Challenge Problems Focusing on Equality and Combinatory Logic: Evaluating Automated Theorem-Proving Programs*

Larry Wos and William McCune

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4844

CONF-8805110--2

DE88 010057

Abstract In this paper, we offer a set of problems for evaluating the power of automated theorem-proving programs and the potential of new ideas. Since the problems published in the proceedings of the first CADE conference [McCharen76] proved to be so useful, and since researchers are now far more disposed to implementing and testing their ideas, a new set of problems to complement those that have been widely studied is in order. In general, the new problems provide a far greater challenge for an automated theorem-proving program than those in the first set do. Indeed, to our knowledge, five of the six problems we propose for study have never been proved with a theorem-proving program. For each problem, we give a set of statements that can easily be translated into a standard set of clauses. We also state each problem in its mathematical and logical form. In many cases, we also provide a proof of the theorem from which a problem is taken so that one can measure a program's progress in its attempt to solve the problem. Two of the theorems we discuss are of especial interest in that they answer questions that had been open concerning the constructibility of two types of combinator. We also include a brief description of a new strategy for restricting the application of paramodulation. All of the problems we propose for study emphasize the role of equality. This paper is tutorial in nature.

1. Introduction

To estimate the possible value and power of an automated theorem-proving program or of a new approach, one needs various test problems. One cannot simply make diverse computations in the abstract about CPU cycles, conclusions drawn and discarded, conclusions drawn and retained, and such. Rather, one must attempt to solve problems from various areas—mathematics and logic, for example—with one's program or with the new approach. After all, the value of a discovery, such as an inference rule or strategy, rests mainly with its effectiveness for problem solving.

To determine how well a given program is doing in its attempt to prove some given theorem or solve some specified problem, one usually requires access to a proof or solution to measure the program's progress. For, without knowing what the answer is, how can one estimate how close the program is to solving the assigned problem? Therefore, to facilitate and encourage the needed experimentation, we offer in this paper various test problems, and for each we include a solution for measuring progress. All of the problems we present emphasize the role of equality.

Each problem focuses on a theorem taken from combinatory logic. In general, the problems we propose for study are far more challenging than those usually used for evaluating a theorem-proving program or a new concept. As evidence of their difficulty, for almost all of them, from what we know, no proof has ever been obtained with a theorem-proving program. These problems are not only hard for a computer program to solve, but, in many cases, also hard for a person to solve. Indeed, one of the theorems (Theorem C3) we include answers a question that had been open, a question that concerns the constructibility of a particular type of combinator. Theorem C3 is also of interest in that it illustrates the excellent meld between automated theorem proving and combinatory logic, for its proof depends on various properties of unification.

For each problem, we shall first state it as a logician would. To make the presentation self-sufficient, we shall, where necessary, give the needed background. Except in Section 2.1, our discussion of the required concepts will be terse. In that section, we do give a rather lengthy treatment of the problem under discussion. We take this action in part to provide a sample of how one can proceed and to focus on various strategies for restricting paramodulation [RobinsonG69]—especially for those who are new to automated theorem proving—and in part to promote a sharp increase in experimentation in general. In particular, we include three short proofs of a simple theorem (Theorem C1.1) to illustrate the role of different strategies. We conjecture that, with the rapid growth in the interest

*This work was supported by the Applied Mathematical Sciences subprogram of the office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

in automated theorem proving, and with the new breed of researcher who is far more excited about implementing and testing ideas, the field is eager for a set of problems that includes some perhaps beyond the capability of any program now in existence.

To complement the mathematical and logical statement of a problem, we shall give a set of statements — abbreviated clauses — that can be used to submit the problem for attack by a theorem-proving program. If the paradigm on which a particular program is based does not rely on the use of clauses, the mathematical description that we give for each problem will make it possible to map the problem accordingly.

