Unearthing Gold: Groups, Semigroups, and Unsolved Problems

Larry Wos

Mathematics and Computer Science Division Argonne
National Laboratory Argonne, IL 60439
wos@mcs.anl.gov

1. Wellspring, Menu, and Problems of Interest

In addition to learning about research, you will be offered here problems for your enjoyment, challenges that you may indeed find stimulating. Besides the various methodologies I shall detail, you might find useful, in attempting to meet a challenge and find a sought-after proof, R. Veroff's sketches strategy and his hints strategy.

The source, or wellspring, of a study can be a book, a paper, a remark, an e-mail, or a rumor. The source for this study was a paper, written by Sherman Stein, followed by communication with my colleague Michael Beeson. In that paper, Stein evinces an interest in short proofs. No surprise to those who are even vaguely familiar with my interests, my attention was immediately seized. What was to occur, as this note- book relates, is the (in part) discovery, or rediscovery, of needed methodology and, most important from the viewpoint of mathematics, the unearthing of gold, in the form of solutions to (perhaps) previously unsolved problems.

The various methodologies do not require the use of one specific reasoning program; indeed, they are quite general. In fact, much of what you find here regarding approaches to reaching some goal can, with effort, be accomplished by hand. I, however, clearly prefer to use a computer program, an automated rea- soning program. Known to many, but not to all, the typical way such a program knows it has reached the sought-after goal is to discover a contradiction. You, the researcher, ordinarily deny or negate the goal to be reached. If a proof is found, as you will see throughout this notebook, the last line of the proof is the sought-after result. The negated item will have a number attached to it, a number that you will usually find in the UNIT CONFLICT statement; further, that numbered item typically does not appear as a parent of one of the deduced items, nor is it considered to be the last line of the proof. To be even clearer, the negated item is placed, for OTTER (the program I use), in list(passive), a list that is used only for detecting that a proof has been completed or, in some cases, for subsumption. Although not of interest at the moment, when a clause, equation, is shown with a pair of numbers, such as 4,3, the first of the pair is a number that occurs when the item is treated as a demodulator, while the second appears when the item is used as a par- ent.

To be absolutely clear, I do not know at this time whether I will, in this notebook, offer to the reader fascinating problems that remain unsolved; however, I believe I shall offer questions whose answers I do not know. The methodology I offer may prove of immense value to the researcher in various contexts that include finding a shorter proof, finding a first proof (of, for example, a conjectured theorem), or converting one perhaps difficult-to-read proof to a proof far more transparent. You will learn of approaches that suc- ceeded in attacking various types of problem that include the following, phrased as questions, not listed in the order of importance nor in the order to be discussed.

First, if given a theorem and its proof that focuses on one area, say, groups, how can you find, if it exists, a similar theorem for a related area, say, semigroups? In that context, you may well know that

This work was supported in part by the U.S. Department of Energy, under Contract DE-AC02-06CH11357.

mathematicians often consider such a question. A natural activity for a researcher may focus on the possi- bility that a theorem can be generalized in a manner that shows its correspondent applies to a weaker the- ory. In a somewhat related fashion, another activity asks about removing axioms of a given theorem or removing axioms for a given field (variety), but without weakening the theorem or the theory. For example, more than one hundred years ago, when the field of group theory was studied, it was found that the usual set of axioms could be weakened—some could be removed—without losing any of the theorems. In other words, among the typical set of axioms, still used, dependencies exist. In particular, you will ordinarily find that a group is defined as a nonempty set with an operation, sometimes written "*", called multiplica- tion that is associative. Indeed, for any elements a, b, and c, $(ab)c = a(bc)$; also, for any two elements, their product is present in the set. In addition, in the set, the group, there exists an element, e, that is a two-sided identity, $ex = xe = x$. Further, with respect to e, for every x, there exists a two-sided inverse of x, call it y, such that $xy = yx = e$. The study of more than one hundred years ago showed that the variety, called groups, could be studied with a weaker set of axioms; you could just require that the identity be a left iden- tity, that $ex = x$, and require that every x have a left inverse, for every x there exists a y with $yx = e$. The axioms of right identity and right inverse are dependent; a similar result holds if "right" is replaced by "left". In this notebook, I shall show how various theorems can be generalized in the sense that correspon- dents can be found for theories weaker than originally in focus. The methodology, presented in detail, may enable you to extend some of your own research. In some of the sections, you will find copious detail that illustrates precisely what actions I took. You will witness how research can work and, perhaps, gain some insight into its nature.

Second, given a proof in one notation, say, a Knuth-Bendix style (to be illustrated first in Section 2), how can you find a more detailed proof whose steps are more comprehensible because of being more explicit? The idea is to begin familiarizing you with two types of proof, each of which has advantages. A Knuth-Bendix approach yields proofs in which (possibly) many steps are not explicitly present, because of the use of *demodulation*, a procedure that automatically simplifies and canonicalizes expressions. Proofs that you find in published papers and books on mathematics often have this property, although the *demodu- lators* are not typically cited. I prefer proofs in which far more is explicit, where each step, when equality (for example) is involved, can (often with difficulty) be seen to follow from its two parents that are each cited. In other words, demodulation is not used. (Demodulation does play a key role throughout this note- book, here where equality is the only relation that is studied.) I conjecture that you can learn more, and more easily, from such proofs. The inference rule that I use, and that is used by many in automated reason- ing, is *paramodulation*, to be illustrated in some detail in Section 2. (For a small taste of what you will find in this notebook, a single application of paramodulation to a given set of parents can yield a number of con- clusions because the rule is term-oriented, rather than literal-oriented.) You will also find in Section 2 two short proofs of the same theorem, one obtained with a Knuth-Bendix approach (which employs paramodu- lation and also demodulation) and one obtained with paramodulation alone. I shall also include there remarks germane to the second question, briefly discussing actions that aid you in producing a demodula- tion-free proof from a Knuth proof.

Third, with the knowledge that a given theorem has been proved with induction, how might you pro- ceed to discover a first-order proof? By way of total clarity, I do not provide a means for taking an induc- tion proof and extracting information from it, or using it, to obtain a first-order proof. I certainly prefer first-order proofs to those based on some form of induction. Among other elements, I like to see which lemmas are used in which manner.

Fourth, given a specific proof of a particular theorem, how can you find a shorter proof, perhaps far shorter? Of course, not always is it the case. But, usually, shorter proofs are more elegant. Researchers of the past clearly evinced an interest in shorter proofs, researchers that include C. A. Meredith and A. N. Prior, and more recently D. Ulrich. The methodology I detail also often applies to finding a first proof of, for example, a conjectured theorem. If you have browsed in my notebooks found on my website, automate- dreasoning.net, you are most familiar with my interest in finding shorter proofs.

In Section 3, I discuss experiments that pertain to the first question, providing some detail about gen- eralizing a result obtained, say, for groups to a result relevant to semigroups. (If you should in some way attempt to duplicate what you find here, and if you get different clause numbers, for example, the explanation rests in part with parameters and options I used and in part with other items in the input file used for the experiment in question. Input files will be given near the end of this notebook.) I start with this aspect in part because, at least for so-called simple theorems, the conversion approach is straightfor- ward. I shall first consider a simple example, discussing how you might proceed. When I eventually focused on a proof of length 830 obtained with a Knuth-Bendix approach, the task of obtaining a demodu- lation-free proof, a proof depending on paramodulation alone, offered much difficulty. You now know the type of problem in focus here, although, no doubt, other types will occasionally be discussed.

All the theorems in focus here are taken from abstract algebra, specifically, from group theory, from semigroup theory, and from varieties between these two. Equality is primarily the relation present. As many of you know, equality-oriented reasoning is often far harder to control than are other types of reason- ing, for example, (in logic) condensed detachment. As you will see, with our emphasis on automated rea- soning and the use of W. McCune’s powerful reasoning program OTTER, the inference rule in use is paramodulation, to be illustrated and discussed in some detail later in this narrative.

When equality is the relation that dominates a study, as is so typical of research concerned with areas of abstract algebra, from the viewpoint of automated reasoning two obvious approaches present themselves. In the first approach, you rely on a Knuth-Bendix approach, an approach that strongly emphasizes the use of demodulation. My research suggests that the choice of a Knuth-Bendix approach, in the beginning, increases the likelihood of finding a proof, a first proof, for the theorem under consideration. As you will see, I do sometimes use this proof to guide my and OTTER’s search for a proof more to my liking, which might mean a proof free of demodulators or a proof that is quite short. Often, in fact, I have no proof in the beginning, so a Knuth-Bendix approach can prove to be crucial.

Unfortunately, such Knuth-Bendix proofs are often difficult to follow because, at various points, four or far more demodulators are applied in a single step. Such demodulators can be found in the input (as part of the problem description), or they may be deduced by the program as it attempts to find a desired proof. On the other hand, such proofs will appear to the person who referees mathematics papers quite natural. Indeed, proofs found in published papers typically do not offer all steps explicitly.

As noted, I prefer proofs that rely on the so-called other approach, an approach that does *not* rely on Knuth-Bendix and, further, relies in no way on the use of demodulation. Instead, the type of proof I prefer relies (mainly) on paramodulation, a rule (familiar to the experienced in automated reasoning) that, as shown in Section 2, is a generalization of equality substitution. What you will find here are Knuth-Bendix proofs, proofs of the type I prefer, and methodology that enables you (with the assistance of, say, OTTER) to convert the former to the latter.

If you have little or no interest in automated reasoning, you may still enjoy much of this notebook. The reason rests with the proofs that are offered, some that may evince more beauty and charm than might be expected. Some of that beauty rests with the shortness of the proof, and some rests with the smallness of the size of the (input) axioms required. Further, as I have attempted to do in other notebooks, I shall try to convey the excitement and intrigue that can result from pursuing goals that would have been out of reach, from the viewpoint of automation, less than a decade ago. The startlingly large numbers I shall cite in the context of but one experiment provide ample evidence.

Proof length is but one measure of complexity. Complexity also can be measured in size, in the total number of symbols in the deduced items. It can also be measured in terms of the so-called longest deduced item. Still other measures of complexity exist. Researchers often seek less complex proofs. Here, you will be presented with methodology for proof shortening, sometimes dramatic proof shortening. Most pleasing, OTTER provides much assistance in obtaining short proofs or first proofs. In contrast, the task of produc- ing a demodulation-free proof from a Knuth-Bendix proof can be most daunting, as you will learn later in this notebook. You will be offered methodology for producing a demodulation-free proof when given a Knuth-Bendix proof.

I shall offer early in this notebook a short example of a proof that resists conversion. However, the conversion is not of the type just discussed. Rather, the conversion, or perhaps extension is more accurate, concerns taking a proof for group theory and producing from it a proof for semigroups, a task (for the theo- rem under study) that proved to be indeed challenging. You see, with groups you

typically have axioms that focus on an identity, denoted by *e*, whereas in semigroups these axioms are absent. In the example to be given, I believe it will be T17, I begin with a nice 13-step proof of a theorem concerning groups and seek its so-to-speak correspondent for semigroups. I was certain the task would require little more than deleting, or commenting out, all items in the input (problem description) that mentioned the identity *e*. I was wrong, as you will see. The approach that succeeded will be discussed. And if that part of this note- book provides you with less drama than you seek, perhaps the journey that begins with the discovery of an
830-step proof, not counting the occurrences of demodulation, will produce substantial wonder, especially when you see where the journey leads.

The methodologies and approaches offered here are not dependent on the areas of mathematics in focus. Rather, they are general and apply to many areas of mathematics and of logic. Since one of the activities of concern to mathematicians and to logicians is that of proving theorems, if such is your interest some of the time, you may find most useful what is offered in this notebook. You need not master paramodulation or fully understand the proofs obtained with it, for the methodologies discussed here can be applied in many contexts. For a foretaste of what is to come, one of the approaches that can lead to total success asks you, the researcher, to make a series of runs, each building on its predecessor by using so- called subproofs obtained along the way. This approach, which has been featured in various writings of mine, is called *lemma adjunction*.

Before turning to one of the appetizers for the meal focusing on research, you might share with me some amusement. As I began this notebook, I felt quite certain I would deliver to you a rather neat package with few loose ends. Well, at least at this time in May 2012, I find that some of my goals remain out of reach. In other words, many loose ends remain, challenges I now expect to leave for you.

The first appetizer illustrates a Knuth-Bendix approach, followed by the corresponding proof that is demodulation-free. The second proof to be given relies solely on paramodulation, which may offer some difficulty. However, fear not; indeed, before the two proofs are presented, I shall provide a full and detailed explanation of paramodulation. For some of you this explanation may simply be a review.

2. Knuth Proofs versus Demodulation-Free Proofs and the Nature of This Study

At this point, the journey begins, a journey that will feature various methodologies and diverse proofs obtained with them, approaches applied by McCune's OTTER, an automated reasoning program written in 1988. Perhaps to increase your excitement, I almost immediately offer you a small challenge. If the actual theorems in focus are of less interest to you, I conjecture that the approaches will still be, providing you with new ideas. For the challenge, I turn to the goal shared by the theorems to be proved.

The Stein theorems that are central to this notebook share a common goal, namely, the proof of com- mutativity, for all *x* and for all *y*, *xy = yx*. Here is your first challenge: Prove that commutativity holds in groups in which, for all elements *x*, *xx = e*, where *e* denotes the identity. The theorem you are challenged to proved was, if memory serves, published in the mid-1950s as a "classroom exercise". (For those who enjoy commentary and history, when I discussed it with my Ph.D. advisor, at the time, Reinhold Baer, he strongly advised me never to publish such a result, not deep enough.) And here you encounter a marked difference between the approach taken by an automated reasoning program and that taken by an individual. The most likely proof that you will produce will rely on instantiation, the replacement of *x* in *xx = e* by some expres- sion. Of course, I shall not say which instantiation so that the challenge is not marred. I know of no effec- tive automated reasoning program that relies on instantiation in the form just discussed, where the program decides how to instantiate; indeed, the number of instances, that might be useful, of an equation or formula is, ordinarily, far too great, and a powerful strategy for selecting among them is not available. (You will find in this notebook examples of how I used instantiation, where I picked the values to assign to various constants found in a hypothesis.) Groups with the property in focus are called *Boolean groups* or groups of exponent 2. In 1968, with "*" for product, Meredith and Prior found the following single axiom, a single equation from which you can deduce the required properties of a Boolean group.

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

(If you are now somewhat intrigued with single axioms, for groups of odd exponent *n*, with *n* greater than or equal to 3, McCune and I used OTTER to find single axioms for these varieties in the early 1990s;

further, Ken Kunen produced some excellent results in this context. As you might expect, when groups of exponent 3 are studied, you do find that they are not necessarily commutative.)

In contrast to odd-exponent groups, those of even exponent behave far less well. Kunen eventually mastered those of exponent 4, groups in which the fourth power of *x* (for every element *x*) is the identity *e*, finding single axioms. Where the function *f* denotes product, the following axiom was Kunen's first suc- cess.

f(x,f(f(x,f(f(x,x),f(y,z))),f(z,f(z,z)))) = y.

He later found two additional single axioms, the following.

f(f(x,f(f(x,x),f(f(x,y),z))),f(z,u(z,z))) = y.
f(f(x,x),f(f(x,f(f(x,y),f(z,f(z,z)))),z)) = y.

As far as I know, the questions of the existence of a short single axiom for groups of exponent 6, 8, ..., are still open. If one enjoys a quite different research challenge, one focusing on automated reasoning, the fol- lowing may prove of interest.

Research Problem. Obtain a proof with an automated reasoning program, without guidance and in a single run, establishing that the first Kunen equation is a single axiom for groups of exponent 4.

With the challenge, concerning groups of exponent 2, under consideration, or met, next in order is a treatment of paramodulation, followed by two proofs for comparison (of the same theorem). An important caveat: when paramodulation is chosen, and even when a Knuth-Bendix approach is chosen, you are strongly advised to include the clause that asserts *x* = *x*. With OTTER, you place this clause in list(usable). Its presence is recommended in part because, often, the program deduces an equation of the form *a* = *a* for some expression *a*. The theorem to be presented has a piquant relation to the theorem featured in the chal- lenge, as you will see when you view its hypothesis.

In this section, I detail the nature of paramodulation and then contrast a Knuth-Bendix approach for obtaining proofs with an approach that avoids the use of demodulation by relying exclusively on paramodu- lation. I borrow from a book I wrote many years ago, titled *Automated Reasoning and the Discovery of Missing and Elegant Proofs*. Many of you will find this inference rule unnatural; indeed, I conjecture that its use is *not* found in many published papers. Further, I doubt that many individuals be comfortable apply- ing the rule unaided by a computer. Nevertheless, its use sharply increases the power of an automated rea- soning program. The proofs yielded by applying paramodulation alone might be far from transparent. What follows may aid you in understanding such a proof. So, although familiar to some of you, here come details.

The following three examples, of graduated complexity, provide some insight into the obstacles and subtleties encountered when equality is the dominant relation. (Each of the three examples is presented, in the strictest sense, in the clause language.) The third example may in fact illustrate unexpected complexity and power even for the experienced researcher.

For the first example, equality-oriented reasoning applied to both the equation *a* + (<*a*) = 0 and the statement "*a* + (<*a*) is congruent to *b*" yields in a single step the conclusion or statement "0 is congruent to *b*". An automated reasoning program captures this straightforward bit of reasoning with the following three clauses by means of the inference rule *paramodulation*, *from* the first clause *into* the second clause, to obtain the third.

EQUAL(sum(a,minus(a)),0).
CONGRUENT(sum(a,minus(a)),b).
CONGRUENT(0,b).

Indeed, even in the presence of the unwanted seven, both axioms for the identity *e*, and both inverse axioms, the program found no proofs. Yes, for T31, I could not even touch group theory. Nothing fancy occurs in this example of a simple use of equality substitution.

For the second example, equality-oriented reasoning applied to both the equation *a* + (<*a*) = 0 and the statement "*x* + (<*a*) is congruent to *x*" yields in a single step the conclusion or statement "0 is congruent to *a*". Again, paramodulation suffices, reasoning *from* the first of the following three clauses, *into* the second, obtaining the third.

```
EQUAL(sum(a,minus(a)),0).
CONGRUENT(sum(x,minus(a)),x).
CONGRUENT(0,a).
```

This example illustrates some of the complexity of the use of equality and of paramodulation; in particular, the second occurrence of the variable x in the *into clause* becomes the constant a in the conclusion, but the term containing the first occurrence of x becomes the constant 0 in the conclusion. A small bit of thought returns the observation that the left-hand argument of the equation (first clause) can be directly applied to the left-hand argument of the congruency statement (second clause) if one merely sets the variable x to the constant a. For a program such as OTTER, all happens automatically through the use of *unification* when the focus is on the two left-hand arguments. Although the unification of the first argument of the *from clause* with the first argument of the *into clause* temporarily requires both occurrences of the variable x (in the second clause) to be replaced by the constant a, paramodulation then requires an additional term replacement justified by straightforward equality substitution, the replacement of $a + (<a)$ by 0.

Finally, for the third and complex example—an example that illustrates some perhaps unexpected subtleties—equality-oriented reasoning applied to both the equation $x + (<x) = 0$ and the equation $y + (<y + z) = z$ yields in a single step the conclusion $y + 0 = <(<y)$, a conclusion that might at first seem to be unsound. Paramodulation suffices here as well as earlier, applied to the following three clauses, *from* the first *into* the second, to logically deduce the third (again without producing any intermediate clauses).

```
EQUAL(sum(x,minus(x)),0).
EQUAL(sum(y,sum(minus(y),z)),z).
EQUAL(sum(y,0),minus(minus(y))).
```

Of course the conclusion corresponds to a well-recognized truth. But how does it follow from flawless reasoning applied to the two hypotheses?

To see that this last clause is in fact a logical consequence of its two parents, one unifies the argument sum(x,minus(x)) with the term sum(minus(y),z), applies the corresponding substitution to both the *from* and *into clauses*, and then makes the appropriate term replacement justified by the typical use of equality. The substitution found by the attempt to unify the given argument and given term requires substituting minus(y) for x and minus(minus(y)) for z. In order to prepare for the (standard) use of equality in this third exam- ple—and here you encounter a key feature of paramodulation—a nontrivial substitution for variables in both the *from* and the *into clauses* is required, which illustrates how paramodulation generalizes the usual notion of equality substitution. In contrast, in the standard use of equality substitution, a nontrivial replace- ment for variables in both the *from* and the *into* statements is not encountered.

Summarizing, a successful use of paramodulation combines in a single step the process of finding the (in an obvious sense) most general common domain (through unification) for which both the *from* and *into clauses* are relevant and applying standard equality substitution to that common domain. The complexity of this inference rule rests in part with its unnaturalness (if viewed from the type of reasoning people employ), in part with the fact that the rule is permitted to apply nontrivial variable replacement to both of the statements under consideration, and in part with the fact that different occurrences of the same expres- sion can be transformed differently. As an illustration of the third factor contributing to the complexity of paramodulation, in the last example given, the (term containing the) first occurrence of z is transformed to 0, but the second occurrence of z is transformed to minus(minus(y)). The third example illustrates what I mean when I say that, rather than literal-oriented (as is the case for hyperresolution and condensed detach- ment), paramodulation is a term-oriented inference rule that generalizes equality substitution.

A single application of paramodulation can lead to the deduction of an interesting conclusion that the usual mathematical reasoning requires more than one step to yield. In contrast to paramodulation, the mathematician or logician picks the instances—the substitutions of terms for variables—that might be of interest and then applies (the usual notion of) equality substitution. Paramodulation, on the other hand, picks the (maximal) instances of the *from* and *into clauses* automatically and—rather than deducing intermediate conclusions by applying the corresponding replacement of terms for variables—combines the instance picking with a standard application of equality substitution.

The groundwork is essentially complete for the comparison of two proofs of a theorem, one obtained with a Knuth-Bendix approach and one obtained with a demodulation-free approach relying solely on paramodulation. One of the original theorems that Stein offered, called T15, with a proof as I understand it, asserts that you can prove commutativity for groups that satisfy the following equation.

```
y*x = (x* y) ˆ n.
```

Here, *n* is a positive integer. Actually, Stein proves commutativity for semigroups, I believe. (In Section 4, you will again encounter *n*; but here, as well as in that section that concerns a theorem I call T17, you will not be treated to items, in proofs that are offered, focusing on exponents.) I found it interesting to try the case in which *n* = 2, a theorem I shall call T15a. So I asked OTTER to prove commutativity, *xy = yx* for all *x* and *y*, in the presence of *yx = (xy)(xy)*. And you see why I consider this weaker theorem to be related to the challenge in which you were asked to prove commutativity for groups in which *xx = e*, where (as a reminder) *e* is the identity.

Because of earlier research focusing on the original Stein theorem, which I shall call T15, and other studies of more of his theorems, I did not actually focus on groups. Specifically, I did not include in the input axioms concerning inverse, for all *x* there exists a two-sided inverse *y* with *xy = yx = e*. In other words, my experiment was actually for monoids, a variety (area of mathematics) between groups and semi- groups. (For semigroups, you have only associativity and, for each two elements, their product.) The pres- ence of axioms concerning an identity *e* has you studying monoids. The following segment of the input file, which will be discussed, shows you how I proceeded, ignoring for now parameters and options offered by OTTER.

```
list(usable).
x=x.
e*x = x.
x*e = x.
x* (y*z) = (x*y)*z.
end_of_list.

list(sos).
y*x = (x* y) * (x * y).
(x* y) * (x * y) = y*x.
end_of_list.

list(passive).
a*b != b*a | $ANS(thm15).
end_of_list.
```

(As you immediately note, the so-called key equation is present twice, with the second occurrence being the flip of the first; inclusion of both forms, as shown by experimentation, can aid the program in its attempt to complete a proof.) The list sos contains the elements of the problem description that are used to begin the search; the list usable contains elements used to complete inferences; the list passive contains the negation of the goal to be reached, in this case, the denial that commutativity holds. (I also place in the passive list negations of intermediate goals, items that show progress is being made if and when any of them are reached.) Perhaps you would enjoy at this point, before reading further, seeking your own proof, of course noting that "*" denotes product. The only inference rule, from among those available to OTTER, that was used was paramodulation.

Although I have not given you much time or space to complete your own proof, I shall, nevertheless, give you the proof that OTTER returned. I shall then turn to a discussion of an attempt to prove the theo- rem (T15a) with a Knuth-Bendix approach, an approach that invokes paramodulation. In the following proof, the numbers correspond to input statements, many of which (as you see from the proof) were not used, and to deduced items. Also note that, in the proofs given here, the last line is in fact the goal of the search; indeed, the inequality is used to inform the program that the target has been reached, and its number is found in the UNIT CONFLICT line.

A Proof of T15a by Paramodulation Alone

```
----> UNIT CONFLICT at   0.00 sec ----> 85 [binary,84.1,25.1] $ANS(thm15).

Length of proof is 4.  Level of proof is 4.

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

2 [] e*x=x.
3 [] x*e=x.
23 [] y*x= (x*y)*x*y.
25 [] a*b!=b*a|$ANS(thm15).
31 [para_into,23.1.1,3.1.1,flip.1] (e*x)*e*x=x.
59 [para_into,31.1.1.1,2.1.1] x*e*x=x.
75 [para_into,59.1.1.2,2.1.1] x*x=x.
84 [para_into,75.1.1,23.1.2] x*y=y*x.
```

Because the last step of the just-given proof may (understandably) present some difficulty, especially in that paramodulation is not what a mathematician usually relies upon, some discussion is now in order.

First, produce 23a by renaming the variables in 23 to get, with standard mathematical notation, *uv = (vu)*(*vu*). Second, in 75, with instantiation (term substitution), obtain 75a, (*wz*)(*wz*) = *wz*. Third, with the usual equality substitution focusing on the second argument of 23a and the first argument of 75a, temporar- ily get (*vu*)(*vu*) = *vu*, followed by the justified replacement to get *uv = vu*. For the person new to paramodu- lation or new to automated reasoning, a program such as OTTER does not actually keep such items as 23a and 75a but simply produces the desired cited result by the appropriate unification and substitution, hence, paramodulation. How interesting that associativity was not used!

Now, as promised, I turn to a Knuth-Bendix proof of the same theorem, that commutativity can be deduced from (*xy*)(*xy*) = *yx* in the presence of associativity and the two identity axioms. The following proof was obtained; for those who use OTTER, I simply set(knuth_bendix) and commented out set(para_from) and set(para_into).

A Proof of T15a with Knuth-Bendix Alone

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on
steamroller, Tue May 15 09:12:12 2012
The command was "otter".  The process ID is 16717.
----> UNIT CONFLICT at   0.01 sec ----> 56 [binary,55.1,1.1] $ANS(thm15).

Length of proof is 4.  Level of proof is 4.

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

1 [] a*b!=b*a|$ANS(thm15).
4,3 [] e*x=x.
5 [] x*e=x.
7 [] x*y*z= (x*y)*z.
9,8 [copy,7,flip.1] (x*y)*z=x*y*z.
42 [] x*y= (y*x)*y*x.
44,43 [copy,42,demod,9,flip.1] x*y*x*y=y*x.
49 [para_into,43.1.1.2.2,5.1.1,demod,4,4] x*x=x.
```

55 [para_into,49.1.1,8.1.1,demod,44] x*y=y*x.

Again, some discussion is in order.

As a reminder, when you use a Knuth-Bendix approach, certainly with OTTER, you rely for an inference rule on paramodulation, but you also rely on automatic demodulation (to canonicalize and simplify). When you see a pair of numbers, such as 9,8, it means that the corresponding equation is used by paramod- ulation with its second number and by demodulation with its first number. For but one example, right iden- tity, 5, is not used as a demodulator, but left identity, 4,3, is used both for paramodulation and for demodu- lation. Next, an examination of clause (49), especially when compared with clause (75) in the preceding proof (that did not use demodulation), shows how demodulation is automatically invoked. In particular, clause (75) in the preceding relies on clause (59) as a parent, which itself relies on clause (31) as a parent. To obtain clause (75), left identity is used twice. In order to obtain clause (49), in the Knuth proof, left identity is also used twice, but without exhibiting the intermediate results. Clause (75) in the first proof is identical to clause (49) in the second proof. The rest of the details of the just-given Knuth proof I leave to you as an exercise, if such is your inclination.

You might, in view of the perhaps obviousness of the demodulation present in the second proof, won- der more than ever why I prefer a demodulation-free proof. Well, as I noted earlier, transparency, or at least greater ease of reading, is at stake. The following item (just the history of the deduced equation), taken from a long, long proof, provides a very small taste of what can be found in a Knuth proof.

65478 [para_into,513.1.1.2,103.1.1,demod,31,26,31,26,3103,1412]

You see that six demodulators, some repeated, were used: the first four (investigation would show) are from the input, and the last two are among the deduced equations. I can assure you that, during the years with OTTER and with McCune, he and I witnessed items in a proof in which hundreds of demodulators were used. Little doubt exists concerning ease of reading and degree of clarity when you are presented with a proof some of whose steps rely on six or more demodulators. Also, as noted but not to be exhibited, the use of numerous demodulators can make far more difficult the task of locating (hidden) lemmas that are present in a proof.

To be completely explicit and answer the second of the questions posed in the beginning, in order to obtain a demodulation-free proof, especially when you have in hand a Knuth proof, you comment out (with a percent sign) the set(knuth_bendix) command and, if necessary, comment in set(para_from) and set(para_into). You can also provide aid to your program by using the deduced equations of the Knuth proof, not as lemmas, but as what are called *resonators*. A resonator is an equation or formula, often placed for OTTER in weight_list(pick_and_purge), with a assigned value that reflects its conjectured value; the smaller the number assigned the resonator, the greater the preference it is given to any equation, or formula, that matches it. The use of resonators, or of Veroff's hints, can make a huge difference in the likelihood of finding a proof and often can reduce the CPU time greatly. Also, as you will see with examples, ordinarily you might try a Knuth approach first, since this approach has, in many cases, a far better chance of finding a proof than does an approach that does not rely on demodulation. By using a Knuth approach first, you then have access, if successful, to its deduced steps to be used as resonators in your attempt to obtain a demodu- lation-free proof relying on paramodulation alone. I shall illustrate this point in Section 3, which focuses on so-called generalizations of T15a; and, in that section, I also offer a result (in the context of T15a) that startled me.

3. Generalizing a Theorem

As noted, and featured in the first question posed near the beginning of Section 1, various ways exist for generalizing a result. For example, my experiments with T15a were, in my mind, focusing on group theory. An examination of the first proof (by paramodulation alone) presented in Section 2 shows that the result holds for monoids; indeed, no mention was made of inverse in the proof. In fact, of the axioms, other than the added hypothesis that $(xy)(xy) = yx$, only those for left and right identity were used. Hence, you conclude that the theorem was proved for monoids, which implies it holds for groups. So, a natural line of inquiry asks about the correspondent to T15a holding for semigroups.

You can quickly and easily conduct the study by merely commenting out the axioms for left and right identity and, of course, (if present) those for left and right inverse, and use the amended input file. You are simply supplying to the program associativity and the hypothesis $(xy)(xy) = yx$. Although I did know at the time that the likelihood of success would be increased if I first relied on Knuth-Bendix, I instead simply sought a proof based solely on paramodulation. OTTER returned the following proof.

For Semigroups, a Proof of T15a by Paramodulation Alone

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on octagon,
Mon May 14 13:58:00 2012
The command was "otter".  The process ID is 28709.
----> UNIT CONFLICT at  84.95 sec ----> 84239 [binary,84238.1,18.1] $ANS(thm15).

Length of proof is 4.  Level of proof is 3.

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

16 [] y*x= (x*y)*x*y.
17 [] (x*y)*x*y=y*x.
18 [] a*b!=b*a|$ANS(thm15).
25 [para_into,16.1.2.1,16.1.1,flip.1] ((x*y)*x*y)*y*x=x*y.
30 [para_into,16.1.2.2,16.1.2,flip.1] ((x*y)*x*y)*y*x= (x*y)*x*y.
13640 [para_into,30.1.1,25.1.1,flip.1] (x*y)*x*y=x*y.
84238 [para_into,13640.1.1,17.1.1] x*y=y*x.
```

Here you see what might be termed an unexpected sequence of equations, for associativity is not used.

