was subsequently used to obtain the desired proofs. Also, though indirectly, those discussions set the stage and perhaps even dictated the number and nature of the experiments central to this article; see Section 4 in relation to Sections 3.1 and 3.2.

3.1. Possible Approaches

Although finding any proof regardless of its properties might prove more than challenging, the most obvious—and perhaps most formidable—obstacle is the requirement of purity. Three approaches to deriving pure proofs are possible: an approach based on a *forward* search, using the denial of the conclusion only to detect proof completion; an approach based on a *backward* search, using the axioms and hypotheses of the theorem only to complete applications of an inference rule; and an approach based on a *bidirectional* search, reasoning both forward and backward.

In [Wos95d] where the focus is on circles of pure proofs for thirteen shortest single axioms of equivalential calculus, *forward proofs* are a virtual necessity in that condensed detachment is the inference rule in use (from the viewpoint of logic). (In the study of this area of logic, a backward search is ordinarily impractical, for reasoning backward from the denial of some desired conclusion typically forces the program to cope with very long clauses.) However, for the current study of Moufang loops, having the program produce a proof relying solely on reasoning forward—thus making its examination for purity straightforward (on the surface) although perhaps painful—might be impractical from the perspective of automated reasoning. (Indeed, one learns in Section 3.2 that, at least when demodulation is used, one cannot merely examine the deduced steps of the proof to see if the unwanted equation is present and, if not, then accurately claim that the proof is pure; indeed, an unwanted equation might be present as an intermediate step that results from demodulation.) In fact, as frequently occurred in the experiments featured here, a strictly forward search failed to yield any proof. (However, if one insists, one can instruct OTTER to make a forward search by placing the negation of the target equation in the passive list of clauses. Clauses in that list are used only for detecting unit conflict, which signals proof completion, and for applying forward subsumption.)

If a forward proof is out of reach, one is then forced to instruct the program to seek a *backward proof* or a *bidirectional proof*. The following syntactic example of a backward proof (included simply for illustration) shows that a mere reading of the proof does not suffice where purity is concerned. The axioms of the example are, in clause form, (1) through (3).

(1) P.
(2) -P | Q.
(3) -Q | R.

The theorem to prove asserts that *P* implies *R*. The following proof is one of contradiction, beginning with clause (4), the negation or denial of the conclusion of the theorem, and reasoning backward.

(4) -R.

The assignment is to produce a pure proof, pure with respect to the axiom represented with clause (5).

(5) Q.

The proof uses clause (4) to deduce clause (6), which is then used to deduce clause (7), which contradicts clause (1).

(6) -Q.
(7) -P.

The claim that the proof consisting of clauses (1) through (4) followed by clauses (6) and (7) is pure with respect to clause (5) is, as my colleague McCune has (in effect) pointed out, total sophistry. Of course, the backward proof does not explicitly contain clause (5), but, if it is transformed into a forward proof, clause (5) is indeed present. Put another way, clause (5) is ''really'' present in the proof, even if not explicitly. One thus sees that the cited obstacle of finding a proof that is indeed pure is subtler than it might first appear.

The third possible approach, a *bidirectional proof*, proceeds forward and backward, using the negative equality (arising, from assuming, say, that Moufang 2 is not deducible from Moufang 1) to deduce additional negative equalities. Unfortunately, when a bidirectional search produces a proof— just as is the case when a backward search produces a proof—one must transform the proof into a strictly forward proof to be then checked to see that an unwanted equation is not present. Such checking is error prone if done by hand; one can, of course, have OTTER do the work instead, but some effort and some additional instructions to the program will be required. I shall give an example of how to proceed in that regard at the close of Section 8.

3.2. The Means to Seek Purity

Independent of the approach taken, there is the problem of devising a method (for the program to apply) to avoid the use, implicitly or explicitly, of some unwanted equation. OTTER offers two mechanisms to block the use of some unwanted equation (or clause): weighting [McCharen76,Wos87,Wos92] and demodulation [Wos87,Wos92].

The following command and weight_list illustrate how weighting is used.

```
assign(max_weight,20).
weight_list(purge_gen).
% Blocking use of Moufang 3.
weight((x * (y * (x * z)) = ((x * y) * x) * z),1000).
end_of_list.
```

The inclusion of the cited command and weight_list in the input file will prevent OTTER from retaining the clause equivalent of Moufang 3, for it will be assigned a purge_gen weight strictly greater than the max_weight. (The purge_gen weight is used for deciding whether or not to retain a clause, in contrast to the pick_given weight which is used to decide where next to focus attention. The max_weight places an upper bound on the weight, or priority, of retained clauses. When no weight template applies, the purge_gen weight and the pick_given weight are measured solely in terms of symbol count.)

Because variables in a weight template are treated as indistinguishable, the use of weighting to block, say, the retention of Moufang 3 will also block the retention of any clause that resembles Moufang 3, where the differences rest with the particular occurrences of variables. Whether or not the preceding property is a disadvantage, there does exist a far more serious problem in using weighting to seek pure proofs if, because of other considerations, demodulation is also in use. (The use of demodulation might be virtually required to prevent the program from retaining a huge number of clauses that would otherwise be simplified and then purged with subsumption.)

Regarding the more serious problem, for but one example, the program may apply an inference rule to a set of hypotheses, then apply a sequence of demodulators, and one of the intermediate clauses (before the demodulation is complete) may be Moufang 3. Weighting would have no effect in the situation under discussion. In such an event, although Moufang 3 would not be explicitly present in the proof, it would be used implicitly; therefore, the proof (in a true sense) would not be pure with respect to Moufang 3.

In place of weighting to block the use of an unwanted equation, such as Moufang 3, OTTER offers demodulation used in the following manner.

```
list(demodulators).
% Blocking use of Moufang 3.
EQ((x * (y * (x * z)) = ((x * y) * x) * z), $T).
EQ((((x * y) * x) * z = x * (y * (x * z))), $T).
end_of_list.
```

(Note that this use of demodulation focuses on demodulating an entire clause, in contrast to the far more common use that focuses on a term. Also note that, rather than a single demodulator, two demodulators are included, one to cope with Moufang 3 as given in this article, and one to cope with the case in which its arguments are interchanged.) When OTTER demodulates a clause to $T, **true**, the clause is automatically purged. Just as the use of weighting to seek pure proofs has pitfalls, so also does the use of demodulation. With demodulation as the means for blocking the participation, explicitly or implicitly, of an unwanted equation, three pitfalls exist.

The first pitfall, easily avoided but commonly encountered, concerns the orientation of demodulators. With OTTER, one can give a lexical ordering in the following way.

```
lex([$T,a,b,c,d,e,1,_*_,rs(_,_),ls(_,_),=(_,_)]).
```

As given, $T is treated as lighter than other terms. Were one to err (as I in fact did) and place $T at the righthand end of the list, then the cited equalities (for blocking the use of Moufang 3) would not apply.

The second pitfall concerns the use of back demodulation, a procedure that is so often of value in increasing the effectiveness of a program. To instruct the program to back demodulate clauses stored in its database—which means to rewrite by simplifying and canonicalizing each, if possible, with every newly adjoined demodulator—one uses the following command.

```
set(back_demod).
```

In the presence of this command, when a newly retained clause is made into a demodulator, the program attempts to apply it to *all* clauses currently retained. The key to the possible pitfall is contained in the modifier ''all'', for, indeed, even existing demodulators can be back demodulated. Therefore, if misfortune occurs, the two input demodulators (used to block the retention of Moufang 3) can be back demodulated. As a matter of history, in some of the experiments preceding those reported in this article, such occurred, necessitating the commenting out of the command, thus avoiding the use of back demodulation.

The final pitfall concerns the choice of demodulating inside out or outside in. The default in OTTER is inside out, correctly suggesting that, in the majority of cases, that is the preferred choice. However, such demodulation can cause the program to present a proof in which, say, Moufang 3 is implicitly present. Indeed, when a sequence of demodulators is being applied inside out, one of the intermediate clauses might in fact be the clause equivalent of the unwelcome equation, the equation whose presence prevents the proof from being pure. At that point, it may well happen that neither of the two cited equalities (used to block, for example, Moufang 3) gets applied, because of the inside-out path. The appropriate action to take is to include the following command.

```
set(demod_out_in).
```

4. Theorems and Experiments

Two possible circles of pure proofs exist for the three equivalent Moufang identities. Circle 1, if it exists, consists of three proofs: a proof that Moufang 1 implies Moufang 2, a proof that is pure with respect to Moufang 3; 2 implies 3, pure with respect to 1; and 3 implies 1, pure with respect to 2. In other words, for Circle 1, the identities are ordered 1, 2, 3, 1. The only other possible order is 1, 3, 2, 1, and corresponding to that order is Circle 2. Therefore, to explore the possible existence of either of the two circles, there exist six possibly true theorems to attack.

For each of the six purported theorems—taking into account the discussion of Section 3 and my continued interest in the use of strategy—a sequence of five experiments are in order. For each of the six purported theorems, the first experiment was designed to determine whether *any* proof can be found by OTTER, ignoring purity. This program offers a wide variety of options and strategies. However, at least currently, no algorithm or metarules exist for making effective choices; experience and intuition are generally the basis for option choosing. (In Wos95a, Chapter 8 does offer some biased guidelines in that regard.) The fact that a theorem has already been proved by some means (typically by a researcher) is, at this time (in 1995), no guarantee that a proof of it is within range of an automated reasoning program.

Since for all six theorems OTTER produced a proof, the other four experiments (in the sequence of five) each focused on finding a pure proof. The second experiment focused on the use of weighting to block the unwanted Moufang identity. Again, in all six cases, the second experiment produced a proof, but a proof not guaranteed to be pure. Therefore, as discussed in Section 3, to guarantee that proofs produced are pure, weighting was replaced by demodulation in the third experiment. The third experiment succeeded in all six cases. Actually, as already discussed, a caveat must be issued: In particular, if the proof is bidirectional, then the lack of purity may be hidden within that part of the proof that proceeds backward from the assumed falseness of the theorem. In such an event, either some hand computation is needed, or some not-so-standard use of OTTER is required. In either situation, misfortune is possible in that purity may be absent from the bidirectional proof.

In all six cases, a proof was produced by the third experiment and, for the fourth and fifth experiments, efficiency became an equal concern. The fourth experiment explored the use of the hot list strategy [Wos95a,Wos95b,Wos95d] to see what effect its use would have on efficiency and on proof length. Because the fourth experiment succeeded (in all six cases), the fifth experiment explored a heavier use of the hot list strategy. The fifth experiment in all cases succeeded. However, as it turned out, in four of the six cases (in the context of Experiment 5), an attempt at a (strictly) forward proof failed, necessitating action to (so to speak) convert a bidirectional proof into a forward proof; see Section 8 for the actions that were taken. (Incidentally, I chose not to be concerned about converting bidirectional proofs for any of Experiments 1 through 4 because purity was proved for all six theorems in the context of Experiment 5, after the converting procedure was applied to the four cases where it was needed.)

5. Successes and Options

To gain some understanding and some insight into the options that were used and the significance of the changes in them within a sequence of experiments, some background information is in order. In addition to providing a clearer picture of how the basic problem under study was solved, this background information is intended to aid researchers in the use of OTTER in totally unrelated areas, to show how decisions were made, and to in part satisfy the frequent request for giving guidance for making choices from among the numerous options offered by McCune's program.

Although (as Ken Kunen did in some successful experimentation [Kunen94]) one can include cancellation laws among the axioms, my preference is for a representation in which no nonunit clauses (or their equivalent) are present. This preference is consistent with my goal of demonstrating that the use of an automated reasoning program often requires little guidance in the context of adding various lemmas. Because equality is the relation, *paramodulation* is the choice. (Whether a Knuth-Bendix approach is effective is left to future research, possibly by others.) Also different from Kunen, no input demodulators were included with the purpose of using them for canonicalization. Instead, OTTER was instructed to set(process_input), a command that makes some or all input clauses into demodulators. To permit new demodulators to be adjoined during the run, the command set(dynamic_demod) was included. As expected from the discussion of Section 3, set(demod_out_in) was the command, to instruct demodulation to be from the outside in (rather than inside out).

As those familiar with my views expect, I emphasized the use of strategy: Three were used, part or all of the time. With the command assign(pick_given_ratio,3), McCune's *ratio strategy* [McCune94,Wos92,Wos95a] was used. With this command, OTTER was instructed to choose (for the focus of attention to direct the program's reasoning) three clauses by weight (symbol complexity, in this case), one by first come first serve (or breadth first), then three, then one, and the like. The ratio strategy blends (subject to the assignment of the pick_given_ratio parameter) a bit of level saturation with (usually more) choosing of clauses for the focus of attention by weight.