In all cases, we shall supply a mathematical proof, an outline of such a proof, or a proof in abbreviated clause notation—sometimes, more than one of the three. Since almost all of the problems we pose have the property that no proof, as far as we know, has ever been obtained with a theorem-proving program, we include no statistics obtained from an attempt to obtain a solution with one of our programs. We would, of course, be very interested in any statistics obtained by a researcher who is successful in solving one of the posed problems—if the solution is obtained with a program. Such statistics provide an important measure of a problem's difficulty and of a program's effectiveness. As an example, the statistics found in the earlier paper [McCharen76]—that focusing on problems and experiments and published in the first CADE conference proceedings—have proved most useful.

In addition to our primary goal of encouraging researchers to test and evaluate programs, we have the secondary goal of causing others to supply various problems for this purpose. Although an earlier attempt to stimulate such contributions clearly did not succeed, the changes evidenced in the past four years may be of sufficient magnitude that this new attempt will in fact succeed. Consistent with our stated goals, we plan to make available some time in the future a database of test problems, including those presented in this paper and others we use for study. This database will be accessible by electronic mail.

We focus on problems heavily emphasizing equality in part because of the importance of this relation to so many possible applications of automated theorem proving, and in part because of a view we have concerning the history of automated theorem proving. In particular, were we sipping brandy and having a pleasant conversation with friends, we would suggest that automated theorem proving would have progressed far more rapidly had it not been for the dominant practice of treating equality as just another relation. Specifically, until relatively recently, a large fraction of the discussion, research, and experimentation focusing on problems in which equality—from the viewpoint of mathematics and logic—naturally plays a vital role was in terms of the so-called P-formulation. For example, to avoid the obstacles presented by directly coping with equality, the axiom of left identity in a group was almost always represented as

P(e,x,x)

rather than as

f(e,x) = x

which is unfortunate. Perhaps this practice is justified by the fact that the field had been in existence for only a few years; on the other hand, perhaps researchers should have been more aggressive. Now, at any rate, we recommend that, when the equality relation naturally dominates the description of a problem domain, the P-formulation be avoided where possible.

With this introduction in hand, let us now turn to a brief discussion of notation, and then to the field of combinatory logic from which we have taken the problems. Combinatory logic, one of the deepest areas of mathematics and logic, offers many problems to test—and perhaps surpass—a theorem-proving program's capacity to solve problems. This field also offers many opportunities to use such a program to answer various currently open questions, and challenges one to formulate new approaches and strategies. With regard to the former, we include problems taken from our successful attack on some of those questions. To illustrate the latter, we include a brief discussion of a new strategy which was formulated to increase the effectiveness of our programs when used for studying various types of combinators. The object of the new strategy is to sharply restrict the application of paramodulation. Since the strategy did in fact prove very useful for our studies in combinatory logic, it or a variant of it might be of use for studies of other fields of mathematics or logic.

2. Combinatory Logic

Before we discuss the first problem, let us supply the needed background, beginning with notation. In all of the problems we offer, as commented earlier, we heavily emphasize equality and equality-oriented notation. Because we wish to emphasize that the equality relation is treated as a built-in relation, we write = or ≠ between the arguments of a literal, rather than using a predicate such as EQUAL or ¬EQUAL followed by its two arguments.

In other words, we do not precisely present clauses to characterize each problem but, instead, use abbreviated clauses, which we simply call clauses. One can, of course, attack the problems we present within ordinary first-order predicate calculus, which is obviously necessary if one's program lacks the facility for treating equality as built in. Nevertheless we strongly recommend that, if possible, the problems be considered within the extended first order predicate calculus where equality does not require axiomatization. If one chooses to follow our recommendation, then one must of course include reflexivity as an axiom if paramodulation is the inference rule to be used. We in fact did use paramodulation heavily in our studies of combinatory logic, the field from which we have taken the problems.

Combinatory logic [Curry58,Curry72,Smullyan85,Barendregt81] is particularly significant for mathematics and logic because it is concerned with the most fundamental aspects of both fields. This field is also of potential interest to automated theorem proving because it challenges the researcher to formulate new strategies to control the reasoning needed to solve problems focusing on combinators and their various properties. In addition, combinatory logic offers one the intriguing opportunity of attempting to answer a number of currently open questions, many of which are amenable to attack with a theorem-proving program playing the role of research assistant. To pique one's curiosity, we shall list some of these open questions.