However, that axiom is used when you turn to a Knuth-Bendix approach to this theorem, as you see in the following proof.

For Semigroups a Proof of T15a by Knuth-Bendix Alone

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on octagon,
Mon May 14 14:03:07 2012
The command was "otter".  The process ID is 28761.
----> UNIT CONFLICT at   0.00 sec ----> 55 [binary,54.1,1.1] $ANS(thm15).

Length of proof is 6.  Level of proof is 6.

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

1 [] a*b!=b*a|$ANS(thm15).
3 [] x*y*z= (x*y)*z.
5,4 [copy,3,flip.1] (x*y)*z=x*y*z.
28 [] x*y= (y*x)*y*x.
30,29 [copy,28,demod,5,flip.1] x*y*x*y=y*x.
34,33 [para_into,29.1.1.2.2,4.1.1,demod,5] x*y*z*x*y*z=z*x*y.
35 [para_into,29.1.1.2,4.1.1,demod,34,5] x*y*z=z*x*y.
36 [copy,35,flip.1] x*y*z=y*z*x.
54 [para_into,36.1.1,29.1.1,demod,5,30] x*y=y*x.
```

Demodulation is used six times in this proof. Were the proof written out fully, it would be much longer than the proof preceding it, based on a demodulation-free proof. I, at this moment, draw no conclusions from the observation. I do note, as you see in the line giving the CPU time, that a Knuth-Bendix approach completed much more quickly than did the approach based solely on paramodulation.

Of the myriad experiments that presented themselves, in the context of T15a or similar theorems, I chose to see whether OTTER could prove commutativity in the presence of (*xy*)(*xy*)(*xy*) = *yx*, which I shall call theorem T15b. I therefore placed the following two equations in list(sos), reminding you that I have found it profitable in the context of using OTTER to include the flip of certain equations.

```
y*x = (x* y) * (x * y) * (x * y)  %  T15b
(x * y) * (x* y) * (x * y) = y*x.
```

Although I commented out axioms for inverse, identity, and such, I inadvertently left in items pertaining to arithmetic that will be brought into play much later in this notebook. Those so-called unwanted and extra items—unwanted in the long run because the goal was to be sure that the result focused on semigroups— did not, I am sure, interfere with nor aid the search; in fact, failure resulted. You would think that I would know better—after all, I earlier noted that a Knuth-Bendix approach has a higher probability of succeed- ing—but my preference for demodulation-free proofs, I guess, got in the way. Yes, I chose, in the experi- ment that turned out to fail, to rely solely on paramodulation. Therefore, I did turn to Knuth-Bendix, with no other changes.

Not unexpected, the Knuth approach succeeded, yielding the following proof.

A Knuth Proof of T15b

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on
octagon, Mon May 14 14:48:19 2012
The command was "otter".  The process ID is 29364.
----> UNIT CONFLICT at   0.00 sec ----> 55 [binary,54.1,1.1] $ANS(thm15).

Length of proof is 6.  Level of proof is 6.

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

1 [] a*b!=b*a|$ANS(thm15).
3 [] x*y*z= (x*y)*z.
5,4 [copy,3,flip.1] (x*y)*z=x*y*z.
28 [] x*y= (y*x)* (y*x)*y*x.
30,29 [copy,28,demod,5,5,flip.1] x*y*x*y*x*y=y*x.
34,33 [para_into,29.1.1.2.2.2.2,4.1.1,demod,5,5] x*y*z*x*y*z*x*y*z=z*x*y.
35 [para_into,29.1.1.2.2.2.2,4.1.1,demod,5,34,5] x*y*z=z*x*y.
36 [copy,35,flip.1] x*y*z=y*z*x.
54 [para_into,36.1.1,29.1.1,demod,5,5,5,30] x*y=y*x.
```

The only input axioms that are present in this proof are associativity, and its flip, and the hypothesis, and its flip. So, you now have a proof that T15b holds for semigroups. Therefore, with added axioms, the proof holds, which in turn implies that T15b holds for monoids and also for groups. Demodulation was used eleven times, mainly with associativity. The details I again leave to you.

Perhaps because I did not supply all the detail, I still did not have a demodulation-free proof of T15b. When I relied (for guidance with resonators) on the steps of the proof just given and turned to paramodula- tion alone, success was still not mine, nor OTTER's. Fortunately, OTTER offers various options, for the search, at the outermost level. So I replaced the general approach I was taking with an approach based on level saturation. (As a reminder, the input items that describe a problem are at level 0; a deduced item has level n+1 when at least one of its parents has level n.) With level saturation, regardless of the complexity of the deduced equations or formulas encountered, a program first explores level 1, then level 2, and so on. The level of the Knuth proof just given is 6, and the axiom set to be used is small; therefore, from a practi- cal viewpoint, a level-saturation approach seemed reasonable. Yes, success resulted, producing the follow- ing demodulation-free proof of T15b.

A Proof of T15b with Paramodulation Alone

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on
octagon, Mon May 14 15:59:57 2012
The command was "otter".  The process ID is 30487.
----> UNIT CONFLICT at 1019.97 sec ----> 32191 [binary,32190.1,25.1] $ANS(thm15).

Length of proof is 6.  Level of proof is 3.

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

3 [] (x*y)*z=x*y*z.
17 [] y*x= (x*y)* (x*y)*x*y.
18 [] (x*y)* (x*y)*x*y=y*x.
25 [] a*b!=b*a|$ANS(thm15).
32 [para_into,17.1.2.1,17.1.1,flip.1] ((x*y)* (x*y)*x*y)* (y*x)*y*x=x*y.
43 [para_into,17.1.2.2.2,17.1.2,flip.1] ((x*y)* (x*y)*x*y)* ((x*y)* (x*y)*x*y)*y*x= ((x*y)*x*y)*x*y.
52 [para_into,17.1.2,3.1.2,flip.1] ((x*y)*x*y)*x*y=y*x.
307 [para_into,32.1.1.2.1,18.1.2] ((x*y)* (x*y)*x*y)* ((x*y)* (x*y)*x*y)*y*x=x*y.
1629 [para_into,52.1.1,43.1.2] ((x*y)* (x*y)*x*y)* ((x*y)* (x*y)*x*y)*y*x=y*x.
32190 [para_into,1629.1.1,307.1.1] x*y=y*x.
```

This proof of T15b, avoiding the use of demodulation, clearly took longer to complete. If you experiment with a Knuth approach, or with the use of demodulation to cope with simplification and canonicalization, the time required to reach a goal is often reduced because of the removal of information not used in the resulting proof. The use of level saturation can result in an increase in CPU time before the goal is reached because the size of the levels usually grows wildly. Also, except for the conclusions that are obviously shared, commutativity, little else is shared by the sets of deduced clauses. As for the input equations that were relied upon, this second proof (free of demodulation) did not use both forms of associativity. More- over, the second proof has level 3, rather than 6; but more CPU time was required by OTTER to find the proof, even in the presence of guidance supplied by the deduced steps of the Knuth proof. Such guidance, as you will see throughout this notebook and, if you experiment with problems of the type featured here, is often vital.

At this point, you might welcome another challenge. In this case I have as yet not met the challenge by using an approach that relies solely on paramodulation. Can you find a proof of what I shall call T15c for semigroups, where the hypothesis of T15c is $(xy)(xy)(xy)(xy) = yx$, for all x and for all y, and the goal is to prove commutativity with a proof that is demodulation-free?

Next in order is a focus on the result that startled me. You will also immediately get a glimpse into the wanderings of my mind. Specifically, I decided to return to T15a, whose hypothesis is $(xy)(xy) = yx$. Although I had a proof of that theorem for semigroups, for monoids, and, as a matter of fact, for groups, I wondered what would occur if I enabled OTTER to focus on the following two axioms for inverse. After all, a group has such axioms, and the proof I had (A Proof of T15a by Paramodulation Alone) relied on the two identity axioms but did not mention inverse.

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

In these two equations respectively for left inverse and right inverse, the function *i* denotes inverse for OTTER; you could, however, have used another function if such was your preference. Relying on the two identity axioms, the axiom for left inverse and that for associativity, OTTER quickly returned the following proof that, if read with a bit of thought, leads you to a correct but odd, or perhaps surprising, conclusion.

A Proof of T15a in Group Theory with a Surprise Ending

```
----- Otter 3.3g-work, Jan 2005 -----
```

```
The process was started by wos on
steamroller, Tue May 15 09:47:35 2012
The command was "otter".  The process ID is 17260.
----> UNIT CONFLICT at   3.24 sec ----> 34811 [binary,34810.1,27.1] $ANS(thm15).

Length of proof is 12.  Level of proof is 8.

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

2 [] e*x=x.
3 [] x*e=x.
4 [] x*y*z= (x*y)*z.
6 [] i(x)*x=e.
25 [] y*x= (x*y)*x*y.
27 [] a*b!=b*a|$ANS(thm15).
31 [para_into,25.1.1,6.1.1,flip.1] (x*i(x))*x*i(x)=e.
35 [para_into,25.1.1,3.1.1,flip.1] (e*x)*e*x=x.
70 [para_into,35.1.1.1,2.1.1] x*e*x=x.
91 [para_into,70.1.1.2,2.1.1] x*x=x.
1064 [para_into,31.1.1,35.1.1] i(e)=e.
1123 [para_into,1064.1.2,6.1.2,flip.1] i(x)*x=i(e).
1142 [para_from,1064.1.2,2.1.1.1] i(e)*x=x.
2696 [para_from,1123.1.2,1142.1.1.1] (i(x)*x)*y=y.
23898 [para_into,2696.1.1,4.1.2] i(x)*x*y=y.
34655 [para_into,23898.1.1.2,91.1.1] i(x)*x=x.
34746 [para_into,34655.1.1,6.1.1] e=x.
34810 [para_into,34746.1.1,34746.1.1] x=y.
```

By reading the eleventh deduced step, you can immediately share my being startled. Indeed, it says that all elements of the group are in fact equal to the identity: The group has order 1. For the hypothesis—Stein terms such items identities—(xy)(xh) = yx to hold in the presence of the two axioms focusing on the iden- tity element e and the presence of the axiom for left inverse, the group must be the trivial group, that of order 1. You might wonder why the proof did not quit at the eleventh step. Well, for the program to know a proof has been completed, typically a contradiction must be found. The twelfth step contradicts the assumed, negation, that commutativity does not hold. You might also ask about clause (27) in that it does not appear as a parent of any of the deduced clauses. Clause (27) provides the contradiction, and it is cited in the UNIT CONFLICT line. More general remarks in this context were given in the first section of this notebook.

On the other hand, theorem T15b focuses on the cube of $xy = yx$, and, as I learned from Dean Hickerson, any direct product of cyclic groups of order 2 satisfies the hypothesis of T15b.

```
y*x = (x* y) * (x * y) * (x * y)  %  T15b
```

Further, if you are interested in a small group that does not satisfy the hypothesis of T15b, you need only consider a group of order 6, the symmetric group on three elements. That this group fails to satisfy the hypothesis in focus follows from the fact that this group is not commutative and, as proved, the fact that (from theorem T15b), groups satisfying the hypothesis of interest are commutative. As what might be thought of as an addendum, I mentioned to my colleague, John Halleck, the implication of the last proof given, that the group must have order 1, and I asked him to independently consider the result. He in turn mentioned it to his colleague, LeRoy N. Eide. You may by now have supplied a simple math proof showing that the group must have order 1. Edie's proof, quite nice indeed, can be compared with yours, if you like. As he observed, you begin with the hypothesis $(xy)(xy) = yx$ and set y equal to e the group identity. After simplifying this equation, by using axioms concerned with e, you have $xx = x$. Therefore, $yx = yxx$. Finally, using the left inverse axiom and simplifying, you have $x = e$.

To close this section, I offer you a challenge I continue to fail to meet. Stein's T15 uses the following hypothesis, with the goal of proving commutativity.

y*x = (x*y)^n.

I can take this hypothesis and prove commutativity for monoids, and hence for groups, relying for axioms on just left and right identity, and, of course, the hypothesis, and not even using associativity (or any state- ments of the number-theoretic kind to cope with the integer *n*). Your challenge is to find a proof for semi- groups, with the just-given Stein identity as hypothesis. You may be forced to formulate something about *n*, some type of arithmetic for exponentiation (as my colleague Beeson has, as shown in Section 6 when a truly difficult theorem is in focus).

I believe you might enjoy leaving T15 and its variants and turning to a new theorem, a different iden- tity from Stein. New obstacles will be encountered and new methodologies detailed.

4. Another Theorem, T17, to Prove and Additional Methodology

Stein offered a number of what he calls identities, and, for each, the goal was to prove commutativity. You could choose to focus on group theory, the theory of monoids, or the theory of semigroups. In each of my studies (central to this notebook), my eventual goal was to prove commutativity from a given identity and, eventually, do so for semigroups. Unless things change before I complete this notebook, I note that I was not always successful; indeed, sometimes I was able only to get to monoids, varieties in which *ex = xe*
= x, and not able to reach the domain of semigroups, although I often did get to semimonoids. To be totally clear, I offered you in the preceding section the challenge of focusing on T15 in its full generality with the goal of proving commutativity for semigroups. In addition to the motive of providing you stimulation and entertainment, I did so also because I cannot meet the challenge myself yet, and I would like to have in hand such a proof.

For this section, the focus is on a theorem I call T17, that of proving commutativity in the presence of the following identity.

x*y = y*x^n*y*x.

Yes, "*" denotes product, and, yes, *n* is some positive integer. I invite you on a rather long journey covered in this section. Your reward will, in part, consist of methodology not seen earlier in this notebook. Further, you will also, if your reaction matches that of my colleagues and me, experience the finding of some trea- sure. In the first stage of the journey, the goal was to find a Knuth-Bendix proof of T17 for semigroups. In the second stage, the goal was to rely on the proof found in the first stage, if such was indeed obtained, and use that proof to obtain a demodulation-free proof, relying solely on paramodulation, that holds for semi- groups. I chose this path in keeping with the observation that a Knuth-Bendix approach often succeeds where an immediate attack with paramodulation alone may fail. Also, as you will soon see, I (as is so typi- cal of these studies) began by focusing on monoids. The axioms *ex = x* and *xe = x*, where *e* denotes the identity, prove quite useful at the start of a study. I almost never began by focusing on groups (as a whole) so that I could use the axioms for left inverse and right inverse; I am not certain why this was the case.

So typical of the experiments reported here, I generally include, with the hypothesis under considera- tion, its "flip"; therefore, in the list(sos) of an input file, you would find the following.

```
y*x = x* y^n * x * y.
x* y^n * x * y = y*x.
```

In list(usable), where you find items used to complete the application of an inference rule, I usually placed the flip of associativity. Both forms of the hypothesis and of associativity contributed to efficiency.

Before the ship leaves port, you might find it interesting to note that two major paths exist for con- ducting the first stage. The goal, in the first stage, is a Knuth proof for T17, a derivation of commutativity. That proof, as noted, will be used in the second stage. The simplest, at least at first, approach is to in part emulate that which worked for versions of T15, namely, include axioms for left and right identity. And immediately you have a bifurcation. On one path, *A*, you can seek a Knuth proof that depends on

axioms involving *e* and then seek from this proof, a proof that still involves *e* but in which demodulation is absent. If successful, you can take the resulting proof and use it to then find a proof in which *e* is absent and, still, demodulation is absent. Or, on the second path, *B*, you can take a Knuth proof, if you can find such, that depends on *e*, and then try to find a second Knuth proof that does *not* involve *e* in any way. On the *B* path, you then take, if you have such, this second Knuth proof, and try to use it to find a demodulation-free proof that, of course, relies in no way on *e*. (You could, in some manner, blend the two paths.) I chose to get a Knuth proof and, after various moves, get from it a second Knuth proof free of the use of *e*. On the way, I sometimes abandoned Knuth. However, for a small taste of what is to come in the second stage, I briefly focus on the highlights that led to a Knuth proof, used in the second stage, said (by OTTER) to have length
18, but actually offering sixteen deduced steps as targets, the last of which is commutativity.

As in so many of my successful experiments, a crucial aspect concerns the resonators that are employed, equations (or formulas) that direct the program's search. The path I pursued was, to a great extent, aimed at finding resonators that would enable the program to start with the (general) hypothesis of T17 and, eventually, deduce commutativity. I say general because, in contrast to T15, I did not pause to study given values of *n*, 2, 3, and the like. Perhaps the first set of resonators, seven of them, were obtained from a Knuth proof for T17, a proof in which *e* (the monoid identity) was present. In that proof, said by the program to be of length 9, left identity, right identity, associativity, and the hypothesis each were used. Also, because of the way in which the Knuth approach works, before any deduced steps were cited, both associativity and the hypothesis were adjoined but flipped, which explains why the program said the length was 9. The second set of resonators that I found came from a run that produced three proofs of commuta- tivity, from which I chose the last, a proof of length 15. Before the fifteen deduced equations were listed, you find left identity, right identity, associativity, and the hypothesis of T17. For this run, Knuth was not used, for reasons I cannot, at the moment, fathom. Also, I relied on a level-saturation search, with set(sos_queue), for reasons that now escape me; no resonators were used in this run.

The next set of resonators, thirty in number, was obtained from a run not relying on Knuth, but not deducing commutativity either. That set of resonators was obtained by taking a set of proofs of so-called intermediate steps and sorting to remove duplicates. Focusing on intermediate steps, or targets, is a typical move I make. The targets, thirteen of them, were taken from a Knuth proof that relied, in but one step, on *e*. I cannot find, at the moment, when and how I produced this 15-step (as cited by OTTER) Knuth proof. To be patently clear, a run can serve well even when the goal, in this case commutativity, is not reached, especially when and if other targets are included in the run. Usually, such targets are negations (denials) of deduced steps in some proof.

Finally, after various moves that included "cramming" (to be explained), nineteen resonators were obtained from a proof not using Knuth but using various sets of resonators, a proof free of *e* and demodula- tion. The object of the cramming strategy, a strategy that will reappear in the second stage, is to force or cram chosen equations or formulas into a sought-after proof. An effective way to try to do so is to use a level-saturation approach while placing in list(sos) the items you wish to cram into a proof. For the curi- ous, one of the powerful uses of this strategy concerns the attempt to shorten a proof of, say, the join of three goals, *F*, *G*, and *H*. One approach has you choose a proof, one you like, of one of the three goals, and place its deduced steps in list(sos) with the intention of proving all three in such a manner that the proof steps of the chosen goal play double duty or more. You will see this strategy used later, not in the context of proving a join but, rather, in the context of proving a single item.

So, summarizing, at this point I had an 18-step proof, sixteen of whose steps were deduced equations. The proof relied on demodulation, the proof was free of occurrences of *e*, and the port had been reached. The next step of the voyage was to aim at a demodulation-free proof, still free of *e*. Therefore, next in order is a treatment of the second stage.

The 18-step proof to be used in the second stage, actually sixteen of its (deduced) equations as res- onators, took less than 1 CPU-second to be found. I strongly suspected that the experiments or runs that would be needed to complete the second stage, to return a demodulation-free proof of T17 for semigroups, would each require more computer time, perhaps far, far more. Indeed, in the Knuth proof I had in hand, although it had the pleasing property of applying to semigroups and, therefore, using only associativity and the hypothesis of T17, relied on sixteen demodulators. If all the demodulators were among the input clauses, then, perhaps, I could have used the *hot list strategy* or the *dynamic hot list strategy* to shorten this part of the journey; each will be detailed later. (McCune merits full credit for generalizing my original notion of the hot list strategy, in two ways: I had thought of it in terms of paramodulation, and he general- ized it to apply to all inference rules; and my version was static in that the contents of the hot list were unchanged during the run, whereas he added the dynamic version in which the program can amend the hot list.) But, as you see in the Knuth proof to be used, a proof I now exhibit, some of the demodulation focuses on deduced items.

An 18-Step Knuth Proof of T17 for Semigroups

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Sat Mar 31 08:10:39 2012
The command was "otter". The process ID is 15660.
----> UNIT CONFLICT at   0.10 sec ----> 681 [binary,680.1,14.1] $ANS(thm17).

Length of proof is 18.  Level of proof is 9.

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

14 [] a*b!=b*a|$ANS(thm17).
16 [] x*y*z= (x*y)*z.
18,17 [copy,16,flip.1] (x*y)*z=x*y*z.
45 [] x*y=y*x^n*y*x.
47,46 [copy,45,flip.1] x*y^n*x*y=y*x.
53,52 [para_into,46.1.1.2.2,17.1.1,demod,18] x*y*z^n*x*y*z=z*x*y.
54 [para_from,46.1.1,17.1.1.1,demod,18,18,18,flip.1]  x*y^n*x*y*z=y*x*z.
85 [para_into,54.1.1.2.2.2,54.1.1] x*y^n*x*z*y*u=y*x*z^n*y*z*u.
86 [para_into,54.1.1.2.2.2,52.1.1] x*y^n*x*z*y*u=y*x*u*z^n*y*u*z.
92 [para_into,54.1.1.2.2,52.1.1] x*y^n*z*x*y=y*x*z^n*x*y*z.
99 [copy,85,flip.1] x*y*z^n*x*z*u=y*x^n*y*z*x*u.
104 [copy,92,flip.1] x*y*z^n*y*x*z=y*x^n*z*y*x.
355 [para_into,86.1.1,54.1.1,flip.1]  x*y*z*x^n*x*z*x=x*y*x*z.
511,510 [para_into,99.1.1.2,92.1.1,demod,53,flip.1]  x*y^n*x*x*y*x=y*y*x*x.
518 [para_into,99.1.1,52.1.1,demod,511] x*y*x=y*y*x*x.
523 [copy,518,flip.1] x*x*y*y=y*x*y.
664,663 [para_into,104.1.1.2,54.1.1,flip.1]  x*y^n*y*x*y=y*y*x*y.
670,669 [back_demod,355,demod,664] x*y*x*x*z*x=x*y*x*z.
672,671 [back_demod,510,demod,670,47,flip.1]  x*x*y*y=x*y.
674,673 [back_demod,523,demod,672,flip.1]  x*y*x=y*x.
680 [back_demod,663,demod,674,47,674,674] x*y=y*x.
```

My main targets, as the second stage was begun, were the negations (denials) of the sixteen deduced equa- tions in the just-cited Knuth proof. To initiate the attack, I relied on the hypothesis of T17 and its flip, both placed (for OTTER users) in list(sos). To complete applications of paramodulation, the only inference rule to be used, I relied on associativity and its flip, placed in list(usable). Finally, being certain that the pro- gram would need substantial direction in its search, I included sixteen resonators, each corresponding to one of the sixteen deduced equations of the Knuth proof and (again, for OTTER users) placed in weight_list(pick_and_purge).

Two words of advice: At least with OTTER, you can often profitably include the flip of each nega- tion along with the negation, and (with OTTER especially) the CPU time will be reduced if you do not use *ancestor subsumption*, a procedure that aids you in finding shorter proofs (to be detailed). On the other hand, if you include set(ancestor_subsume), then also include set(back_sub). To make things simpler for me to quote from the actual short trip, which focuses on the first experiment, let the sixteen denials be

numbered 7 through 22, the last being the denial of commutativity (a*b != b*a). Proofs were found for 7,
8, 9, 13, 15, 17, and 18, not in that order; in other words, of the sixteen, in this first try, the program proved items 1, 2, 3, 7, 9, 11, and 12. The seven proofs were completed in less than 6 CPU-seconds; more time was allowed, but nothing of interest resulted.

The next run was based, as you shall see, on the preceding, and it proved, from the original sixteen, items 6, 8, and 13, but not in that order. If you pause immediately, you might ask yourself how the run I am about to describe could build on its predecessor. Perhaps you will formulate a method I did not use; if so, I would enjoy learning about it in detail from you.

I did not, as might be expected, amend the set of resonators; instead, I continued to rely on the same sixteen. However, I did put to use the results of the preceding run by extracting the deduced steps of the proofs that were completed in that run. I then turned to the cramming strategy, an approach that has you adjoin to the list(sos) of the preceding run the proof steps after sorting to remove duplicates, in this case, fifteen equations. I also now invoked a level-saturation approach. Yes, I was hoping to force, cram, the new steps, or at least some of them, into proofs not yet in hand, proofs of some of the sixteen targets and, if luck was present, perhaps even commutativity. As I noted, commutativity still eluded me; but, if measured by the number of items proved from the sixteen, I was getting closer. However, from this run, I did extract five resonators, correspondents of proof steps of the three proved, from among the sixteen targets.

The third move (run) found OTTER proving items 14 and 15, of the sixteen targets. I again relied on cramming, adjoining in the last cited five proof steps to list(sos). I also used three sets of resonators, the first set consisting of the already-discussed sixteen, the second set consisting of the already-mentioned fif- teen corresponding to proof steps from the first run, and the third set consisting of five corresponding to the just-cited five proof steps (resulting from the second run). Again, level saturation was the choice. But I still had not proved commutativity.

And now you can share my problem; indeed, I had, after approximately 307 CPU-seconds, in the third run a key equation, namely, x*y*x=y*x. That run also had access to two other key equations, namely, x*y^n*y*x*y=y*y*x*y and x*y^n*x*y=y*x. What, I wondered, prevented OTTER from using the cited three equations to deduce commutativity? After all—and yes, I did look carefully at the Knuth 18-step proof—the three equations, used appropriately, enabled OTTER, with Knuth, to deduce commutativity. You can, and may enjoy doing so, verify this statement by re-examining the 18-step Knuth proof of T17 for semigroups I presented earlier. You will find that the last two equations were used as demodulators on the equation cited just before they were cited. What could I do to get OTTER to give me enough so that, using what it found, a proof of commutativity could be completed?

To detail what worked, I shall introduce you to the *hot list strategy*, which some of you may already know about. The hot list strategy (which is somewhat reminiscent of what is found in proofs in logic and in mathematics) instructs the program to immediately visit, revisit, and the like user-chosen items among those that present the theorem to be proved. The user-chosen items are placed, at least for OTTER, in list(hot). The degree to which the so-called revisiting occurs is governed by the *heat* parameter, for exam- ple, assign(heat,4). When the program decides to retain a new conclusion, *before* another conclusion is chosen as the focus of attention to drive the program's reasoning, the hot list strategy causes the new con- clusion to initiate applications of the inference rules in use, with the remaining hypotheses (or parents) that are needed all chosen from the hot list.

The action I took to find additional equations that would enable me, once I put all my work together, was to turn to the hot list strategy. I placed on the hot list the three equations in focus and also placed them in list(sos). The program was then, in effect, instructed to consider, from list(sos), each of its members and use, when a conclusion was deduced and retained, members of the hot list. I assigned the value 4 to the heat parameter, not knowing ahead of time precisely what path the program would take. I also relied on a level-saturation approach. Not only did I get some useful information, as the proof I now offer you shows; I found what was needed to reach commutativity.

A Useful Proof Segment

```
----- Otter 3.3g-work, Jan 2005 -----

The process was started by wos on vanquish,
Mon May 21 19:37:11 2012
The command was "otter".  The process ID is 5455.
----> UNIT CONFLICT at 1209.57 sec ----> 151739 [binary,151738.1,21.1] $ANS(thm17).

Length of proof is 8.  Level of proof is 5.

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

1 [] x*y^n*y*x*y=y*y*x*y.
21 [] a*b!=b*a|$ANS(thm17).
22 [] x*y^n*y*x*y=y*y*x*y.
23 [] x*y*x=y*x.
24 [] x*y^n*x*y=y*x.
25 [para_into,1.1.1,1.1.1] x*x*y*x=x*x*y*x.
26 [para_into,1.1.2,1.1.2] x*y^n*y*x*y=x*y^n*y*x*y.
32 (heat=1) [para_into,25.1.1.2,23.1.1,flip.1] x*x*y*x=x*y*x.
37 (heat=1) [para_into,26.1.1.2.2,23.1.1,flip.1] x*y^n*y*x*y=x*y^n*x*y.
130 (heat=2) [para_into,32.1.1,22.1.2] x*y^n*y*x*y=y*x*y.
171 (heat=2) [para_into,37.1.2,24.1.1] x*y^n*y*x*y=y*x.
867 (heat=3) [para_into,130.1.2,23.1.1] x*y^n*y*x*y=x*y.
151738 [para_into,867.1.1,171.1.1] x*y=y*x.
```

Before visiting the final port, some examination of the just-given proof is merited. You see that the needed portion was not obtained immediately. You find that eight steps were needed, and you find that commutativity itself was proved. As for the eight deduced steps, you learn how much the hot list strategy was used, to the point that one of the eight asserts heat=3, which says how much revisiting occurred. As for the items listed before the deduced items, you find equations listed twice, for each, once because it was chosen as the focus of attention to drive the reasoning, and once because of its use and appearance in the hot list. In order that you can, if you wish, participate in the trip, I ask you the following questions. What do you conjecture was done with the eight-step proof? Further, what actions do you suppose were taken to enable the program to begin with associativity and the hypothesis of T17 and search until commutativity was deduced?