More important to the research was the use of the *hot list strategy*. Use of the hot list strategy typically requires the researcher to choose from among the input statements that present the question or problem those that are conjectured to merit immediate visiting (as hypotheses) and, if the value assigned to the heat parameter is greater than 1, even immediate revisiting. The chosen statements are placed in the (input) hot list, and they are used one at a time or in combination (depending on the inference rule being applied) to *complete* inference rule applications, rather than to *initiate* applications. Members of the hot list are *not* subject to any restriction regarding unit and nonunit clauses. Potentially powerful choices for members of the hot list include the elements of the special hypothesis and elements of the initial set of support. (The special hypothesis of a theorem of the form **if** *P* **then** *Q* refers to that part of *P*, if such exists, that excludes the underlying axioms and lemmas. For example, in the study of the theory that asserts that rings in which the cube of $x = x$ are commutative, the special hypothesis consists of the equation $xxx = x$.) Also, when an inference rule in use relies on the use of some clause as a nucleus (as in hyperresolution), then placing a copy of such a clause in the hot list is often profitable. Further, if a needed nucleus is absent from the hot list, then the program cannot use the hot list strategy for that case, for the strategy requires that, other than the initiating clause, all remaining clauses used in an application of the corresponding inference rule be present in the hot list.

When the program decides to retain a new conclusion, *before* another conclusion is chosen from list(sos) as the focus of attention to drive the program's reasoning, the hot list strategy causes the new conclusion to initiate applications of the inference rules in use, with the remaining hypotheses (or parents) that are needed all chosen from the hot list. Members of the hot list, whether input or adjoined during a run, are not subject to back demodulation or back subsumption; in other words, clauses in the hot list are never rewritten with the discovery of new demodulators (rewrite rules), nor are they purged because of being captured by a newly retained clause.

If one wishes the program *during* the run to adjoin new members to the hot list, one uses the *dynamic hot list strategy* [Wos95b,Wos95d] (which McCune formulated as an extension of the hot list strategy). If that is the choice, then one assigns an integer, positive or negative, to the dynamic_heat_weight parameter with the intention that conclusions that are retained and that have a *pick_given weight* (complexity) less than or equal to the assigned value will be adjoined to the hot list. (A clause, or its equivalent, technically has two weights, its pick_given weight which is used in the context of choosing clauses as the focus of attention to drive the program's reasoning, and its purge_gen weight which is used in the context of clause discarding; often the two weights are the same.) The dynamic_heat_weight assignment places an upper bound on the pick_given weight of clauses that can be adjoined to the hot list during the run.

When the heat parameter is assigned the value 1, conclusions that result from consulting the hot list (whether one is using the hot list strategy or the dynamic hot list strategy) have *heat level* 1. (By definition, input clauses have level 0, and a deduced clause has a level one greater than that of its parents.) If the program decides to retain a conclusion of heat level 1 and if the (input) heat parameter is assigned the value 2, then *before* another conclusion is chosen as the focus of attention, the heat-level-1 conclusion is used to initiate the search for conclusions of heat level 2 (with the hypotheses needed to complete the corresponding application of the inference rule chosen from the hot list). The heat parameter can be assigned to whatever positive integer the researcher chooses, with the objective of placing corresponding emphasis on the immediate use of the members of the hot list. If the parameter is assigned the value 0, then the hot list strategy will not be used.

Finally, as expected if one is familiar with my research, the *set of support strategy* [Wos87,Wos92] was used. Because attempts at completing forward proofs (each in a single run) failed the majority of the time, the denial of the conclusion to be proved was placed in list(sos), the initial set of support. The only other member of that list was the hypothesis. For example, when attacking the theorem that asserts the deducibility of Moufang 1 from Moufang 3, list(sos) contained two clauses, the equivalent of Moufang 3 and the equivalent of the negation of Moufang 1. (For the researcher wishing some information immediately regarding placing all clauses in the initial set of support, I can report with satisfaction that, other than in a few cases, the results for all five experiments, for each of the six theorems attacked, required less CPU time when the two-element set of support was used; see Section 6.1 for more detail.)

Regarding option changes from experiment to experiment, the minimal changes were made in order to gain the most meaningful information in the context of effectiveness. For example, the only differences between the third and the fifth experiment when attacking the theorem that asserts that Moufang 2 implies Moufang 3, with the constraint that Moufang 1 not be used (explicitly or implicitly), was the addition of a hot list (for the fifth experiment) and the setting of the heat parameter. For a second example, three differences existed between the fourth experiment focusing on proving that Moufang 1 implies Moufang 3 with Moufang 2 not participating *at all* and the fourth experiment focusing on proving that Moufang 2 implies Moufang 1 with Moufang 3 not participating *at all*. First, regarding list(sos) (the initial set of support), Moufang 1 and the negation of Moufang 3 are replaced by Moufang 2 and the negation of Moufang 1. Second, regarding list(demod), which is the input list of demodulators, two equalities corresponding to Moufang 1 are replaced by two corresponding to Moufang 2. Perhaps unexpected, two equalities are included, for the program must be in a position to cope with the deduction of the unwanted identity with the lefthand and righthand arguments interchanged. The third difference rests with the contents of list(hot), the initial hot list. When Moufang 1 is the hypothesis, a copy of it appears in list(hot) in addition to appearing in list(sos). When Moufang 2 is the hypothesis, then a copy of it appears in list(hot) as well as in list(sos).

Regarding the five experiments taken as a sequence, for a given implication, say Moufang 3 implies Moufang 2, many options were shared. Among them, max_weight was assigned the value 20, pick_given_ratio was assigned the value 3, max_proofs was assigned the value 2, and max_mem was assigned the value 20000 (20 megabytes). Also in common were set(dynamic_demod) (to adjoin new demodulators during the run), set(demod_out_in) (for outside-inside demodulation), set(process_input) (to make some or all input clauses into demodulators), and set(para_into) and set(para_from) (both for the inference rule paramodulation). With respect to the latter two set commands, OTTER is instructed, for each clause chosen as the focus of attention to drive its reasoning, to paramodulate *from* that clause and also to paramodulate *into* that clause with all of the clauses available in list(usable). Because paramodulation's performance deteriorates markedly when a nonunit clause is involved, in all five experiments (for each of the six theorems under attack), the commands set(para_into_units_only) and set(para_from_units_only) are included. Other than clauses corresponding to a Moufang identity or the negation of one, all input clauses were placed in list(usable), were not placed in the initial set of support.

For each of the six theorems, the first experiment in the sequence of five focused merely on provability. Indeed, no actions were taken to block the presence in a proof of the unwanted Moufang identity. Despite the lack of a guarantee when using weighting to block an unwanted equation or formula (as discussed in Section 3), Experiment 2 extended Experiment 1 by including a single weight template in weight_list(purge_gen), an inclusion of the following type.

```
weight_list(purge_gen).
% Blocking use of Moufang 3.
weight((x * (y * (x * z)) = ((x * y) * x) * z),1000).
end_of_list.
```

In the third through the fifth experiment (for each theorem of the six), to guarantee (with one proviso) purity, weighting was replaced by demodulation, in the following manner.

```
list(demodulators).
% Blocking use of Moufang 3.
EQ((x * (y * (x * z)) = ((x * y) * x) * z), $T).
EQ((((x * y) * x) * z = x * (y * (x * z))), $T).
end_of_list.
```

As for the proviso, forward proofs were often difficult to complete, necessitating in many cases a bidirectional search. The example given in Section 3.1 (in terms of *P*, *Q*, and *R*) correctly suggests that proofs containing reasoning backward from a negative clause can ‘‘hide’’ the presence of an unwanted equation or formula. As (in effect) promised earlier, I show in Section 8 how OTTER was used to replace the so-called backward portion of the proof with a forward portion, maintaining the restriction of blocking the unwanted identity.

The fourth and fifth experiments in each of the six sequences extended the third experiment by adding the use of the hot list strategy. In the fourth experiment, the (input) hot list contained a single element. For example, when Moufang 3 was the hypothesis, then a copy of it was placed in list(hot). In the fifth experiment also included in list(hot) were the axioms of a loop, those clauses (excluding that for reflexivity) that were also present in list(usable).

5.1. The First Circle

At this point, comparisons are in order. For the first circle, three proofs are of interest: a proof that Moufang 1 implies Moufang 2, *pure* with respect to Moufang 3; a proof that Moufang 2 implies Moufang 3, *pure* with respect to Moufang 1; and a proof that Moufang 3 implies Moufang 1, *pure* with respect to Moufang 2. As commented in Section 1, (from what I know) the third proof was absent in prior studies of Moufang loops. Instead, the proof that Moufang 3 implies Moufang 1 relied on the use of Moufang 2.

The first theorem for discussion asserts that Moufang 1 implies Moufang 2. However, purity is *not* relevant to the first of the sequence of five experiments, for that experiment is concerned merely with completing *any* proof. (All experiments in this article were conducted on a SPARCstation-10. For brevity, although the CPU times that are cited are indeed approximate figures, each is given without the reminder of ‘‘approximately’’.) In 125 CPU-seconds, a forward proof was produced of length 73 (paramodulation steps) and level 20, completing with retention of clause (1876). (By definition, input clauses have level 0, and a deduced clause has a level one greater than that of its parents; demodulators do not count among parents of a deduced clause.) The use of demodulators in deducing a clause affects neither proof length nor proof level.

The second and third experiments (of the sequence of five) produced the same proof with similar statistics.

In the fourth experiment (in which the hot list strategy was introduced), some ground was gained, and some was lost. A shorter proof (based solely on reasoning forward) was found, one of length 58 and level 19, completing with the retention of clause (2707). However, the CPU time required to find the proof increased to 589 CPU-seconds. For the curious, quite often more CPU time is required to complete a shorter proof.

The fifth experiment was most satisfying. A length 28 and level 14 proof was found, completing with retention of clause (611). In addition to the sharp reduction in proof length, less than 4 CPU-seconds were required. The only drawback at all was that the proof contains one deduced step produced by reasoning backward from the denial of the theorem. Two experiments sufficed to replace that step with a forward reasoning step. Of less interest, in 30 CPU-seconds (in the same run) OTTER did complete a forward proof of length 82 and level 23, with retention of clause (1478). Each of the five experiments in fact yielded a pure proof.

The cited successes with the study of Moufang 1 implies Moufang 2 immediately called for the study of Theorem 2, Moufang 2 implies Moufang 3. The first experiment—testing the difficulty of obtaining *any* proof, regardless of purity—produced a proof in 771 CPU-seconds with retention of clause (2783). The proof has length 55 and level 16. Except for one step, the proof is forward; a subsequent experiment produced the desired replacement step to yield a forward proof. The second experiment essentially imitated the first. The third differed somewhat, producing a proof in 866 CPU-seconds with retention of clause (2778). The proof has length 52, level 16, and, except for one step, is a forward proof. A single experiment provided the needed replacement step to yield a forward proof. Even the fourth experiment was similar, yielding a 57-step proof of level 14, completing with retention of clause (2013). To obtain the proof, 591 CPU-seconds were required. Again, one additional experiment sufficed to replace one step to produce a forward proof.

The breakthrough in the context of CPU time, as in the study of the first theorem of Circle 1, occurred with the fifth experiment. Indeed, a 50-step proof of level 17 was produced in 19 CPU-seconds with retention of clause (1294). The proof contains one step resulting from reasoning backward from the negation of the theorem, a step that was easily replaced with one experiment. It turned out that, again, (with the cited step replacements) the proofs yielded by the five experiments focusing on the second theorem of Circle 1 are pure (with respect to Moufang 1).

To complete the study of Circle 1, the third theorem became the focus, the theorem asserting the deducibility of Moufang 1 from Moufang 3. Of course, purity with respect to Moufang 2 was key. In 555 CPU-seconds, the first experiment yielded a 47-step proof of level 16 with retention of clause (2359). The proof contains two steps of so-called backward reasoning. The experiment to replace the two steps with forward reasoning succeeded, but, rather amusing, only one replacement step was needed. (On the other hand, as shown by example in Section 8, sometimes the replacement of one step of backward reasoning can require the use of two steps of forward reasoning.) The second and third experiments essentially mirrored the first. The fourth experiment produced a 55-step proof of level 16 in 1910 CPU-seconds with retention of clause (2491). As in the first experiment, the single experiment designed to replace two backward-reasoning steps with forward-reasoning steps found one step that was sufficient.

Again, the fifth experiment was the most satisfying, but not as dramatically as with the first and second theorems. In 103 CPU-seconds, OTTER produced a 68-step proof of level 22, completing with retention of clause (1860). As was becoming the custom, one experiment produced a forward-reasoning step to replace the single step resulting from reasoning backward. With the cited step replacements, all five proofs are pure.

5.2. The Second Circle

Three theorems, 4, 5, and 6, are the focus for Circle 2. Theorem 4 asserts that Moufang 2 implies Moufang 1, of course, with the additional objective of finding a proof pure with respect to Moufang 3. Theorem 5 has Moufang 3 for its hypothesis and Moufang 2 as its conclusion, and purity is with respect to Moufang 1. Finally, Theorem 6 has Moufang 1 as its hypothesis and Moufang 3 as its conclusion, and purity is with respect to Moufang 2.