Combinatory logic can be viewed as an alternative foundation for mathematics—which was Curry's proposal—or viewed as a programming language. On the one hand, the logic offers the same generality and power as set theory in the sense that essentially all of mathematics can be embedded in it. On the other hand, any computable function can be expressed in combinatory logic, and the logic can be used as an alternative to the Turing machine. In fact, combinators have been used as the basis for the design of computers. This logic is concerned with the abstract notion of applying one function to another.

For a more formal definition, we can borrow from Barendregt who defines combinatory logic as a system satisfying the combinators S and K (defined shortly) with S and K as as constants, and satisfying reflexivity, symmetry, transitivity, and two equality substitution axioms for the function that exists implicitly for applying one combinator to another. In other words, since the majority of combinatory logic can be studied strictly within the first-order predicate calculus, this logic is clearly within the province of automated theorem-proving. Even further, although one can clearly operate within ordinary first-order predicate calculus and rely on inference rules such as hyperresolution and UR-resolution, we recommend that one instead study combinatory logic within the extended calculus and use paramodulation as the inference rule. (Of course, one might prefer to rely on an alternative inference rule for building in equality.) Therefore, to study the entire logic, we need only choose an appropriate function symbol, such as *a* to stand for "apply", and supply the axiom for reflexivity and those for the combinators S and K.

x = x
a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
a(a(K,x),y) = x

Even though one can study all of combinatory logic in terms of S and K, one can also study the field or subsets of it by choosing other combinators to replace S and K. Indeed, our focus will shift from one set of combinators to another, depending on the type of problem to be studied. For each combinator of interest, we supply an equation that gives the behavior of the combinator. In such an equation, the combinator appears as a constant. Strictly speaking, a combinator is a member of a class of objects that exhibits the behavior given by its equation. For example, were we being more rigorous, we would say that if the combinator E satisfies

(Ex)y = x

for all combinators *x* and y, then we would say that E is a K since K satisfies

(Kx)y = x,

which we stated earlier in clause form. Therefore, when a problem asks for the construction of a combinator E from a set P of combinators, the object is to find an expression in terms of the elements of P that exhibits the behavior that E does. The solvability of such problems is one of the reasons that the system consisting of S and K alone is studied, for one can always succeed in finding the required expression. Formally, one says that the set consisting of S and K alone is *complete*. One can find other complete sets of combinators by reading one of the general texts on combinatory logic [Curry58,Curry72,Smullyan85,Barendregt81].

To complete the background—especially for those who are new to automated theorem proving—we point out that, if one follows our recommendation of using paramodulation as the inference rule, one will directly encounter the impressive obstacle of coping with equality-oriented reasoning. Overcoming the difficulties inherent when equality plays a vital role is one of the most important research areas in the field, which is one reason for illustrating (with three proofs of Theorem C1.1) the use of different restriction strategies. The strategies for controlling the application of paramodulation include allowing or preventing paramodulation *from* a variable, *into* a variable, *from* the left side of an equality, *from* the right side of an equality, and *into* terms satisfying some given condition concerning their relative position within a statement. Since the advantages and disadvantages vary widely depending on which combination of strategies is employed, by discussing this aspect in the context of different proofs, we may succeed in increasing the interest in the corresponding research.

2.1. Problem 1

For the first problem in this section, we focus on one of the interesting properties, the *weak fixed point property*, that is sometimes present and sometimes absent for some given set P of combinators. From what we can tell, Raymond Smullyan deserves credit as the one to introduce and then study this property. His book *To Mock a Mockingbird* [Smullyan85] is an excellent source for problems and open questions, and a delight to read. We therefore need the following definition.

Definition. If P is a given set of combinators, then the *weak fixed point property* holds for P if and only if for all combinators x there exists a combinator y such that $y = xy$.

For this paper, we can only give the following small hint about why the weak fixed point property and the to-be-defined strong fixed point property are of interest. Gödel's self-referential sentence and Kleene's recursion theorem can be interpreted as applications of fixed point combinators [Barendregt81]. Also, fixed point combinators were known as paradoxical combinators in the early days of combinatory logic, because the Russell Paradox and other paradoxes can be formulated in terms of fixed point combinators.