I relied heavily on the resonance strategy, including five sets of resonators. The first set was the now- familiar fifteen; the second set was the described five from cramming on the fifteen; the third set was the six from cramming on the preceding two sets; the fourth set, which is where the 8-step proof comes into play, consisted of the eight correspondents to the proof segment just cited; and the fifth and last set was the six- teen taken from the Knuth proof in stage 1. I abandoned the hot list strategy and the use of level saturation. In just over 526 CPU-seconds, the program produced the following 31-step proof of commutativity for semigroups.

A 31-Step Proof of T17 by Paramodulation Alone

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Mon May 21 21:00:22 2012
The command was "otter".  The process ID is 8656.
----> UNIT CONFLICT at 526.14 sec ----> 120491 [binary,120490.1,24.1] $ANS(thm17).

Length of proof is 31.  Level of proof is 11.

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

3 [] (x*y)*z=x*y*z.
```

```
4 [] y*x=x*y^n*x*y.
5 [] x*y^n*x*y=y*x.
24 [] a*b!=b*a|$ANS(thm17).
27 [para_into,4.1.1,3.1.1,flip.1] x* (y*z)^n*x*y*z=y*z*x.
31 [para_into,4.1.2.2.2,3.1.1,flip.1] (x*y)*z^n*x*y*z=z*x*y.
36 [para_from,4.1.1,3.1.1.1] (x*y^n*x*y)*z=y*x*z.
37 [para_from,4.1.1,3.1.2.2,flip.1] x*y*z^n*y*z= (x*z)*y.
106 [para_into,31.1.1,3.1.1] x*y*z^n*x*y*z=z*x*y.
162 [para_into,36.1.1,3.1.1] x* (y^n*x*y)*z=y*x*z.
228 [para_into,37.1.1.2,27.1.1,flip.1] (x*y*z)*u=x*y*z*u.
606 [para_from,228.1.1,162.1.1.2] x*y^n*x*y*z=y*x*z.
626 [para_into,606.1.1.2.2.2,606.1.1] x*y^n*x*z*y*u=y*x*z^n*y*z*u.
632 [para_into,606.1.1.2.2.2,106.1.1,flip.1] x*y*z*u^n*x*z*u=y*x^n*y*u*x*z.
653 [para_into,606.1.1.2.2,106.1.1,flip.1] x*y*z^n*y*x*z=y*x^n*z*y*x.
833 [para_into,626.1.1,3.1.2] (x*y^n)*x*z*y*u=y*x*z^n*y*z*u.
862 [para_into,626.1.2.2,606.1.1] x*x^n*x*y*x*z=x*y*x*z.
874 [para_into,626.1.2,106.1.1] x*y^n*x*x*y*x=x*y*x.
952 [para_into,632.1.2,606.1.1] x*y*z*x^n*x*z*x=x*y*x*z.
1000 [para_into,653.1.1.2,606.1.1,flip.1] x*y^n*y*x*y=y*y*x*y.
1016 [para_into,653.1.1,106.1.1,flip.1] x*x^n*y*x*x=y*x*x.
1785 [para_from,833.1.1,3.1.1] x*y*z^n*x*z*u=y*x^n*y*z*x*u.
1856 [para_into,862.1.1.2.2,606.1.1] x*x^n*y*x*z=x*y^n*x*y*z.
2403 [para_into,952.1.2,5.1.1] x*y^n*y*x^n*x*y*x=y*x.
2781 [para_from,1016.1.1,626.1.2.2] x*y^n*x*x*y*x=y*y*x*x.
4221 [para_into,1856.1.2,606.1.1] x*x^n*y*x*z=y*x*z.
4453 [para_into,2403.1.1.2.2,1000.1.1] x*y^n*x*x*y*x=y*x.
6143 [para_into,2781.1.1.2.2,1000.1.2] x*y^n*y*x^n*x*y*x=y*y*x*x.
12300 [para_from,4221.1.1,1785.1.1.2,flip.1] x*y^n*x*x*y*z=y*y*x*z.
15826 [para_into,4453.1.1,874.1.1] x*y*x=y*x.
18132 [para_into,6143.1.1,2403.1.1,flip.1] x*x*y*y=x*y.
28370 [para_into,15826.1.1.2,15826.1.2] x*x*y*x=y*x.
31189 [para_from,18132.1.2,4.1.2.2.2,flip.1] x*y^n*x*x*y*y=y*x.
43951 [para_into,28370.1.1,12300.1.2] x*y^n*x*x*y*y=x*y.
120490 [para_into,43951.1.1,31189.1.1] x*y=y*x.
```

Of the sixteen deduced steps of the Knuth proof found in stage 1, only eleven of them are found among the thirty-one of the completed proof of T17.

By now, you may be wondering what the point was with the positive integer *n* in the hypothesis of T17. Well, and here is the charming property of the proof OTTER eventually presented to me, the 31-step proof: no equations that precede those obtained with paramodulation mention *n*, except, of course, (a flip of) the hypothesis of T17. Indeed, no axioms regarding exponentiation were required. Now, is that not charming—and even startling? Further, I have in hand a 21-step proof of T17; it too uses no information concerning exponents, merely relying on the hypothesis and associativity. So, in an odd sense, semigroups benefit again, if commutativity is the prime concern. (You will see later that I was not always able to go as deep as semigroups, an observation that will become totally clear.)

Since the trip has ended, a new topic demands attention, the topic of proof shortening. Proof shorten- ing addresses one of the important aspects of proof simplification, of finding proofs of less complexity. As I noted in the beginning of this notebook, my attention was captured when I learned that Stein himself was interested in shorter proofs. The discussion of proof shortening will bring to some of you a myth that could easily be believed.

5. Proof Shortening

The problem central to this section asks for approaches that can be used to find so-called short proofs. Of course, the notion of short is relative. Indeed, if you are given by some researcher a proof of, say, 135 (deduced) steps, and if you eventually return to that researcher a proof of length 50, much pleasure is ordinarily experienced. Yes, I have done so in answer to a request by McCune in the context of lattice theory, of course, with invaluable aid from McCune's OTTER. If, as will be discussed later in this note- book, you on your own find a proof of length 830, and if that length does not count the numerous demodu- lators that are present (in that Knuth-Bendix) proof, you consider it a success when and if you eventually find a proof of length 200—or less. So, the problem I pose for you—not with total fairness in that I have studied it for decades—asks for methodologies for proof shortening. While you invent or formulate possi- ble solutions, I shall pause for a bit of history and for the discussion on an understandable myth, but, indeed, a myth.

Decades ago, McCune and I often worked together on projects, some in areas of logic, some in areas of mathematics. If memory serves, my interest in proof shortening began with theorems of Lukasiewicz (which McCune and I were both studying in the context of automated reasoning), in the area of infinite-val- ued (or many-valued) sentential calculus. When McCune learned of my interest in finding shorter proofs, specifically, for a theorem in the cited area, he again—as he did so often—amended what was then the ver- sion of OTTER, today in 2012 offering far more than in the early 1990s. Specifically, McCune formulated *ancestor subsumption*. Ancestor subsumption is a procedure that compares derivation lengths to the same conclusion and prefers the strictly shorter. To use this splendid procedure, with OTTER, you include set(ancestor_subsume). The use of this command is the simplest and most direct way of finding shorter and still shorter proofs, as the following will show. But first, you might enjoy considering a myth that exists for new researchers, one that, with an example, I shall dispel.

To set the stage, first imagine that you are asked to provide a proof from the hypothesis *H* of the join, **and**, of three conditions, *A*, *B*, and *C*. Next imagine that you are presented with two proofs, **P** and **Q**, of respective lengths, say, forty-five and fifty-two. But, in the beginning, you do not know the respective lengths. Instead, you merely know that, within **P**, the length of the proof of *B* is, say, twenty-three and, within **Q**, the length of the proof of *B* is eighteen. You could easily quickly guess that the length of **P** is greater than the length of **Q**. Clearly, a reasonable guess! In effect, to obtain **P** from **Q**, the first step is to replace the 18-step proof of *B* by a 23-step proof. But, then, why is **P**, eventually, shorter than **Q**? Well, the cited replacement next allows some of the new steps, among the twenty-three, to be used to lead to dif- ferent proofs of one or both of *A* and *C*. The newer proofs of each of the two, since they share enough steps with the proof of *B*, permit the completion of a proof **P** that is five steps shorter than is **Q**. You see that finding a shorter proof than that in hand may not always be straightforward. In particular, with the case just discussed as an example, you cannot simply seek a shorter proof of *B* and proceed further.

Unfortunately and not totally obvious, the aphorism "shorter subproofs do not necessarily a shorter total proof make" nicely summarizes the complexity of seeking proof abridgment. Indeed, for another small taste of what can go wrong (that illustrates again what has just been exemplified), even when the goal is a single item, such as commutativity, consider the following. Let **P** be a 30-step proof of the theorem **T**; let *A* be the twentieth step; let the proof **S** of *A* within **P** consist of various axioms and the fifteenth through the twentieth step inclusive. The length of the proof **S** is 6, and **S** is a subproof of **P**. Now let **R** be a proof of *A* of length 5 such that the only deduced step present in both **R** and **S** is *A*. If all the steps of **S** are used as parents for steps later than *A* in **P**, and if the first four steps of **R** are of no use in this context, then replacing **S** by **R** will in many cases require for a proof of **T** a sequence of steps whose length is strictly greater than 30. You thus see how the aphorism applies, whether the shorter subproof is obtained by hand or by a program through the use of ancestor subsumption; indeed, the obvious shortcut to proof shortening often fails. And now to yet another new theorem.

The new hypothesis, for a theorem I call T41x4, (and its flip), is expressed this way.

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

Again, the object of T41 is to prove, from the given hypothesis in the presence of associativity alone,

commutativity; in other words, the area in focus is semigroups. For this section, I located the input file I had used a month earlier that led to the 28-step proof and amended it by using ancestor subsumption. I allowed the program to use as much CPU time as it desired. The program found seven proofs of respective lengths 22, 21, 20, 19, 18, 17, and 16. The first of the seven was found in less than 4 CPU-seconds, and the last was found in a bit more than 1625 CPU-seconds. No new resonators were adjoined; indeed, this run relied on various sets of resonators, some extracted from proofs in which the identity element *e* was present. Of note, and of some amusement for me, the 16-step proof contains eight equations not present in the
28-step proof.

Perhaps by now you wonder about other approaches to proof shortening. Indeed, I shall immediately discuss a sharply different approach. The theorem to be studied is closely related to T41x4, and I shall call it T41x6; its hypothesis and flip are the following.

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

A comparison of the hypothesis of T41x6 with that of T41x4 discloses the naming convention, x6 versus x4. The experiment I now describe, with the goal of proving commutativity but, more important, of illustrating a radically different approach to proof shortening, began with the input file that was used in an ear- lier study of T41x6, conducted on April 25, 2012. On that day, OTTER produced a 38-step proof for T41x6. That file did not employ ancestor subsumption, nor did my current experiments (of May 25, 2012). I modified the input file in two ways. I added list(demodulators), whose use will become clear almost immediately, and I added weight(junk,1000), which will also be quickly explained. Although I originally (in the 1960s) formulated demodulation to increase the efficiency of what were then called automated theo- rem-proving programs by canonicalizing and simplifying deductions, I discovered an odd way to use demodulators. Specifically, demodulation could be used for information the researcher classed as unwanted. The following example illustrates this use.

```
EQ(x*y*x*y=x*y,junk).
```

If and when the cited equation, contained in the EQ clauses, is deduced, it will be demodulated to “junk”. Then, as promised regarding the cited weight template, you also include in weight_list(pick_and_purge) the following item to give expressions involving junk a high weight, higher that the maximum the user has assigned.

```
weight(junk,1000).
```

When the program combines the effects of such a demodulator with such a weight template, the cited equa- tion is purged, and it cannot therefore participate in a proof.

Finally, with the preceding, you may have guessed what actions you can take for finding a proof shorter than that in hand. For example, I took an input file that yielded a 38-step proof of T41x6, with just the emendations of the cited weight template and a demodulator list. I used a program—again, written as a courtesy for me by McCune when I informed him of what I could do by hand—that allowed me to apply an iterative approach. I had this program, otter-loop, block one at a time the thirty-eight steps of the proof, by demodulating each to junk, to see what would happen. If progress occurred, I amended the file with the demodulator that caused progress, and repeated the process. With otter-loop, and with iteration, I found proofs of successive lengths 33, 31, and 30. The 30-step proof contains four equations not present in the (original) 38-step proof. The following three demodulators were used in reverse order.

```
EQ((x*x)*x*x=      (x*x)*x,junk).
EQ(x*   (x*x)*y=   (x*x)*y,junk).
EQ(x*y*x*y=x*y,junk).
```

Of course, with this demodulation-blocking procedure, you can paint yourself into a corner. In particular, using a different set of demodulators could and probably would produce a different result. No, I did not try the huge number of experiments that present themselves. However, and this aspect might have just occurred to you, I took the file that yielded the 30-step proof and modified it in one small way. Have you surmised what that is? Yes, I added the use of ancestor subsumption. The program then found a

23-step proof of T41x6. Statistics can provide such entertainment. The 23-step proof contains for equations not in the 30-step proof, but an oddity occurs (if I have not lost my mind). Compared with the 38-step proof, the

23-step proof has just two equations that are new. At this point, you must wish for more food for thought, for new challenges. So, I shall oblige and, in doing so, supply evidence of what huge numbers can be encountered, numbers that a computer program copes well with, but, I venture, an individual might find overwhelming

I have in my collection of proofs a proof for T41x6 of length 19. Now, if you wish to seek such, you may find a path that involves smaller numbers than I show in the following line that quotes the CPU-seconds and clause numbers that were involved.

```
----> UNIT CONFLICT at 101255.62 sec ----> 4334346 [binary,4334345.1,25.1] $ANS(thm41x6). If
```

you find variations on the T41 theme, I suggest you start with the following simpler case, T41x2.

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

You might then turn to T41x3, which, in my studies, proved quite a challenge, with the following hypothesis, still to prove commutativity.

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

Of course, I have tackled, and won, for T41x4, the following.

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

I have, for now, not gone beyond T41x6, but I am rather certain all will proceed in a fine manner, although much CPU time may be required. Finally, I leave T41-type problems with one last (so-to-speak) gasp, especially since I thrill to large numbers. The case in question is T41x5; the following line tells quite a story.

```
----> UNIT CONFLICT at 661696.84 sec ----> 8207454 [binary,8207453.1,34.1] $ANS(thm41x5x5).
```

6. A Most Difficult Theorem to Prove

The question of finding a first-order proof is now in focus. Many theorems are proved by induction. But I, in particular, prefer first-order proofs. With that type of proof, you can more easily learn from it, can see, for example, which lemmas play what role in the proof. The program I use, OTTER, provides little or no assistance in finding proofs by induction; however, as many years of experimentation have shown, this program is often invaluable in finding a first-order proof. The theorem T33 was proved by Stein with induction. In this section, you see how it was proved in a first-order manner.

Now when an integer, such as *n* or *k*, is present in the hypothesis of a theorem, you might naturally suspect that induction is in order. Thanks to my colleague Beeson, for the theorem to be studied here, T33, appropriate axioms to cope with exponentiation are in hand. His formulation, for exponentiation, is the first I have seen for automated reasoning. With them, I shall present a study of T33, a study that resulted in first-order proofs rather than proofs by induction. Indeed, this section in effect answers the third question posed in Section 1. The proofs Stein presents in general rely on induction. As T33 is considered, you will again visit proof shortening and again visit proof conversion from Knuth-Bendix to paramodulation alone.

The goal, as is the case throughout this notebook, is to prove commutativity. The hypothesis of T33, expressed as a clause, is the following.

```
y*x = x^ (3 o k) * y^ (3 o k) * x^ (3 o k) * y^ (3 o k).
```

The meaning of the hypothesis will become clear as I almost immediately cite from an e-mail sent to me by Beeson. But, before doing so, I note that you need not master his material on exponentiation to follow and learn from the approach I detail. In particular, the elements of the methodology to be offered apply to a wide variety of problems you may encounter, and an understanding of how to treat exponents is not needed to see how proof conversion occurred and more. Therefore, if you are so inclined, you might prefer to browse quickly through the next long paragraph until you find the methodology that was employed.

Stein's theorems make use of exponential notation, where *x*n means the product of *n* copies of *x*. To formalize theorems such as T33 (and the challenge concerning T15 offered earlier), you could require a two-sorted predicate calculus and could incorporate a theory of the natural numbers. Beeson formalized these theorems in a simpler first-order theory, without requiring the natural numbers or mathematical induc- tion. In essence, he just supposes that the exponents belong to some commutative semigroup, not necessar- ily the natural numbers. Beeson allows addition and multiplication of exponents. He uses the "+" symbol for addition of exponents and the symbol "o" (lower-case letter "o") for multiplication of exponents, since * is already used for the semigroup multiplication. There exists a constant 0 (for use in exponents), and he wrote down the natural laws relating exponentiation to the semigroup or group operation. The following is a list of the axioms concerning exponentiation. (In this form, they can be used directly in OTTER; some of the spaces between symbols that look awkward to a person are required by OTTER.)

```
x^ 0 = e.   % Use this one only with groups, not with semigroups
x^ 1 = x.
1+1 = 2.
2+1 = 3.
x^ (xp+xq) = x^ xp * x^ xq.
x^ (xp+1) = x^ xp *x.    % deducible but perhaps useful to have explicit.
%  exponents and addition
x+y=y+x.
x+0 = x.
0+x = x.
(x+y) + z = x + (y+z).
%  exponents and multiplication
0 o x = 0.
x o 0 = 0.
1 o x = x.
x o 1 = x.
(x o y) o z = x o y o z.
x o (y + z) = x o y + x o z. (x
+ y) o z = x o z + y o z. x o y
= y o x.
(x^y) ^ z = x^ (y o z).
```

The underlying theory here is a two-sorted predicate calculus, with one sort for semigroup elements and another sort for exponents.

As is true throughout this notebook, I almost never focused on group theory, although I frequently focused on monoids. The distinction, as noted, concerns axioms about inverse, axioms I used infrequently. However, I did use axioms about identity, namely, *ex = xe = x*, justified when studying monoids. When the goal concerned semigroups, other than the hypothesis of the specific theorem in focus, associativity (and often its flip) was all that was allowed. For T33, two major paths existed with the goal of proving, for semi- groups, commutativity. For the first path, which was in fact the path I pursued first, I could remove axioms involving the identity element *e*, find a Knuth proof, (perhaps) shorten it still with a Knuth-Bendix approach, then convert it (if I could) to a demodulation-free proof, and finally shorten the result. If all went according to plan, I would have in hand a rather short, or very short, proof relying solely on paramodulation of T33 that held for semigroups. I could, on a second path, retain the axioms involving the group identity element and find a Knuth proof, then from it find a shorter Knuth proof, then convert it to a demodulation- free proof, then shorten that proof, and finally seek a proof for semigroups in which, therefore, axioms involving *e* were not present. On either path, I would rely on the Beeson axioms for exponentiation. A proof should, on the surface, be easier to obtain at the beginning because of the additional axioms involving the identity element *e*. On the other hand, perhaps on that path, conversion from Knuth to strictly paramod- ulation, and conversion from group theory to semigroup theory, might be more difficult because of deduced steps that rely on *e*. Both paths merit detailing.

I began with an input file similar to that used in studies reported earlier; sample input files will be given later. I did provide some direction to the search for a Knuth proof of T33 by including a few resonators gleaned from studies of T17. I avoided, by commenting out, the use of axioms that involved *e* and asked OTTER to find a Knuth-Bendix proof. As for the title of this section, the motivation rests in part from the fact that more than 1 CPU-hour was required before a proof was found, which was certainly longer than for studies of variants of T15 and T17. Another contribution to the title rests with the impres- sive length of the proof OTTER presented to me, a proof of length 830. Yes, a proof of this length is unusual in my experience, especially when a Knuth-Bendix approach is used. In that I planned to convert whatever Knuth proof I found to a proof relying on paramodulation alone, free of demodulation, an obvious move was to try for a shorter proof—much shorter, I hoped.

One of the simplest ways to find a shorter proof is, as experience suggests, to take the file that pro- duced the too-long proof and crammed it by adjoining the many deduced steps as resonators. When I did so, after intervening runs, the program offered me a proof of length 324. Although this sharp reduction in proof length was pleasing, I stored the result away and then turned to the use of ancestor subsumption and the hot list strategy, still relying on many, many resonators. As for the contents of the hot list, based on my years of experimentation, I placed a copy of the hypothesis of T33. In addition, I placed copies of five of the Beeson axioms regarding exponentiation. I cannot recall my reason for choosing these elements for the hot list. Just perhaps, as I attacked this theorem in many ways simultaneously, I stumbled onto the positive effect of doing so. The program found a 214-step proof, a proof that I immediately used, its deduced steps, as resonators. That 214-step proof was the third proof yielded in the experiment. The use of ancestor sub- sumption typically leads to finding more than one proof, although the proofs do not always have decreasing lengths. Those resonators, in the next experiment, enabled the program to complete three proofs, the last and shortest of length 206. The negations of those 206, actually 204 because of the way OTTER counts length in a Knuth proof, steps would be the targets for much of this part of the journey, aimed at producing a demodulation-free proof. To be totally clear, the plan was not to replace the deduced equations of the
206-step Knuth proof but rather to fill in the gaps, the gaps that were (in effect) present because of demodu- lation. The idea was, therefore, to find iteratively enough additional equations so that, in the end, a demod- ulation-free proof would be completed, deducing commutativity. Instead of detailing each experiment, including those that led nowhere, I shall now outline the approach I took and, when I discuss the second path, provide far more details.

You might naturally and reasonably suggest that a shorter Knuth proof would make a feasible starting point. I did try for a bit to find such. Among my attempts, I used the deduced equations of the Knuth proof as resonators. Unfortunately, and to my amusement, the result was, if memory serves, the completion of a proof of length 282. No, I do not know why such occurred, nor did I investigate.

The iterative approach that I used had me make a series of runs, later ones building on earlier ones by adding to list(sos) proof steps that were found. In particular, if in run *k* the program found proofs of some of the targets, then the deduced steps of those proofs were added as so-to-speak lemmas in run *k*+1. Also, sometimes, the set of resonators being used was amended by adjoining resonators corresponding to the cited proof steps. Of course, the trip was not as smooth as (possibly) implied by this general description.

Indeed, although I usually relied on what might be termed a complexity-preference search, occasion- ally I turned to level saturation. By a complexity-preference search I mean the program chooses where next to focus its attention to drive the reasoning based on the so-called simplest but not-yet-used deduced item. As evidence that the trip was not straightforward, with level saturation I occasionally resorted to cramming, having the program attempt to force newly adjoined proof steps into proofs still to be found. Even with these extra moves, yet one more important measure was taken, one that I like to think of as in some way sinister. In particular, the journey on this part of the first path, aimed at producing a proof based solely on paramodulation, encountered an equation, among the deduced equations of the 206-step proof, that resisted being proved, resisted the usual moves I made. An inspection of the hard-to-reach equation often, but not always, revealed that, in the Knuth proof, in addition to the two parents, three or more demodulators were used. A reasonable conclusion asserts that a few equations must be found to explicitly take the place of the demodulators. And you come to a bit of methodology that might not be expected. Specifically, what works, almost always, is to make a run in which the parents and the demodulators that are cited (in the Knuth proof) are each placed in list(sos) as its only members, to drive the reasoning, and each placed in a hot list, to be consulted and used as each new conclusion is drawn and retained. A template is immediately in order.

Assume that the equation that has not yet been proved, or reached, in the attempt to avoid demodula- tion has parents *A* and *B* and that three demodulators are used again, *F*, *G*, and *F*. The run will contain the following.

```
list(sos).
A
B
F
G
end_of_list.

list(hot).
A
B
F
G
end_of_list.
```

Some of you may be familiar with the fact that one more addition must be made, something like the follow- ing.

```
assign(heat,4).
```

That instruction is placed among the other parameters and options given to OTTER. The value 4 ensures that enough visiting and revisiting will occur. When I made this rather odd run, with the target the negation of the equation resisting proof, I typically did get the needed missing equations. I was surprised, some- times, by finding that eight equations were presented, when I expected four or fewer.

My travels through the country of proof conversion completed with a fully unexpected—and I may have lost track of what actually occurred—proof of length 94, a proof relying exclusively on paramodula- tion. Many, many cities were visited in my travels, but progress was occurring.

Now, through care exercised by Beeson and his e-mail and conversations with me, I must note that the 94-step proof does not apply to semigroups. The fault rests with three axioms, from among those found in the input, that were cited in the proof, the following.