Theorem 4 was proved in the first experiment in 193 CPU-seconds with retention of clause (1980). The proof has length 24 and level 12. A single experiment provided a forward-reasoning step to replace the one step that resulted from reasoning backward. The second and third experiments were essentially copies of the first. The fourth yielded a 37-step proof of level 12 in 202 CPU-seconds with retention of clause (1562). A single experiment provided a forward-reasoning step to replace the one step that resulted from reasoning backward. The fifth experiment was minutely pleasing, for 51 CPU-seconds sufficed to produce a proof, one of length 62 and level 19, completing with retention of clause (1640). A single experiment provided a forward-reasoning step to replace the one step that resulted from reasoning backward. With the cited step replacements, the proofs yielded by the five experiments focusing on the first theorem (Theorem 4) of Circle 2 are pure.

Theorem 5 proved to be more difficult than did Theorem 4, requiring in the first experiment 573 CPU-seconds and the retention of clause (2351). The resulting 47-step proof of level 16 is a forward proof. The second and third experiments essentially mirrored the first. The fourth required 752 CPU-seconds and the retention of clause (2030) to complete a forward proof of length 66 and level 17. The fifth experiment produced the desired proof in 205 CPU-seconds with retention of clause (2091), a proof of length 60 and level 22 that is a forward proof. All five proofs are pure and are also forward proofs.

The study of Theorem 6 was more rewarding than was the study of either Theorem 4 or Theorem 5, as one sees from the following data. The first experiment produced a proof of length 59 and level 19 in 41 CPU-seconds with retention of clause (1622). One experiment sufficed to produce a step to convert the 59-step backward proof to a forward proof. The second and third experiments essentially matched the first. The fourth experiment proved to be a singular failure, requiring 667 CPU-seconds and the retention of clause (2740) to complete a 93-step proof of level 25. The proof contains one step resulting from reasoning backward, easily replaced to produce a forward proof by conducting a single experiment.

The fifth experiment, however, was indeed satisfying. Less than 3 CPU-seconds were required to produce a 30-step proof of level 15, and only 614 clauses were retained before the proof was completed. In addition, the proof offered a piquant property: To replace the only step resulting from backward reasoning, to produce a forward proof, required two forward-reasoning steps. With the cited step replacements, the proofs yielded by the five experiments focusing on the third theorem (Theorem 6) of Circle 2 are pure.

6. Insight Regarding the Use of Strategy

To complement the preceding material of this article, this section is devoted to a more detailed discussion of various strategies. Program performance can indeed be sharply enhanced by a judicious use of strategy. The evidence is drawn, as expected, from experimentation.

6.1. The Set of Support Strategy

In the thirty experiments reported here, a natural question concerns the role of the set of support strategy, a strategy that restricts a program's reasoning to drawing conclusions that are recursively traceable to the input clauses placed in list(sos). In particular, what is the effect on program performance if this strategy is not used? Much of the answer is contained in a comparison of the results obtained from conducting experiments in pairs, each pair consisting of one experiment *with* and one *without* the set of support strategy. To conduct an experiment without the set of support strategy, (with OTTER) one places all clauses in list(sos). (As a practical matter, the clause for reflexivity of equality, $x = x$, can be placed in the input usable list, for hardly ever is paramodulation applied *into* or *from* a variable.) In keeping with an accurate account of history, I note that I conducted thirty experiments without using the set of support strategy *before* conducting the thirty with its use, the latter being the basis of this article. To have done so is not in keeping with my usual approach of heavily relying on the set of support strategy, as anyone familiar with my views can attest. The explanation (for my omission of the use of the set of support strategy) rests either with inadequate attention to detail or with a preoccupation with the problem under study.

Among the favorable comparisons, Theorem 2 provides relevant evidence, the theorem that asserts the deducibility of Moufang 3 from Moufang 2. In the first experiment (in which no attempt was made to block the use of Moufang 1), *without* the set of support strategy, a 52-step proof of level 18 was completed, requiring 1897 CPU-seconds. On the other hand, *with* the set of support strategy, a 55-step proof of level 16 was completed, only requiring 771 CPU-seconds. Why the shorter proof without the strategy, yet requiring roughly 2.5 times as much CPU time? First, in the unsupported proof, the first three steps are yielded by applying paramodulation to combinations of various axioms for a loop (see Section 2 for the definition), and none of the parents of any of these steps is Moufang 2 or the negation of Moufang 3. Nevertheless, amusing enough, two of the three steps are found in the proof obtained by using the set of support strategy with list(sos) containing only Moufang 2 and the negation of Moufang 3. When the axioms of a theory are placed in list(sos), OTTER is given permission to explore the underlying theory, seeking lemmas and such. To ensure that the members of list(sos) are considered *before* other clauses for driving the program's reasoning, the following command is included.

set(input_sos_first).

The second factor regarding the difference in CPU time and the difference in proof length rests with the fact that, by placing Moufang 2 and the negation of Moufang 3 in list(sos) and the axioms in list(usable), the program is prevented from exploring the basic theory. Although the result will usually be the inaccessibility to various lemmas, in general far fewer paths of inquiry are traversed. Indeed, without the use of the set of support strategy, to complete the proof, perhaps as many as 303,551 clauses were generated of which 1575 were retained. When the strategy was used, the proof was completed after generating perhaps as many as 289,428 clauses of which 1563 were retained. (Precision regarding how many clauses were generated is lacking, for the run was not terminated immediately after finding the first proof.) More significant, to obtain a proof without the aid of the strategy, 363 clauses were chosen as the focus of attention to drive the program's reasoning. In contrast, with the strategy, 340 clauses were chosen.

For a second example, still focusing on Theorem 2 (of the first circle), the fourth experiment also shows the value of using the set of support strategy. Without the strategy, 1811 CPU-seconds were required to obtain a proof; in contrast, with the strategy, 591 CPU-seconds were required. As noted, the fourth experiment of each sequence of five relies on the use of the hot list strategy, where the (initial) list(hot) contains the hypothesis of the theorem and no other clauses.

A third example is provided by staying with Theorem 2 but switching to the fifth experiment, that in which the (initial) hot list was expanded to include the axioms of a loop. Without the set of support strategy, 171 CPU-seconds were required to produce the desired proof; with the strategy, only 19 CPU-seconds were required.

For additional evidence, the focus switches to Theorem 5, the theorem asserting the deducibility of Moufang 2 from Moufang 3. This theorem is in an obvious sense the converse of Theorem 2, the theorem that was just under scrutiny. In the context of the first of the five experiments, without the set of support strategy, 1615 CPU-seconds were required to obtain a proof; with the strategy, 573 CPU-seconds were required. Perhaps unexpected in view of the decrease in CPU time, 2311 clauses were retained in order to complete a proof without the strategy, in contrast to the requirement of retaining

2351 with it. The additional CPU time is mainly accounted for by the need to choose as the focus of attention 362 clauses, when the strategy was not used, in contrast to choosing 331 clauses as the focus of attention when the strategy was used. With the strategy, OTTER generated roughly 250,000 clauses and retained 1344 to produce a proof. In contrast, without the set of support strategy, to complete a proof, the program generated roughly 277,000 clauses of which 1309 were retained.

An example of proof lengthening by using the set of support strategy is found in the context of Theorem 2. Theorem 5 provides an example of proof shortening: In the context of Experiment 1, without the strategy, the first completed proof has length 54 and level 17; with the strategy, the first completed proof has length 47 and level 16. One might find it piquant to note that the shorter proof was obtained without the use of the three lemmas used as the first three deduced steps in the longer proof, lemmas obtained by applying paramodulation to pairs of axioms for a loop. In other words, the use of three lemmas from the theory of loops (ignoring the property of being Moufang) did not aid OTTER in finding a shorter proof.

The fourth and, to some extent, the fifth experiments (focusing on Theorem 5) also provide evidence of the value of using the set of support strategy. In the fourth experiment, without the strategy, 2420 CPU-seconds were required to produce a proof; with its use, 752 CPU-seconds were required. The first of the two cited proofs has length 34 and level 11; despite the reduction in the required CPU time, the second has length 66 and level 17. In the fifth experiment (in which an extensive hot list was used), 256 CPU-seconds were required to produce a proof without the aid of the set of support strategy; with its aid, 205 CPU-seconds were required. Where the first of the two cited proofs has length 35 and level 16, the second has length 60 and level 22.

The story would hardly be complete without an example of an increase in CPU time when the set of support strategy is use. For that example, Theorem 1 suffices, in the context of the fourth experiment. As a reminder, the theorem asserts the deductibility of Moufang 2 from Moufang 1. The experiment's objective is to find a proof that is pure with respect to Moufang 3, using the hot list strategy with but one element in list(hot), namely, Moufang 2. Without the set of support strategy, 364 CPU-seconds is enough to yield a 68-step proof of level 17. However, although this strategy is considered by many researchers in automated reasoning to be the most powerful strategy for restricting a program's reasoning, 589 CPU-seconds were required to complete a proof. The proof has length 58 and level 19.

The cited data supports my usual recommendation, the following.

Recommendation. In a majority of cases, the power of a program's attack is significantly increased if one uses the set of support strategy. Without additional knowledge, ordinarily, the most effective choice for the initial set of support is the union of the *special hypothesis* and the negation (or denial) of the conclusion. In the presence of a set S of axioms, the special hypothesis of a theorem of the form P implies Q is P. For example, in ring theory, the additional hypothesis that the cube of $x = x$ (for every x) is sufficient to prove commutativity. If the given recommendation is followed regarding which input clauses are to be placed in list(sos), then the elements of (the initial set of support) are the clauses corresponding to $xxx = x$ and the assumed falseness of commutativity.

6.2. The Resonance Strategy

In addition to the set of support strategy and the hot list strategy, OTTER offers other strategies that can sharply increase program performance, for example, the resonance strategy [Wos95c]. The objective of the resonance strategy is to enable the researcher to suggest equations or formulas, called *resonators*, each of whose symbol pattern (all of whose variables are considered indistinguishable) is conjectured to merit special attention for directing a program's reasoning. To each resonator, one assigns a value reflecting its relative significance; the smaller the value, the greater the significance. My notion in formulating this strategy was that the steps of a known proof of one theorem might indeed be effective if used to direct an automated reasoning program in its search for a proof of a related theorem. Resonators are *not* lemmas, do *not* have **true** or **false** values. Instead, any equation or formula that matches an included resonator is given the value of the matching resonator and is, therefore, given a corresponding preference for being chosen as the focus of attention to direct the program's reasoning. A clause matches a resonator if and only if, when all variables in both are treated as the same variable, the two are identical.

After completing the research on which this article is mainly based, it seemed natural to experiment briefly with the resonance strategy to see what effect its use would have, leaving a more thorough investigation either to a future article or to other researchers. My notion was to choose one proof from among the proofs produced by the thirty experiments featured in this article, and use the (positive) deduced steps of that proof as resonators. Equations or formulas containing logical **not** are never used as resonators. What criteria would be reasonable for choosing the key proof?

Although, other than aesthetics, no persuasive justification can be given, three criteria taken together appealed to me. First, narrow the choice to one from among those six experiments in which a substantial hot list was used, the fifth of each sequence of five for the six theorems. Intuitively, if the use of such a hot list could sharply reduce the CPU time required to obtain a proof—which it often did, as reported here—then perhaps the proofs thus obtained offered some special property. Second, narrow the list further by focusing on experiments that yielded a proof in less than 30 CPU-seconds. More intuition: perhaps such a degree of effectiveness would be inherited. Third, from the narrowed list, choose the experiment that reduced the required CPU time the most. Applying the criteria selects the fifth experiment focusing on Theorem 2, that in which Moufang 2 as hypothesis is used to deduce Moufang 3 while blocking the participation of Moufang 1.

From the first proof yielded by the fifth experiment concerned with Theorem 2, forty-nine resonators were chosen, the positive deduced steps. Each was assigned a pick_given weight of 2 (to give any matching clause that is retained a high priority for being chosen as the focus of attention), and a weight_list(pick_and_purge) was used. The same set of support was used. In each of the following experiments, the only change from its original form was the addition of the use of the resonance strategy (as described).

The third criterion for choosing the resonators to be used suggested which of the thirty experiments to revisit first, namely, the first experiment focusing on Theorem 2. From the viewpoint of CPU time, the target was 771 CPU-seconds, the time required to obtain a proof on the first visit to the experiment under consideration. How satisfying: Only 12 CPU-seconds (approximately) were required. Where the first visit completed a proof upon retention of clause (2783), the second visit completed a proof upon retention of clause (895). As for the explanation for the dramatic reduction in CPU time, the first visit required choosing 340 clauses as the focus of attention to drive the reasoning; in contrast, the second visit required but 63. When the proof was completed in the first visit, 29,000 clauses had been generated of which 1563 were retained. In contrast, in the revisiting, 13,000 were generated of which 519 were retained upon completion of the proof.

Closer inspection of the two runs revealed additional riches. Where the proof obtained in the first visit has length 55 and level 16, the proof obtained in the second visit has length 40 and level 12. As one can see from the data, the use of 40 clauses (for the proof) out of 519 is more impressive than using 55 out of 1563. Of the 40 deduced steps in the shorted proof, 20 were *not* present in the longer proof.