To study a combinator of the type in which we are interested in here, an equation giving the behavior of the combinator is required. We restrict our attention to combinators that are called *proper*, one such that the left side of its equation is left associated and consists of the combinator followed by some nonempty list of distinct variables, and the right side consists of some or all of the variables that occur on the left side. For example, the combinator S is defined with the equation

((Sx)y)z = (xz)(yz),

which explains why we can, when studying S, use the second clause of the four clauses given earlier, where the function a is given explicitly to show that one combinator is being applied to another.

Theorem C1. The weak fixed point property holds for the set P consisting of the combinators S and K alone, where ((Sx)y)z = (xz)(yz) and (Kx)y = x.

Problem 1 asks for a proof of Theorem C1. The following clauses, in abbreviated notation, characterize this problem. (In contrast, combinatory logic does not explicitly employ a function symbol such as a and observes the convention that all expressions are left associated unless otherwise indicated.)

x = x
a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
a(a(K,x),y) = x
y ≠ a(f,y)

If one assigns a chosen automated theorem-proving program the task of finding a proof for some specific theorem—in particular, for Theorem C1—with the object of testing and evaluating the program, some means must exist for measuring the program's progress. The most obvious means—and perhaps the only significant one—focuses on what percentage of a proof has been found by the program. One must, therefore, have a proof in hand, or be able to complete a proof by using whatever information the program has found. To meet this requirement for Theorem C1, we shall, as promised earlier, supply our own proof. Before we give that proof, let us focus on some simpler problems to provide additional information that might prove useful for attacking Theorem C1 in various ways—Problem 1 admits a number of distinct solutions—and, even more, might prove useful for formulating general strategies. These simpler problems illustrate some of the interesting aspects of the coupling of strategy and inference rule. Each of the simpler problems focuses on proving Theorem C1.1, but proving it under different restrictions.

Theorem C1.1. The weak fixed point property holds for the set P consisting of the combinators S, B, C, and I, where ((Sx)y)z = (xz)(yz), ((Bx)y)z = x(yz), ((Cx)y)z = (xz)y, and Ix = x.

As part of the background, one might find it interesting and useful to note that, just as S and K form a complete set of combinators for combinatory logic, S, B, C, and I form a complete set for that part of the logic known as non-eliminating. A combinator is non eliminating if all of the variables that appear on the left side of its equation also appear at least once on the right side. In other words, from the set consisting of S, B, C, and I, one can construct a combinator of any desired type from these four combinators providing, of course, that the combinator to be constructed is non-eliminating. Naturally, when a given set of combinators is complete for the entire logic or for a large fraction of the logic, one should expect to encounter various hazards when focusing on such a set. Among the obstacles one encounters are the propensity for requiring the use of long expressions to complete a desired construction, the need to strongly consider permitting paramodulation *from* a variable, and the need to strongly consider permitting paramodulation *into* a variable. Each of the simpler problems focusing on proving Theorem C1.1, which we discuss on the path to giving a proof for Theorem C1, illustrates some of these difficulties.

The following six clauses can be used.

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(a(B,x),y),z) = a(x,a(y,z))
(4) a(a(a(C,x),y),z) = a(a(x,z),y)
(5) a(I,x) = x
(6) y ≠ a(f,y)

From this set of clauses, we can quickly and easily give three proofs of Theorem C1.1. Each of the proofs satisfies some given restriction on paramodulation, and corresponds to one of the simple problems we promised to examine. We include all three proofs to illustrate the use of different strategies that one might consider using in other studies. Note that, in the first of the three, we use paramodulation in a fashion that is contrary to our usual recommendations for its use—in particular, paramodulation both *from* and *into* variables occurs. As a form of compensation, we do not allow paramodulation *from* the left side of any equality, and only terms in negative clauses are allowed to be *into* terms. In other words, from a technical viewpoint, we place clause (6) only in the set of support and require every paramodulation to be related to what is called in combinatory logic an *expansion*. For the first proof, therefore, the strategy consists of using set of support and restricting paramodulation to expansions into terms confined to the second argument of an inequality.