```
0+x=x.
0 o x=0.
x o 0=0.
```

The presence of these three axioms, so Beeson points out, could eventually be used to deduce *xˆ0*x = x*, which in turn asserts that for each element *x* there exists an identity element. Therefore, although the proof is not restricted to monoids, it does apply to what you could call semimonoids. In particular, the proof does *not* imply that an identity *e* exists such that for all *x*, *ex = xe = x*. (At the time I write this material, I have not yet succeeded in dispensing with the three unwanted axioms; however, I shall proceed as if I am focus- ing on semigroups, now telling the tale of proof shortening.)

I began the second half of the voyage (focusing on the first path designed to prove commutativity for semigroups for T33) by invoking ancestor subsumption and by relying on 94 resonators, each correspond- ing to one of the 94 steps of the proof that relied solely on paramodulation. The run produced an 85-step proof and no other proofs of commutativity, which was a bit of a surprise in that, at least early on a trip of this type, the use of ancestor subsumption often produces more than one proof. Yes, I had not, and did not through to the end, comment out the three unwanted axioms, for, at the time, I was unaware that semi- monoids were in focus rather than semigroups. The use then of the 85 resonators corresponding to the deduced steps of the 85-step proof yielded nothing, that is, nothing until I invoked the hot list strategy with heat=1. The hot list contained the following four equations, including, as you see, another unwanted,

unwanted in the context of semigroups.

x+0 = x.
0+x = x.
0 o x = 0.
x o 0 = 0.

The hot list led to finding two new proofs, of respective lengths of 82 and 80. You see, the use of the hot list strategy (in effect) allows the program to look ahead, sometimes far ahead. Indeed, with level saturation and heat=1, well before the program has finished with level *j*, the program is examining areas of levels of greater and even greater value; with heat=4, the program looks dramatically ahead of where it would have with heat=0.

The use of the 80 resonators in place of the cited 85 next led to a 79-step proof. I then used 79 res- onators instead of the 80 but assigned heat the value 0, rather than 1; I am not sure why. The result was the finding of a 75-step proof. I, like you, was curious about running an experiment almost identical to that just cited, but with heat=1; the result was a 78-step proof.

Now, as you might expect from other notebooks I have on my website, automatedreasoning.net, I almost always run experiments in parallel (so to speak). You might say I lack patience, or you might say I am eager to obtain more proofs. In this case, I had, as noted, a 79-step proof, which I used, in the form of resonators, and found a 76-step proof. And the next bit of methodology you may indeed find intriguing and even useful in your own research. In particular, in the run that yielded the 76-step proof, I had also asked the program to find proofs, perhaps subproofs is more precise, of various late steps in the 79-step proof. The motive for this action was definitely not curiosity; rather, I thought that I might be able to put to use the information that was gleaned. I was relying on years of experience that showed such information to be sometimes useful, especially in the following context—and here is the intriguing methodology. I found a
62-step proof of an equation, in the 79-step proof, that occurs two steps before the 79-step proof was com- pleted. Cramming was its destination, or the destination of whichever subproof I chose to focus on. You see, a means for proof shortening rests on the fact that many, many proofs usually exist for a given theorem. Sometimes, two or more of them share the first *j* steps, in this case, if all went well, 62 of them. I placed those 62 equations in list(sos), still retaining there the hypothesis as well as copies of certain other axioms, and chose level saturation with the intent of cramming, forcing, the program (if possible) to complete a new proof, one different from the 79-step proof. Clearly, it succeeded, or I would not be including this bit of research in the notebook. That run (in effect) asserted that but five additional equations would complete a deduction of commutativity, rather than relying on seventeen equations. When I used, as resonators, corre- spondents to the sixty-two and the five, voila! a proof of length 66 was found. I was so pleased, in part at the nice reduction in proof length for T33, and in part for the newer evidence of the power of using the cramming strategy in this manner.

One of the lessons some researchers believe merit learning says that when an approach succeeds, repeat its use, perhaps again and again. I did exactly that, first finding through a small path that I cannot recover a slightly different 66-step proof. I chose a 50-step subproof, a proof of one of the late equations in the 66-step proof, and crammed on it to see what would occur. To my temporary disappiontment, the run in focus suggested sixteen equations to use. However, when I used as resonators fifty corresponding to the initial segment of the newer 66-step proof, followed by sixteen corresponding to the suggested sixteen equations to use, OTTER found in the newest run two proofs of respective lengths 65 and 57, by employing McCune's ancestor subsumption. As you may have surmised, the use of resonators corresponding to a suc- cessful proof completion often leads to a fast turn-around, a small amount of computer time to obtain the next result. The two cited proofs were obtained, respectively, in less than 2 CPU-seconds and a bit more than 217 CPU-seconds. By continuing to stay with the approach in focus, using the 57-step proof as res- onators and cramming on it to obtain a possible path to a proof yet smaller in length, I obtained two more proofs. The first proof, of length 58, was completed in just over 3 CPU-seconds; but the second proof, as it turned out, caused me to wait and wait and wait to see what might be found. After just over 14334 CPU- seconds, the wait proved worthwhile;

indeed, the program presented me with a 56-step proof. (Of course, to some, an improvement of one in length produces little pleasure; but, to me, each decrease in proof length, regardless of how small, produces satisfaction.) Just as a reminder, I am not detailing those experi- ments that yielded little of value.

Reliance on the cited 56 equations, as resonators, led to a 53-step proof, in less than 2 CPU-seconds. Immediate use as resonators of the 53 deduced equations led to two proofs in the same run, the first of length 53 (which often happens) and the second of length 52 in just over 4717 CPU-seconds. Yes, a key element in this type of research, as well as other not-so-similar types, is that of waiting. But, from the viewpoint of the unaided researcher, unaided by any computer program, days and days can elapse before a sought-after proof is in hand. For the curious, the 52-step proof contains four deduced equations not present in the 53-step proof. The use, as seen again and again, of the 52-step proof led to finding a proof of length 50 in just over 8598 CPU-seconds; it was the second of two proofs. That proof relied on a clause numbered (98171), denoting how many new equations were retained. I, unfortunately, did not use a report, a feature that would have told me how many new clauses had been generated. Finally, in the context of a proof I present next, the use of the 50-step proof as resonators led to the completion of a 48-step proof in less than 1 CPU-second. I leave it to you to go further, if that is your preference. My efforts in that regard failed.

The first path has now been detailed; commutativity has been proved for T33. Unwanted axioms remain in the proof, as you will immediately see; semimoniods is the area, even though semigroups was the goal. This bit of research began February 15, 2012, and completed March 23, 2012; the proof follows.

A 48-Step Proof for T33

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Fri Mar 23 11:34:58 2012
The command was "otter". The process ID is 20643.
----> UNIT CONFLICT at   0.26 sec ----> 2196 [binary,2195.1,35.1] $ANS(thm).

Length of proof is 48.  Level of proof is 19.

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

2 [] x*y*z= (x*y)*z.
3 [] x^1=x.
4 [] 1+1=2.
5 [] 2+1=3.
6 [] x^ (xp+xq)=x^xp*x^xq.
7 [] x^ (xp+1)=x^xp*x.
8 [] x+y=y+x.
9 [] x+0=x.
10 [] 0+x=x.
12 [] 0 o x=0.
19 [] x o y=y o x.
20 [] (x^y)^z=x^ (y o z).
21 [] y*x=x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k).
25 [] x o y=y o x.
26 [] (x^y)^z=x^ (y o z).
35 [] a*b!=b*a|$ANS(thm).
56 [para_into,21.1.1,6.1.2,flip.1] (x^y)^ (3 o k)* (x^z)^ (3 o k)* (x^y)^ (3 o k)* (x^z)^ (3 o k)=x^ (z+y).
59 [para_into,21.1.2.2.1.2,19.1.2,flip.1] x^ (3 o k)*y^ (k o 3)*x^ (3 o k)*y^ (3 o k)=y*x.
61 [para_into,21.1.2.2.1,20.1.1,flip.1] x^ (3 o k)*y^ (z o 3 o k)*x^ (3 o k)* (y^z)^ (3 o k)=y^z*x.
69 [para_from,21.1.1,2.1.2.1,flip.1] (x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k))*z=y*x*z.
181 [para_into,26.1.2.2,12.1.1] (x^0)^y=x^0.
188 [para_into,56.1.1,21.1.2] x^y*x^z=x^ (y+z).
```

```
203 [para_into,69.1.1,2.1.2] x^ (3 o k)* (y^ (3 o k)*x^ (3 o k)*y^ (3 o k))*z=y*x*z.
225 [para_from,181.1.1,61.1.1.2.2.2] x^ (3 o k)*y^ (0 o 3 o k)*x^ (3 o k)*y^0=y^0*x.
240 [para_into,188.1.2.2,10.1.1] x^0*x^y=x^y.
241 [para_into,188.1.2.2,9.1.1] x^y*x^0=x^y.
242 [para_into,188.1.2.2,8.1.2] x^y*x^z=x^ (z+y).
250 [para_into,203.1.1.2,2.1.2] x^ (3 o k)*y^ (3 o k)* (x^ (3 o k)*y^ (3 o k))*z=y*x*z.
280 [para_into,225.1.1.2.1.2,12.1.1] x^ (3 o k)*y^0*x^ (3 o k)*y^0=y^0*x.
298 [para_into,240.1.1.2,3.1.1] x^0*x=x^1.
339 [para_from,241.1.2,240.1.1.2] x^0*x^y*x^0=x^y.
362 [para_into,242.1.1.1,3.1.1,flip.1] x^ (y+1)=x*x^y.
393 [para_into,250.1.1.2.2,2.1.2] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
483 [para_into,298.1.2,3.1.1] x^0*x=x.
555 [para_into,362.1.1.2,5.1.1,flip.1] x*x^2=x^3.
556 [para_into,362.1.1.2,4.1.1,flip.1] x*x^1=x^2.
564 [para_into,362.1.1,7.1.1] x^y*x=x*x^y.
593 [para_into,393.1.1.2,393.1.1] x^ (3 o k)*x*y*z=y*x*x^ (3 o k)*z.
606 [para_into,483.1.1,280.1.2] x^ (3 o k)*x^0*x^ (3 o k)*x^0=x.
624 [para_from,555.1.1,2.1.2.1] x*x^2*y=x^3*y.
643 [para_into,556.1.1.2,3.1.1,flip.1] x^2=x*x.
704 [para_from,564.1.1,2.1.2.1] x^y*x*z= (x*x^y)*z.
785 [para_into,606.1.1.2,339.1.1] x^ (3 o k)*x^ (3 o k)=x.
875 [para_from,643.1.2,2.1.2.1,flip.1] x^2*y=x*x*y.
955 [para_from,704.1.1,593.1.1] (x*x^ (3 o k))*y*z=y*x*x^ (3 o k)*z.
961 [para_into,785.1.1.1.2,25.1.2] x^ (k o 3)*x^ (3 o k)=x.
974 [para_into,785.1.1,643.1.2] (x^ (3 o k))^2=x.
989 [para_from,785.1.1,2.1.2.1] x^ (3 o k)*x^ (3 o k)*y=x*y.
1046 [para_from,875.1.1,624.1.1.2] x*x*x*y=x^3*y.
1090 [para_into,955.1.1,2.1.2] x*x^ (3 o k)*y*z=y*x*x^ (3 o k)*z.
1164 [para_into,974.1.1,26.1.1] x^ ((3 o k) o 2)=x.
1324 [para_into,1046.1.2.1,26.1.1] x^y*x^y*x^y*z=x^ (y o 3)*z.
1385 [para_into,1090.1.1,875.1.1] x*x* (x^2)^ (3 o k)*y*z=y*x^2* (x^2)^ (3 o k)*z.
1422 [para_into,1164.1.1.2,25.1.2] x^ (2 o 3 o k)=x.
1599 [para_into,1385.1.1.2.2.1,26.1.1] x*x*x^ (2 o 3 o k)*y*z=y*x^2* (x^2)^ (3 o k)*z.
1626 [para_into,1599.1.2.2.2.1,26.1.1] x*x*x^ (2 o 3 o k)*y*z=y*x^2*x^ (2 o 3 o k)*z.
1629 [para_into,1626.1.2.2,875.1.1] x*x*x^ (2 o 3 o k)*y*z=y*x*x*x^ (2 o 3 o k)*z.
1654 [para_into,1629.1.1.2.2.1,1422.1.1,flip.1]  x*y*y*y^ (2 o 3 o k)*z=y*y*y*x*z.
1662 [para_into,1654.1.1.2.2.2.1,1422.1.1] x*y*y*y*z=y*y*y*x*z.
1760 [para_into,1662.1.2,1324.1.1] x*y^z*y^z*y^z*u=y^ (z o 3)*x*u.
1870 [para_into,1760.1.1.2,1324.1.1] x*y^ (z o 3)*u=y^ (z o 3)*x*u.
1935 [para_into,1870.1.1.2,961.1.1,flip.1]  x^ (k o 3)*y*x^ (3 o k)=y*x.
2065 [para_from,1935.1.1,59.1.1.2] x^ (3 o k)*x^ (3 o k)*y=y*x.
2195 [para_into,2065.1.1,989.1.1] x*y=y*x.
```

Especially for those who enjoy statistics and comparisons, of the forty-eight deduced equations in the just- given proof, thirty-five are not in the 830-step proof that (in effect) prompted the discovery of this 48-step proof. If you succeed in dispensing with the unwanted axioms, thus proving T33 for semigroups, regard- less of the length of the proof, even if the proof is of Knuth-Bendix nature, I would enjoy viewing it. Dur- ing my travels on the first path, I, as is typical of my research, studied various other Stein theorems. For T33, I conducted more than 350 experiments.

I did learn from those weeks of experiments, as you will shortly see when I detail my journey on the second path, a path designed to deduce commutativity from the hypothesis of T33, with the goal of having the domain be semigroups. For convenience, I repeat here the hypothesis of T33.

y*x = x^ (3 o k) * y^ (3 o k) * x^ (3 o k) * y^ (3 o k).

To review the meaning of the symbols, I refer you back to the earlier part of this section. There you will also find Beeson's treatment of exponentiation. This second path, as noted, has you keep the axioms involving the identity element *e* for a while, until you are finished with the shortening of a Knuth proof. You then turn to converting the (so-to-speak) short Knuth proof to a proof free of demodulation. When you have this proof based on paramodulation alone, you turn to shortening it. Then, finally, you try to eliminate all references to the identity element *e*. If all the gold is mined, you have, after the appropriate experiments, a proof based on paramodulation alone that applies to semigroups. I believe at the moment that passing from semimonoids to semigroups, if success occurs, presents the most difficulty. I invite you to again join me; the trip will be quicker. Indeed, as evidence that it took less time, no matter how time is measured, and as evidence that I learned from following the first path—if my notes are correct—I began following the sec- ond path on May 26, 2012, and arrived at almost my destination on May 28, 2012. You will learn, near the end of the next few paragraphs, why the "almost" is written.

I began in a manner similar to my travels on the first path, with the goal of finding a Knuth proof of T33. I submitted the following three lists to OTTER.

```
list(usable).
x=x.
% semigroup axioms
e*x = x.
x*e = x.
x* (y*z) = (x*y)*z.
(x*y)*z = x* (y*z).
% exponentiation
xˆ 0 = e.
xˆ 1 = x.
1+1 = 2.
2+1 = 3.
xˆ (xp+xq) = xˆ xp * xˆ xq.
xˆ (xp+1) = xˆ xp *x.    % deducible but perhaps useful to have explicit.
% exponents and addition
x+y=y+x.
x+0 = x.
0+x = x.
(x+y) + z = x + (y+z).
% exponents and multiplication
0 o x = 0.
x o 0 = 0.
1 o x = x.
x o 1 = x.
(x o y) o z = x o y o z.
x o (y + z) = x o y + x o z. (x
+ y) o z = x o z + y o z. x o y
= y o x.
(xˆy) ˆ z = xˆ (y o z).
end_of_list.

list(sos).
y*x = xˆ (3 o k) * yˆ (3 o k) * xˆ (3 o k) * yˆ (3 o k). xˆ
(3 o k) * yˆ (3 o k) * xˆ (3 o k) * yˆ (3 o k) = y*x.
end_of_list.

list(passive).
a*b != b*a | $ANS(thm33).
```

```
end_of_list.
```

In reverse order, the list(passive) contains the negation of the goal, commutativity; list(sos) contains the hypothesis, and its flip, to initiate the search for a proof; list(usable) contains items used to complete appli- cations of paramodulation, which in turn is employed by a Kunth-Bendix approach. In this last list, you find two copies of associativity, the second being the flip of the first. Inclusion of flips of equations often aids OTTER in finding a sought-after proof. Although, in principle, I was aiming at a proof for the domain of group theory, as you see, no axioms for inverse were included. Therefore, to be precise, the domain under study was monoids. The program presented me with a 99-step proof; you might recall that its count for length, with a Knuth-Bendix approach, includes flips that are cited. With the use of ancestor subsump- tion, the run found three proofs of respective lengths 99, 93, and 58; less than 45 CPU-seconds sufficed.

As you have surmised, the next action to take was using fifty-six resonators, each corresponding to one of the deduced equations in the shortest of the three proofs. Two proofs were found of respective lengths 60 and 52. Yes, a brief examination of the 52-step proof showed that both axioms regarding the identity element, left and right, both axioms for associativity, the original and its flip, and both axioms for the hypothesis of T33, the original and its flip, were present in the cited 52-step proof. In that I preferred, only being influenced by the decimal system, to have a proof of length less than 50, I tried other experi- ments. One of them, based on two sets of resonators (to be detailed almost immediately), eventually pro- duced a 48-step proof in less than 10 CPU-seconds. The first set of resonators corresponded to the deduced steps of a 56-step Knuth proof, and the second set corresponded to the deduced steps of the earlier-cited
830-step Knuth proof. I made one other move, a move that I think made a significant difference. In partic- ular, I assigned the value 2, rather than 4, to McCune's pick_given_ratio, a parameter probably not available in many automated reasoning programs. If the value *k* is assigned to the pick_given_ratio parameter, then the program will choose *k* clauses based on complexity, 1 based on breadth first, *k*, 1, and so forth.

I was thus, for purely aesthetic reasons, satisfied with this proof of length less than 50, and I turned from a Knuth-Bendix approach to an approach based on paramodulation alone (avoiding the use of demod- ulation entirely). The two key commands on which I relied for the rest of my trip on this second path were the following.

```
set(para_from).
set(para_into).
```

And, especially for OTTER users, I relied on the following assignments, assignments that can have a pro- found effect on what the program retains, how it conducts its search, and how it restricts its search.

```
assign(pick_given_ratio,2).
assign(max_proofs,-1).
assign(max_distinct_vars,5).
assign(max_mem, 840000).
assign(max_weight,40).
```

The assignment of the value -1 to max_proofs instructs the program to find as many proofs as it can within memory and time constraints that are chosen by the researcher. The assigned value of 5 to max_dis- tinct_vars has the program immediately discard any newly deduced item when that item relies on six or more distinct variables. With the given assignments, memory is restricted to 840 megabytes. Finally, if no other action is taken, any deduced item relying on strictly more than 40 symbols (not counting commas and parentheses) is discarded. The most common action that can be taken concerns the use of weight templates, such as occurs with the inclusion of resonators. You see, as noted earlier, you can assign small numbers to long equations or formulas with, say, appropriate resonators. You can, for example, have OTTER treat quite long, or very long, expressions as if they are quite short, which does indeed have a large effect on directing the search and on which new information is retained.

I was now prepared to set sail for the port known as proof conversion, having in hand the 48-step Knuth proof that marked a successful completion of the first part of the second-path voyage. The target, as noted, was a proof of commutativity from the T33 hypothesis, hopefully in the domain of semigroups. Intermediate targets always play a role in this aspect of research. I had forty-six of them, the negations of the deduced steps of the 48-step proof. You can view the forty-six deduced equations as, in effect, provid- ing an outline of the proof I sought, although I might reach commutativity without finding demodulation- free subpaths to all of the forty-six. So I placed in list(passive) the forty-six negations. Of course, I would have to iterate, proving a few of the targets at a time, using the corresponding subproofs as I progressed. And, I almost forgot to note, I abandoned the use of ancestor subsumption because its use slows the pro- gram markedly sometimes and shortness of proof was not my concern during this part of the trip.

The pursuit of a demodulation-free proof began with great promise. Indeed, twenty-one of the forty- six were proved, with the corresponding proofs, after sorting to remove duplicate equations, providing me with, coincidentally, forty-six (so-to-speak) lemmas for the next run. Each of the lemmas to be used was taken from one of the twenty-one proofs. Each of these forty-six lemmas was adjoined to the list(sos) to be used, with level saturation, to seek proofs of more of the targets, the (what I call) intermediate equations, those occurring before or at commutativity. When a level-saturation approach is employed, automatically the pick_given_ratio strategy is ignored. As I traveled on the way to a proof based solely on paramodula- tion, I did not check to see whether any of the targets was proved more than once. This can happen because, not seldom, more than one equation can be deduced, of different generality, that contradicts a given negation that itself (here) is free of variables.

With these choices, the program found proofs for three additional members of the forty-six interme- diate targets. Those added lemmas enabled OTTER to make progress. However, as a sign of the difficulty of converting the proof in hand, almost 3900 CPU-seconds was required to prove the last of the three new proofs. Some of you, perhaps all of you, may be puzzled by the following observation. Specifically, the forty-six added equations, in list(sos), are considered by the program, enabling it to add to the list of suc- cesses in the context of intermediate targets. But those equations were already present in, say, run *A*. So why were they forced to wait to be used in, say, run *B*? Why did the program fail to use them as run *A* was still in focus? Well, a guess, but a very good guess, is that too many other equations were in the way, too many equations remained to become the focus of attention to drive the reasoning before some or all of the forty-six were chosen. Put in a slightly different way, by placing those forty-six equations in the list(sos) of run *B*, they were almost immediately chosen to drive the reasoning, waiting for essentially no other equa- tions to be considered. If you are using OTTER, you can ensure the consideration of all elements of list(sos) before considering any newly deduced item by including the following command.

```
set(input_sos_first).
```

And, with surprise and even disappointment, I just checked that the cited command was included as I trav- eled the second path, and I found that it was absent. So, some of the CPU times I shall list are greater than they might have been if the command had been included. With level saturation, as you will correctly observe, this input_sos_first command is irrelevant, by the very nature of level saturation. As for the first path, detailed earlier in this section, my checking suggests I did include the input_sos_first command throughout.

As I so often do, I paralleled in a sense the actions just described with but one difference: I replaced level saturation with a search based on the pick_given_ratio strategy. The program found nine proofs to supplement the twenty-one found near the beginning of this phase.
###Larry - Do you mean Among the nine, two of the three that had been found with level saturation were found again; the third was the last and most time-consuming.
###Shouldn’t you leave out the part you have in the next sentence "found with level saturation" since you had already said that it was one of three found with level saturation Among the nine, two of the three found with level saturation were found again; the third was the last and most time-consuming, found with level saturation.
###LARRY - do you mean they again required more than they had with the ratio -- so move the word again to the start of the sentence? Again, the two found with level saturation required roughly 50% more time than with the ratio strategy. (When level saturation is in use, the ratio strategy is, in effect, blocked, having no meaniing.) One reason the third proof found with level saturation was not found with the ratio strategy rests with the fact that level saturation forces the program to focus on equations regardless of their weight, complexity. In contrast, with the ratio strategy, a complex equation might never be chosen

to drive the reasoning because of its weight.

For a few experiments, I chose not to take into account the proof (of an intermediate target) found with level saturation in favor of the nine proofs found without its use. However, as you will learn, that third proof was put to good use after a while.

Various runs produced no additional proofs of the intermediate targets. After a few failures, I adjoined to list(sos) forty sorted proof steps, based on the nine proofs already cited. I also adjoined, after the forty, six equations taken from a 6-step proof found with level saturation, the third proof that I said you would find put to use. In less than 2 CPU-seconds, OTTER proved five of the intermediate targets. Although I had the program continue for quite a while, no other proofs were found.

Odd though it may seem, numerically, those five proofs provided me with, after sorting, five additional equations to adjoin (in the spirit of lemma adjunction) to list(sos). I say it was odd, at least to me, in that ten equations were required to prove the five intermediate targets; obviously, duplication was rampant. This latest run yielded exactly one proof of an intermediate target, and in less than 5 CPU-seconds.

I was concerned that, at this rate, many days might pass before, if ever, I had the desired proof of commutativity. Therefore, I repeated the experiment with but one change; namely, I turned back to level saturation. You are witnessing the piquancy of the methodology. Indeed, sometimes I turn away from level saturation to using ratio, and sometimes I turn away from ratio to level saturation. Well, I had to wait a long time if measured in CPU-seconds, or measured in CPU-minutes; but, in addition to the one (so-to- speak) new proof, OTTER gave me two other proofs of interest: the first, the duplicate, in less than 3 CPU- seconds; the second in more than 3936 CPU-seconds (more than 1 CPU-hour); the third in more than 7424
CPU-seconds. (Fortunately, while this time was passing, I was studying various other Stein theorems, some of which may appear later in this notebook.)

Being aware that some of the proofs already discussed, on the second path, may be of the same inter- mediate target (as discussed earlier in this section), next in order was a run without extra lemmas to see where things stood. I wished to know how many of the forty-six targets had been proved. The correspond- ing run showed that twenty-seven had been proved, so, indeed, some duplication of proving intermediate targets had occurred. Now, I concluded that some new action must be taken if I were ever to have a demod- ulation-free proof of commutativity. Put another way, if the goal was to prove all of the forty-six intermedi- ate targets, or replace them, at the rate progress was occurring, a long, long voyage was in store.

I made two moves. I added as resonators five such, each corresponding to a step of a proof found earlier, where the program had proved three intermediate targets. I also decided to try using the hot list strategy, with the view that the search space would be explored in a quite different manner from that in which the twenty-seven proofs were obtained. I assigned the value 1 to heat. Well, these two moves—and I do not know which had which effect—were most effective. Indeed, OTTER proved thirty-eight of the forty-six. the first of the thirty-eight was proved in less than 1 CPU-second, and the last in just over 555
CPU-seconds. Possibly the additional five resonators, although few, provided important direction to the reasoning. Quite likely, with heat=1, the program was looking farther and farther ahead, rather than waiting until key items were deduced and were chosen as the focus of attention to drive the program's reasoning.

With but eight targets still unproved, I was most pleased. That run producing the thirty-eight proofs provided me with 106 (in effect) lemmas to place in list(sos), in addition to the hypothesis of T33 and its flip. I did not include corresponding resonators, and I am not sure why I did not. The run found a single proof. To provide you with a sense of what was occurring, that single proof, completed in less than 13 CPU-seconds, had a length of 2, just two new equations. Still, progress was occurring, but not fast enough, so I thought. And I decided to imitate that which had worked before, namely, to try the same experiment or run but with level saturation. It proved to be a good choice.

Five proofs were returned. And you are now to be visited by large numbers. In particular, although the first proof was completed in just over 4 CPU-seconds, the last of the five required just over 46,304 CPU-seconds. The first proof relied on a clause numbered (11918), while the last relied on a clause numbered (1068951).

A pause for some comment on the use of the combination of level saturation and the hot list strategy might prove instructive and explain in part how these large numbers were encountered. Even without the use of a hot list, the size of the levels grows most rapidly, far more rapidly than some people originally believed. With a hot list, even with heat=1, the size of the levels grow much faster; well, technically, the size of the levels is not growing differently, but the program is looking ahead. Indeed, after a short while, the program is considering items on level 2 though it is focusing in the main on level 1. Still later, the pro- gram can be considering items at level 5 even though in the main it is browsing through elements at level 3. Things get worse all the time, in the sense of looking farther and farther ahead.

Specifically, the level under study, when the last of the five proofs was completed, was 5; before the proof was completed, 10,386 equations had been chosen as the focus of attention to drive the reasoning. McCune, fully aware of how a level-saturation search can explode, provided the following two commands.

```
assign(change_limit_after,600).
assign(new_max_weight,22).
```

With commands of this type, the researcher can instruct OTTER to change the assigned value to max_weight, usually lowering it quite a bit, to decrease the rate of growth of the levels to be explored. Yes, McCune was indeed brilliant.

Again, a run was made to see where things stood. Many sets of resonators were employed, from var- ious earlier runs, and the only members of list(sos) were the T33 hypothesis and its flip. That run presented forty proofs, forty of the forty-six. Just in case some confusion is occurring regarding some of the strategy, ancestor subsumption was still not in use, because of its slowing effect on a search. The addition of another small set of resonators enabled the program to return forty-one proofs, leaving five to prove, some of which might be difficult to conquer.

Acting on that assumption, I turned to a direct attack, used on the first path, on one of the missing tar- gets, one that had as yet not been proved. In general, the idea is to go directly to the 48-step Knuth proof, locate the equation that is offering resistance, and copy its parents and any and all demodulators that were used to obtain it in the Knuth proof. You then remove all the equations that were being used in the preced- ing run and, for this new run, place the identified parents and demodulators in list(sos). You also place the negation of the sought-after equation in list(passive), to be the target. Does this remind you somewhat of cramming? Indeed, you are limiting the program’s access to information—at the beginning, just to the cho- sen parents and demodulators—attempting to force it to derive, perhaps not easily, the resisting equation. Level saturation is often my choice. Sometimes you are met with a surprise, expecting, because of the number of demodulators that were used in the Knuth proof, a proof of length, say, *k*. Now, you realize that you are not controlling completely what the program does, where it looks, and which items in which order it employs to direct the reasoning. So, occasionally, rather than a *k*-length proof, if one is in fact found, the program presents you with a longer proof, even a few steps longer than expected. Because in the case under discussion I expected a proof of length 4, I chose to use a hot list with an assignment of 4 to the heat parameter. As for the contents of list(hot)—you may indeed have surmised what I placed there—the list mirrored the list(sos) that contained the identified parents and demodulators.

Be warned, when using this bit of methodology, specifically when demodulation is applied three or more times not necessarily with different demodulators, you may be forced to assign quite a high value to the heat parameter. For example, say that two parents are involved, as must be the case with paramodula- tion, (not necessarily different) and demodulation is applied seven times, just three different demodulators are used to obtain the resisting equation. If you follow the approach just described, the entire size of the new list(sos) will be five, and that will be the size of the hot list. The two lists will be identical. You might expect to find, if all goes well, a proof of length, say, eight. Well, you might be presented with a proof of length eleven, and you might have been forced to assign the value 5 or higher to the heat parameter. The program, with level saturation, might require examining a few levels before finding the treasure. When and if your program wins, the proof steps of the completed proof can be used as resonators in the next run, or as lemmas adjoined to the list(sos), or in both ways.

When I used the methodology, the program returned to me a proof of length 4, which is not to be thought of as significant even though 4 was the assigned value to heat. I chose, for the next run, to add to my sets of resonators a set consisting of four members, each corresponding to one of the deduced steps of the completed 4-step proof.

When I made the called-for run, as predictable, the program presented me with forty-two proofs. The result confirmed, as I thought, that one additional (out of forty-six) intermediate target had been reached. I therefore continued in this manner, accruing resonators, occasionally relying on lemma adjunction, fre- quently employing the hot list strategy with the value 1 assigned to heat, and, when I encountered an ele- ment among the forty-six that offered substantial resistance, relying on the just-detailed bit of methodology. Sometimes I assigned the value 40 to max_weight, and sometimes 50; sometimes I assigned the value 2 to pick_given_ratio, and sometimes 4. Almost at the end of the journey, I again applied the parent-demodula- tor methodology to obtain additional resonators. Their use not only led to a proof of the resisting equation; the run under discussion—most unexpectedly—proved commutativity, offering a proof of length 64 for T33. Immediately I turned to ancestor subsumption, assigning the value 0, rather than 1, to heat. As you have perhaps predicted, I removed (by commenting out) all the resonators that I had used to obtain the
64-step proof, replacing them with sixty-four, each corresponding to one of the sixty-four deduced equa- tions. OTTER found a 57-step proof, the following.

A 57-Step Proof of T33

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Mon May 28 15:19:08 2012
The command was "otter". The process ID is 31183.
----> UNIT CONFLICT at   1.35 sec ----> 9606 [binary,9605.1,31.1] $ANS(thm33).

Length of proof is 57.  Level of proof is 22.