Immediately, one might wonder about the direct role of the resonators. In particular, how many of the 40 steps are among the 49 resonators? The answer is 31. In that resonators are treated as if their variables are indistinguishable, the natural question to answer next concerns how many of the 40 deduced steps match (in the sense discussed earlier in the context of the resonance strategy) a resonator. That question can easily be answered by taking the 49 resonators and making all of their variables the variable x, then applying the same actions to the 40 deduced steps, and finally making a comparison. Noting that 31 of the 40 steps are among the resonators, I fully expected that almost all of the 40 steps would match a resonator (in the sense used here). I was wrong: Indeed, the number increased merely by 1, from 31 to 32.

Among other conclusions, one could envision having a strong preference for using the hot list strategy, having, therefore, run Experiment 5 for Theorem 2 first, and then (out of curiosity, at least) using the corresponding resonators to run Experiment 1. Unfortunately, such a sequence of experiments would have hidden the value of the hot list strategy. Still, all in all, the cited results show what can be accomplished by combining the set of support strategy, the hot list strategy, and the resonance strategy. The impressive (to me) reduction in CPU time virtually demanded some additional experimentation. Of the other twenty-nine experiments that might be revisited, clearly the most charming and provocative focused on Theorem 5, for the following reason. Theorem 5 is the converse of Theorem 2, interchanging the hypothesis and conclusion of the latter. Therefore, to expect any significant decrease in CPU time to obtain a proof of Theorem 5 by using resonators from a proof of Theorem 2 is indeed counter-intuitive. To provide the simplest comparison, the obvious choice from among the sequence of five experiments to revisit was the first of the five.

Where in the first visiting 573 CPU-seconds were required to obtain a proof, in the revisiting (with the use of the resonance strategy) 238 CPU-seconds were required. In the first of the two runs, the completed proof has length 47 and level 16; in the second, the proof has length 33 and level 15. Of the 33 deduced steps, 6 are not present in the 47-step proof. Regarding the direct role of the resonance strategy, 18 of the 33 steps are not identical to one of the 49 resonators. In the context of resonator matching, 15 steps do not match a resonator, which (when compared with the preceding experiment) in part explains why the decrease in required CPU time was less dramatic.

From among the remaining twenty-eight experments meriting revisiting, I chose but three for testing the power of the resonance strategy, keying on the same forty-nine resonators; the other twenty-five must wait for future research by me or by another researcher. The first of the three focused on Theorem 4, Experiment 1. (My emphasis on the first of the sequence of five experiments throughout this subsection is explained by the conjecture that the most interesting results would be found in that context.) The first visit to that experiment yielded a 24-step proof of level 12 in 193 CPU-seconds, with retention of clause (1980). The revisiting (with the resonance strategy) produced a proof of length 32 and level 12 in 14 CPU-seconds, with retention of clause (917). Of the 32 deduced steps, 19 are not found among the deduced steps of the proof produced in the first visiting. Only 6 of the 32 are not identical to one of the 49 resonators, and only 5 do not match one of the resonators (in the sense used in the context of the resonance strategy in which all variables are treated as indistinguishable). Again one finds a rather impressive decrease in CPU time in the context of the first visit and the (second) revisiting, and also finds that a large fraction of the deduced steps of the proof produced by the revisiting match one of the included resonators. One could hardly call such correlations merely coincidental.

The second of the three experiments focused on Theorem 1, (in effect) the converse of Theorem 4. Again, the first of the sequence of five experiments was the target. In 125 CPU-seconds, a 73-step proof of level 20 was produced, with retention of clause (1876). With the resonance strategy, only 6 CPU-seconds were required, completing a proof of length 28 and level 12, with retention of clause (817). Of the 28 deduced steps, 11 are not present in the much longer proof. Regarding the role of the resonance strategy, 10 of the 28 steps are not identical to any of the 49 resonators, and 6 do not match any of the resonators.

For the final experiment (of the three relying on the 49 resonators), I chose a so-to-speak self-referential experiment. Specifically, what would occur in the contexts of CPU time and proof length when the 49 resonators taken from the proof of Theorem 2, Experiment 5, are used to influence the search for a proof of that same theorem when revisiting the same experiment? The answers are that the CPU time increased from 19 CPU-seconds to 25 CPU-seconds, but the proof length decreased from 50 to 41; the level decreased from 17 to 15 also. In the revisiting, the proof completed upon retention of clause (1834), after 56 clauses had been chosen as the focus of attention to drive the reasoning. In the first visit, the proof completed upon retention of clause (1294), after 77 clauses had been chosen as the focus of attention. Where did the extra CPU time go, in view of finding a shorter proof? The additional 6 CPU-seconds is almost equally divided with the weighing of clauses—which is to be expected in view of the inclusion of 49 weight templates as opposed to none—and the use of the hot list strategy, which I cannot explain.

The preceding data strongly supports the thesis that, through the use of the resonance strategy, deduced steps of the proof of one theorem *can* be of substantial value in seeking the proof of a related theorem. However, as the following data shows—and as one would almost certainly predict—the choice of resonators sometimes aids the program not at all, or even interferes with its performance. When, for example, one takes as resonators the positive steps of the first proof found for Theorem 1 in Experiment 5, of which there are 27, and uses them to seek a proof of Theorem 2, Experiment 1, the performance of the program is harmed. Indeed, without the resonators, OTTER completes a proof in 771 CPU-seconds; with the 27 resonators, the CPU time increases to 1404 CPU-seconds. Also, the proof length increases, from 55 to 64, and the level increases, from 16 to 18. But the situation is often confusing and often lacks consistency, as the following shows. When the same 27 resonators are used to seek a proof of Theorem 1, Experiment 1, the CPU time decreases to 4 CPU-seconds from 125, the proof length decreases to 33 from 73, and the level decreases to 15 from 20.

6.3. The Dynamic Hot List Strategy

Closely related to the hot list strategy, which plays a key role in obtaining the results featured in this article, is the dynamic hot list strategy. Although I did not use the latter strategy, its use might indeed prove valuable in a study of this type and might provide an interesting research topic.

McCune, recognizing the value of enabling a program to adjoin elements to the hot list *during* a run, extended the hot list strategy to the dynamic hot list strategy. The same heat parameter governs the degree to which the hot list is revisited when a new clause is retained. On the other hand, unlike the hot list strategy, a second input parameter affects the dynamic hot list strategy, namely, the dynamic_heat_weight. The dynamic_heat_weight assignment places an upper bound on the pick_given weight of clauses that can be adjoined to the hot list during the run.

For example, one might wish the program to add to the hot list during the run any retained clause whose pick_given weight (whether determined by a weight template or by symbol count) is less or equal 4. In that event, one includes the following command.

```
assign(dynamic_heat_weight,4).
```

A far more intriguing and intricate example concerns the case in which the researcher wishes the program to adjoin to the hot list during the run certain clauses *if they are deduced and retained*. As an illustration, one might wish the hot list to contain the clause equivalent of the property of commutativity of product *if* the corresponding equation is deduced and retained. The following actions suffice. First, one begins by relying on a strategy that combines the power of the resonance strategy with that of the hot list strategy (in its dynamic incarnation); see [Wos95b]. Second, one assigns, say, the value 2 to the dynamic_heat_weight (by using the just-cited assign command with 4 replaced by 2). Finally, one includes the following.

```
weight_list(pick_and_purge).
weight(EQUAL(prod(x,y),prod(y,x)),2).
end_of_list.
```

Of course, one must assign to the max_weight a value greater than or equal to 2 to permit the program to retain the clause encoding of commutativity of product, if that clause is deduced. If the cited equality clause is deduced, it will be assigned a weight of 2, of course, assuming that no earlier weight template applies and that the clause is not demodulated. The clause will be retained if subsumption does not purge it. Because of having a weight of 2, which is also the value assigned to the dynamic_heat_weight parameter, the clause will be adjoined to the hot list during the run—as desired.

6.4. An Alternative Strategic Approach

For studying Moufang loops and other areas in which equality plays a major role, one other strategy and one other approach merit mention. The strategy is the *tail strategy* [Wos95a], a strategy that gives preference to equations whose second argument is short. To instruct OTTER to use the tail strategy, one includes the following command, which can be included in either of two ways.

```
weight(EQ($(1),$(2)),1).
weight(($(1) = $(2)), 1).
```

This strategy, due to McCune, served nicely for studying equivalential calculus [Wos90], Robbins algebra, and other areas.

Regarding a sharply different alternative than used in this article, by including the following command, one can rely on a Knuth-Bendix approach.

```
set(knuth_bendix).
```

By including the cited command, the program is instructed to continually attempt to make all positive equalities into demodulators (rewrite rules) and, more generally, to rely (as much as possible) on seeking a complete set of reductions. Studies focusing on various aspects of group theory, for example, have profited from using Knuth-Bendix; see McCune92] and [Wos93]. Unfortunately, too often, the use of this approach consumes an inordinate amount of CPU time in demodulation. Therefore, a most attractive (to me) but difficult research problem to attack concerns some compromise between full reliance on Knuth-Bendix and full reliance on paramodulation (as shown in the included input file).

7. Apparently Weaker Axioms

This section is devoted to what might be termed an alternative treatment of Moufang loops, but one that contains some unexpected riches regarding the equivalence of the three Moufang identities. (My colleague McCune is given full credit for this material.)

Partly for review and partly for background, note that quasigroups can be studied in three ways. One can study them in terms of a binary operator (multiplication) satisfying left and right cancellation with the added property that a left and right solution exist. Instead, one can replace the cancellation laws with the property that left and right solutions are unique. Finally, one can study quasigroups in terms of three functions, multiplication, left solution (ls), and right solution (rs), as was done in this article. With the third approach, one uses the four axioms cited in Section 2 for the three functions. To move to the study of loops, one adds the existence of a two-sided identity, 1, with $1x = x1 = x$. Then, for the study of Moufang loops, one adjoins any one of Moufang 1, 2, or 3; the three are equivalent, in the presence of the four axioms for the functions *ls* and *rs* and the two for the constant 1. In addition to being able to prove the cited equivalence, one can also prove that there exists a two-sided inverse in Moufang loops, an equation of the form inv(x)*x = 1 = x*inv(x). And now to the unexpected riches.

To extract the riches, one begins with axioms that assert a two-sided identity and, with respect to that identity, a two-sided inverse. In other words, one replaces the four clauses featuring the functions *ls* and *rs* with two clauses featuring inverse. If Moufang 2 and nothing more is then added, OTTER can prove Moufang 3, Moufang 1, and, regarding the quasigroup axioms, the left and right solution properties and their uniqueness. One has thus arrived at an apparent weakening of the axioms for a Moufang loop, in the sense that four axioms focusing on the functions *ls* and *rs* are replaced by two focusing on inverse. Further, one is on the way to proving the equivalence of the three Moufang identities. If one then turns to an emphasis of Moufang 3, as a replacement for Moufang 2, again all goes well (as just discussed).

All that remains is to switch to keying on Moufang 1 in place of Moufang 2 or 3 and have OTTER attack the corresponding purported theorems. If all goes well, the equivalence has been established. In particular, one has a simpler set of equalities for studying Moufang loops.

But Moufang 1 turns out to be too weak. One cannot prove Moufang 2 or 3 and, even more fascinating, one cannot prove the quasigroup axioms. Indeed, with his program MACE for generating models and counterexamples, McCune found a 3-element model satisfying a two-sided identity, a two-sided inverse, and Moufang 1, but failing to satisfy Moufang 2 or Moufang 3 or the quasigroup axioms regarding solvability. Even further, one of the 3-element models (of the type under discussion) is commutative (with respect to multiplication) and Boolean, in the sense that $xx = e$ for all x. Also, every element has a unique inverse, and different elements have different inverses. Therefore, counterintuitive though it may be, with the aid of OTTER and MACE, one finds that the three Moufang identities are *not* equivalent in a barely weakened axiom system. However, as noted, if Moufang 1 (which is satisfied by the model) is replaced by Moufang 2 or Moufang 3, one returns to the subject of Moufang loops.

To produce the appropriate multiplication table, one needs to know only the value of 0*2; it is 0. The other entries of the table can be quickly written in, using the fact that 1 is the identity, that each element is its own inverse, and that the model is commutative.

As for the relative difficulty of finding proofs when using the two axioms for inverse in place of the four for the functions *ls* and *rs*, the following data sheds some light on the subject. For the first data point, when the approach used in the first of the sequence of five experiments was applied to proving that Moufang 2 implies Moufang 3—with, of course, the two inverse axioms replacing the four in the functions *ls* and *rs*—OTTER completed a proof in 200 CPU-seconds with the retention of clause

(1358). The proof has length 40 and level 14, and it contains one step resulting from reasoning backward from the denial (or negation) of Moufang 3. The same technique for replacing the cited step with one resulting from reasoning forward (to thus produce a direct proof) was successfully applied. When the cited approach was amended by adding the use of the hot list strategy with the (input) heat parameter assigned the value 1 and with the hot list containing clauses for Moufang 2, left and right identity, and left and right inverse, the program completed a 37-step proof of level 14 in 50 CPU-seconds. To obtain the proof, OTTER was required to retain 1060 clauses. Again, the proof contained one deduced step resulting from reasoning backward.