Proof 1 of Theorem C1.1

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(a(B,x),y),z) = a(x,a(y,z))
(4) a(a(a(C,x),y),z) = a(a(x,z),y)
(5) a(I,x) = x
(6) y ≠ a(f,y)

from the second argument of clause (3) into term a(f,y) of clause (6)
(7) a(y,z) ≠ a(a(a(B,f),y),z)

from the second argument of clause (5) into the second occurrence of the term z in clause (7)
(8) a(y,z) ≠ a(a(a(B,f),y),a(I,z))

Clause (8) and clause (2) form a unit conflict, which can be seen by letting the variables in clause (2) be *x*, *u*, and *v* in the order in which they occur, and applying the substitution a(B,f) for *x*, I for *u*, and a(a(S,a(B,f)),I) for *v*, *y*, and *z*. Therefore, a contradiction is obtained, and the proof is complete.

Proof 1 shows that indeed the weak fixed point property holds for the set P consisting of S, B, C, and I. Note that the combinator C plays no role in this proof.

To use the proof of Theorem C1.1 to obtain the value for the existentially quantified variable *y* that occurs in the definition of the weak fixed point property, one can adjoin the ANSWER literal—in this case, the literal ANSWER(y)—to the clause corresponding to the denial of the theorem. The answer literal will contain at each point in the proof the current instantiation of the universally quantified variable *y* that exists because of assuming Theorem C1.1 false. One simply takes the argument of the ANSWER literal when unit conflict is found and, taking into account that we began by assuming the theorem false, replaces the constant *f* by the variable *x*. Summarizing, an examination of the unification that establishes unit conflict, the negation of the conclusion of Theorem C1.1, and the path that leads to the unit conflict shows that, for the existentially quantified *y* that one is seeking, one can choose the square of ((S(Bx))I).

As we see in the second proof we give shortly, we can avoid paramodulation *from* and *into* variables if we allow paramodulation *from* the left sides of the various clauses – allow paramodulations related to what are called *reductions* in combinatory logic. The avoidance of paramodulating *from* and *into* variables is often essential for, as is known to those how have experimented with paramodulation or other approaches to building in equality, allowing *from terms* or *into terms* to be variables usually destroys the effectiveness of a theorem-proving program. The reason for such total destruction, for those who may be new to this aspect of the field, is that variables always unify with any chosen expression. This property of never failing to unify would not necessarily be so damaging if a program could separate the needed deductions from the unneeded, but no one has come close at this point in time to discovering a strategy that produces anything resembling such a separation. Such a discovery would deserve and receive overwhelming acclaim, if it is ever made.

Proof 2 of Theorem C1.1

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(a(B,x),y),z) = a(x,a(y,z))
(4) a(a(a(C,x),y),z) = a(a(x,z),y)
(5) a(I,x) = x
(6) y ≠ a(f,y)

from the second argument of clause (3) into term a(f,y) of clause (6)
(7) a(y,z) ≠ a(a(a(B,f),y),z)

from the second argument of clause (2) into term a(a(a(B,f),y),z) of clause (7)
(8) a(v,a(u,v)) ≠ a(a(a(S,a(B,f)),u),v)

from the first argument of clause (5) into term a(u,v) of clause (8)
(9) a(v,v) ≠ a(a(a(S,a(B,f)),I),v)

Clause (9) unit conflicts with clause (1), and the proof is complete.

We can even get a proof that prevents paramodulation from using variables as *from terms* or as *into terms*, and also restricts it to the use of expansions only. However, where the first two proofs fail to use C, the third proof fails to use I.