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

2 [] e*x=x.
4 [] x*y*z= (x*y)*z.
5 [] (x*y)*z=x*y*z.
6 [] x^0=e.
7 [] x^1=x.
8 [] 1+1=2.
9 [] 2+1=3.
10 [] x^ (xp+xq)=x^xp*x^xq.
11 [] x^ (xp+1)=x^xp*x.
12 [] x+y=y+x.
13 [] x+0=x.
14 [] 0+x=x.
17 [] x o 0=0.
23 [] x o y=y o x.
24 [] (x^y)^z=x^ (y o z).
25 [] y*x=x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k).
26 [] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)=y*x.
31 [] a*b!=b*a|$ANS(thm33).
41 [para_into,25.1.1,11.1.2] x^ (y+1)=x^ (3 o k)* (x^y)^ (3 o k)*x^ (3 o k)* (x^y)^ (3 o k).
42 [para_into,25.1.1,10.1.2] x^ (y+z)= (x^z)^ (3 o k)* (x^y)^ (3 o k)* (x^z)^ (3 o k)* (x^y)^ (3 o k).
60 [para_from,25.1.1,5.1.1] x^ (3 o k)* (y*z)^ (3 o k)*x^ (3 o k)* (y*z)^ (3 o k)=y*z*x.
61 [para_from,25.1.1,5.1.1.1] (x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k))*z=y*x*z.
75 [para_into,26.1.1.1,24.1.1] x^ (y o 3 o k)*z^ (3 o k)* (x^y)^ (3 o k)*z^ (3 o k)=z*x^y.
94 [para_from,41.1.2,25.1.2] x^y*x=x^ (y+1).
95 [para_into,42.1.1.2,12.1.2] x^ (y+z)= (x^y)^ (3 o k)* (x^z)^ (3 o k)* (x^y)^ (3 o k)* (x^z)^ (3 o k).
```

```
96 [para_from,42.1.2,25.1.2] x^y*x^z=x^ (y+z).
165 [para_into,61.1.1.1,60.1.1] (x*y*z)*u= (x*y)*z*u.
183 [para_into,75.1.1.2.2.1,24.1.1] x^ (y o 3 o k)*z^ (3 o k)*x^ (y o 3 o k)*z^ (3 o k)=z*x^y.
191 [para_into,94.1.1.1,7.1.1] x*x=x^ (1+1).
229 [para_into,95.1.2,42.1.2] x^ (y+z)=x^ (z+y).
238 [para_into,96.1.1.1,7.1.1] x*x^y=x^ (1+y).
250 [para_into,96.1.2.2,14.1.1] x^0*x^y=x^y.
251 [para_into,96.1.2.2,13.1.1] x^y*x^0=x^y.
312 [para_into,165.1.2,5.1.1] (x*y*z)*u=x*y*z*u.
372 [para_into,191.1.2.2,8.1.1] x*x=x^2.
545 [para_from,238.1.2,229.1.2] x^ (y+1)=x*x^y.
558 [para_into,250.1.1.1.2,17.1.2] x^ (y o 0)*x^z=x^z.
770 [para_from,312.1.1,61.1.1] x^ (3 o k)*y^ (3 o k)* (x^ (3 o k)*y^ (3 o k))*z=y*x*z.
865 [para_from,372.1.1,5.1.1.1] x^2*y=x*x*y.
894 [para_into,545.1.1,11.1.1] x^y*x=x*x^y.
961 [para_into,558.1.1,251.1.1] x^ (y o 0)=x^0.
1040 [para_into,770.1.1.2.2,5.1.1] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
1102 [para_into,865.1.1,94.1.1] x^ (2+1)=x*x*x.
1246 [para_from,894.1.2,5.1.1.1] (x^y*x)*z=x*x^y*z.
1253 [para_into,961.1.1.2,23.1.2] x^ (0 o y)=x^0.
1293 [para_into,1040.1.1.2.1.2,23.1.2] x^ (3 o k)*y^ (k o 3)*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
1300 [para_into,1040.1.1.2.2.2,251.1.1] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)=y*x*y^0.
1306 [para_into,1040.1.1.2,1040.1.1] x^ (3 o k)*x*y*z=y*x*x^ (3 o k)*z.
1374 [para_into,1102.1.1.2,9.1.1] x^3=x*x*x.
1495 [para_into,1246.1.1,5.1.1] x^y*x*z=x*x^y*z.
1590 [para_into,1253.1.2,6.1.1] x^ (0 o y)=e.
1715 [para_from,1300.1.1,26.1.1] x*y*x^0=x*y.
1932 [para_from,1374.1.2,312.1.1.1] x^3*y=x*x*x*y.
2068 [para_from,1495.1.1,1306.1.1] x*x^ (3 o k)*y*z=y*x*x^ (3 o k)*z.
2307 [para_from,1590.1.2,2.1.1.1] x^ (0 o y)*z=z.
2315 [para_into,1715.1.1.2.2,6.1.1] x*y*e=x*y.
2446 [para_into,1932.1.1.1,24.1.1] x^ (y o 3)*z=x^y*x^y*x^y*z.
2970 [para_from,2068.1.1,1306.1.2.2] x^ (3 o k)*x*y*z*u=y*z*x*x^ (3 o k)*u.
3280 [para_from,2307.1.1,183.1.1.2.2] x^ (0 o 3 o k)*y^ (3 o k)*y^ (3 o k)=y*x^0.
3297 [para_into,2315.1.1.2.2,6.1.2] x*y*z^0=x*y.
3647 [para_from,2446.1.1,1293.1.1.2] x^ (3 o k)*y^k*y^k*y^k*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
4433 [para_into,3297.1.2,2.1.1] e*x*y^0=x.
4863 [para_from,4433.1.1,2.1.1] x=x*y^0.
5111 [para_from,4863.1.2,3280.1.2] x^ (0 o 3 o k)*y^ (3 o k)*y^ (3 o k)=y.
5220 [para_into,5111.1.1,2307.1.1] x^ (3 o k)*x^ (3 o k)=x.
5311 [para_from,5220.1.1,4.1.2.1] x^ (3 o k)*x^ (3 o k)*y=x*y.
5337 [para_into,5311.1.1.2,2970.1.1] x^ (3 o k)*y*z*x*x^ (3 o k)*u=x*x*y*z*u.
5428 [para_from,5311.1.1,2970.1.2.2.2.2] x^ (3 o k)*x*y*z*x^ (3 o k)*u=y*z*x*x*u.
5752 [para_from,5428.1.1,5337.1.1] x*y*y*y*z=y*y*y*x*z.
6204 [para_into,5752.1.2.2.2.2,5311.1.1] x^ (3 o k)*y*y*y*x^ (3 o k)*z=y*y*y*x*z.
6711 [para_from,6204.1.1,3647.1.1] x^k*x^k*x^k*y*x^ (3 o k)*z=x*y*z.
6828 [para_into,6711.1.1,2446.1.2] x^ (k o 3)*y*x^ (3 o k)*z=x*y*z.
7089 [para_into,6828.1.1.1.2,23.1.2] x^ (3 o k)*y*x^ (3 o k)*z=x*y*z.
7778 [para_into,7089.1.2.2,5220.1.1] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)=x*y.
9605 [para_into,7778.1.1,26.1.1] x*y=y*x.
```

As you see, just two of the input axioms featuring the identity element *e* were relied upon, the following. e*x=x.

x^0=e.

By way of a summary, at this point I had shortened the demodulation-free proof found on the second path, finding the given 57-step proof. However, in view of the two cited and unwanted axioms, I was not yet in the domain of semigroups. If I could find a proof avoiding the first of the two, I would at least be in the domain of semimonoids.

I removed, by commenting out, the axioms of left and right identity, avoided the use of ancestor sub- sumption, and otherwise proceeded in a manner that yielded the cited 57-step proof. My goal, of course, was to get to the domain of semimonoids. OTTER found such a proof, one of length 65. The following three equations did, however, prevent the proof from applying to semigroups.

```
x^0=e.
x+0=x.
0+x=x.
```

Based on observations made by Beeson, in addition to the cited three, and of course the axioms of left and right identity, the following two axioms must not be used in the sought-after proof, if the proof is to apply to semigroups.

```
0 o x = 0.
x o 0 = 0.
```

I am, as I write these lines, trying to obtain the desired proof, a proof that avoids seven unwanted axioms that were present in the study of T33 in the beginning.

And I have an update as we (so to speak) examine what is occurring. I have a proof in which, of the seven unwanted equations, only the following two remain in a proof just found.

```
4 [] x^0=e.
12 [] 0+x=x.
```

(I permitted their actual numbers to be included in order to enable you to more closely share what is hap- pening.) The experiment, or run, that just completed provided a most unusual bonus. To have you share in my being startled, I note that I had tried to find a proof of length strictly less than 57 for the semimonoids case. I even tried the technique I often employ, that of blocking (with demodulation) the steps of the proof in hand one at a time. That approach yielded nothing, no progress. Well, the experiment in focus not only produced the proof with just two unwanted equations but also found a 54-step proof. As a caustic observer might say, so much for systematic, hard work!

I present you with some treasure, the 54-step proof, offering you the challenge of finding a proof, perhaps longer, that avoids all seven of the unwanted axioms.

A 54-Step Proof for T33 in the Domain of Semimonoids

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Fri Jun  8 07:45:35 2012
The command was "otter".  The process ID is 26509.
----> UNIT CONFLICT at   1.08 sec ----> 8585 [binary,8584.1,27.1] $ANS(thm33).

Length of proof is 54.  Level of proof is 25.

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

2 [] x*y*z= (x*y)*z.
3 [] (x*y)*z=x*y*z.
4 [] x^0=e.
5 [] x^1=x.
6 [] 1+1=2.
```

```
7 [] 2+1=3.
8 [] x^ (xp+xq)=x^xp*x^xq.
9 [] x^ (xp+1)=x^xp*x.
10 [] x+y=y+x.
12 [] 0+x=x.
19 [] x o y=y o x.
20 [] (x^y)^z=x^ (y o z).
21 [] y*x=x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k).
22 [] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)=y*x.
27 [] a*b!=b*a|$ANS(thm33).
36 [para_into,21.1.1,9.1.2] x^ (y+1)=x^ (3 o k)* (x^y)^ (3 o k)*x^ (3 o k)* (x^y)^ (3 o k).
37 [para_into,21.1.1,8.1.2] x^ (y+z)= (x^z)^ (3 o k)* (x^y)^ (3 o k)* (x^z)^ (3 o k)* (x^y)^ (3 o k).
53 [para_from,21.1.1,3.1.1] x^ (3 o k)* (y*z)^ (3 o k)*x^ (3 o k)* (y*z)^ (3 o k)=y*z*x.
54 [para_from,21.1.1,3.1.1.1] (x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k))*z=y*x*z.
66 [para_into,22.1.1.1,20.1.1] x^ (y o 3 o k)*z^ (3 o k)* (x^y)^ (3 o k)*z^ (3 o k)=z*x^y.
85 [para_from,36.1.2,21.1.2] x^y*x=x^ (y+1).
86 [para_into,37.1.1.2,10.1.2] x^ (y+z)= (x^y)^ (3 o k)* (x^z)^ (3 o k)* (x^y)^ (3 o k)* (x^z)^ (3 o k).
87 [para_from,37.1.2,21.1.2] x^y*x^z=x^ (y+z).
116 [para_into,54.1.1.1,53.1.1] (x*y*z)*u= (x*y)*z*u.
127 [para_into,66.1.1.2.2.1,20.1.1] x^ (y o 3 o k)*z^ (3 o k)*x^ (y o 3 o k)*z^ (3 o k)=z*x^y.
134 [para_into,85.1.1.1,5.1.1] x*x=x^ (1+1).
137 [para_into,85.1.2.2,12.1.1] x^0*x=x^1.
151 [para_into,86.1.2,37.1.2] x^ (y+z)=x^ (z+y).
180 [para_into,87.1.1.1,5.1.1] x*x^y=x^ (1+y).
226 [para_into,116.1.2,3.1.1] (x*y*z)*u=x*y*z*u.
251 [para_into,134.1.2.2,6.1.1] x*x=x^2.
405 [para_from,180.1.2,151.1.2] x^ (y+1)=x*x^y.
608 [para_from,226.1.1,54.1.1] x^ (3 o k)*y^ (3 o k)* (x^ (3 o k)*y^ (3 o k))*z=y*x*z.
695 [para_into,405.1.1,85.1.2] x^y*x=x*x^y.
698 [para_into,405.1.2.2,251.1.2] x^ (2+1)=x*x*x.
785 [para_into,608.1.1.2.2,3.1.1] x^ (3 o k)*y^ (3 o k)*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
921 [para_from,695.1.2,3.1.1.1] (x^y*x)*z=x*x^y*z.
927 [para_into,698.1.1.2,7.1.1] x^3=x*x*x.
1039 [para_into,785.1.1.2.1.2,19.1.2] x^ (3 o k)*y^ (k o 3)*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
1050 [para_into,785.1.1.2,785.1.1] x^ (3 o k)*x*y*z=y*x*x^ (3 o k)*z.
1099 [para_into,921.1.1,3.1.1] x^y*x*z=x*x^y*z.
1244 [para_from,927.1.2,226.1.1.1] x^3*y=x*x*x*y.
1555 [para_from,1099.1.1,1050.1.1] x*x^ (3 o k)*y*z=y*x*x^ (3 o k)*z.
1590 [para_into,1244.1.1.1,20.1.1] x^ (y o 3)*z=x^y*x^y*x^y*z.
2161 [para_from,1555.1.1,1050.1.2.2] x^ (3 o k)*x*y*z*u=y*z*x*x^ (3 o k)*u.
2373 [para_from,1590.1.1,1039.1.1.2] x^ (3 o k)*y^k*y^k*y^k*x^ (3 o k)*y^ (3 o k)*z=y*x*z.
2528 [para_into,137.1.1.1,4.1.1] e*x=x^1.
2657 [para_into,2528.1.2,5.1.1] e*x=x.
2733 [para_into,2657.1.1.1,4.1.2] x^0*y=y.
3065 [para_from,2733.1.1,695.1.1] x=x*x^0.
3089 [para_into,3065.1.2.2,4.1.1] x=x*e.
3139 [para_into,3089.1.2.2,4.1.2] x=x*y^0.
3191 [para_into,3139.1.2,2733.1.1] x^0=y^0.
4396 [para_into,3191.1.1,20.1.1] x^ (y o 0)=z^0.
4416 [para_into,4396.1.1.2,19.1.2] x^ (0 o y)=z^0.
4505 [para_from,4416.1.2,2733.1.1.1] x^ (0 o y)*z=z.
4649 [para_from,4505.1.1,127.1.1.2.2] x^ (0 o 3 o k)*y^ (3 o k)*y^ (3 o k)=y*x^0.
4697 [para_into,4649.1.2,3139.1.2] x^ (0 o 3 o k)*y^ (3 o k)*y^ (3 o k)=y.
```

```
4813 [para_into,4697.1.1,4505.1.1] xˆ (3 o k)*xˆ (3 o k)=x.
4898 [para_from,4813.1.1,2.1.2.1] xˆ (3 o k)*xˆ (3 o k)*y=x*y.
4919 [para_into,4898.1.1.2,2161.1.1] xˆ (3 o k)*y*z*x*xˆ (3 o k)*u=x*x*y*z*u.
5012 [para_from,4898.1.1,2161.1.2.2.2.2] xˆ (3 o k)*x*y*z*xˆ (3 o k)*u=y*z*x*x*u.
5438 [para_from,5012.1.1,4919.1.1] x*y*y*y*z=y*y*y*x*z.
6046 [para_into,5438.1.2.2.2.2,4898.1.1] xˆ (3 o k)*y*y*y*xˆ (3 o k)*z=y*y*y*x*z.
6371 [para_from,6046.1.1,2373.1.1] xˆk*xˆk*xˆk*y*xˆ (3 o k)*z=x*y*z.
6425 [para_into,6371.1.1,1590.1.2] xˆ (k o 3)*y*xˆ (3 o k)*z=x*y*z.
6493 [para_into,6425.1.1.1.2,19.1.2] xˆ (3 o k)*y*xˆ (3 o k)*z=x*y*z.
7008 [para_into,6493.1.2.2,4813.1.1] xˆ (3 o k)*yˆ (3 o k)*xˆ (3 o k)*yˆ (3 o k)=x*y.
8584 [para_into,7008.1.1,22.1.1] x*y=y*x.
```

7. Methodology by Instantiation

Perhaps thirty years ago Woody Bledsoe, then at the University of Texas, sat in my office at Argonne National Laboratory as we discussed the formulation of algorithms. We talked of our approach to finding, for all positive integers *n*, a procedure that would accomplish the task at hand. It might have been a surprise to either or both of us to learn that what we did was consider the case *n* = 1, then, if successful, 2, then
3. If an algorithm was found for the values 1, 2, and 3, our approach, separately, was to attempt to general- ize the method. You might say we were instantiating *n* to these small values. Somewhat similarly, the early challenge—I offered focusing on *xx = e*, the identity element in a group, with the goal of proving commuta- tivity—can be met, and often is, by instantiating *x* to *yz* and proceeding with a proof.

In this section, you will learn of an approach that is concerned with a theorem, called T29, that involves positive integers *a*, *b*, and *c*, with the goal of proving, yes, again, commutativity. You will be asked what actions to take to deal with three positive integers. You will be revisiting T15, which involved a single positive integer. Will you suggest Beeson's approach to exponentiation? Will you have an interesting alternative? Will you offer induction? Well, the approach I took was based on instantiation. No, I cannot promise I completed the research in question. But what comes now might provide you with yet one more approach to proving a theorem that strongly resists proof finding.

The theorem, as noted, is T29, whose hypothesis is the following.

```
y*x = x * yˆa * xˆb * yˆc.
```

On the surface, this theorem might offer substantial difficulty if compared with T17. Indeed, here you are asked to cope with three positive integers, whereas with T17 you were coping with one, *n*. As you would suggest, I relied on the Beeson approach to exponentiation.
###Larry - what I meant was that above, you ask what the reader would do - use exponentiation? induc- tion? then you say you chose instantiation. But right here you say exponentiation. The goal was not only to prove commutativity for semigroups but to do so with a proof that was demodulation-free. As was the case with so many of the studies on which this notebook is based, I began with a Knuth-Bendix approach; and, in imitation of other successes, I did not include in the axiom set axioms those for left and right inverse. So, the domain in focus was monoids.

In less than 1 CPU-second, OTTER presented me with three Knuth proofs of respective lengths 33, 30, and 35. No, I was not using ancestor subsumption. The explanation for the three proofs rests with the last line of each, the following.

```
x*y=z*x.
x*y=y*z.
x=y.
```

Each of these three equations clearly contradicts the only member of the passive list, the denial of commu- tativity.

```
a*b!=b*a.
```

Did your glance at the three cited last lines, especially the third of them, give you pause? Did you draw the same, most unexpected, conclusion that I did?

I concluded that since *x = y* was deduced (everything equals everything), the monoid, hence group, must have order 1, must be the so-called trivial group if the hypothesis (of T29) holds. This conclusion sheds no light, or provides nothing that I could think of using, in the context of obtaining a proof of com- mutativity for semigroups satisfying the T29 hypothesis.

So, rather than devoting much thought to this possible puzzle, I plowed ahead, still aiming at a proof of commutativity for semigroups. For no particular reason, I sought, with Knuth-Bendix, a proof of com- mutativity after commenting out three of the unwanted (according to Beeson) axioms, the following.

```
% xˆ 0 = e.
% 0 o x = 0.
% x o 0 = 0.
```

Immediately, OTTER found a Knuth proof of length 19. If you are momentarily puzzled by this length, the explanation rests with the presence, in the proof, of both left and right identity and the axioms *x+0 = 0+x = x*. Still relying on Knuth (because of the increased likelihood of finding a proof quickly), next in order for me was to comment out the left and right identity axioms and seek the desired proof.

In less than 1 CPU-second, OTTER found a proof. Well, numbers can provide piquancy, promise, and pleasure. As suggested, added axioms can lead to shorter proofs. Conversely, the removal of axioms can lead to longer proofs. In this case, the proof that was obtained has length 31—nice, indeed, as an illus- tration of the implied study. I had, therefore, a proof for T29 for the variety semimonoids. The next and obvious move was to remove, by commenting out, the remaining two unwanted axioms, *x+0 = x* and *0+x = x*. If successful, the race would be won: A proof for semigroups would be in hand. But, no proof was found for semigroups, although I now had a proof for T29 in the context of semimonoids. You might enjoy experimenting with omitting, one at a time, each of the two cited axioms. I did not pursue this avenue but instead—finally, says the reader—turned to instantiation.

You may have surmised at this point what could be done. Indeed, I chose values for the three positive integers, *a*, *b*, and *c*. With the plan of experimenting with various triples, I began with 2, 3, 4. Also, with the intention of making progress quickly, I omitted all (input) axioms but associativity, and its flip, and the T29 hypothesis, and its flip. For the convenience of the reader, the following was the basis of the experi- ment, with a Knuth-Bendix approach to be used, with a level-saturation approach (chosen for reasons I can- not recall).

```
list(usable).
x=x.
x*   (y*z)   =   (x*y)*z.
(x*y)*z   =   x*   (y*z).
end_of_list.

list(sos).
y*x = x * y*y * x*x*x * y*y*y*y. x *
y*y   *   x*x*x   *   y*y*y*y   =   y*x.
end_of_list.

list(passive).
a*b != b*a | $ANS(thm29).
end_of_list.
```

In less than 1 CPU-second, Knuth conquered, returning to me a 31-step proof whose steps were peppered with the citing of demodulation. Even the first step of the proof relied on five demodulators, all the same, the flip of associativity. One of the twenty-nine deduced equations (two of the thirty-one were cited from the input) relied on twelve demodulators. All five equations that were just given in the three lists were used. Because of the demodulation, an attempt to produce, or convert, this proof to a proof relying solely on paramodulation would seem daunting. It was, and still is at the moment. If you would find intriguing the prospect of completing a demodulation-free proof of the 2,3,4 instance of T29, I offer you the following proof.

A 31-Step Knuth Proof of the 2,3,4 Instance of T29

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Fri May 25 19:48:11 2012
The command was "otter".  The process ID is 18252.
----> UNIT CONFLICT at   0.01 sec ----> 105 [binary,104.1,1.1] $ANS(thm29).

Length of proof is 31.  Level of proof is 14.

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

1 [] a*b!=b*a|$ANS(thm29).
3 [] x*y*z= (x*y)*z.
5,4 [copy,3,flip.1] (x*y)*z=x*y*z.
6 [] x*y=y*x*x*y*y*y*x*x*x*x.
7 [copy,6,flip.1] x*y*y*x*x*x*y*y*y*y=y*x.
9 [para_into,7.1.1.2.2.2.2.2.2.2.2,4.1.1,demod,5,5,5,5,5] x*y*z*y*z*x*x*x*y*z*y*z*y*z*y*z=y*z*x.
11 [para_into,7.1.1.2.2.2.2.2,4.1.1,demod,5,5,5] x*y*z*z*x*y*x*y*x*y*z*z*z*z=z*x*y.
14,13 [para_from,7.1.1,4.1.1.1,demod,5,5,5,5,5,5,5,5,5,flip.1]  x*y*y*x*x*x*y*y*y*y*z=y*x*z.
15 [para_into,9.1.1.2.2.2.2.2.2,7.1.1] x*x*x*x*x*x*x*x=x*x*x.
18,17 [para_into,11.1.1.2.2.2.2,7.1.1] x*x*x*x*x*x=x*x*x.
20,19 [back_demod,15,demod,18] x*x*x*x*x=x*x*x.
22,21 [back_demod,17,demod,20] x*x*x*x=x*x*x.
26,25 [back_demod,7,demod,22] x*y*y*x*x*x*y*y*y=y*x.
27 [para_into,13.1.1.2.2.2.2.2.2.2.2,13.1.1,demod,14,flip.1]  x*y*x*x*x*x*x*x*x*x*z=x*y*z.
30 [para_into,13.1.1.2.2.2.2.2.2.2,9.1.1,demod,22,26,22,22,22,22,22,22,22,22,22,flip.1]  x*y*x*x*x=x*y.
32 [para_into,13.1.1.2.2.2.2.2.2,13.1.1] x*y*y*x*x*x*y*y*z=y*x*y*y*y*y*y*y*z.
33 [para_into,13.1.1.2.2.2.2.2,9.1.1,demod,22,22,22,22,22] x*y*y*x*x*y*y*x=y*x*x*x*x*y*y*y.
41,40 [para_from,21.1.1,13.1.1.2.2.2.2.2.2.2,demod,22,26,flip.1]  x*y*x=x*y.
45,44 [para_from,21.1.1,4.1.1.1,demod,41,5,41,5,flip.1]  x*x*x*y=x*x*y.
46 [para_from,21.1.1,13.1.1.2.2.2.2.2.2.2.2,demod,41,41,41,41,45,41] x*y*y*x*x*y*y=y*x*y*y.
50 [back_demod,33,demod,41,45,45] x*y*y*x*x*y*y*x=y*x*x*y*y.
53,52 [back_demod,30,demod,41] x*y*x*x=x*y.
59,58 [back_demod,32,demod,45,45,45,45,45] x*y*y*x*x*y*y*z=y*x*y*y*z.
61,60 [back_demod,27,demod,45,45,45,45,45,45] x*y*x*x*z=x*y*z.
62 [back_demod,46,demod,53] x*y*y*x*x*y*y=y*x.
65,64 [back_demod,50,demod,59,61,flip.1]  x*y*y*x*x=x*y*y.
66 [back_demod,58,demod,61] x*y*y*x*x*y*y*z=y*x*z.
68 [back_demod,62,demod,65,65] x*y*y=y*x.
74,73 [para_from,40.1.1,4.1.1.1,demod,5,5,flip.1]  x*y*x*z=x*y*z.
89,88 [para_into,66.1.1.2.2.2.2.2.2,4.1.1,demod,5,74,74,5,5,74,74,5] x*y*z*x*x*y*z*u=y*z*x*u.
91 [para_into,66.1.1.2.2.2.2,40.1.1,demod,5,41,5,41,41,74,41,5,41,5,flip.1]  x*x*y*y=x*y.
95,94 [back_demod,66,demod,89] x*x*y*z=x*y*z.
97,96 [back_demod,91,demod,95] x*y*y=x*y.
104 [back_demod,68,demod,97] x*y=y*x.
```

As for hints for the bit of research involved, I have made some progress. Indeed, one of the runs I made relied on assigning the value 5 to heat, with the program focusing on just the parents and demodulators that were used to deduce an early step, in the 31-step proof, that was resisting conversion.

However, I now had, for semigroups, a proof of one of the possible instantiations of T29, namely, 2, 3, and 4 for the respective (implied) integers *a*, *b*, and *c*. Yes, as you hasten to comment, no use of any of the Beeson axioms for exponentiation was required. My current efforts (in the context of converting the
31-step proof), in many runs, produce too many proofs of the same intermediate step, a target equation from among the twenty-nine that were deduced to complete the cited 31-step proof.

With that success in hand, I sought proofs for other triples, namely, 3,4,5,; 4,5,6; and 8,7,6. Each experiment, with Knuth, yielded a proof in less than 1 CPU-second. As I write this, I thought that some might wonder whether the success was related to the closeness of the values chosen in the triple investigated. Therefore, to allay any such suspicion, I just tried 11,8,6, and, yes, again, success was the result, in just over 17 CPU-seconds, completing a 44-step proof. All of the proofs for instances of T29 I studied apply to semigroups, with no use of axioms for exponentiation. Yes, it seems logical that more time is required with larger values for *s*, *b*, and *c*.

My conclusion, and likely yours, is that a proof exists for T29, in its full generality with no specific assignment to its positive integers, for semigroups, a first-order proof relying solely on paramodulation (with no demodulation). At this time, I leave to you the challenge of producing this sought-after proof.

I almost forgot a peculiar addendum. Specifically, the experiment, run, that yielded the 31-step proof did *not* also deduce *x = y*, which would have implied that the semigroup has order 1. In view of that deduc- tion cited earlier, now why is that? Is there some problem with the clauses for exponentiation, for example?

8. Methodology at Work

In 1982, when I was writing my first book, my division director, Paul Messina, made an observation that, at the time, I thought was idealistic and even sophomoric. I was so wrong. Messina told me that I would learn from writing the book. Well, the foregoing I am about to prove has taught me more about proving the type of theorem in focus throughout this notebook. To provide evidence, I shall revisit theo- rems I studied last February, six months ago.

I returned to a study of T15, in its full generality, which involves exponents. For your convenience, I note that the hypothesis of T15, in its full generality, is the following.

```
y*x = (x* y) ^ n.
```

I decided, this time, in contrast to the earlier attempts, to stay with Knuth-Bendix. In successive runs, some of which were based on level saturation, I was able to find proofs by deleting, one after the other, items among the seven unwanted axioms. Success was mine until I finally obtained a proof with but one of the unwanted still being used, namely, *0+x = x*. In particular, in less than 1 CPU-second, OTTER produced what it asserted was a 10-step proof. My efforts to find a proof that was free of all seven unwanted have, so far, failed. So, I now have a proof of T15 in its full generality for semimonoids. Ideally, if T17 is the stan- dard, a proof can be found that depends on just associativity, and its flip, and the hypothesis of T15, and its flip. I was unable to locate that proof, if it exists, which may present a challenge. The proof I now have depends on the following axioms from the input.