To complete the proof of the equivalence of Moufang 2 and Moufang 3, in the presence of the two axioms for inverse and the two for the identity 1, Moufang 3 was made the hypothesis and Moufang 2 the conclusion. In 111 CPU-seconds, OTTER completed a 19-step proof of level 10, with retention of clause (1090). When the hot list strategy was added to the approach, 143 CPU-seconds were required to complete a 31-step proof of level 16, with retention of clause (1236). Of course, I am disappointed that the hot list strategy addition increased the required CPU time. Also, where without its use the proof that was returned is based solely on forward reasoning, with the hot list strategy one of the deduced steps is based on reasoning backward.

The next data points focus, respectively, on proving that Moufang 2 implies Moufang 1 and that Moufang 3 implies Moufang 1. Regarding the first of the two results, without the hot list strategy, 155 CPU-seconds sufficed to find a 26-step proof of level 12, completing with retention of clause (1222). With the addition of the hot list strategy, 98 CPU-seconds sufficed to yield a 28-step proof of level 12, completing with retention of clause (1204). Both proofs contained a single step resulting from reasoning backward from assuming the theorem false. For the second of the two theorems (Moufang 3 implies Moufang 1), without the hot list strategy, 136 CPU-seconds sufficed to produce a 31-step proof of level 14, completing with retention of clause (1159). Almost immediately, in the same run, OTTER found a 29-step proof of level 14, completing with retention of clause (1172). The first of the two proofs contains one step resulting from reasoning backward; the second contains two such steps. With the hot list strategy, the theorem was proved in 128 CPU-seconds, having a length of 38 and level 17, completing with retention of clause (1143). Again, almost immediately the program found a 34-step proof of level 17, completing with retention of clause (1145). Different from the preceding case, both proofs contain two steps resulting from reasoning backward.

8. Proof Anomalies and Proof Conversion

The oddities and anomalies that one finds in proofs can contain clues to means for increasing program performance. Three anomalies are cited here without any suggestion regarding their significance. First, the replacement of a step resulting from reasoning backward by one or more steps resulting from reasoning forward often can require the use of paramodulation rather than the use of demodulation. Indeed, sometimes the backward reasoning step benefits by focusing on an equation that is free of variables, thus permitting demodulation to be used. When that part of the reasoning is replaced by reasoning forward from the appropriate equations, often the program must focus on equations in which variables abound, thus requiring paramodulation, for two-way unification can be required.

Before turning to the second anomaly, an example of the preceding phenomenon is in order, an example showing how a single step of backward reasoning can require as a replacement two steps of forward reasoning. The example is taken from the conversion of a backward proof to a forward proof in the context of Experiment 5 focusing on the deducibility of Moufang 3 from Moufang 1. Perhaps a thorough study of the example can show how to proceed in obtaining forward proofs in a single run, rather than obtaining backward proofs that then are transformed into forward proofs. The first set of clauses is taken from the backward proof.

```
----> UNIT CONFLICT at   2.97 sec ----> 615 [binary,614.1,10.1] $ANS(m3).
10 [] x=x.
27 [copy,26,flip.1] ((a*b)*a)*c!=a* (b* (a*c))|$ANS(m3).
29,28 [para_into,24.1.1.1.2,19.1.1,demod,22] (x*y)*x=x* (y*x).
331,330 (heat=1) [para_into,316.1.1.2,4.1.1,flip.1] rs(rs(x,1),y)=x*y.
585,584 (heat=1) [para_into,573.1.1.1.2.1,6.1.1,demod,271,396,331,526,129] (x* (y*x))*z=x* (y* (x*z)).
```

614 [para_from,330.1.2,27.1.2.2.2,demod,29,585,331] a* (b* (a*c))!=a* (b* (a*c))|$ANS(m3).

To replace the cited negative equality, clause (614), the following two-step proof was used (based on forward reasoning).

```
----> UNIT CONFLICT at   0.17 sec ----> 25 [binary,23.1,7.1] $ANS(m3).
Length of proof is 2.  Level of proof is 2.
---------------- PROOF ----------------
2 [] (x*y)*x=x* (y*x).
3 [] (x* (y*x))*z=x* (y* (x*z)).
5 [] rs(rs(x,1),y)=x*y.
7 [] ((a*b)*a)*c!=a* (b* (a*c))|$ANS(m3).
17,16 [para_into,5.1.2,2.1.2] rs(rs(x,1),y*x)= (x*y)*x.
23 [para_from,5.1.2,3.1.1.1,demod,17] ((x*y)*x)*z=x* (y* (x*z)).
```

Regarding a possible template for the choice and precise placement of the various clauses OTTER uses to produce the needed steps of forward reasoning, see the example given at the close of this section, just after the discussion of the third anomaly. However, note that one may be forced to modify the template, for example, by commenting out dynamic_demod or by adding clauses to the input list of demodulators or by taking some other (not necessarily) small action.

Regarding the second anomaly, when one closely examines a proof in which backward reasoning occurs, where the goal is to discover the obstacle to completing a proof in which only forward reasoning occurs, the following can be unearthed. The forward step merely requires the paramodulation *from* a clause *A* *into* a clause *B* to yield a clause *C* which, if nothing interferes, would complete a proof by contradiction by providing unit conflict with the negation of the conclusion or with the negation of the conclusion with its two arguments interchanged. In such an event, one might theorize that the forward proof could be obtained merely by placing in list(passive) both the negation of the conclusion and its flip, with the two arguments interchanged. When I tried an experiment of the type under discussion, the sought-after forward proof was *not* produced. Instead, I found that indeed paramodulation was applied *from A into B*, but interference took place. Specifically, before unit conflict was detected, demodulation was applied to *C*, where the demodulators are a form of associativity (whose general and well-known form does not hold in Moufang loops), followed by demodulation with *B*. To my surprise, the result of the cited demodulation was a clause that was an instance of reflexivity of equality, and was thus subsumed by $x = x$ which was present in list(usable).

For the third anomaly, which was discovered directly as a result of the just-cited experiment, one might experience something like bewilderment by trying the following. One selects an experiment reported in this article or one selects an experiment from one's own research and revisits it with one small modification, the removal of the clause $x = x$ for reflexivity. I was motivated to do so by the possibility that a forward proof would be discovered in the absence of reflexivity and its power for subsumption. My choice was to revisit one of the experiments featured in Section 6.2, the experiment in which forty-nine resonators were used in the study of proving Moufang 3 from Moufang 2, Experiment 5. Rather than rediscovering the 39-step proof that was found in the experiment, OTTER found a shorter proof, one of length 35. I do find such oddities piquant, disturbing, and, most of all, counterintuitive.

At this point, I provide an example (as promised) of how steps resulting from reasoning backward from the negation of the conclusion can be replaced by steps whose nature is forward reasoning. The focus is on the proof of Theorem 2, Experiment 5. That proof contains one step resulting from reasoning backward from the flip of the negation of the theorem. The flip of an equality is produced by OTTER by simply interchanging the lefthand and righthand arguments. The following three clauses are present in which != occurs, the third being a deduced clause.

```
25 [] a* (b* (a*c))!= ((a*b)*a)*c|$ANS(m3).
26 [copy,25,flip.1] ((a*b)*a)*c!=a* (b* (a*c))|$ANS(m3).
138 [para_into,26.1.1.1.1,57.1.2,demod,58,28] (a* (b*a))*c!=a* (b* (a*c))|$ANS(m3).
```

Clause (138) is used to complete the proof when considered with the following clause, clause (1294), to obtain unit conflict.

1294 (heat=1) [para_from,1281.1.1,3.1.1.1,demod,1087] (x* (y*x))*z=x* (y* (x*z)).

The object is to replace clause (138) with a clause resulting from reasoning forward, a clause that gives unit conflict when considered with the negation of Moufang 3 or with its flip, clauses (25) and (26), respectively.

The first step of the procedure is intended to prevent reasoning backward. Therefore, clauses (25) and (26) are placed in list(passive). Clauses in that list do not participate in the reasoning; they are used only to determine proof completion (by detecting unit conflict) and for forward subsumption. The next move is designed to have the paramodulation involve clause (1294) rather than clause (138); clause (138) must not be derived. With clause (26) in this converting procedure in list(passive) and from the facts that clause (57) was paramodulated *into* clause (26) and the result was demodulated with clause (58) and clause (28), two actions are taken. Clause (1294) is placed as the only clause in list(sos), and (the following) clauses (27) and (57) are placed in list(usable), along with the clause for reflexivity.

58,57 (heat=1) [para_into,29.1.1,9.1.1,demod,20,22,flip.1] ls(x,1)*y=x*y.
28,27 [para_into,23.1.1.1.1,19.1.1,demod,20] (x*y)*x=x* (y*x).

A pair of numbers, such as 58,57, says that clause (58) is a copy of clause (57), the former used as a demodulator, and the latter used for paramodulation (in this case). Of course, all of the history is stripped from the clauses when they are placed in the various input lists. Also, as is obvious, no need exists for placing clauses (57) and (58) both in list(usable), for they are identical; a similar remark holds for clauses (28) and (27). What is not obvious is that none of the cited clauses is placed in list(demodulators), the reason being that I conjectured that demodulation with these clauses would not be needed. (Sometimes, to find the needed forward steps, clauses may need to be inserted into the input demodulator list.)

As for options, with a few exceptions, one can use those of the experiment that produced the proof being converted to a forward proof. One can drop the use of the hot list strategy, and one is advised to avoid the use of set(process_input), thus preventing any of the input clauses from being demodulated.

What may surprise the researcher, as it did me, is the fact that clause (27) and clause (1294) were enough to produce (by applying paramodulation) the desired clause to unit conflict with clause (26).

9. Extending Purity to a Fourth Moufang Axiom

To test the possible value and scope of a new methodology, a profitable approach is to find a problem that was not in the set of problems that precipitated the study that led to the methodology. This section features such a problem, brought to my attention by Ken Kunen. In particular, he notes that focusing on three equivalent Moufang axioms is a historical accident, for a fourth (the following expressed in a notation acceptable to OTTER) is of equal interest.

% Moufang 4:
x * ((y * z) * x) = (x * y) * (z * x).

When he informed me of this fact, he also suggested that, rather than seeking circles of pure proofs, one might search for a set of pure proofs that, if found, shows that all possible orderings of the four Moufang axioms admit a corresponding circle of pure proofs. Twelve proofs are needed.

One could begin by seeking a proof that Moufang 1 implies Moufang 2 such that the proof is pure both with respect to Moufang 3 and to Moufang 4, in contrast to requiring (as was the case earlier in this article) purity with respect to Moufang 3 alone. Almost immediately upon learning of this fourth axiom, I applied the methodology used in this article to searching for the desired twelve proofs. Here I give the results and briefly detail some of the highlights.

Rather than a sequence of experiments, my choice was to attempt to go to the heart of the matter, in particular, by focusing on the fifth experiment. After all, the approach on which the fifth experiment is based had proved the most powerful, when compared with the other four elements of the sequence of five. My plan was simply to take the corresponding input file for attacking, say, Moufang 1 implies Moufang 2 and merely add demodulators to block the use of Moufang 4. For a second illustration, for the sought-after proof of Moufang 1 from Moufang 4, the plan called for taking the input file (for

Experiment 5) that includes, say, Moufang 3 as hypothesis and the negation of Moufang 1 (as the denial of the sought-after conclusion) and replacing Moufang 3 with Moufang 4 and adds two demodulators for blocking the use of Moufang 3.

In most respects, the plan worked beautifully. Indeed, ten of the twelve proofs were yielded quite readily. However, the proof that Moufang 2 implies Moufang 1 that is pure with respect to both Moufang 3 and Moufang 4 and the proof that Moufang 3 implies Moufang 1 that is pure with respect to both Moufang 2 and Moufang 4 gave trouble. The culprits had to be the two demodulators that were added to block the use (in a proof) of Moufang 4, for no other change was made from the corresponding two experiments that had succeeded (as discussed earlier in this article). One of culprits in fact caused OTTER to deduce -$T, where $T denotes **true**; hence, the empty clause was deduced, where it was not wanted.

In order to circumvent this unfortunate occurrence, $T was replaced throughout the input file by the constant junk, and a weight_list was included containing one template, the following.

weight(junk,1000).

The idea is to purge any clause containing the constant junk. Therefore, weight_list(purge_gen) was used. A smaller value than 1000 would have sufficed, as long as the value exceeded the max_weight.

The modification succeeded, and the final two proofs (of the desired twelve) were completed. In point of fact, many of the runs yielded two proofs of the sought-after result. Measured in CPU time, the most difficult to obtain required approximately 1084 CPU-seconds (on a SPARCstation-10). The theorem was that of proving Moufang 1 from Moufang 2, and it was the second proof that required the time. That proof was sought because its use made easier the task of converting the backward-reasoning fraction of a bidirectional proof into a forward proof; see Section 8 for the approach. Its length is 48, and its level is 17. Of the set of proofs that were produced, the longest consists of 83 deduced steps; its level is 23; and the theorem is that of deducing Moufang 2 from Moufang 1, the second proof. On the other hand, some proofs were produced almost immediately, having length 2 and level 2.

Summarizing, the research featured in this article suggests that use of the hot list strategy is most effective for this type of problem. Also, the use of demodulation to prevent the participation of unwanted equations succeeds. Finally, motivated by Kunen's suggestion, I applied the methodology for seeking pure proofs to the ensemble of the four Moufang axioms, yielding not only their equivalence, but in addition yielding the desired twelve pure proofs. As a corollary, all orderings of the four Moufang axioms admit a circle of pure proofs, for example, the ordering 1, 3, 2, 4, 1 and the ordering 1, 4, 2, 3, 1.

To put totally to rest any notion that, with enough effort, CPU time, and ingenuity, pure proofs can always be found, a brief review of the study of the thirteen shortest single axioms for equivalential calculus suffices. Were the parallel situation to hold for equivalential calculus, then one could select any two of the thirteen shortest single axioms and prove that the first implies the second with a proof that is pure with respect to the other eleven. However, quite the opposite is true. Specifically, although one can of course select any of the thirteen and prove any of the remaining twelve—for each by itself is a complete axiom system—for many pairs of formulas, purity of proof is often unobtainable, regardless of the effort, CPU time, and ingenuity involved. For a striking example of an impenetrable barrier, note that the first application of condensed detachment (the inference rule often used to study this area of logic) to the axiom known as *UM* with itself yields that known as *XGF*. Therefore, with *UM* as hypothesis, any sequence of deduced steps must contain *XGF* as the first step, in turn implying that only one pure proof (with respect to the other shortest single axioms) is possible, namely, that of *XGF*.

Because of the use of demodulation (which captures instances of demodulators), and because of (where needed) application of the technique of Section 8 for replacing steps of backward reasoning with steps of forward reasoning, all twelve proofs are even pure with respect to instances of the undesired equations, and none of the proofs contains hidden use of an unwanted equation; see the syntactic example given in Section 3.1 regarding a backward proof. In other words, among other properties, none of the proofs relies upon as an intermediate step (resulting from demodulation) the use of an equation to be avoided, which is where the outside-in demodulation comes into play. The presence of *instance purity* for the proofs, although not an objective, adds nicely to the result.

10. Review, Conclusions, and Future Research

Featured in this article is the problem of finding, if such exist, three *pure* proofs that together prove the equivalence of three identities known as the Moufang identities. A proof that Moufang 1 implies Moufang 2 is pure with respect to Moufang 3 if and only if Moufang 3 is not present in the proof, explicitly or implicitly.

Although Bruck supplied proofs of the equivalence of the three identities [Bruck71], an open question remained, one that (in effect) is concerned with the following flaw in the proofs. The equivalence was established by showing that Moufang 1 implies Moufang 2, 2 implies 3—and here is the flaw—rather than showing directly that 3 implies 1, so-to-speak two proofs were presented respectively showing that 3 implies 2 and that 2 implies 1. Aesthetically, in mathematics and in logic, when asked to establish the equivalence of a set of properties or definitions, one prefers a *circle of pure proofs* (as formally defined in Section 2). To answer the implied open question, one must order the three identities and then supply three proofs that, for the ordering, are each pure. For example, the ordering might be 1, 3, 2, which would then ask for a proof that 1 implies 3 that is pure with respect to 2, a proof that 3 implies 2 that is pure with respect to 1, and a proof that 2 implies 1 that is pure with respect to 3.

This article shows that both orderings admit a circle of pure proofs, namely, 1, 2, 3, and 1, 3, 2. Perhaps equally important, the method for obtaining the required proofs with substantial assistance from McCune's program OTTER is detailed. Access to the details enables one to verify the results independently and to extend the studies on which this article is based. In addition, this information provides some insight for researchers who might wish to use OTTER for attacking questions totally unrelated to those featured here. To further that possibility, the appendix offers a sample input file and three proofs (for the first circle of pure proofs). In Section 9, the method is successfully applied to the study of a fourth Moufang axiom (brought to my attention by Ken Kunen) in the context of obtaining pure proofs. The method, through the use of demodulation outside-in, produces a bonus: The proofs are free of the use of instances of the unwanted equations, even in the context of the intermediate steps resulting from demodulation. Nor do the proofs contain the hidden use of an unwanted equation (as discussed in Section 3.1). That aspect was verified by finding steps resulting from forward reasoning to replace any that resulted from reasoning backward. I thus continue the practice of acquainting researchers with the use of OTTER and its options, showing how and why to make various choices.

The two strategies that played the key role are the *set of support strategy* (see Section 5 and Section 6.1) and the *hot list strategy* (see Section 5). The first of these two strategies derives its power from restricting the reasoning, thus avoiding numerous paths of inquiry. The second derives its power from rearranging the order in which conclusions are drawn, enabling the program to draw conclusions far sooner than it would otherwise. Indeed, use of the hot list strategy permits an automated reasoning program to ''look ahead''.

Throughout this article, data is sprinkled regarding CPU time, clause retention, proof length, and the like. As one quickly discovers, anomalies occur among the data. Research that culminates in explaining the anomalies might be quite useful in the fuller context of using an automated reasoning program to solve problems in mathematics and in logic. Such research might also shed needed light on the topic of option selection. Indeed, research that succeeds in producing some effective metarules for option choosing would clearly be significant.

Among the features of OTTER covered here, features that can be put to good use in a variety of contexts, are techniques for blocking the use of unwanted equations or formulas and diverse strategies for increasing the effectiveness of n automated reasoning program. Of the two techniques for preventing the participation of an unwanted equation or formula, weighting and demodulation, the latter can be used to guarantee one's objective, where the former (though useful) admits loopholes. The picture and subtleties are discussed in some detail in Section 3.

In the context of future research, Section 6.2 focuses on the *resonance strategy*, a strategy that encourages the user of the program to supply symbol patterns (in the input) that can sharply influence the preference given to retained conclusions. The preference or priority assigned to retained conclusions directs the program in its choice of where next to focus its attention. Also of interest for future research is the *dynamic hot list strategy* (see Section 6.3), formulated by McCune as an extension of the hot list strategy. This strategy enables an automated reasoning program to adjoin new clauses to the hot list *during* the run.

Although the question central to this article was answered by finding two circles of pure proofs, many of the proofs lacked one aesthetic property. In particular, many of the proofs contained a step resulting from reasoning backward from the negation of the conclusion to be proved, and, occasionally, two such steps were present. Therefore, to satisfy the aesthetic constraint of having access to proofs all of whose steps result from forward reasoning, additional runs were needed; see Section 8. An interesting and challenging research topic focuses on finding some means to obtain in a single run the desired forward proof, for each of the six theorems that were proved in this article. Also meriting research is the study of ''short'' proofs and the study of ''elegant'' proofs; the two properties are often tightly coupled, but not necessarily. The strategies discussed in this article have proved useful for research of the type just cited.

The data presented here support various conclusions. First, quite likely, the question central to this article would still be open were it not for McCune's program OTTER. Second, of a more general character, the data given throughout this article provides substantial evidence of the power of the set of support strategy, of the power of the hot list strategy, and of the virtual requirement of using strategy when attacking deep questions and hard problems with such a program. Indeed, a glance at the CPU times with and without the strategies is most encouraging regarding the value of strategy. No longer is memory the obstacle to effective research with the assistance of a reasoning program; CPU time is the dominant concern.

Appendix

To aid and stimulate research, we present here a sample input file and the proofs of which the first circle consists. The following sample input file provides a beginning. When a line contains a ''%'', the characters from the first ''%'' to the end of the line are treated by the program as a comment. In the proofs, two copies of an input clause denotes its presence in two input lists, one of which is the hot list. Also, as an example, [para_into,24.1.1.1.2,19.1.1,demod,22] says that clause (24) is the *into clause*, clause (19) is the *from clause*, and clause (22) is used as a demodulator. The notation 24.1.1.1.2 says that the term of interest is the second subargument of the first subargument of the first argument of the first literal. When OTTER lists in a proof a pair of numbers, such as 28,27, to designate the clause number of a retained clause, the clause is used both as a demodulator and as a parent in the application of some inference rule; the higher number designates the use as a demodulator. Finally, the proof length cited by OTTER can be greater than the actual length; for example, OTTER counts for its computation of length clauses obtained by ''flip'', interchanging the two arguments of an equality.

Sample Input File

```
op(400,xfx, *).  %  make all association explicit
%  set(ur_res).
%  Following two commands prevent paramodulation involving a nonunit clause.
set(para_into_units_only).
set(para_from_units_only).
set(para_into).  %  Paramodulate into the chosen clause.
set(para_from).  %  Paramodulate from the chosen clause.
set(order_eq).  %  Flip arguments if righthand heavier than lefthand.
set(dynamic_demod).  %  Adjoin demodulators during the run.
%  set(back_demod).  %  Apply new demodulators to retained clauses.
set(lrpo).  %  Activate the LRPO ordering for orienting equalities and deciding dynamic demodulators.
set(demod_out_in).  %  Demodulate from outside to inside.
set(process_input).  Treat input clauses as if they were generated by applying an inference rule.
clear(print_kept).  %  Do not enter in the output file clauses as they are retained.
lex([$T,a,b,c,d,e,1,_*_,rs(_,_),ls(_,_),=(_,_)]).  %  Order the symbols for demodulation.
```

```
assign(max_weight,20).  %  Limit the complexity of retained clauses.
assign(max_mem,20000).  %  Limit the memory use to 20 megabytes.
assign(pick_given_ratio,3).  %  Three clauses are chosen by weight and one by breadth first, repeatedly.
assign(max_proofs,2).  %  Limit the number of completed proofs to 2.
assign(report,90).  %  Every 90 CPU seconds, write a statistical summary into the output file.
assign(heat,1).  %  Limits the recursion through the hot list.

%  Following list can be used to purge unwanted equations of various types.
%  weight_list(purge_gen).
%  Blocking use of Moufang 1.
%  weight(((x * y) * (z * x)  = (x * (y * z)) * x),1000).
%  Blocking use of Moufang 2.
%  weight((((x * y) * z) * y  = x * (y * (z * y)),1000).
%  Blocking use of Moufang 3.
%  weight((x * (y * (x * z))  = ((x * y) * x) * z),1000).
%  end_of_list.

%  Used to complete applications of inference rules.
list(usable).
x = x.

x * rs(x,y) = y.  %  right solvable
rs(x, x * y) = y.  %  right solution is unique (implies left cancellation)
ls(x,y) * y = x.  %  left solvable
ls(x * y , y) = x.  %  left solution is unique (implies right cancellation)

%  identity:
1 * x = x .
x * 1 = x.

%  left cancellation
%  x*y != u | x*z != u | y = z.
%  right cancellation
%  y*x != u | z*x != u | y = z.
end_of_list.

%  Used to initiate applications of inference rules.
list(sos).
%  A consequence of left and right surjective:
%  It's all we need here.
%  x * R(x) = 1.
%  L(x) * x = 1.
%  Following negate left and right inverse.
%  d * y != 1.
%  y * e != 1.
%  actually, L and R turn out to be the same in a Moufang loop

%  Axiom, Moufang 1:
%  (x * y) * (z * x)  = (x * (y * z)) * x.
%  Axiom, Moufang 2:
((x * y) * z) * y  = x * (y * (z * y)).
%  Axiom, Moufang 3:
%  x * (y * (x * z))  = ((x * y) * x) * z.
%  Axiom, Moufang 4:
%  x * ((y * z) * x) = (x * y) * (z * x).
```

```
% Negation Axiom, Moufang 1:
% ((a * b) * (c * a) != (a * (b * c)) * a) | $ANS(m1).
% Negation Axiom, Moufang 2:
% ((a * b) * c) * b != a * (b * (c * b)) | $ANS(m2).
% Negation Axiom, Moufang 3:
a * (b * (a * c)) != ((a * b) * a) * c | $ANS(m3).
% Negation Axiom, Moufang 4:
% a * ((b * c) * a) != (a * b) * (c * a) | $ANS(m4).
end_of_list.

% Used mainly to detect proof completion and to monitor progress.
% list(passive).
% ((a * b) * (c * a) != (a * (b * c)) * a) | $ANS(m1).
% ((a * b) * c) * b != a * (b * (c * b)) | $ANS(m2).
% a * (b * (a * c)) != ((a * b) * a) * c | $ANS(m3).
% a * ((b * c) * a) != (a * b) * (c * a) | $ANS(m4).
% end_of_list.

% Following list can be used to purge unwanted equations of various types.
list(demodulators).
% Blocking use of Moufang 1.
EQ(((x * y) * (z * x) = (x * (y * z)) * x), $T).
EQ(((x * (y * z)) * x = ((x * y) * (z * x))), $T).
% Blocking use of Moufang 2.
% EQ((((x * y) * z) * y = x * (y * (z * y))), $T).
% EQ(((x * (y * (z * y))) = (((x * y) * z) * y)), $T).
% Blocking use of Moufang 3.
% EQ((x * (y * (x * z)) = ((x * y) * x) * z), $T).
% EQ((((x * y) * x) * z = x * (y * (x * z))), $T).
% Blocking use of Moufang 4.
% EQ((x * ((y * z) * x) = (x * y) * (z * x)), $T).
% EQ(((x * y) * (z * x) = x * ((y * z) * x)), $T).
end_of_list.

list(hot).
% Axiom, Moufang 1:
% (x * y) * (z * x) = (x * (y * z)) * x.
% Axiom, Moufang 2:
((x * y) * z) * y = x * (y * (z * y)).
% Axiom, Moufang 3:
% x * (y * (x * z)) = ((x * y) * x) * z.
% Axiom, Moufang 4:
% x * ((y * z) * x) = (x * y) * (z * x).
x * rs(x,y) = y. % right solvable
rs(x, x * y) = y. % right solution is unique (implies left cancellation)
ls(x,y) * y = x. % left solvable
ls(x * y , y) = x. % left solution is unique (implies right cancellation)
% identity:
1 * x = x .
x * 1 = x.
end_of_list.
```