```
6 [] a*b!=b*a|$ANS(thm15).
8 [] x*y*z= (x*y)*z.
10,9 [copy,8,flip.1] (x*y)*z=x*y*z.
16 [] x^ (y+1)=x^y*x.
20 [] 0+x=x.
27,26 [] x o 1=x.
33 [] (x^y)^z=x^ (y o z).
35 [] x*y= (y*x)^n.
37,36 [copy,35,flip.1] (x*y)^n=y*x.
```

I offer you two challenges. The first is to emulate what occurred with T17 in the context of relying on fewer, perhaps none, of the axioms for exponentiation. Failing that, the second challenge is to find a proof that avoids the remaining unwanted axiom, thus obtaining a proof for semigroups, a first-order proof. After all, if I understand things, Stein has such a proof, one based on induction. You could add a third challenge, that of producing a demodulation-free proof, say, for semimonoids; I have not attacked that challenge myself at this time, preferring to revisit other theorems.

Next, perhaps in order, for my revisiting was T13, with the following hypothesis.

y*x = (xˆp)* (yˆq).

Here, implicitly, *p* and *q* are positive integers. As a review, the goal is to prove commutativity, which should be easily reached with Knuth if no other requirements are given. That goal, as was true in the begin- ning of each voyage (named by the theorem under study), was for groups. As is noted throughout, the two axioms for inverse were seldom included; therefore, the initial study, in each journey, focused on monoids. The additional requirements were to prove the theorem in question for semigroups, and, if the urge was present, to obtain eventually a demodulation-free proof (dependent on paramodulation alone, and not on Knuth). Always, rather than induction, the intent was to find first-order proofs. In February, when I initi- ated my study of T13, I quickly turned to proofs avoiding demodulation, non-Knuth. I did find such a proof, one of length 15, with the following axioms to be eventually avoided, if all went according to plan.

2 [] e*x=x.
3 [] x*e=x.
5 [] xˆ0=e.
14 [] 0 o x=0.

The progress that I am about to report, after recounting what occurred in March and April, exhibited
Messina's observation about learning: namely, I stuck with Knuth.

In March, my attempts to remove all unwanted did not succeed. But in April I did obtain an 18-step proof that relied on just three unwanted axioms, the following.

8 [] x+0=x.
9 [] 0+x=x.
11 [] 0 o x=0.

In other words, I was able to move from group theory, past monoids, to semimonoids. But I got no fur- ther—that is, until here and now in June, where I learned from experience and followed what I have said to you earlier in the notebook, namely, Knuth proofs are easier to find.

I began this revisiting with no restrictions placed on the axiom set, recognizing that perhaps all of the seven unwanted would be present in the proof I would find. In less than 1 CPU-second, OTTER found the sought-after Knuth proof, a 7-step proof that did rely on four of the unwanted, the following.

4,3 [] e*x=x.
6,5 [] x*e=x.
11,10 [] xˆ0=e.
24 [] 0 o x=0.

Immediately, I turned to the called-for experiment, removing (by commenting out) the two axioms for left and right identity. OTTER did not return a proof. So I, instead, commented out the following three equa- tions.

% xˆ 0 = e.
% 0 o x = 0.
% x o 0 = 0.

In particular, I allowed the program to use, in a proof if found, the left and right identity axioms and the two of the respective from *x+0 = x* and *0+x = x*. Indeed, in the 7-step proof, of the unwanted, the following two were relied upon.

11,10 [] xˆ0=e.
24 [] 0 o x=0.

OTTER quickly succeeded, and you now will see how the removal of axioms can affect proof length. Specifically, the proof that was found has length 43.

I then removed, from the four unwanted and still possibly participating, the axiom for left identity, and you come to what might be termed an anomaly. The program succeeded; but, instead of finding an even longer proof, OTTER presented me with a 37-step proof. That proof relied on the right identity axiom *xe = x* and on the axiom *x+0 = x*. The obvious move to make, since I was aiming at semigroups, although perhaps I would be stopped at semimonoids, was to continue on this path but now remove the right identity axiom. In just under 182 CPU-seconds—did I fail to wait long enough in the earlier experi- ments?—OTTER found a 287-step proof, a proof relying on, in addition to various Beeson axioms for exponentiation, just *0+x = x* from among the unwanted seven. So I was almost there, almost to semi- groups. Of course I had not yet considered, in this revisiting, the prospect of turning from Knuth to paramodulation alone. Of the 287 deduced equations, in 192 of them demodulation occurred. Sometimes, more than five demodulators were applied, although not necessarily distinct demodulators. Efforts to find a proof for T13 for semigroups have, at this point in time, failed; but some odds and ends merit mention. For one item, I tried for a proof with none of the unwanted but right identity. The program found a 37-step proof, which of course applies to monoids, which is not the goal. I had in hand proofs for specific values assigned to *p* and *q*, for example, *p = 4* and *q = 3*. Such proofs were delightfully pure, as a proof of T17 eventually was. Indeed, associativity and the T13 hypothesis sufficed; no axioms for exponentiation were needed. I recently tried *p = 4* and *q = 6*, with Knuth, and a 29-step proof was obtained, relying on just associativity and the T13 hypothesis and their flips. Level saturation and the other main top-level approach, complexity mixed with level saturation through the use of McCune's ratio strategy, work. With effort, I assume I could convert each instantiated proof to a proof free of demodulation. More important, being so close to obtaining a proof for T13, close in various respects, leads easily to the conjecture that a first-order proof for T13 can be found for semigroups and then, with much work perhaps, converted to a proof based on paramodulation alone. I have obtained an 11-step proof with, of the seven unwanted, just right identity and a few axioms for exponentiation. I tried this experiment in the perhaps-wild hope of gaining insight into an approach that would lead to the desired semigroup proof. I then returned to the experiment that avoided six of the seven unwanted, just relying on *0+x = x*, and added the use of ancestor subsumption. In less than 5 CPU-seconds, the program found three proofs, the last and shortest having length 97. That run relied on 286 resonators taken from the cited 287-step proof. Had you wished to try for an even shorter proof, you could—as I did not at this time—have replaced the 286 resonators with those corresponding to the deduced steps of the 97-step proof. You could also have applied one or more methods I use for finding shorter proofs, such as that based on blocking proof steps one at a time with demodulation. I experimented with ancestor subsumption for this theorem without using any resonators. So far, that run has found five proofs of decreasing lengths. The longest and first has length 258, and the shortest and last has length 205. The first proof was completed in just over 4923 CPU-seconds, and the last in just under 5156 CPU-seconds. Yes, more and more evidence accrues for relying on Knuth in the beginning. Nevertheless, I still prefer proofs free of demodulation, because an examination of such a proof often leads to insights in various con- texts. (By the way, especially because of Stein's interest in the length of deduced equations, the proofs I am finding for the different theorems I study do not rely on complex equations in general, whatever complex means.) I made a run without ancestor subsumption and with *x+0 = x* avoided, finding a proof of length
149 relying on various axioms for exponentiation, in less than 4 CPU-seconds. Still avoiding the use of ancestor subsumption, but now allowing just one of the seven unwanted to participate, *0+x = x*, the pro- gram found, after much more time, a proof of length 113. Specifically, just over 411 CPU-seconds were required. By citing these various experiments, I may provide you with some intuition that will in turn lead you to the formulation of more methodology. To answer a question that likely has occurred to you, by using the methodology developed and presented in this notebook, I was able to reach more goals than I did earlier, in February, March, and April.

As yet another example, I now turn to a theorem I call T25. The hypothesis of T25 is the following.

y*x = x * y^a * x^b * y.

Again, the goal was to prove commutativity. The plan was to reach the variety, semigroups. The hope was to avoid instantiation and, instead, reach the goal by using Knuth-Bendix.

With another theorem to prove, how does a researcher choose the approach? In the case of T25, I was no doubt influenced by my wish to quickly obtain a demodulation-free proof, a proof that I could then refine or modify so that the domain in focus was semigroups. Therefore, although my studies of Stein theo- rems strongly suggested that a Knuth-Bendix approach gave me the best chance of quickly finding some proof, I instead turned in late February 2012 to relying on paramodulation alone, avoiding the use of demodulation. To increase the likelihood of finding a proof to modify, I permitted OTTER to use any and all of the unwanted seven, unwanted from the viewpoint of semigroups. Well, the program succeeded, but not quickly, returning to me a 24-step proof in just under 2451 CPU-seconds, a proof relying on the follow- ing four of the seven to eventually be avoided.

```
2 [] e*x=x.
3 [] x*e=x.
5 [] xˆ0=e.
15 [] x o 0=0.
```

In that same run a second proof was completed, one of length 22 relying on the same four axioms. (I won- der: If I had included at the start the axioms for left and right inverse, would I have learned things that would have proved of value?)

Because it seemed clear that my chances for reaching my goal, a demodulation-free proof of T25 for semigroups, would be increased by concentrating on a shorter proof, if such could be easily found, I made a second run, using twenty-two resonators corresponding to the deduced steps of the shorter of the two cited proofs. Yes, just to be absolutely clear, demodulation was indeed not present. That run quickly yielded an
18-step proof, again relying on the cited four unwanted. Indeed, in less than 17 CPU-seconds, illustrating the power of the resonance strategy, three proofs were completed of respective lengths 22, 20, and 18.

Now I cannot be certain, because I conduct many studies of many theorems more or less in parallel, but I believe various attempts to get close to my goal failed in the following sense. I did get a proof, of length 29, that applied to semimonoids, by commenting out the two axioms of left and right identity. But that proof now relied on a different four of the seven unwanted, the following.

```
3 [] xˆ0=e.
9 [] x+0=x.
10 [] 0+x=x.
12 [] 0 o x=0.
```

A week’s worth of on-and-off experimentation left me rather far from my goal, although I did have a rea- sonably short demodulation-free proof. So I ceased searching for the sought-after proof for semigroups until late June, when I resumed various studies.

Consistent with the experiments of late June, I returned to Knuth, in my study of T25. Perhaps because of what I had learned in late February and early March about the seven unwanted, I permitted the program to use just the following four.

```
x+0 = x.
0+x = x.
0 o x = 0.
x o 0 = 0.
```

In other words, forsaking demodulation-free, perhaps for a long time, I concentrated at this point on semi- monoids. The approach worked quickly, returning to me almost immediately a 90-step proof. All four of those to eventually be avoided, if I were to reach the domain of semigroups, were still present. The obvious experiment, based on earlier results with other Stein theorems, called for commenting out the third and fourth of the cited items and seeking a different proof. And we come to another oddity: OTTER found a proof; but rather than being substantially longer, expected because of having fewer axioms to work with, the proof that was completed has length 43. Further, that proof relies on just the following unwanted.

```
14 [] 0+x=x.
```

Yes, I was much encouraged. Was I about to prove another Stein theorem, but, more important, for semi- groups?

All of these particular experiments were occurring over a span of but a few minutes. I commented out the cited unwanted and made the next run. And I found out why those February-March experiments had so much difficulty. Specifically, a quick glance at the proof—yes, a proof was found—showed that the items, before those that were deduced, cited no use of 0 and no use of *e*. The mountain had been climbed. I had reached semigroups. Further, and you see why I had so much difficulty earlier, the proof that was pre- sented to me has length 157. The proof relies on twelve of the Beeson axioms for exponentiation, associa- tivity and its flip, and, of course, on the T25 hypothesis and its flip. You guessed it: I have not yet even begun to seek a demodulation-free proof of T25 for semigroups. Far easier, when the goal is finding a shorter proof, is the approach that invokes ancestor subsumption. I did so, without using any resonators from the 157-step proof, and the program found three proofs; the last and shortest has length 100.

Next for discussion is a theorem I call T19. That theorem, in contrast, say, to T17 or T25 or T29, might at quick glance appear to offer different obstacles, as the form of its hypothesis suggests.

```
y*x = x^n * y * x * y.  %  T19 hypothesis
```

What I have in mind is that the right side of the equality does not begin with an isolated variable, such as *x*.

```
y*x = x*y^n x*y.  %  T17
y*x = x * y^n * x^m * y.  %  T25
y*x = x * y^p * x^q * y^r.  %  T29
```

To study T19, in mid-February 2012, I relied on Knuth-Bendix, including in the input all of the Beeson axioms for exponentiation, associativity, left and right identity, and, of course, the T19 hypothesis to drive the reasoning. OTTER produced a 36-step proof, deducing *x = y*, which might sound familiar. The step before the final one asserted *x = e*, the identity element. So, as occurred in an earlier study, the only mon- oid, or group, satisfying the T19 hypothesis is the trivial group, that of order 1. When I turned immediately to avoiding demodulation and removing axioms, if my notes are correct, I got nowhere.

When I renewed my study with the methodology in hand, the use of Knuth yielded the same result, in the presence of all the usual axioms. Why wait, I said, and commented out all of the seven unwanted axioms except *x+0 = x* and *0+x = x*. As the program retained clause (5088), a proof of commutativity was completed, a proof of length 118, relying on both of the included unwanted axioms—unwanted if semi- groups was the intended domain. Additional experiments with T19 in its full generality, including relying on one of the two remaining unwanted axioms along, yielded nothing. Well, you perhaps saw the not-so- well-veiled hint. In particular, I turned to instantiation in order to possibly gain some insight into the larger picture. When I assigned specific values to *n*, instantiation, 2, 3, 5, and 9, OTTER presented me with a proof for semigroups; indeed, in each of the proofs, none of the unwanted axioms was relied upon. Put a bit differently, all that was needed, in each case, was associativity and the T19 hypothesis. Some of the proofs that were found provided me with satisfaction (I like large numbers). Specifically, one of the proofs required more than 6579 CPU-seconds; one required more than 9006 CPU-seconds; one required just over
37374 CPU-seconds. The secret to such longevity rests totally with the avoidance of demodulation.

A closer look yielded an oddity; in particular, although permitting demodulation, often a great deal of it, tends to be coupled with smaller, sometimes much smaller, cited proof lengths, such is not the case here. What is seen, when the statistics are read, is much larger clause numbers upon proof completion, when Knuth is not employed. For example, one of the longest in CPU time is in the context of a 39-step proof of T19 with *n* assigned the value 3; and, far more impressive, the clause number cited for the contradiction is (816390). If you wish to view numbers larger than I may have ever seen, you need only consult my experi- ments with T19 with *n* assigned the value 5. Indeed, the proof I sought was based solely on paramodula- tion. The eventual result—eventual is clearly the correct designation—was a 21-step proof that completed with the retention of clause (8207453), in just under 661697 CPU-seconds. In answer to a natural question, my memory asserts that I have never before known of a proof that required, for completion, a clause num- bered in excess of 5,000,000.

You have possibly shared my conclusion? It strongly appears that a proof of T19 in its general from, with exponent *n*, must exist for semigroups, a proof that is first-order. Perhaps—and I have not yet deter- mined how—the proofs for *n = 2*, 3, 5, 9, and more could be used in some way to produce this sought-after proof. You might, for example, take some of the deduced steps of the proofs and replace various subex- pressions by *x^n*. Well, for now the challenge is yours, that of producing a proof of T19 in its fully general form, a proof that is first-order and that relies on none of the unwanted seven and, therefore, that applies to semigroups. Whether you use the Beeson axioms for exponentiation or some formulation of your own, should you succeed, I would enjoy hearing from you by e-mail.

9. Bigger Challenges, Harder Theorems

At the simplest level, a challenge is bigger, according to what you are reading, often because the author is having trouble meeting it. That a theorem is harder to prove typically means that the author, in this case, me, is achieving little in its consideration even with the assistance of that powerful program OTTER. Perhaps it is wisest to start with a medium-size challenge.

Stein's theorem, which I call T21, again has the goal of proving commutativity from the following hypothesis.

x*y=y^a*x^b*y*x.

If I have all correct, Stein has a proof of this theorem for semigroups, an induction proof. Of course, as is so patently clear by now, I vastly prefer first-order proofs. When I, in emulation of what has most worked, chose a Knuth-Bendix approach, omitting the unwanted seven, nothing occurred. So I turned to the other end of the spectrum, permitting not only the unwanted seven axioms to be used but also the following two axioms for inverse.

i(x)* x = e.
x*i(x) = e.

Now what do you think resulted? Indeed, did OTTER find some proof, any proof, of anything? And, if it did, which of the various (input) axioms were used?

Well, McCune's program presented me with an 18-step proof. What domain did the proof apply to? The proof applied to group theory; indeed, the axioms used to obtain the proof consisted of left and right identity, left and right inverse, associativity and its flip, and the T21 hypothesis and its flip. None of Bee- son's axioms for exponentiation were used, which, yes, did surprise me a bit. Since the usual axioms for a group include dependent axioms, I removed right identity and right inverse (which are dependent on the remaining), and again OTTER returned a proof, one of length 23, still using no axioms for exponentiation. Now Veroff, with his hints, would have begun adjoining numerous so-to-speak lemmas, including many he knew did not hold, and then iterating; in other words he would have turned to his sketches, which is indeed a powerful approach. In contrast, I gave up, leaving you the challenge of finding a first-order proof of com- mutativity, for semigroups, for T21. Such a proof must exist. By the way, if you consider instantiation, as I did, for the values 4 and 6, respectively, for the two exponents in the T21 hypothesis, all works perfectly for semigroups.

Next in order is a theorem of Stein that I call T23, with the following hypothesis.

y*x = x^a* y * x^b * y.

When Knuth is applied to its consideration, along with left and right identity axioms and axioms for left and right inverse, as well as others among the so-called unwanted, after proving commutativity, the pro- gram proves in the same run with a 29-step proof that *x = y*. In other words, as you saw earlier in this note- book, for the T23 hypothesis to hold in groups, the group must have order 1. In a second run, with the omission of the following three unwanted axioms, again the program proves *x = y* with a 49-step proof.

x^ 0 = e.
0 o x = 0.
x o 0 = 0.

And your challenge now appears. In particular, my brief experiments with further omissions yielded no proofs of any type. Since the theorem does apply to semigroups, as Stein notes, a first-order proof must exist, a proof that depends on no more than associativity and the T23 hypothesis, with perhaps the inclusion of their flips. Yes, I did try instantiation with the values 3 and 3, and OTTER returned to me a 35-step proof almost immediately, a proof for semigroups.

Finally, a coup de grace, a theorem for which I have made no progress, outside of a brief visit to instantiation. The theorem is T31, with the following hypothesis.

y*x = x^a* y * x^b * y^q.

Does the difficulty lie with three exponents in the hypothesis? Well, if you assign the respective values 2, 3, and 5, OTTER quickly finds a proof, a proof of length 31. As I expected, all that was needed was associativity, its flip, the T31 hypothesis, and its flip. To emphasize the point, no input axioms concerning exponentiation were needed. Now, why does this situation continually occur when specific values are assigned to exponents in the type of theorem featured here?

With the preceding sections as well as this one, I am sure you can supply your own challenges. You might even turn to induction, but, of course, not with the aid of OTTER. Stein clearly did.

This notebook soon will come to a close, but first—perhaps in tacit acknowledgment of the length of this notebook and the many, many ideas presented—I shall summarize some of the main points. Of course, I may well also introduce some new results. My hope is that these will provide you with ideas for research and, if all goes well, lead you to startling new results for subsequent publication.

10. Challenges Summarized and Other Observations Presented

The following list of Stein identities may provide you with questions to answer and challenges to meet. I have studied these identities myself, not always reaching the goals that (so I understand) can be met.

T13: yx = x^p y^q.
T15: yx = (xy)^n.
T17: yx = xy^n xy.
T19: yx = x^n yxy.
T21: y*x = x^n * y^m * x * y.
T23: y*x = x^a * y * x^n * y.
T25: y*x = x * y^n * x^m * y.
T29: y*x = x * y^p * x^q * y^r.
T31: y*x = x^a * y * x^p * y^q.
T33: y*x = x^ (3 o k) * y^ (3 o k) * x^ (3 o k) * y^ (3 o k).
T35: y*x = x^ (3 o k + 1) * y^ (3 o k + 1) * x^ (3 o k + 1) * y^ (3 o k).
T37: y = (x * y)^n * x. % and also should prove c*c = d*d
T39: y = (x * y)^ (2 o n) * x. % to prove a = b

In the spirit of an overview of what has been presented in the earlier sections, you might now wonder about possible algorithms. From what I know, and from what my colleague Beeson knows, for the type of problem in focus here, where you are asked to prove one identity from another, no algorithm exists. Never- theless, from the evidence I offer, you see that such problems can be solved —sometimes solved readily. And sometimes the resulting proof evinces surprising properties.

Paramodulation. If you were intent on finding a difficult research problem to solve, a problem whose solution would merit publication, you could consider the formulation of strategies to effectively control paramodulation. Arguably, as the evidence of decades shows, paramodulation is immensely valuable. But it is also far too fecund. Blocking paramodulation *from* or and *into* variables sharply reduces possible fecundity. However, far more is needed.

Ancestor Subsumption. Introduced by McCune many years ago, ancestor subsumption was (at least from my research viewpoint) a boon, a gift of immeasurable value. Its use provides an excellent strategy when seeking proofs shorter than that in hand, and, more important from the larger picture, shorter than found in the literature. However, this procedure is not a panacea. No procedures are, in fact. I have runs from diverse areas of study in which, with the use of ancestor subsumption, the program found proofs of, say, lengths 32, 29, 25, 34, 41, 50, 36, and the like. Ironically, my research has unearthed many situations in which the replacement of a shorter subproof, **P**, of one member of a conjunction, say *B*, by a substantially longer proof of *B* results in the discovery of a proof of the conjunction shorter than that relying on **P**. On the so-called other side of such experiments is the case in which proofs of members of a conjunction are found that are shorter and still shorter while the proofs that are found of the entire conjunction get longer and still longer.

Number of Axioms and Proofs. In general, the greater the number of (input) axioms, the shorter the proof that will result, when and if a proof is found. If, for example, I had usually included in my studies offered here the axioms of left and right inverse, I would have, in the beginning of each journey, found a proof shorter than I typically did. I was likely influenced by my wish to quickly move to the domain (variety) of semigroups, an area in which inverse is not present. When you are conducting a study and you are omitting various axioms that are justified with the result that nothing of interest is happening, you might try includ- ing some or all of those omitted axioms to get started. As you find in this notebook, I usually began with an axiom set weaker than group theory (by omitting the axioms concerned with inverse), monoids to be specific, and, if all went ideally, concluded with a result that applied to semigroups. On the other hand, a study of group theory, for example, in which inverse axioms are omitted still applies to groups. Certainly a useful approach is to begin with a full set of axioms and, as successes occur, gradually drop more and more of the axioms, an approach somewhat reminiscent of Veroff's sketches. So, you might, even with the goal of proving a theorem for semigroups, begin with a full set of axioms for group theory, including those that are dependent, and see what happens. McCune often studied group theory without including right inverse and right identity axioms; they are dependent on the remaining. In contrast, I generally included those two when I sought new results in group theory. You thus witness contrasting styles between two researchers who frequently worked together closely and effectively.

Methodologies. To many readers, the proofs, the theorems, and the commentary will not be the key; rather, the methodology will be what counts, for it can be applied to many diverse areas of mathematics and logic. This notebook has discussed the use of cramming, demodulation, blocking of proof steps, lemma adjunc- tion, and level saturation versus the use of McCune's ratio strategy, among other methodologies. You have also been treated to a discussion of a myth concerning a straightforward approach to finding shorter proofs. By now, especially if you have browsed in earlier notebooks, you are sharply aware of the view I share with others that asserts the difficulty of proof finding, proof shortening, conjecture testing, and the like, a diffi- culty that can be overcome only with the use of diverse strategies. Indeed, almost five decades ago, the set of support strategy was formulated to address the obstacle that reasoning programs faced focusing on the full set of axioms, special hypothesis, and denial of the theorem under study. Without the use of that strat- egy, a program could spend almost all of its time exploring the general domain from which the theorem was taken. Of a quite different nature—and I do not believe I am trying to entice you to read other notebooks— the subformula strategy was formulated to enable an automated reasoning program to cope with very long equations and formulas and to cope with expressions relying on many distinct variables.

Demodulation. In this writing, demodulation plays a key role, especially when a Knuth-Bendix approach is in use. But, in the following sense, demodulation offers the researcher, sometimes, monumental obsta- cles to overcome. In particular, you can obtain from a program a proof in which a single step relies on fif- teen or more demodulators, not necessarily distinct. When the goal is to produce a proof relying solely on paramodulation, avoiding any use of demodulation, such a step can be a giant to kill. If you have read much mathematics or, even more, refereed papers on some area of mathematics intended for publication, you are indeed familiar with the use of demodulation, although it is not cited as such. Often, for example, a step occurs in a proof in such a paper that relies, implicitly, on a number of applications of demodulation for, say, associativity, right identity, left inverse, and more. In order to see precisely what occurs in a proof, which lemmas are used in which ways and the like, a demodulation-free proof is best.

An Exercise in Proving Commutativity. If you return to T15a, whose hypothesis says that the square of $xy = yx$, you can (as I did) prove commutativity. A nice exercise, for the person who likes models, asks for the smallest group that fails to satisfy the T15a hypothesis. If you read no further, the exercise is yours. If you instead wish the answer, I give it now. The smallest group that fails to satisfy the T15a hypothesis is the symmetric group on three letters, a group of order 6. How can you prove this? Well, that group is not commutative, and the rest of the details I leave to you.

Hot List. In this notebook, I have made references to the hot list strategy and to the dynamic hot list strat- egy, in the context of proof finding and in the context of proof shortening. For either activity, if you desire to apply a more sophisticated approach to your study, either strategy may serve you well. The notion of sophistication is just the right word, for your choice of the contents of the hot list, your choice of the value to assign to heat, and, if the dynamic hot list strategy is to be put to use, your choice of which newly deduced items are to be adjoined to the hot list each can have a monumental effect on the program's perfor- mance. For but two examples (to begin with), if you place too many elements in the hot list or if you choose unwisely what you place there, the program can get immersed in the deduction of information that is of little or no value. I do sometimes place associativity in the hot list, but with some trepidation. After all, with even reasonably long equations, associativity readily applies to each newly deduced item. There- fore, the inclusion in an input list(hot) of very general laws can be indeed dangerous. For a more potent inclusion that might bury the program, you need only consider placing one of the distributive laws for ring theory in the initial hot list.

On the other hand, a fine choice for an element of the initial hot list is the hypothesis, sometimes known as the special hypothesis, of the theorem. In ring theory, for example, you can prove commutativity when you know that *xxx =x/fR. This hypothesis, the cube of x* is *x* for all *x*, is the type of item that, when placed in list(sos) in the input, promotes finding a proof. If you consult published proofs that commutativ- ity holds for such rings, you will find that this property, *xxx = x*, is frequently used in proof steps. In logic, you will often find what amounts to heavy use of heat, in proofs by Meredith and by Lukasiewicz, in partic- ular. In classical propositional predicate calculus, Meredith provided the following single axiom, most likely, the shortest possible.

```
% Following is Meredith's single axiom.
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).
```

In 2004, I found a 51-step proof that derives, from the given Meredith single axiom, the Lukasiewicz 3-axiom system for this area of logic, a proof in which twenty-two of the fifty-one derived steps relied on the use of the hot list strategy. One of those twenty-two steps showed heat=4, which I shall immediately explain. The only member of the initial hot list was Meredith's single axiom. When a newly deduced for- mula was retained, it was immediately considered with the Meredith (for the two required parents) axiom to yield, say, *A*, with heat=1 as the designation. Next *A* was immediately considered with Meredith's axiom to yield *B*, with heat=2. Then *B* was considered with Meredith's axiom to yield *C*, with heat=3. Finally, *C* and Meredith's axiom were considered together to yield *D*, with heat=4. You might comment that a form of recursion was in use. In that 51-step proof, you would find a sequence of five consecutive steps with heat=1 or greater. (My colleague B. Fitelson sometimes assigned the value 8 to heat.) The assigned value instructs the program how much to visit and revisit members of the hot list, not necessarily the same mem- ber. You had best be careful with your assignment to the heat parameter; too large a value can drown the program. So you see that there are indeed traps with the (static) hot list strategy.

Still, as I write these paragraphs, I have found a proof by using this strategy, with an assignment of the value 1 to heat, a proof shorter than I knew of a day ago. I had tried ancestor subsumption, but to no avail. When I combined it with the hot list strategy, instead of the 21-step proof that I just reproduced with a run, I obtained a 19-step proof. The theorem is T17, in its purest form, with no use of axioms for expo- nentiation. The 19-step proof relies on three steps not present in the 21-step proof. Both proofs are demod- ulation-free, how nice! Thus you have a pleasing example of the value the hot list strategy offers, occasion- ally.

And now in focus, and most deservedly, is McCune's powerful dynamic hot list strategy. As noted, with this strategy you enable a program to adjoin, during a run, elements to the hot list. Its use addresses the situation in which you have identified some formula or formulas or equation or equations that you suspect are key to finding a desired proof. Further, you suspect that the identified objects are to be used repeatedly, as elements of the hot list are visited and revisited, if success is to be the result. OTTER offers what is needed. You place the identified objects on weight_list(pick_and_purge), each with an assigned weight less than or equal to the value you assign to the dynamic_heat_weight. The objects in focus are not available in the beginning, not included in the input; rather, they are such that you conjecture they can be deduced. Therefore, if and when such objects are deduced, you wish them adjoined to the hot list. It all works this way.

Imagine that you conjecture that the following equation, if deduced and retained, would be of much assistance in reaching your goal.

x*y*z^n*x*y*z=z*x*y.

(This equation was used four times in a 31-step proof of T17.) You first enter the following in weight_list(pick_and_purge).

weight(x*y*z^n*x*y*z=z*x*y,02).

You then add to the instructions the following.

assign(dynamic_heat_weight,-1).

Of course, you had best assign a value to max_weight that is greater than or equal to -1, for the key item to be retained. If the cited equation is deduced and retained, because it will be assigned a weight of -2, it will be adjoined, during the run, to the hot list. That newly adjoined element in the hot list will now be visited, and perhaps revisited depending on the assigned value to heat, as each new item is retained. In other words, McCune's impressive contribution of the dynamic hot list strategy is reminiscent of *if-then*. Indeed, if the program happens to deduce and retain an item you believe is crucial if used repeatedly, and if you are cor- rect, you can aid the search significantly with this strategy.

Other Theorems. Now I turn to other results, other theorems suggested to me by Stein. A glance at the various theorems discussed so far in this notebook shows that, at most, three (implicit) positive integers are present in a hypothesis. In a rather late e-mail, Stein suggested theorems for study whose hypothesis had, usually, four specific positive integers in the right side of the equality, the hypothesis, each greater than 1. For example, you are asked to prove commutativity for semigroups when the following equation holds.

x*y = y*y* x*x*x*x * y*y*y* x*x*x*x*x*x*x.

For me, this theorem asks that I prove, for semigroups, commutativity from 2,4,3,7, the hypothesis. OTTER proved the theorem in less than 1 CPU-second, with Knuth-Bendix, a proof of length 27. The proof relies on much demodulation. A frequently used demodulator is associativity; indeed, in one of the deduced steps, that demodulator occurs fifteen times. Hence, the conversion to a demodulation-free proof presents quite a challenge. Some of the demodulators, as expected, are equations that are deduced and used later.

A far more difficult theorem to prove, from the reactions of Stein in response to an e-mail of mine, is the following, which can be referred to as 5,6,4,3. The goal is again commutativity, and the hypothesis is the following.

x*y = y*y*y*y*y* x*x*x*x*x*x * y*y*y*y* x*x*x.

OTTER, to the total amazement of Stein, found a proof of commutativity from the given hypothesis in the first try, actually two proofs of respective lengths 86 and 53. The 86-step proof was completed in a bit over 82 CPU-seconds, and the 53-step proof was completed in a bit under 297 CPU-seconds. I was curious about the results of using the hot list strategy with much heat. Therefore, I made a second run with an assignment of 4 to heat. I was sure that far more CPU time would be required because of the program's continual referral to the hot list. After all, as you will see immediately with the following, the hot list con- tained a possibly dangerous—in the context of fecundity and time—associativity.

x*y = y*y*y*y*y* x*x*x*x*x*x * y*y*y*y*x*x*x.
y*y*y*y*y* x*x*x*x*x*x * y*y*y*y*x*x*x = x*y.

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

Well, to my surprise, less than 21 CPU-seconds were required; OTTER presented me with a 28-step proof, which is a nice example of how the hot list strategy can be of much aid. All of the Knuth proofs found for 5,6,4,3 are for semigroups, and none of the Beeson items for exponentiation were used. As for proof con- version, in that I had again used Knuth, the following deduced clause (equation) shows how difficult would be the task.

```
437 (heat=4) [para_into,420.1.1.2,12.1.1,demod,19,19,19,19,19,19,19,19,19,19,19,19,19,19,93,
19,19,19,19,19,19,19,93,19,19,19,19,19,19,19,93,19,19,19,19,19,19,19,19,19,19,19,19,19,19,93,
19,19,19,19,19,19,19,93,19,19,19,19,19,19,19,93,19,19,19,19,19,19,19,93]
x*x*x*x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*x*x*x*x*y*y*y=x*y.
```

A quick count says that demodulation was applied 70 times to obtain this equation. The most frequently used demodulator here is 19, the following.