Three Proofs for the First Circle, in Order

Moufang 1 implies Moufang 2

```
----- Otter 3.0.3i, May 1995 -----
The job was started by wos on merlin.mcs.anl.gov, Mon Jun 19 18:50:32 1995
The command was "otter303i".

----> UNIT CONFLICT at   3.84 sec ----> 612 [binary,611.1,10.1] $ANS(m2).
Length of proof is 29.  Level of proof is 14.
---------------- PROOF ----------------
4 [] x*rs(x,y)=y.
5 [] rs(x,x*y)=y.
6 [] ls(x,y)*y=x.
7 [] ls(x*y,y)=x.
8 [] 1*x=x.
9 [] x*1=x.
10 [] x=x.
11 [] x*rs(x,y)=y.
14,13 [] rs(x,x*y)=y.
15 [] ls(x,y)*y=x.
17 [] ls(x*y,y)=x.
20,19 [] 1*x=x.
22,21 [] x*1=x.
23 [] (x*y)* (z*x)= (x* (y*z))*x.
24 [copy,23,flip.1] (x* (y*z))*x= (x*y)* (z*x).
26 [] ((a*b)*c)*b!=a* (b* (c*b))|$ANS(m2).
----------------------------
28,27 [para_into,24.1.1.1.2,19.1.1,demod,22] (x*y)*x=x* (y*x).
29 [para_into,24.1.1.1.2,15.1.1,demod,28,flip.1] (x*ls(y,z))* (z*x)=x* (y*x).
31 [para_into,24.1.1.1.2,11.1.1,demod,28,flip.1] (x*y)* (rs(y,z)*x)=x* (z*x).
35 [para_into,24.1.2.1,11.1.1,demod,28] x* ((rs(x,y)*z)*x)=y* (z*x).
41 (heat=1) [para_into,27.1.1.1,4.1.1,flip.1] x* (rs(x,y)*x)=y*x.
46,45 (heat=1) [para_from,27.1.1,7.1.1.1] ls(x* (y*x),x)=x*y.
71,70 (heat=1) [para_from,31.1.1,5.1.1.2] rs(x*y,x* (z*x))=rs(y,z)*x.
115 [para_from,24.1.1,13.1.1.2] rs(x* (y*z),(x*y)* (z*x))=x.
128,127 (heat=1) [para_into,115.1.1.2.1,9.1.1,demod,71,20] rs(x,x)*y=y.
138,137 [para_into,127.1.1,21.1.1] rs(x,x)=1.
174 [para_from,41.1.1,13.1.1.2] rs(x,y*x)=rs(x,y)*x.
200 (heat=1) [para_into,174.1.1.2,8.1.1,demod,138,flip.1] rs(x,1)*x=1.
205 [para_from,200.1.1,24.1.2.2,demod,28,22] x* ((y*rs(x,1))*x)=x*y.
209 [para_from,200.1.1,24.1.2.1,demod,28,20] rs(x,1)* ((x*y)*rs(x,1))=y*rs(x,1).
212,211 [para_from,200.1.1,17.1.1.1] ls(1,x)=rs(x,1).
219 (heat=1) [para_from,205.1.1,5.1.1.2,demod,14,flip.1] (x*rs(y,1))*y=x.
242 (heat=1) [para_from,209.1.2,7.1.1.1,demod,46] rs(x,1)* (x*y)=y.
244 [para_from,29.1.1,13.1.1.2,demod,71] rs(ls(x,y),x)*z=y*z.
246 (heat=1) [para_into,244.1.1,9.1.1,demod,22] rs(ls(x,y),x)=y.
260 [para_into,219.1.1.1.2,246.1.1,demod,212] (x*y)*rs(y,1)=x.
269,268 (heat=1) [para_into,260.1.1.1,6.1.1,flip.1] ls(x,y)=x*rs(y,1).
270 (heat=1) [para_into,260.1.1.1,4.1.1] x*rs(rs(y,x),1)=y.
314 [para_into,242.1.1.1,246.1.1,demod,269,20] x* (rs(x,1)*y)=y.
328 (heat=1) [para_into,314.1.1.2,4.1.1,flip.1] rs(rs(x,1),y)=x*y.
519 [para_from,270.1.1,35.1.1.2.1,flip.1] x* (rs(rs(y,rs(z,x)),1)*z)=z* (y*z).
524,523 [para_from,270.1.1,242.1.1.2,flip.1] rs(rs(x,y),1)=rs(y,1)*x.
526,525 (heat=1) [para_from,519.1.1,5.1.1.2,demod,524,524] rs(x,y* (z*y))= ((rs(x,1)*y)*z)*y.
611 [para_from,328.1.2,26.1.2,demod,526,524,128] ((a*b)*c)*b!= ((a*b)*c)*b|$ANS(m2).
612 [binary,611.1,10.1] $ANS(m2).
```

Moufang 2 implies Moufang 3

```
----- Otter 3.0.3i, May 1995 -----
The job was started by wos on merlin.mcs.anl.gov, Mon Jun 19 21:21:02 1995
The command was "otter303i".

----> UNIT CONFLICT at  19.71 sec ----> 1296 [binary,1294.1,138.1] $ANS(m3).
Length of proof is 51.  Level of proof is 17.
---------------- PROOF ----------------
3 [] ((x*y)*z)*y=x* (y* (z*y)).
4 [] x*rs(x,y)=y.
5 [] rs(x,x*y)=y.
6 [] ls(x,y)*y=x.
7 [] ls(x*y,y)=x.
8 [] 1*x=x.
9 [] x*1=x.
11 [] x*rs(x,y)=y.
13 [] rs(x,x*y)=y.
16,15 [] ls(x,y)*y=x.
17 [] ls(x*y,y)=x.
20,19 [] 1*x=x.
22,21 [] x*1=x.
23 [] ((x*y)*z)*y=x* (y* (z*y)).
25 [] a* (b* (a*c))!= ((a*b)*a)*c|$ANS(m3).
26 [copy,25,flip.1] ((a*b)*a)*c!=a* (b* (a*c))|$ANS(m3).
----------------------------
28,27 [para_into,23.1.1.1.1,19.1.1,demod,20] (x*y)*x=x* (y*x).
29 [para_into,23.1.1.1.1,15.1.1] (x*y)*z=ls(x,z)* (z* (y*z)).
30 [para_into,23.1.1.1.1,11.1.1] (x*y)*rs(z,x)=z* (rs(z,x)* (y*rs(z,x))).
34,33 [para_into,23.1.1.1,21.1.1,demod,20] (x*y)*y=x* (y*y).
36,35 [para_into,23.1.1.1,11.1.1,flip.1] x* (y* (rs(x*y,z)*y))=z*y.
39 (heat=1) [para_into,27.1.1.1,4.1.1,flip.1] x* (rs(x,y)*x)=y*x.
58,57 (heat=1) [para_into,29.1.1,9.1.1,demod,20,22,flip.1] ls(x,1)*y=x*y.
68 (heat=1) [para_from,29.1.2,5.1.1.2] rs(ls(x,y),(x*z)*y)=y* (z*y).
108 (heat=1) [para_from,33.1.1,5.1.1.2] rs(x*y,x* (y*y))=y.
118 [para_from,23.1.1,17.1.1.1] ls(x* (y* (z*y)),y)= (x*y)*z.
120 [para_from,23.1.1,13.1.1.2] rs((x*y)*z,x* (y* (z*y)))=y.
127,126 (heat=1) [para_into,118.1.1.1,6.1.1,flip.1] (ls(x,y* (z*y))*y)*z=ls(x,y).
134 (heat=1) [para_into,120.1.1.2,6.1.1,demod,127] rs(ls(x,y),x)=y.
136 [para_into,57.1.1,21.1.1,demod,22] ls(x,1)=x.
138 [para_into,26.1.1.1.1,57.1.2,demod,58,28] (a* (b*a))*c!=a* (b* (a*c))|$ANS(m3).
140,139 [para_from,136.1.1,134.1.1.1] rs(x,x)=1.
224 [para_from,39.1.1,29.1.2.2] (x*rs(y,z))*y=ls(x,y)* (z*y).
229 [para_from,39.1.1,13.1.1.2] rs(x,y*x)=rs(x,y)*x.
236 (heat=1) [para_into,224.1.2.2,8.1.1,demod,16] (x*rs(y,1))*y=x.
260 (heat=1) [para_into,229.1.1.2,8.1.1,demod,140,flip.1] rs(x,1)*x=1.
276 [para_into,260.1.1.1,134.1.1] x*ls(1,x)=1.
281,280 (heat=1) [para_from,276.1.1,5.1.1.2,flip.1] ls(1,x)=rs(x,1).
286 [para_from,260.1.1,23.1.1.1.1,demod,20,flip.1] rs(x,1)* (x* (y*x))=y*x.
288 [para_from,260.1.1,13.1.1.2] rs(rs(x,1),1)=x.
299 (heat=1) [para_into,286.1.1.2.2,6.1.1,demod,16] rs(x,1)* (x*y)=y.
305 [para_from,276.1.1,29.1.2.2.2,demod,281,281,22,281,16] (x*y)*rs(y,1)=x.
308,307 (heat=1) [para_into,305.1.1.1,6.1.1,flip.1] ls(x,y)=x*rs(y,1).
309 (heat=1) [para_into,305.1.1.1,4.1.1] x*rs(rs(y,x),1)=y.
339 [para_into,35.1.2,29.1.2,demod,36,308] (x*rs(y,1))* (y* (z*y))= (x*z)*y.
348,347 (heat=1) [para_into,339.1.1.2.2,6.1.1,demod,308,flip.1] (x* (y*rs(z,1)))*z= (x*rs(z,1))* (z*y).
361 [para_into,236.1.1.1,30.1.1,demod,28,348,20] x* ((rs(x,1)*rs(x,1))* (x*y))=y.
```

366,365 [para_into,236.1.1.1,27.1.1,demod,348] (rs(x,1)*rs(x,1))* (x*y)=rs(x,1)*y.
370 (heat=1) [para_from,361.1.1,5.1.1.2,demod,366] rs(x,y)=rs(x,1)*y.
392 [para_into,299.1.1.1,288.1.1] x* (rs(x,1)*y)=y.
407 [para_into,299.1.1.2,11.1.1] rs(x,1)*y=rs(x,y).
409,408 (heat=1) [para_into,392.1.1.2,4.1.1,flip.1] rs(rs(x,1),y)=x*y.
437 (heat=1) [para_from,407.1.1,7.1.1.1,demod,308] rs(x,y)*rs(y,1)=rs(x,1).
446 [para_from,299.1.1,17.1.1.1,demod,308] x*rs(y*x,1)=rs(y,1).
463 (heat=1) [para_from,446.1.1,5.1.1.2,flip.1] rs(x*y,1)=rs(y,rs(x,1)).
536,535 [para_from,309.1.1,299.1.1.2,flip.1] rs(rs(x,y),1)=rs(y,1)*x.
613 [para_from,370.1.2,33.1.2.2,demod,34,flip.1] x*rs(y,rs(y,1))=x* (rs(y,1)*rs(y,1)).
652,651 (heat=1) [para_into,613.1.1,6.1.1,demod,308,536,536,34,140,20,flip.1] (x* (y*y))* (rs(y,1)*rs(y,1))=x.
973 [para_from,437.1.1,108.1.1.1,demod,409] x* (rs(x,y)* (rs(y,1)*rs(y,1)))=rs(y,1).
1005,1004 (heat=1) [para_from,973.1.1,5.1.1.2] rs(x,rs(y,1))=rs(x,y)* (rs(y,1)*rs(y,1)).
1033,1032 [para_from,463.1.1,407.1.1.1,demod,1005,flip.1] rs(x*y,z)= (rs(y,x)* (rs(x,1)*rs(x,1)))*z.
1073,1072 [para_from,535.1.1,407.1.1.1,flip.1] rs(rs(x,y),z)= (rs(y,1)*x)*z.
1084 [para_from,535.1.2,35.1.1.2.2.1.1,demod,1073,140,20] rs(x,1)* (y* ((rs(y,x)*z)*y))=z*y.
1087,1086 (heat=1) [para_from,1084.1.1,5.1.1.2,demod,1073,140,20,flip.1] x* ((rs(x,y)*z)*x)=y* (z*x).
1281 [para_into,68.1.1.2.1,407.1.1,demod,308,1033,1005,652,1073,140,20] (x*y)* (rs(y,z)*x)=x* (z*x).
1294 (heat=1) [para_from,1281.1.1,3.1.1.1,demod,1087] (x* (y*x))*z=x* (y* (x*z)).
1296 [binary,1294.1,138.1] $ANS(m3).