```
19,18 [copy,17,flip.1] (x*y)*z=x*y*z.
```

You also see how powerful McCune's program is, to cope with such computations in such a short CPU time.

For those who enjoy history, my first exposure to McCune's implementation, in OTTER, of the hot list strategy was indeed eventful. I had been studying a 3-axiom system, that Church appeared to favor, for classical propositional calculus, a system actually due to Lukasiewicz. My goal was to find a short proof that derived from it a well-known (and different) 3-axiom system of Lukasiewicz. I had found four 22-step proofs but could get no further. When McCune called and asked me to test his implementation, I immedi- ately did so, focusing on the so-called Church system. To my utter delight, OTTER found a 21-step proof. Yes, others, such as Fitelson, Z. Ernst, and K. Harris, have used the hot list strategy most profitably in vari- ous areas of logic.

Level versus Length. For those interested in seeking shorter and still shorter proofs of some chosen theo- rem, you might keep in mind the following observation. When the level of your latest success is very close to the length of that success, say, 28 and 30, most likely you will be forced to pursue a different line of study. Indeed, when the level of a proof is close to its length, usually no further progress can be made in the context of proof shortening if you concentrate on such a proof.

Induction. OTTER provides essentially no assistance if you are intent on obtaining a proof by induction. As you have seen throughout this notebook, I always seek—even when a notebook is not in production—first-order proofs. I am, in fact, not clear about how to proceed in the context of proof shortening if given a proof by induction. Nor, to be totally open about it, do I enjoy induction proofs.

Using Lemmas. Now, if you would enjoy emulating an approach often taken by a mathematician—or if you are a mathematician and you would find intriguing to have some computer aid in this approach—a pro- gram like OTTER will serve you well. In particular, sometimes a mathematician suspects that a set of axioms implies some property. You could call such a situation a purported theorem. The approach that can be taken is to guess at lemmas along the way, guess about how a proof might proceed. If so, you then set out to prove the various lemmas and, if all goes well, fill in the gaps until you have a proof. With OTTER, you can apply this approach by placing in list(passive) the negations of the lemmas you suspect will be of use, and then instruct the program to prove as many as it can in a first run. You then take, for the next run, the last lines of each proved lemma or take all of the proof steps of the proofs and adjoin them to list(sos). You also place resonators that correspond to all of the proof steps in weight_list(pick_and_purge). You continue iterating in this manner until, if the gods are kind and you have made a reasonably accurate con- jecture about the nature of the sought-after proof, success is yours. What I have just described might remind you of consulting with another with the request of that person for proofs of thought-to-be-important lemmas. You are, in effect, providing an outline of a possible proof and instructing the automated reason- ing program to supply the details and needed subproofs.

Using Negations of Steps. Closely related is my practice of often placing negations of steps of some proof, for example, of a proof that relies on demodulation when the goal is to find a demodulation-free proof. I do not assume that all of the steps of the proof relying on demodulation will be found in the sought-after demodulation-free proof, but some of them might. When and if any are proved, to me, it signi- fies progress and that, perhaps, the desired proof is nearing completion. Similarly, I sometimes place nega- tions of the steps of some other proof, often somewhat related to the proof I am seeking, motivated by the thought that the two theorems in question might share, to be proved, lines of reasoning. Again, I am trying to measure progress, and, in many cases, I will use as adjoined lemmas proof steps of proofs of such inter- mediate targets.

Axiom Removal. Of a distinctly different nature is the topic of proof length and its relation to the set of axioms to be used. As you would conjecture, the removal of axioms, as you make one run after another, results, typically, in longer and longer proofs of the theorem under study. For example, as I attempted to prove commutativity from the hypothesis of T25, I began deleting, by commenting out, various axioms. At one point, the following two axioms of the so-called unwanted seven were still present.

```
x+0 = x.
0+x = x.
```

That run produced a 43-step proof of commutativity. In the next run, I removed, by commenting out, the first of the two cited axioms. That run yielded a 63-step proof. Then, when I removed the other item of the cited two, producing an input file that relied on none of the unwanted seven, OTTER found a proof of length 157. This 157-step proof, featured earlier in this notebook, does rely on Beeson axioms for expo- nentiation but does not rely on any of the unwanted seven; it, therefore, applies to semigroups. The axiom- removal procedure can, once in a while, lead you to the discovery that the theorem or theorems you are studying hold for varieties weaker than you thought. Indeed, you might have been initially studying group theory with some set of theorems of interest, only to discover that each is true for monoids and, eventually, find that each is true for semigroups. In the rarest of cases, you will bring to the world a new variety, a new area that merits study.

If you should begin a study of T13, or return to your study (if you made one), you might encounter another, more dramatic, example of proof length changing as axioms are deleted. If you begin with all seven unwanted, the following, you can find a 7-step proof of commutativity.

```
e*x  =  x.
x*e  =  x.
xˆ 0  =  e.
x+0 = x.
0+x = x.
0  o  x  =  0.
x o 0 = 0.
```

If you delete the last three of the cited seven (unwanted), you can find a 43-step proof. If you then follow this move by deleting the first two, those focusing on the identity e*e*, you reach drama—a proof of length
287 can be found. For the given three experiments, the first two complete in less than 1 CPU-second. The third requires almost 282 CPU-seconds. Maddeningly, you will find, if you conduct the appropriate experi- ments, that you are close to semigroups. Indeed, just the following axiom stands in the way.

```
0+x=x.
```

Further experimentation can enable you to find an 18-step proof, free of demodulation, and relevant to semimonoids in that but two of the unwanted seven are used, namely, *x+0 = x* and *0+x = x*.

11. Intriguing Input Files and Pleasing Proofs

You might have decided to attempt to meet one or more challenges offered in this notebook. To speed such a journey, I now supply various input files, to illustrate approaches discussed here. The added comments and discussion may indeed provide you with ideas that, eventually, lead to finding gold.

I also present some proofs that might lead you to meeting a challenge or formulating an approach not featured here. The commentary you find in this section may well supplement earlier discussions of methodology.

First in order is an input file that illustrates much of what you will need. This input file, whose elements I shall briefly discuss, is similar to one of the first I used as I begun a study of Stein theorems. The focus is on T15.

An Input File for Studying T15 in Group Theory

```
set(para_from).
set(para_into).
set(order_eq).
set(hyper_res).
% set(ancestor_subsume).
set(back_sub).
op(299, xfy, ^).
op(300, xfy, o). % multiplication of exponents
assign(max_weight,40).
clear(print_kept).
assign(max_proofs,-1).
assign(max_distinct_vars,9).
assign(pick_given_ratio,4).
assign(max_seconds,40).
assign(max_mem,84000).
% set(sos_queue).
assign(report,5400).

weight_list(pick_and_purge).
end_of_list.

list(usable).
x=x.
% group axioms
e*x = x.
x*e = x.
x* (y*z) = (x*y)*z.
% inverse axioms
i(x)* x = e.
x*i(x) = e.
% exponentiation
x^ 0 = e.
x^ 1 = x.
1+1 = 2.
x^ (xp+xq) = x^ xp * x^ xq.
x^ (xp+1) = x^ xp *x.      % deducible but perhaps useful to have explicit.
% exponents and addition
x+y=y+x.
x+0 = x.
0+x = x.
(x+y) + z = x + (y+z).
% exponents and multiplication
0 o x = 0.
x o 0 = 0.
1 o x = x.
```

```
x o 1 = x.
(x o y) o z = x o y o z.
x o (y + z) = x o y + x o z. (x
+ y) o z = x o z + y o z. x o y
= y o x.
(xˆy) ˆ z = xˆ (y o z).
end_of_list.

list(sos).
y*x = (x* y) ˆ n.
end_of_list.

list(passive).
a*b != b*a | $ANS(thm).
end_of_list.
```

Easiest to discuss is the material beginning with list(usable). Almost as an aside, the presence of *x = x* might seem odd. A wise move, when equality is present, is to include this clause because, occasionally, an experiment seeking a proof will deduce *t != t* for some term *t*. If and when such occurs, *x = x* comes into play. (For those interested in history, for quite a few years after I introduced paramodulation, I was often questioned about my definition of that inference rule, with the thought that I should have built reflex- ivity into the rule.) The elements that are input in list(usable) cannot be chosen, by OTTER, to initiate an application of any inference rule. They are used to complete some application. You find there the axioms for a group and Beeson's axioms for exponentiation, where "o" denotes multiplication (not within the group), and "ˆ" denotes exponentiation. The only element in list(sos) is the (clause equivalent of) hypothe- sis of T15. That element is used to initiate the search for a proof. Were you to run (use) the given input file, OTTER would return to you the following proof.

A Short Proof for T15 in Group Theory

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Mon Aug  6 14:51:50 2012
The command was "otter".  The process ID is 21266.
----> UNIT CONFLICT at   0.01 sec ----> 63 [binary,62.1,26.1] $ANS(thm).

Length of proof is 3.  Level of proof is 3.

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

2 [] e*x=x.
3 [] x*e=x.
25 [] y*x= (x*y)ˆn.
26 [] a*b!=b*a|$ANS(thm).
33 [para_into,25.1.1,3.1.1,flip.1] (e*x)ˆn=x.
57 [para_into,33.1.1.1,2.1.1] xˆn=x.
62 [para_into,57.1.1,25.1.2] x*y=y*x.
```

A quick glance at this proof shows that you have proved more than was intended. Indeed, the axioms for left and right inverse were not used to obtain this 3-step proof. Since the axioms involving the identity, *e*, were used, the proof establishes T15 to hold for monoids. As an aside, associativity was not even used.

But Stein was studying semigroups. Therefore, the natural experiment to conduct next is to comment out the two axioms for identity, and of course the two for inverse, and, if you wish to leap across a small river, also comment out the Beeson axioms for exponentiation. As I write these paragraphs, I did jump that river. The program, and the experiment, yielded nothing. Therefore, I commented back in the Beeson axioms; after all, that move is consistent with the study of semimonoids because of allowing some of the unwanted seven to participate. Success resulted, with the following proof.

A Proof for T15 for Semimonoids

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Thu Aug  9 09:58:38 2012
The command was "otter".  The process ID is 6213.
----> UNIT CONFLICT at   0.33 sec ----> 12116 [binary,12115.1,22.1] $ANS(thm).

Length of proof is 15.  Level of proof is 11.

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

2 [] x*y*z= (x*y)*z.
3 [] x^0=e.
4 [] x^1=x.
7 [] x^ (xp+1)=x^xp*x.
10 [] 0+x=x.
21 [] y*x= (x*y)^n.
22 [] a*b!=b*a|$ANS(thm).
23 [para_into,21.1.1,7.1.2,flip.1] (x*x^y)^n=x^ (y+1).
27 [para_into,21.1.2.1,21.1.1,flip.1] ((x*y)^n)^n=x*y.
44 [para_into,27.1.1.1,21.1.2] (x*y)^n=y*x.
82 [para_into,23.1.1.1.2,3.1.1] (x*e)^n=x^ (0+1).
733 [para_into,82.1.1,44.1.1,flip.1] x^ (0+1)=e*x.
9461 [para_into,733.1.1.2,10.1.1] x^1=e*x.
9576 [para_into,9461.1.1,4.1.1,flip.1] e*x=x.
9758 [para_into,9576.1.1,44.1.2] (x*e)^n=x.
9780 [para_from,9576.1.1,44.1.1.1] x^n=x*e.
10583 [para_into,9780.1.1,9758.1.1,flip.1] (x*e)*e=x.
11289 [para_into,10583.1.1.1,9780.1.2] x^n*e=x.
11349 [para_into,10583.1.1,2.1.2] x*e*e=x.
11973 [para_into,11349.1.1.2,9576.1.1] x*e=x.
12058 [para_into,11973.1.1,11289.1.1,flip.1] x^n=x.
12115 [para_into,12058.1.1,44.1.1] x*y=y*x.
```

This proof depended on just two of the unwanted seven. You can get farther along on the trail to semigroups; indeed, I have a proof that depends on just the equation *0+x = x* and none of the other unwanted seven. But, as said earlier in this notebook, I can get no further. Because Stein has proved T15 for semigroups, from what I understand, a proof by induction, the desired first-order proof, for semigroups, must exist, perhaps using the Beeson axioms. Since I cannot find such, I offer you that challenge.

But, just perhaps, you shout "Knuth". So, I first give you an input file for studying T15 with a Knuth-Bendix approach, and then add some comments.

Input File for Studying T15 for Semimonoids with Knuth-Bendix

```
% set(para_from).
% set(para_into).
set(knuth_bendix).
set(lrpo).
lex([e,f(x,x),g(x)]).
set(order_eq).
set(hyper_res).
%  set(ancestor_subsume).
set(back_sub).
op(299, xfy, ^).
op(300, xfy, o).  % multiplication of exponents
assign(max_weight,40).
clear(print_kept).
assign(max_proofs,-1).
assign(max_distinct_vars,9).
assign(pick_given_ratio,4).
%  assign(max_seconds,40).
assign(max_mem,84000).
%  set(sos_queue).
assign(report,5400).

weight_list(pick_and_purge).
end_of_list.

list(usable).
x=x.
%  group axioms
%  e*x = x.
%  x*e = x.
x* (y*z) = (x*y)*z.
%  %  inverse axioms
%  i(x)* x = e.
%  x*i(x) = e.
%  exponentiation
x^ 0 = e.
x^ 1 = x.
1+1 = 2.
x^ (xp+xq) = x^ xp * x^ xq.
x^ (xp+1) = x^ xp *x.      % deducible but perhaps useful to have explicit.
%  exponents and addition
x+y=y+x.
x+0 = x.
0+x = x.
(x+y) + z = x + (y+z).
%  exponents and multiplication
0 o x = 0.
x o 0 = 0.
1 o x = x.
x o 1 = x.
(x o y) o z = x o y o z.
x o (y + z) = x o y + x o z. (x
+ y) o z = x o z + y o z. x o y
= y o x.
(x^y) ^ z = x^ (y o z).
end_of_list.

list(sos).
y*x = (x* y) ^ n.
end_of_list.
```

```
list(passive).
a*b != b*a | $ANS(thm).
end_of_list.
```

If you use this input file, OTTER will present you with the following promising proof.

Knuth Proof for T15 for Semimonoids

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Thu Aug  9 10:05:23 2012
The command was "otter".  The process ID is 6032.
----> UNIT CONFLICT at   0.01 sec ----> 193 [binary,192.1,1.1] $ANS(thm).

Length of proof is 11.  Level of proof is 7.

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

1 [] a*b!=b*a|$ANS(thm).
3 [] x*y*z= (x*y)*z.
5,4 [copy,3,flip.1] (x*y)*z=x*y*z.
7,6 [] x^0=e.
12 [] x^ (y+1)=x^y*x.
16 [] 0+x=x.
23,22 [] 1 o x=x.
32,31 [] (x^y)^z=x^ (y o z).
35 [copy,12,flip.1] x^y*x=x^ (y+1).
40 [] x*y= (y*x)^n.
42,41 [copy,40,flip.1] (x*y)^n=y*x.
43 [para_into,41.1.1.1,35.1.1,demod,32] x^ ((y+1) o n)=x*x^y.
45 [para_into,41.1.1.1,4.1.1,demod,42,5] x*y*z=y*z*x.
81,80 [para_into,43.1.1.2.1,16.1.1,demod,23,7] x^n=x*e.
88 [back_demod,41,demod,81,5] x*y*e=y*x.
94 [copy,88,flip.1] x*y=y*x*e.
153,152 [para_into,88.1.1,45.1.1] x*e*y=x*y.
185,184 [para_into,94.1.1,88.1.1,demod,5,153,flip.1]  x*y*e=x*y.
192 [back_demod,94,demod,185] x*y=y*x.
```

This 11-step proof relies on but two of the unwanted seven. My experiments failed to yield, even with Knuth, the sought-after proof for semigroups, a proof that avoids the cited two unwanted axioms. As noted earlier, I did try Knuth with various values for *n* and found proofs for semigroups—quite encourag- ing. But I do not as yet know how to use those proofs to reach the semigroup goal for T15 in its full gener- ality. For example, if I assign *n* the value 9, to obtain the following equation to be substituted for the (fully general) T15 hypothesis, Knuth works almost immediately.

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

As for the items in the given input file, before list(usable), the (following) “assign” items are easily described.

```
assign(max_weight,40).
assign(max_proofs,-1).
assign(max_distinct_vars,9).
assign(pick_given_ratio,4).
assign(max_seconds,40).
assign(max_mem,84000).
```

```
    assign(report,5400).
```

With the inclusion of these seven, one places a limit on the number of CPU-seconds to be used, places a limit on megabytes (840) of memory to be used, and assigns in CPU-seconds how often a report is to be made to the output file. That report gives various details regarding items such as the number of new con- clusions drawn (generated), the number kept, and other often-useful facts. The value -1 for max_proofs instructs OTTER to complete as many proofs as it can within whichever limits are assigned. The value 9 assigned to max_distinct_vars has the program discard, upon deduction, any new conclusion that relies on ten or more distinct variables. The assigned value of 4 to the pick_given_ratio has the program choose, from the items available, for initiating a line of reasoning 4 clauses based on complexity, 1 by first come first serve (queue), then 4, then 1, and the like. With the assigned value of 40 to max_weight, the program discards immediately any new item that relies on strictly more than forty symbols, not counting parentheses or commas, with one proviso. The weight of an item is based solely on its symbol count unless—and this is crucial—some element in the weight_list(pick_and_purge) affects the computation. For a fine example, when I was studying an area in logic in which a key formula would have been given a weight of 93 (based on symbol count), by including appropriate expressions in the pick_and_purge list, I had the program treat its weight as far, far smaller. The use of resonators are an excellent example of enabling the researcher to influence the calculation of weight and thus sharply affect the direction the program takes.

The following "set" commands are easily understood.

```
    set(para_from).
    set(para_into).
    set(order_eq).
    set(hyper_res).
    % set(ancestor_subsume).
    set(back_sub).
    % set(sos_queue).
```

Here, in particular, ancestor subsumption is commented out as well as is the instruction to base the search strictly on first come first serve, queue, ignoring the weight of an item. Among the other given "set" com- mands, the two that are key focus on paramodulation. You are, with the two, telling OTTER to apply that rule both *from* and *into*, if possible, the item in focus.

The following three commands had to be, or wisely were, included.

```
    op(299, xfy, ^).
    op(300, xfy, o).  % multiplication of exponents
    clear(print_kept).
```

The first two, of the three, enable OTTER to interpret product and exponentiation in Beeson's axioms. The third has the program avoid keeping each clause it decided to retain. Since in many cases the program, dur- ing an attempt to complete an assignment, usually finds a proof and retains one million new conclusions, or more, without this third item, your output file could become far too large.

You now have been treated to some details regarding the given input file. Next in order is a different input file to illustrate what can be done and to give you a basis for experimentation. This input file illus- trates the use of resonators and the weight_list(pick_and_purge), the use of results obtained earlier in a study of a Stein theorem (in this case, t17), and the use of intermediate targets to measure progress and, possibly, to find so-called lemmas to be adjoined to a next run.

A Key Input File for the Successful Study of T17 for Semigroups

```
    set(para_from).
    set(para_into).
    set(order_eq).
    set(hyper_res).
    % set(ancestor_subsume).
    set(back_sub).
```

```
assign(max_weight,40).
% assign(change_limit_after,300).
% assign(new_max_weight,12).
clear(print_kept).
clear(print_back_sub).
clear(print_new_demod).
clear(print_back_demod).
assign(max_proofs,-1).
assign(max_distinct_vars,4).
assign(pick_given_ratio,6).
% assign(max_seconds,40).
% set(sos_queue).
set(input_sos_first).
op(299, xfy, ^).
op(300, xfy, o). % multiplication of exponents

weight_list(pick_and_purge).
% Following 21 may prove t17, .out3h4s, without x+0 and such
weight(x* (y*z)^n*x*y*z=y*z*x,-2).
weight((x*y)*z^n*x*y*z=z*x*y,-2).
weight((x*y^n*x*y)*z=y*x*z,-2).
weight(x*y*z^n*y*z= (x*z)*y,-2).
weight(x*y*z^n*x*y*z=z*x*y,-2).
weight(x* (y^n*x*y)*z=y*x*z,-2).
weight((x*y*z)*u=x*y*z*u,-2).
weight(x*y^n*x*y*z=y*x*z,-2).
weight(x*y^n*x*z*y*u=y*x*z^n*y*z*u,-2).
weight(x*y*z*u^n*x*z*u=y*x^n*y*u*x*z,-2).
weight(x*y*z^n*y*x*z=y*x^n*z*y*x,-2).
weight(x*y^n*x*x*y*x=x*y*x,-2).
weight(x*y*z*x^n*x*z*x=x*y*x*z,-2).
weight(x*y^n*y*x*y=y*y*x*y,-2).
weight(x*y*x*x*z*x=x*y*x*z,-2).
weight(x*y^n*x*x*y*x=y*x,-2).
weight(x*y*x=y*x,-2).
weight(x*x*y*x=y*x,-2).
weight(x*y^n*x*y=y*y*x*y,-2).
weight(x*x*y*x=x*y,-2).
weight(x*y=y*x,-2).
end_of_list.

list(usable).
x=x.
% group axioms
% e*x = x.
% x*e = x.
x* (y*z) = (x*y)*z.
(x*y)*z=x*y*z.
% inverse axioms
% x*i(x) = e.
% i(x)* x = e.
% exponentiation
% x^ 0 = e.
```

```
x^ 1 = x.
1+1 = 2.
x^ (xp+xq) = x^ xp * x^ xq.
x^ (xp+1) = x^ xp *x.      % deducible but perhaps useful to have explicit.
%  exponents and addition
x+y=y+x.
%  x+0 = x.
%  0+x = x.
%  exponents and multiplication
%  0 o x = 0.
%  x o 0 = 0.
1 o x = x.
x o 1 = x.
(x o y) o z = x o y o z.
x o (y + z) = x o y + x o z. (x
+ y) o z = x o z + y o z. x o y
= y o x.
(x^y) ^ z = x^ (y o z).
end_of_list.

list(sos).
y*x = x* y^n * x * y.
x*y^n*x*y=y*x.
%  Following 5 copies of last five in usable
(x o y) o z = x o y o z.
x o (y + z) = x o y + x o z. (x
+ y) o z = x o z + y o z. x o y
= y o x.
(x^y) ^ z = x^ (y o z).
end_of_list.

list(passive).
%  Following is a flip of an early one
a2*a1*a4*a3^n*a2*a4*a3!=a1*a2^n*a1*a3*a2*a4 |
$ANS(test2).
%  following 16 negs of a Knuth proof, apparently avoiding x+0 and such, temp.stein.t17.out3h4d
a1*a2*a3^n*a1*a2*a3!=a3*a1*a2 | $ANS(newinter).
a1*a2^n*a1*a2*a3!=a2*a1*a3 | $ANS(newinter).
a1*a2^n*a1*a3*a2*a4!=a2*a1*a3^n*a2*a3*a4 | $ANS(newinter).
a1*a2^n*a1*a3*a2*a4!=a2*a1*a4*a3^n*a2*a4*a3 |
$ANS(newinter). a1*a2^n*a3*a1*a2!=a2*a1*a3^n*a1*a2*a3 |
$ANS(newinter). a1*a2*a3^n*a1*a3*a4!=a2*a1^n*a2*a3*a1*a4 |
$ANS(newinter). a1*a2*a3^n*a2*a1*a3!=a2*a1^n*a3*a2*a1 |
$ANS(newinter). a1*a2*a3*a1^n*a1*a3*a1!=a1*a2*a1*a3 |
$ANS(newinter). a1*a2^n*a1*a1*a2*a1!=a2*a2*a1*a1 |
$ANS(newinter). a1*a2*a1!=a2*a2*a1*a1 | $ANS(newinter).
a1*a1*a2*a2!=a2*a1*a2 | $ANS(newinter).
a1*a2^n*a2*a1*a2!=a2*a2*a1*a2 | $ANS(newinter).
a1*a2*a1*a1*a3*a1!=a1*a2*a1*a3 | $ANS(newinter).
a1*a1*a2*a2!=a1*a2 | $ANS(newinter).
a1*a2*a1!=a2*a1 | $ANS(newinter).
a1*a2!=a2*a1 | $ANS(newinter).
a*b != b*a | $ANS(thm17).
end_of_list.
```

When compared with the discussion of the preceding input file, three aspects are of substantial inter- est focusing, respectively, on the weight_list, the list(sos), and the list(passive). I shall discuss the three in the order cited. You find in the given input file (for T17), in its pick_and_purge list, twenty-one equations. Each is used here as a resonator, included to direct OTTER's reasoning by giving a small weight, high pri- ority, to deduced items that match any of the twenty-one, where variables are essentially ignored, just treated as being a variable. The twenty-one were taken from a 21-step proof in an earlier experiment, one that occurred before the experiment relying on the given input file. In that earlier experiment, the following two (of the unwanted seven) were present.

```
0 o x = 0.
x o 0 = 0.
```

A glance at the earlier proof did show that neither was used, but the idea was to comment the two out and find the expected proof free of any of the unwanted seven, the presence of any of which would indicate that semimonoids were in focus and not semigroups. The approach that was being used was iterative, gradually removing unwanted items, and, more important for the current discussion, obtaining proof steps to be used as resonators. As a refresher, in a sequence of the type under discussion, that seeking resonators and, even- tually, a proof applying to semigroups, I often do turn to a level-saturation search and, occasionally, to cramming, a strategy in which so-called lemmas are adjoined to list(sos) with the intention of forcing their use in a proof to be completed.

As for the list(sos) in the last given input file, you will notice that it contains, in addition to the hypothesis of T17 (and its flip), five other equations. Those are, as indicated, copies of the last five items in list(usable), each focusing on one of the Beeson clauses for exponentiation. They are hold-overs, left from earlier experiments in the sequence designed to eventually obtain a proof for T17 that applies to semi- groups. Such inclusions do no harm to any proof the results. In general, by amending the initial set of sup- port, you provide the program more avenues to reason from in the beginning. To ensure that OTTER con- siders all items in the initial set of support before focusing on a deduced item, you include in the input file the command set(input_sos_first). As you have learned, or already knew, on the way to completing a jour- ney I often, with lemma adjunction, amend an initial set of support. When I do so, if all goes well, I later remove such amendations so that the resulting proof, if one is found, focuses strictly on the original theo- rem to be proved.

The last of the three items, list(passive), in the input file under discussion illustrates the use of suc- cesses, or partial successes, on the way to winning the prize. You find the negation of an item that had resisted proof in an earlier experiment, present in that earlier experiment to show that, finally, progress had occurred. You also find sixteen equations, each the negation of a proof step for T17, a proof obtained by a Knuth-Bendix approach. The idea of such an inclusion, of negations of steps of an earlier proof, is that of measuring what is happening in the current experiment. As more and more of the included negations in the passive list are proved, encouragement results and the almost-always-correct belief that significant progress is occurring. Also, the proofs of items in the passive list, before the treasure is in hand, provide or can pro- vide items to be used in lemma adjunction.

By now, are you discarding the rest of your fine wine and enjoyable food items, because you wish to have in hand the proof that is obtained with the use of the given file? Well, the proof is now given, a proof that surprised me.

A Surprising Proof for T17 for Semigroups

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on
octagon, Thu Aug  9 17:19:30 2012
The command was "otter".  The process ID is 20793.
----> UNIT CONFLICT at  15.10 sec ----> 22311 [binary,22310.1,40.1] $ANS(thm17).