Moufang 3 implies Moufang 1
----- Otter 3.0.3i, May 1995 -----
The job was started by wos on merlin.mcs.anl.gov, Tue Jun 20 08:43:56 1995
The command was "otter303i".

----> UNIT CONFLICT at 103.87 sec ----> 1861 [binary,1860.1,145.1] $ANS(m1).
Length of proof is 70. Level of proof is 22.
---------------- PROOF ----------------
4 [] x*rs(x,y)=y.
5 [] rs(x,x*y)=y.
6 [] ls(x,y)*y=x.
7 [] ls(x*y,y)=x.
8 [] 1*x=x.
9 [] x*1=x.
12,11 [] x*rs(x,y)=y.
14,13 [] rs(x,x*y)=y.
16,15 [] ls(x,y)*y=x.
18,17 [] ls(x*y,y)=x.
20,19 [] 1*x=x.
22,21 [] x*1=x.
23 [] x* (y* (x*z))= ((x*y)*x)*z.
25,24 [copy,23,flip.1] ((x*y)*x)*z=x* (y* (x*z)).
26 [] (a*b)* (c*a)!= (a* (b*c))*a|$ANS(m1).
27 [copy,26,flip.1] (a* (b*c))*a!= (a*b)* (c*a)|$ANS(m1).
----
28 [para_into,24.1.1.1.1,21.1.1,demod,20] (x*x)*y=x* (x*y).
31,30 [para_into,24.1.1.1.1,15.1.1] (x*ls(x,y))*z=ls(x,y)* (y* (ls(x,y)*z)).
32 [para_into,24.1.1.1.1,11.1.1,flip.1] x* (rs(x,y)* (x*z))= (y*x)*z.
35,34 [para_into,24.1.1,21.1.1,demod,22] (x*y)*x=x* (y*x).
36 [para_into,24.1.1,11.1.1,demod,35,flip.1] x* (y* (x*rs(x* (y*x),z)))=z.
38 (heat=1) [para_into,28.1.1,4.1.1,flip.1] x* (x*rs(x*x,y))=y.
40 (heat=1) [para_from,28.1.1,7.1.1.1] ls(x* (x*y),y)=x*x.
42 (heat=1) [para_from,28.1.1,5.1.1.2] rs(x*x,x* (x*y))=y.
46 (heat=1) [para_into,30.1.1.1,8.1.1,flip.1] ls(1,x)* (x* (ls(1,x)*y))=ls(1,x)*y.

56 (heat=1) [para_into,32.1.1.2.2,9.1.1,demod,22] x* (rs(x,y)*x)=y*x.
59,58 (heat=1) [para_into,32.1.1.2.2,8.1.1,demod,20,22] rs(1,x)*y=x*y.
62 (heat=1) [para_into,32.1.1.2.2,4.1.1] x* (rs(x,y)*z)= (y*x)*rs(x,z).
66,65 (heat=1) [para_from,32.1.1,5.1.1.2] rs(x,(y*x)*z)=rs(x,y)* (x*z).
69 (heat=1) [para_from,34.1.1,7.1.1.1] ls(x* (y*x),x)=x*y.
84,83 (heat=1) [para_from,36.1.1,5.1.1.2,flip.1] x* (y*rs(y* (x*y),z))=rs(y,z).
85 [para_from,24.1.1,17.1.1.1,demod,35] ls(x* (y* (x*z)),z)=x* (y*x).
93 (heat=1) [para_into,85.1.1.1.2.2,4.1.1] ls(x* (y*z),rs(x,z))=x* (y*x).
96,95 (heat=1) [para_into,85.1.1.1.2,6.1.1] ls(x*y,z)=x* (ls(y,x*z)*x).
115 [para_from,28.1.1,17.1.1.1,demod,96,96,35] x* (x* ((ls(y,x* (x*y))*x)*x))=x*x.
119 [para_from,28.1.2,13.1.1.2,demod,66] rs(x,x)* (x*y)=x*y.
124,123 (heat=1) [para_into,115.1.1.2.2.1.1.2.2,8.1.1,demod,20,20,22,22,20,22] ls(x,x)=1.
142,141 (heat=1) [para_into,119.1.1.2,4.1.1,demod,12] rs(x,x)*y=y.
144,143 (heat=1) [para_from,119.1.1,7.1.1.1,demod,124,flip.1] rs(x,x)=1.
145 [para_into,27.1.1.1.2,58.1.2,demod,35,59] a* ((b*c)*a)!= (a*b)* (c*a)|$ANS(m1).
146 [para_from,30.1.2,13.1.1.2,demod,66] rs(ls(x,y),x)* (ls(x,y)*z)=y* (ls(x,y)*z).
154 (heat=1) [para_from,146.1.1,7.1.1.1,demod,18,flip.1] rs(ls(x,y),x)=y.
156 [para_into,34.1.1.1,34.1.2,demod,25,flip.1] x* ((y*x)*x)=x* (y* (x*x)).
166 (heat=1) [para_from,156.1.1,5.1.1.2,demod,14,flip.1] (x*y)*y=x* (y*y).
207,206 [para_from,38.1.1,13.1.1.2,flip.1] x*rs(x*x,y)=rs(x,y).
217,216 (heat=1) [para_from,206.1.1,5.1.1.2,flip.1] rs(x*x,y)=rs(x,rs(x,y)).
241 [para_from,56.1.1,42.1.1.2.2,demod,217,14] rs(x,y*x)=rs(x,y)*x.
253 (heat=1) [para_into,241.1.1.2,8.1.1,demod,144,flip.1] rs(x,1)*x=1.
296 [para_into,253.1.1.1,154.1.1] x*ls(1,x)=1.
304 [para_from,253.1.1,32.1.1.2.2,demod,22,12,flip.1] (x*rs(y,1))*y=x.
307,306 [para_from,253.1.1,42.1.1.2.2,demod,217,14] rs(rs(x,1),1)=x.
308 [para_from,253.1.1,24.1.1.1.1,demod,20,flip.1] rs(x,1)* (x* (rs(x,1)*y))=rs(x,1)*y.
310 (heat=1) [para_into,304.1.1.1,6.1.1,flip.1] ls(x,rs(y,1))=x*y.
315,314 (heat=1) [para_from,304.1.1,7.1.1.1] ls(x,y)=x*rs(y,1).
318 (heat=1) [para_into,308.1.1.2.2,4.1.1,demod,12] rs(x,1)* (x*y)=y.
323,322 (heat=1) [para_from,308.1.1,5.1.1.2,demod,14,flip.1] x* (rs(x,1)*y)=y.
326 [para_from,296.1.1,32.1.1.2.2,demod,22,12,315,20,flip.1] (x*y)*rs(y,1)=x.
328 (heat=1) [para_into,326.1.1.1,4.1.1] x*rs(rs(y,x),1)=y.
333,332 (heat=1) [para_from,326.1.1,5.1.1.2] rs(x*y,x)=rs(y,1).
334 [para_into,46.1.1.2,36.1.1,demod,315,20,84] rs(x,1)*y=rs(x,y).
337 (heat=1) [para_from,334.1.1,7.1.1.1,demod,315] rs(x,y)*rs(y,1)=rs(x,1).
377 [para_from,310.1.2,40.1.1.1.2,demod,315,31,315,307,16] (x*y)* (rs(y,1)*x)=x*x.
384,383 (heat=1) [para_from,377.1.1,5.1.1.2] rs(x*y,x*x)=rs(y,1)*x.
466,465 [para_from,318.1.1,42.1.1.2.2,demod,217,14] rs(rs(x,1),y)=x*y.
467 [para_from,318.1.1,40.1.1.1.2,demod,315] (rs(x,1)*y)*rs(x*y,1)=rs(x,1)*rs(x,1).
472,471 [para_from,318.1.1,17.1.1.1,demod,315] x*rs(y*x,1)=rs(y,1).
478,477 (heat=1) [para_from,467.1.1,5.1.1.2,demod,384,flip.1] rs(x*y,1)=rs(y,1)*rs(x,1).
481,480 (heat=1) [para_from,471.1.1,5.1.1.2,demod,478] rs(x,rs(y,1))=rs(x,1)*rs(y,1).
483,482 [para_into,322.1.1.2,56.1.1,demod,466,flip.1] (x*y)*rs(x,1)=x* (y*rs(x,1)).
501 [para_from,322.1.1,42.1.1.2.2,demod,217,14] rs(x,y)=rs(x,1)*y.
544,543 [para_from,328.1.1,318.1.1.2,flip.1] rs(rs(x,y),1)=rs(y,1)*x.
627,626 [para_into,69.1.1.1.2,322.1.1,demod,315,483,472,483,544,142,flip.1] (rs(x,1)*y)*x=rs(x,1)* (y*x).
676 [para_into,62.1.1.2,328.1.1,demod,481,544,544,flip.1] (x*y)* (rs(y,1)* ((rs(x,1)*y)*z))=y*z.
701,700 (heat=1) [para_from,676.1.1,5.1.1.2] rs(x*y,y*z)=rs(y,1)* ((rs(x,1)*y)*z).
760 [para_from,62.1.2,42.1.1.2.2,demod,217,14,701] rs(x,1)* ((rs(y,1)*x)* (rs(x,y)*z))=rs(x,z).
793,792 (heat=1) [para_into,760.1.1.2.1,4.1.1,demod,544,466,333,323,466,flip.1] rs(x*y,z)= (rs(y,1)*rs(x,1))*z.
851,850 [para_from,166.1.1,326.1.1.1] (x* (y*y))*rs(y,1)=x*y.
884,883 [para_from,206.1.1,332.1.1.1,demod,544] rs(rs(x,y),x)=rs(y,1)* (x*x).
929,928 [para_into,241.1.1.2,38.1.1,demod,793,544,851,627,333,544,207,flip.1] (rs(x,1)* (y*y))*rs(y,x)=rs(x,1)* (y*x).
931,930 [para_into,241.1.1.2,11.1.1,demod,884,929] rs(rs(x,y),y)=rs(y,1)* (x*y).

969 [para_from,337.1.2,328.1.1.2] x* (rs(rs(y,x),z)*rs(z,1))=y.
1003,1002 [para_from,543.1.1,501.1.2.1] rs(rs(x,y),z)= (rs(y,1)*x)*z.
1856 [para_from,969.1.1,93.1.1.1,demod,315,1003,22,931,142,22,1003,flip.1] x* (((rs(x,1)*y)*z)*x)=y* (z*x).
1860 (heat=1) [para_into,1856.1.1.2.1.1,4.1.1,demod,1003,142] x* ((y*z)*x)= (x*y)* (z*x).
1861 [binary,1860.1,145.1] $ANS(m1).

References

[Boyer88] Boyer, R. S., and Moore, J S., ''A Computational Logic Handbook '', Academic Press: New York, 1988 (also Web information http://www.cli.com/s software/nqthm/obtaining.html).

[Bruck71] Bruck, R. H., *A Survey of Binary Systems*, Springer-Verlag: Berlin, 1971.

[Chein78] Chein, O. *Moufang Loops of Small Order*, *Memoirs of the American Mathematical Society* 13, issue 1, no. 197, January 1978

[Kunen94] Kunen, K., Private communication (1994).

[McCharen76] McCharen, J., Overbeek, R., and Wos, L., ''Complexity and related enhancements for automated theorem-proving programs'', *Computers and Mathematics with Applications* **2,** 1-16 (1976).

[McCune92] McCune, W., and Wos, L., ''Application of automated deduction to the search for single axioms for exponent groups'', pp. 131-136 in *Logic Programming and Automated Reasoning, Lecture Notes in Artificial Intelligence,* Vol. 624, ed. A. Voronkov, Springer-Verlag: New York, 1992.

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

[Moufang35] Moufang, R. ''Zur Struktur von Alternativkoerpern'', *Math. Ann.* **110**, 416-430 (1935).

[Wos87] Wos, L., *Automated Reasoning: 33 Basic Research Problems,* Prentice-Hall: Englewood Cliffs, N.J., 1987.

[Wos90] Wos, L., ''Meeting the challenge of fifty years of logic'', *J. Automated Reasoning* **6**, no. 2, 213-222 (1990).

[Wos92] Wos, L., Overbeek, R., Lusk, E., and Boyle, J., *Automated Reasoning: Introduction and Applications,* 2nd ed., McGraw-Hill: New York, 1992.

[Wos93] Wos, L., ''Automated reasoning answers open questions'', *Notices of the AMS* **5**, no. 1, 15-26 (January 1993).

[Wos95a] Wos, L., *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial*, accepted for publication by Academic Press (1995).

[Wos95b] Wos, L., *The Power of Combining Resonance with Heat*, *J. Automated Reasoning* (accepted for publication).

[Wos95c] Wos, L., ''The resonance strategy'', *Computers and Mathematics with Applications* **29,** no. 2, 133-178 (January 1995).

[Wos95d] Wos, L., *Searching for Circles of Pure Proofs*, *J. Automated Reasoning* (accepted for publication).