Length of proof is 21.  Level of proof is 11.
```

```
---------------- PROOF ----------------

3 [] (x*y)*z=x*y*z.
16 [] y*x=x*y^n*x*y.
17 [] x*y^n*x*y=y*x.
40 [] a*b!=b*a|$ANS(thm17).
45 [para_into,16.1.1,3.1.1,flip.1] x* (y*z)^n*x*y*z=y*z*x.
52 [para_into,16.1.2.2.2,3.1.1,flip.1] (x*y)*z^n*x*y*z=z*x*y.
57 [para_from,16.1.1,3.1.1.1] (x*y^n*x*y)*z=y*x*z.
58 [para_from,16.1.1,3.1.2.2,flip.1] x*y*z^n*y*z= (x*z)*y.
262 [para_into,52.1.1,3.1.1] x*y*z^n*x*y*z=z*x*y.
326 [para_into,57.1.1,3.1.1] x* (y^n*x*y)*z=y*x*z.
404 [para_into,58.1.1.2,45.1.1,flip.1] (x*y*z)*u=x*y*z*u.
824 [para_from,404.1.1,326.1.1.2] x*y^n*x*y*z=y*x*z.
845 [para_into,824.1.1.2.2.2,824.1.1] x*y^n*x*z*y*u=y*x*z^n*y*z*u.
851 [para_into,824.1.1.2.2.2,262.1.1,flip.1] x*y*z*u^n*x*z*u=y*x^n*y*u*x*z.
875 [para_into,824.1.1.2.2,262.1.1,flip.1] x*y*z^n*y*x*z=y*x^n*z*y*x.
1111 [para_into,845.1.2,262.1.1] x*y^n*x*x*y*x=x*y*x.
1193 [para_into,851.1.2,824.1.1] x*y*z*x^n*x*z*x=x*y*x*z.
1245 [para_into,875.1.1.2,824.1.1,flip.1] x*y^n*y*x*y=y*y*x*y.
2157 [para_from,1245.1.1,1193.1.1.2.2] x*y*x*x*z*x=x*y*x*z.
3031 [para_into,2157.1.2,17.1.1] x*y^n*x*x*y*x=y*x.
4701 [para_into,3031.1.1,1111.1.1] x*y*x=y*x.
5878 [para_into,4701.1.1.2,4701.1.2] x*x*y*x=y*x.
6067 [para_from,4701.1.1,1245.1.1.2.2] x*y^n*x*y=y*y*x*y.
8471 [para_into,6067.1.1,17.1.1,flip.1] x*x*y*x=x*y.
22310 [para_into,8471.1.1,5878.1.1] x*y=y*x.
```

Have you identified the property or properties of this proof that produced my surprise? Well, from the input equations, just four were used. Of course, the denial of commutativity is among them. But only associativity and the T17 hypothesis and its flip are also present. None of the items from Beeson that focus on exponentiation are present. I, in my files, call this proof "pure" as a result.

Now I turn to a theorem, to be called T43,5643, that Stein, in an e-mail, indicated he would never have tackled. I shall present to you an input file that illustrates the use of the hot list strategy, followed by a proof that it yields. I shall then briefly discuss other proofs you can get for this theorem if you do not use the hot list strategy or if you assign different values to max_weight. You will get a glimpse of how you might cope with assigning an effective value to max_weight. You will also learn that proof length is influ- enced by the choice of strategy and the choice of assigned values.

The following file is essentially that which I first ran those weeks ago. Its list(sos) gives the hypothe-sis of T43,5643; the name in part is derived by the number of occurrences, exponent values, for the various variables.

Input File for T43,5643

```
op(299, xfy, ^).
op(300, xfy, o). % multiplication of exponents
set(knuth_bendix).
set(order_eq).
set(ancestor_subsume).
set(back_sub).
assign(max_weight,70).
assign(change_limit_after,400).
assign(new_max_weight,28).
```

```
clear(print_kept).
clear(print_back_sub).
clear(print_new_demod).
clear(print_back_demod).
assign(max_proofs,-1).
assign(max_distinct_vars,5).
assign(heat,4).
%  assign(max_seconds,40).
set(sos_queue).

weight_list(pick_and_purge).
%  following 6 from a purer proof of xyxyxy, not Knuth.
weight(((x*y)* (x*y)*x*y)* (y*x)*y*x=x*y,-12).
weight(((x*y)* (x*y)*x*y)* ((x*y)* (x*y)*x*y)*y*x= ((x*y)*x*y)*x*y,-12).
weight(((x*y)*x*y)*x*y=y*x,-12).
weight(((x*y)*   (x*y)*x*y)*   ((x*y)*   (x*y)*x*y)*y*x=x*y,-12).
weight(((x*y)*   (x*y)*x*y)*   ((x*y)*   (x*y)*x*y)*y*x=y*x,-12).
weight(x*y=y*x,-12).
%  following 4 from  Knuth proof of xyxyxy version out t15
weight(x*y*z*x*y*z*x*y*z=z*x*y,-15).
weight(x*y*z=z*x*y,-15).
weight(x*y*z=y*z*x,-15).
weight(x*y=y*x,-15).
%  following 12 from and xyxy t15 proof, with inverse
weight((x*i(x))*x*i(x)=e,-12).
weight((e*x)*e*x=x,-12).
weight(x*e*x=x,-12).
weight(x*x=x,-12).
weight(i(e)=e,-12).
weight(i(x)*x=i(e),-12).
weight(i(e)*x=x,-12).
weight((i(x)*x)*y=y,-12).
weight(i(x)*x*y=y,-12).
weight(i(x)*x=x,-12).
weight(e=x,-12).
weight(x=y,-12).
%  Following 4 from Knuth proof, temp.stein.t15.ot3j12, of xyxyxyxy version of t15
weight(x*y*z*x*y*z*x*y*z*x*y*z=z*x*y,-10).
weight(x*y*z=z*x*y,-10).
weight(x*y*z=y*z*x,-10).
weight(x*y=y*x,-10).
end_of_list.

list(usable).
x=x.
%  Semigroup axioms
x* (y*z) = (x*y)*z.
(x*y)*z = x* (y*z).
end_of_list.

list(sos).
x*y = y*y*y*y*y* x*x*x*x*x*x * y*y*y*y*x*x*x.
y*y*y*y*y* x*x*x*x*x*x * y*y*y*y*x*x*x = x*y.
```

```
end_of_list.

list(passive).
% Following 6 negs from a purer proof of xyxyxy, not Knuth
((a1*a2)* (a1*a2)*a1*a2)* (a2*a1)*a2*a1!=a1*a2 | $ANS(inter1506).
((a1*a2)* (a1*a2)*a1*a2)* ((a1*a2)* (a1*a2)*a1*a2)*a2*a1!= ((a1*a2)*a1*a2)*a1*a2 |
    $ANS(inter1506).
((a1*a2)*a1*a2)*a1*a2!=a2*a1 | $ANS(inter1506).
((a1*a2)* (a1*a2)*a1*a2)* ((a1*a2)* (a1*a2)*a1*a2)*a2*a1!=a1*a2 | $ANS(inter1506).
((a1*a2)* (a1*a2)*a1*a2)* ((a1*a2)* (a1*a2)*a1*a2)*a2*a1!=a2*a1 | $ANS(inter1506).
a1*a2!=a2*a1 | $ANS(inter1506).
% Following 4 negs from a Knuth proof of xyxyxyxy of t15
a1*a2*a3*a1*a2*a3*a1*a2*a3*a1*a2*a3=a3*a1*a2 |
$ANS(intert1504). a1*a2*a3=a3*a1*a2 | $ANS(intert1504).
a1*a2*a3=a2*a3*a1 | $ANS(intert1504).
a1*a2=a2*a1 | $ANS(intert1504).
a*b != b*a | $ANS(thm5643).
end_of_list.

list(hot).
x*y  =  y*y*y*y*y*  x*x*x*x*x*x  *  y*y*y*y*x*x*x.
y*y*y*y*y* x*x*x*x*x*x * y*y*y*y*x*x*x = x*y. x*
(y*z) = (x*y)*z.
(x*y)*z = x* (y*z).
end_of_list.
```

You see in the file assign(heat,4) a command that has the program, upon retaining a new conclusion, consider it with each element of the hot list, then consider retained results again with each member, again, and (finally) again. Such consideration occurs before any other item is chosen to drive the reasoning. In list(hot), you see associativity, its flip, the T43,5643 hypothesis, and its flip. A general rule for using the hot list, if no other inspiration is present, is to place there the hypothesis of the theorem to be proved and, often, none of the basic axioms.

Its use yields the following proof.

A Proof of T43,5643 Emphasizing the Use of the Hot List Strategy

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Tue Aug 14 08:54:40 2012
The command was "otter".  The process ID is 29235.
----> UNIT CONFLICT at   3.10 sec ----> 740 [binary,739.1,11.1] $ANS(thm5643).

Length of proof is 28.  Level of proof is 16.

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

11 [] a*b!=b*a|$ANS(thm5643).
12 [] x*y=y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*x*x*x.
13 [] y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*x*x*x=x*y.
15 [] (x*y)*z=x*y*z.
17 [] x*y*z= (x*y)*z.
19,18 [copy,17,flip.1] (x*y)*z=x*y*z.
20 [] x*y=y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*x*x*x.
22,21 [copy,20,flip.1] x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*y*y*y=y*x.
```

```
23 [para_into,21.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2,18.1.1,demod,19,19,19,19,19,19,19,19]
     x*x*x*x*x*y*z*y*z*y*z*y*z*y*z*y*z*x*x*x*x*y*z*y*z*y*z=y*z*x.
25 [para_into,21.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2,18.1.1,demod,19,19,19,19,19,19,19,19]
     x*y*x*y*x*y*x*y*x*y*z*z*z*z*z*z*x*y*x*y*x*y*x*y*z*z*z=z*x*y.
27 (heat=1) [para_into,23.1.1.2.2.2.2.2.2.2.2.2,13.1.1] x*x*x*x*x*x*x*x*x*x*x=x*x*x.
29 (heat=1) [para_from,23.1.1,15.1.1.1,demod,
     19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,flip.1]
     x*x*x*x*x*y*z*y*z*y*z*y*z*y*z*y*z*x*x*x*x*y*z*y*z*y*z*u=y*z*x*u.
35 (heat=2) [para_from,27.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,19,19,flip.1]
     x*x*x*x*x*x*x*x*x*x*x*y=x*x*x*y.
37 (heat=2) [para_from,27.1.1,13.1.1.2.2.2.2.2.2.2] x*x*x*x*x*x*x*x*x*x=x*x.
51,50 (heat=3) [para_into,35.1.1.2.2.2.2.2.2.2.2,13.1.1,demod,22] x*x*x*x*x*x*x*x*y*x=y*x.
56 (heat=3) [para_from,37.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,flip.1]
     x*x*x*x*x*x*x*x*x*x*y=x*x*y.
89,88 (heat=4) [para_from,50.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,flip.1]
     x*x*x*x*x*x*x*x*y*x*z=y*x*z.
106 (heat=4) [para_into,56.1.1.2.2.2.2.2,13.1.1] x*x*x*x*x*y*x=x*x*y*y*y*y*y*y*x*x*x*x*y*y*y.
148 [copy,106,flip.1] x*x*y*y*y*y*y*y*x*x*x*x*y*y*y=x*x*x*x*x*y*x.
172,171 (heat=1) [para_into,148.1.1.2.2.2.2.2.2.2.2.2.2.2,12.1.1,demod,
     19,19,19,19,19,19,19,19,89,19,19,19,19,22,19,19,19,19,19,19,51]
     x*x*y*y*y*y*y*y*x*x*x*y*y*x*y=x*x*x*x*x*y*x.
400 [para_from,21.1.1,18.1.1.1,demod,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,flip.1]
     x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*y*y*y*z=y*x*z.
402 (heat=1) [para_into,400.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2,13.1.1]
     x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*z*y=y*x*y*y*z*z*z*z*z*z*y*y*y*y*z*z*z.
403 [copy,402,flip.1] x*y*x*x*z*z*z*z*z*z*x*x*x*x*z*z*z=y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*z*x.
404 (heat=1) [para_into,403.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2,12.1.1,demod,
     19,19,19,19,19,19,19,19,89,19,19,19,19,22,19,19,19,19,19,19,51,172]
     x*y*x*x*x*x*x*z*x=y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*z*x.
407 (heat=2) [para_from,404.1.1,15.1.1.1,demod,
     19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19]
     x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*z*y*u=y*x*y*y*y*y*y*z*y*u.
416 (heat=3) [para_into,407.1.1,13.1.1,flip.1]  x*y*x*x*x*x*x*x*x*x=x*y.
433 (heat=4) [para_into,416.1.1.2,12.1.1,demod,
     19,19,19,19,19,19,19,19,19,19,19,19,19,19,89,19,19,19,19,19,19,19,89,
     19,19,19,19,19,19,19,89,19,19,19,19,19,19,19,19,19,19,19,19,19,19,89,
     19,19,19,19,19,19,19,89,19,19,19,19,19,19,19,89,19,19,19,19,19,19,19,89]
     x*x*x*x*x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*x*x*x*x*y*y*y=x*y.
436,435 (heat=4) [para_from,416.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,flip.1]
     x*y*x*x*x*x*x*x*x*x*z=x*y*z.
681,680 [para_into,29.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2,25.1.1,demod,89,89,flip.1]
     x*x*x*y*y*y*y*y*y*x*x*x*x*x*x*x*x*y*y*y=x*y*x*x.
683,682 [back_demod,433,demod,681] x*x*x*x*x*x*x*y*x*x=x*y.
699,698 (heat=1) [para_into,682.1.1.2.2.2.2.2.2.2,12.1.1,demod,
     19,19,19,19,19,19,681,19,19,19,683] x*x*x*x*x*x*x*x*x*y=x*y.
701,700 (heat=1) [para_from,682.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,flip.1]
     x*x*x*x*x*x*x*y*x*x*z=x*y*z.
702 (heat=1) [para_from,682.1.1,12.1.1,demod,
     19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,701,19,19,19,19,19,19,19,19,701,
     19,19,19,19,19,19,19,19,701,701,19,19,19,19,19,19,19,19,436,19,19,19,19,19,
     19,19,19,436,19,19,19,19,19,19,19,19,436,19,19,19,19,19,19,19,19,436,19,19,
     19,19,19,19,19,19,436,699,flip.1] x*x*x*x*x*x*y*x*x*x=x*y.
739 (heat=2) [para_into,702.1.1.2.2.2.2.2.2,12.1.1,demod,19,19,19,19,19,19,699,
19,19,19,19,19,19,22,19,19,19,19,19,19,51,51] x*y=y*x.
```

This proof is also wonderfully pure, relying on nothing but associativity and the T43,5643 hypothesis. You immediately may wonder why, in the proof, clauses are cited more than once, clauses from the input before deduced clauses appear. When that occurs, the second (duplicate) citing shows the researcher which items from the hot list were used in the proof. You will also find abundant citing, for the deduced steps, of heat, sometimes heat=4. And, as expected with Knuth, much demodulation is present.

As I was writing these paragraphs, I searched for the original success, and then I modified it to present a file without much extraneous material. When I used the modified file, however, I failed to include the hot list. The result—and here you see how proof length can be affected by your decisions, or, in my case, by unintentional omissions—was the obtaining of a different proof. Actually, because of the presence of ancestor subsumption, eight proofs were found; the longest (and first) has length 40, and the shortest and last has length 33. In addition to the unintentional omission of the hot list, I also intentionally assigned the value 60, rather that 70, to max_weight.

Almost at the same time, I decided to investigate smaller assigned values to max_weight. The assignment of the value 50 led to the program quickly telling me, before any proof was found, that the sos went empty, which meant that no further deductions could be made. So, in effect, you see how you might begin to assign a value to max_weight in your experiments, regardless of the area of study. Indeed, you might begin with a value that you conjecture is too small, and then proceed accordingly. My friend and col- league Ross Overbeek often suggests that a good approach is to assign the value 1, then 2, and continue in this manner.

From the next cited item, you learn that writing a notebook presenting methodology and experimen- tal results can lead to a most unexpected discovery. In particular, when I retained the assigned value of 60 to max_weight, but corrected my omission by restoring the use of the hot list strategy with the hot list iden- tical to that given in the cited input file—gold, indeed—the program returned to me the following proof, a proof of less length than I had ever found in this context before.

A Most Pleasing Proof for T43,5643

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Tue Aug 14 08:49:25 2012
The command was "otter".  The process ID is 29099.
----> UNIT CONFLICT at   1.46 sec ----> 865 [binary,864.1,11.1] $ANS(thm5643).

Length of proof is 24.  Level of proof is 17.

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

11 [] a*b!=b*a|$ANS(thm5643).
12 [] x*y=y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*x*x*x.
13 [] y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*x*x*x=x*y.
15 [] (x*y)*z=x*y*z.
17 [] x*y*z= (x*y)*z.
19,18 [copy,17,flip.1] (x*y)*z=x*y*z.
20 [] x*y=y*y*y*y*y*x*x*x*x*x*x*y*y*y*y*x*x*x.
22,21 [copy,20,flip.1] x*x*x*x*x*y*y*y*y*y*y*x*x*x*x*y*y*y=y*x.
23 [para_into,21.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2,18.1.1,demod,19,19,19,19,19,19,19,19]
   x*x*x*x*x*y*z*y*z*y*z*y*z*y*z*y*z*x*x*x*x*y*z*y*z*y*z=y*z*x.
25 [para_into,21.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2,18.1.1,demod,19,19,19,19,19,19,19,19]
   x*y*x*y*x*y*x*y*x*y*z*z*z*z*z*z*x*y*x*y*x*y*x*y*z*z*z=z*x*y.
27 (heat=1) [para_into,23.1.1.2.2.2.2.2.2.2.2.2,13.1.1] x*x*x*x*x*x*x*x*x*x*x=x*x*x.
31 (heat=2) [para_from,27.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,19,19,flip.1]
```

```
    x*x*x*x*x*x*x*x*x*x*x*x*y=x*x*x*y.
42,41 (heat=3) [para_into,31.1.1.2.2.2.2.2.2.2.2,13.1.1,demod,22] x*x*x*x*x*x*x*x*y*x=y*x.
62,61 (heat=4) [para_from,41.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,
    flip.1] x*x*x*x*x*x*x*x*y*x*z=y*x*z.
168,167 [para_into,61.1.1,25.1.1,flip.1] x*x*y*y*y*y*y*y*x*x*x*x*x*x*x*x*y*y*y=y*x*x.
241 [para_from,61.1.1,23.1.1.2.2.2.2.2] x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*y*y*y*y*y=y*y*x.
245 (heat=1) [para_from,241.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,19,
    19,19,19,19,19,19,19,19,19,flip.1] x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*y*y*y*y*y*z=y*y*x*z.
249 (heat=2) [para_into,245.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2,13.1.1,demod,42]
    x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*y=y*y*x*y*y*y*y.
250 [copy,249,flip.1] x*x*y*x*x*x*x=y*y*y*y*y*x*x*x*x*y*y*y*y*x*x.
254,253 (heat=1) [para_into,250.1.1.2.2,12.1.1,demod,19,19,19,19,19,19,19,19,
    19,62,19,19,19,19,19,19,168,19,19,19,19,19,62,19,19,19,19,19,19,62,flip.1]
    x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*y=y*y*y*y*x*y*y.
256 (heat=1) [para_from,250.1.1,15.1.1.1,demod,254,19,19,19,19,19,19,19,19,19,19,19]
    x*x*x*x*y*x*x*z=x*x*y*x*x*x*x*z.
268,267 (heat=2) [para_from,253.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,
    19,19,19,19,19,19,19,19,19,flip.1]
    x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*y*z=y*y*y*y*x*y*y*z.
277 (heat=2) [para_from,256.1.1,15.1.1,demod,19,19,19,19,19,19,19,19,19,19,19]
    x*y*x*y*z*x*y*x*y*x*y*x*y*u=x*y*x*y*x*y*x*y*z*x*y*x*y*u.
287,286 [back_demod,241,demod,268] x*x*x*x*y*x*x*x*x*x*x=x*x*y.
288 (heat=3) [para_into,267.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2,13.1.1,demod,22]
    x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*z*y=y*y*y*y*x*y*z*y.
332 (heat=3) [para_into,277.1.1.2.2.2.2.2,13.1.1,demod,42,287]
    x*x*x*x*y*x*x=x*x*x*x*x*x*y.
371,370 (heat=4) [para_from,288.1.1,15.1.1.1,demod,19,19,19,19,19,19,19,19,19,
    19,19,19,19,19,19,19,19,19,19,19,19,flip.1]
    x*x*x*x*x*y*y*y*y*y*x*x*x*x*y*z*y*u=y*y*y*y*x*y*z*y*u.
502 [copy,332,flip.1] x*x*x*x*x*x*y=x*x*x*x*y*x*x.
621 (heat=1) [para_from,502.1.1,13.1.1.2.2.2.2.2,demod,19,19,19,19,19,19,371]
    x*x*x*x*y*x*x*x*x*x=x*y.
864 (heat=2) [para_into,621.1.1.2.2.2.2,12.1.1,demod,19,19,19,19,19,19,19,19,
    62,19,19,19,19,19,19,19,19,62,19,19,19,19,22,19,19,19,19,19,19,19,19,42,
    19,19,19,19,19,19,19,19,42,42] x*y=y*x.
```

I leave to you, if you so choose, the analysis of the differences between the 28-step and the 24-step proof. I also offer you, as you may have predicted, the challenge of converting either proof to a demodulation-free proof. What is satisfying in the preceding paragraphs, for T43,5643, is that I was able to reach the domain, variety, of semigroups.

Earlier, I discussed T19 in its full generality, the following, and my inability to complete the results for semigroups, thus leaving for you a challenge.

```
y*x = x^n * y * x * y.  %  T19 hypothesis
```

Although I did not succeed with this general hypothesis, when I assigned to *n* the value 5, all went well, in the purest sense, as you will see from the proof I shall shortly give. I include this input file and proof because, just perhaps, you share my excitement at witnessing success involving extremely large numbers, for clause retention as well as for CPU time. I think it safe to say that, had I used a Knuth approach rather than relying solely on paramodulation, the resulting numbers would have been much, much smaller.

Input File for an Instantiation of T19

```
op(299, xfy, ^).
op(300, xfy, o).  % multiplication of exponents
```

```
set(para_from).
set(para_into).
set(order_eq).
set(hyper_res).
set(ancestor_subsume).
set(back_sub).
assign(max_weight,22).
clear(print_kept).
clear(print_back_sub).
clear(print_new_demod).
clear(print_back_demod).
assign(max_proofs,-1).
assign(max_distinct_vars,6).
assign(pick_given_ratio,6).

weight_list(pick_and_purge).
% Following 12 prove a version of t41, which is related distantly to t17, with xxxx replacing
   xn, temp.stein.t41.out4d
weight(e*e*e*e*x*e*x=x,-8).
weight(x*x*x* (x*y)*x*y=y*x,-8).
weight(e*e*e*e*x*x=x,-8).
weight(e*e*e*x*x=x,-8).
weight(e*e*x*x=x,-8).
weight(e*x*x=x,-8).
weight(x*x=x,-8).
weight(x*x*y=x*y,-8).
weight(x*x*x*y=x*y,-8).
weight(x*x*x*x*y=y*x,-8).
weight(x*x*y=y*x,-8).
weight(x*y=y*x,-8).
% Following 12 proof steps of two proofs of version of t41, but with xxx in place of n
weight(e*e*e*x*e*x=x,-8).
weight(e*e*e*x*x=x,-8).
weight(e*e*x*x=x,-8).
weight(e*x*x=x,-8).
weight(x*x=x,-8).
weight(x*x*x*y=x*y,-8).
weight(x*x*x*y=y*x,-8).
weight(x*x*y=x*y,-8).
weight(x*x*y*x*y=y*x,-8).
weight(x*x*y=y*x,-8).
weight(x*y*x*y=x*y,-8).
weight(x*y=y*x,-8).
% Following 15 prove t17 from temp.stein.t17.out3b, queue.
weight((x*y)*z^n* (x*y)*z= (z*x)*y,1).
weight(x*y^n*y*x^n*y*x=y*x,1).
weight(e*x^n*x=x*e,1).
weight((x*y^n*x*y)*z=y*x*z,1).
weight((x*e)*y^n*x*y= (y*x)*e,1).
weight(x* (y^n*y)*x^n*y*x=y*x,1).
weight(x^n*x=x*e,1).
weight(x*y^n*x*y=y*x*e,1).
weight((x*e)*y^n*x*y=y*x,1).
weight(x* (y*e)*x^n*y*x=y*x,1).
weight((x*y*e)*z=x*y*z,1).
weight(x*x*y=y*x,1).
weight((x*x*e)*y=y*x,1).
weight((e*x)*y=y*x,1).
weight(x*y=y*x,1).
% Following 7 are from a 9-step Knuth proof of t17, .out1.
weight(x*e^n*x=x,3).
weight(x^n*x=x,3).
weight(e^n=e,3).
weight(x*x=x,3).
weight(x*x*y=x*y,3).
weight(x*y*x=y*x,3).
weight(x*y=y*x,3).
end_of_list.

list(usable).
x=x.
x*  (y*z)  =  (x*y)*z.
(x*y)*z  =  x*  (y*z).
end_of_list.

list(sos).
y*x  =  x*x*x*x*x  *  y  *  x  *y.
x*x*x*x*x  *  y  *  x  *y  =  y*x.
end_of_list.

list(passive).
% Following 13 from a 13-step proof with e used, of t17
(a1*a2)*a3^n* (a1*a2)*a3!= (a3*a1)*a2 | $ANS(t1713).
a1*a2^n*a2*a1^n*a2*a1!=a2*a1 | $ANS(t1713).
e*a1^n*a1!=a1*e | $ANS(t1713).
(a1*e)*a2^n*a1*a2!= (a2*a1)*e | $ANS(t1713).
a1* (a2^n*a2)*a1^n*a2*a1!=a2*a1 | $ANS(t1713).
a1^n*a1!=a1*e | $ANS(t1713).
(a1*e)*a2^n*a1*a2!=a2*a1 | $ANS(t1713).
a1* (a2*e)*a1^n*a2*a1!=a2*a1 | $ANS(t1713).
a1*a1*a2!=a2*a1 | $ANS(t1713).
a1*a1!=e*a1 | $ANS(t1713).
a1*a1!=a1 | $ANS(t1713).
a1*a1*a2!=a1*a2 | $ANS(t1713).
a1*a2!=a2*a1 | $ANS(t1713).
a*b != b*a | $ANS(thm41).
end_of_list.
```

If you run this input file as given, as you will now see from the ensuing proof, you must wait a long, long time to see the results for yourself.

A Proof for T19 Instantiated Presenting Very Large Numbers

```
----- Otter 3.3g-work, Jan 2005 -----
The process was started by wos on vanquish,
Fri Apr 27 13:08:48 2012
The command was "otter".  The process ID is 24932.
```

```
----> UNIT CONFLICT at 661696.84 sec ----> 8207454 [binary,8207453.1,34.1] $ANS(thm41).

Length of proof is 21.  Level of proof is 14.

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

3 [] (x*y)*z=x*y*z.
19 [] y*x=x*x*x*x*x*y*x*y.
20 [] x*x*x*x*x*y*x*y=y*x.
34 [] a*b!=b*a|$ANS(thm41).
37 [para_into,19.1.2.2.2.2,3.1.2,flip.1] x*x*x* (x*x)*y*x*y=y*x.
38 [para_into,19.1.2.2.2,3.1.2,flip.1] x*x* (x*x)*x*y*x*y=y*x.
39 [para_into,19.1.2.2,3.1.2,flip.1] x* (x*x)*x*x*y*x*y=y*x.
55 [para_from,37.1.1,20.1.1.2.2] x*x*x*x= (x*x)*x.
57 [para_into,55.1.1,3.1.2] (x*x)*x*x= (x*x)*x.
76 [para_into,38.1.1.2.2,3.1.2] x*x* ((x*x)*x)*y*x*y=y*x.
79 [para_into,57.1.1,3.1.2] ((x*x)*x)*x= (x*x)*x.
125 [para_from,79.1.1,3.1.1.1,flip.1] ((x*x)*x)*x*y= ((x*x)*x)*y.
143 [para_into,39.1.1.2,3.1.2] x* ((x*x)*x)*x*y*x*y=y*x.
8676 [para_from,125.1.2,76.1.1.2.2] x*x* ((x*x)*x)*x*y*x*y=y*x.
5857484 [para_into,8676.1.1.2,143.1.1] x*y*x=y*x.
5857509 [para_into,5857484.1.1,3.1.2] (x*y)*x=y*x.
5857783 [para_into,5857509.1.1.1,5857484.1.2] (x*y*x)*y=x*y.
7197906 [para_into,5857783.1.1,3.1.1] x* (y*x)*y=x*y.
8205374 [para_into,7197906.1.1.2,5857509.1.1] x*x*y=x*y.
8205381 [para_into,7197906.1.1.2,3.1.1] x*y*x*y=x*y.
8205586 [para_into,8205374.1.1.2,8205374.1.2] x*x*x*y=x*y.
8205887 [para_into,8205381.1.1,8205374.1.2] x*x*y*x*y=x*y.
8205949 [para_into,8205586.1.1.2.2,8205374.1.2] x*x*x*x*y=x*y.
8206645 [para_from,8205887.1.1,20.1.1.2.2.2] x*x*x*x*y=y*x.
8207453 [para_into,8206645.1.1,8205949.1.1] x*y=y*x.
```

Now, if you wish to go further than I did in earlier sections, or, more likely, if you wish to study other areas in some way emulating what you find in this notebook, you might amend one of the given input files (offered in this section) with various items. For a first example, consider the case in which you conjecture that the proof you seek would be within reach if some lemma, *L*, were proved and then keyed upon. Fur- ther, let *L* be an equation, or formula, that can be represented with a unit clause, a clause free of " | ", the symbol for logical **or**. Finally, let *R* be a resonator that you wish to include to guide the program's reason- ing. If you included in your input file something like the following, you would get your wish.

```
assign(max_weight,22).
assign(dynamic_heat_wt,-2).
weight_list(pick_and_purge).
weight(L,-2).
weight(R,4).
end_of_list.
```

(Sections 4 and 10 offer a bit more about the dynamic hot list strategy.) You must be careful, as suggested by the given illustration, not to drown your hot list, which explains why the different values were chosen. In particular, you most likely do not wish other deduced items to be adjoined to the hot list during the run, items that match a resonator or that are given a weight based on symbol count.

For a second possibility, in contrast to this discussion of an intention of keying on some deduced item, if and when deduced, you might wish to block the use of some newly deduced item. Of course, you could give that item a weight strictly larger than the assigned value to max_weight. You could, instead, proceed as I often do when seeking a proof shorter than I have in hand or that occurs in the literature. Specifically, you could have a list(demodulators) and place in the list something like EQ(D,junk)., where D represents the unwanted item. (Items in that list can, of course, be used to canonicalize and simplify newly deduced items, which can materially improve the effectiveness of your program.) You would then also place in weight_list(pick_and_purge) something like weight(junk,1000). Especially for the new researcher, you might be asked by a mathematician or logician for a proof that avoids the use of a well-known lemma, and now you have yet aother way to attempt to meet the request. Section 5 addresses the use of "junk".

For a third possibility, you might have the goal of finding a proof of a tough theorem, a theorem that is proven in a book or paper, a proof that appears to you to be longer than necessary. One path that appeals to me involves taking various steps of the published proof, negating each, and placing those negations in list(passive). I would (logically, and notationally, with " | ") or to each such negation something like
$ANS(intermed); see Sections 1 and 6 and this section itself. An iterative approach is most likely called for. In run *n*, for example, you find proofs of some of the steps. In the next run, in the use of lemma adjunction, you adjoin either the last line of each new proof, or all of the new proof steps, to list(sos) for the next run. You, of course, continue until you have a proof. What you have done is fill in the gaps between the chosen steps of the published proof. You can now use various methods for proof shortening, focusing on a fully detailed proof obtained with your program.

Now, if—and, of course, I wish it so—you are interested in and intrigued with experimentation, and if you would like to examine other input files, you can find other notebooks on my website automatedrea- soning.net, notebooks that contain input files for areas other than these Stein theorems.

Thanks to So Many

Indeed merited, on my website, in my notebooks, and elsewhere is gratitude to various people I know and have known. Here I cite those who are recently relevant; many in the past played key roles. To Ross Overbeek for his continued advice and ideas, I owe much; thanks, Ross. Any researcher or curious person who browses in, say, one of my notebooks for excitement or ideas can do so because of Ross; indeed, he played a key role in their existence. Gail Pieper, my editor for decades and co-author more than once, has proved invaluable; thanks, Gail. Gene Rackow has enabled me to continue experimentation; thanks, Gene. Michael Beeson caused this particular notebook to come into existence by brining these theorems to my attention; thanks, Michael. Sherman Stein supplied the theorems, and his e-mail messages added to my zeal; thanks, Sherman. John Halleck discussed some of this research; thanks, John. Finally, though he is no longer on this planet, William McCune designed and implemented OTTER, the program I have used for decades and continue to use; thanks, Bill. On a more personal note, I thank my best friend; she continues to provide me with joy.