Proof 3 of Theorem C1.1

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(a(B,x),y),z) = a(x,a(y,z))
(4) a(a(a(C,x),y),z) = a(a(x,z),y)
(5) a(I,x) = x
(6) y ≠ a(f,y)

from the second argument of clause (3) into term a(f,y) of clause (6)
(7) a(y,z) ≠ a(a(a(B,f),y),z)

from the second argument of clause (4) into term a(a(a(B,f),y),z) of clause (7)
(8) a(z,y) ≠ a(a(a(C,a(B,f)),y),z)

Clause (8) unit conflicts with clause (2)—which can be seen by naming the variables in clause (2) *x*, *u*, and *v*, and by substituting a(C,a(B,f)) for *x*, a(S,a(C,a(B,f))) for *u*, *v*, and *y*, and a(a(S,a(C,a(B,f))),a(S,a(C,a(B,f)))) for *z*—and the proof is complete.

An analysis of this third proof shows that, for the *y* that must exist for the weak fixed point property to hold for Theorem C1.1, one can choose the cube of S(C(Bx)) in contrast to the square of ((S(Bx))I) which was found in the first two proofs. Because the set of combinators consisting of S and K alone is complete, we could use either value of y to lead us to a proof of Theorem C1 by expressing certain combinators in terms of S and K. In particular, we could construct two combinators that, respectively, act like B and I, or two that, respectively, act like B and C. One would then use either value of y found by proving Theorem C1.1, but replace the appropriate combinators by the corresponding expressions in terms purely of S and K. However, this indirect path is not what we have in mind when we suggest proving Theorem C1 as the first problem posed in this paper. That indirect path depends on either making a good guess about which other sets of combinators are (sufficiently) complete -- for example, the set consisting of S, B, C, and I—or making a good guess about which sets have the weak fixed point property. Because we had found the two given values of y in other studies, we were able to pursue the indirect path to proving Theorem C1 quickly and easily with the program ITP. But that is not the objective of Problem 1. Instead, we suggest that the theorem-proving program one is evaluating should attempt to prove Theorem C1 directly—using as axioms that for S, that for K, and that for reflexivity--by simply denying that the weak fixed point property holds for the set P consisting of S and K alone. Our attempt to obtain a computer proof of Theorem C1 along the direct path of inquiry failed, which is one reason why we consider Problem 1 to be an interesting challenge.

Since we are suggesting problems as challenges for various theorem-proving programs rather than for theorem-proving people, we shall complete a proof of Theorem C1 rather than assigning that task to the researcher. We shall use Theorem C1.1 in our proof despite the preceding remarks. We take this action to increase the likelihood of independent and uninfluenced experimentation and, as one might suspect, because of certain pedagogical considerations. We shall employ algebraic notation rather than clause notation for this proof.

A Proof of Theorem C1

From Proof 1 of Theorem C1.1, one can conclude that

(1) $((S(Bx))I)((S(Bx))I) = x(((S(Bx))I)((S(Bx))I))$

is true for all x. One can check this equality by simply applying the equations (given as clauses (2), (3), and (5) in Section 2.1) for S, B, and I to the left side of (1) to obtain the right side. Next, one can prove that

(2) $((SK)K)x = x$

for all x, which translates to the statement that (SK)K is an I, or, equivalently, (SK)K behaves as I does. Then one can show that

(3) $(((S(KS))K)x)y)z = x(yz)$

for all x, y, and z. Equation (3) says that (S(KS))K is a B, or, equivalently, behaves like B. Because of equations (2) and (3) and the remarks made concerning their meaning, we can in effect substitute into equation (1) for both B and I to obtain

(4) $((S(((S(KS))K)x))((SK)K))((S(((S(KS))K)x))((SK)K)) =$
$x(((S(((S(KS))K)x))((SK)K))((S(((S(KS))K)x))((SK)K)))$,

which holds for all x, and the proof of Theorem C1 is complete.

We can improve on the result contained in the proof of Theorem C1 by presenting a simpler value for the y that must exist satisfying

$y = xy$,

the equation that defines the weak fixed point property. In particular, if we let y be the square of $(S(S(Kx)))((SK)K)$, we can apply the equations for S and for K and show that this y also satisfies the equation for the weak fixed point property. The simpler y can be found by taking the term Bx in the square of (S(Bx))I and using equation (3) to write B in terms of S and K, then reducing both occurrences of the replaced term with S, and finally using equation (2) to write I in terms of S and K. We therefore have two solutions to Problem 1, and could even obtain a third by focusing on the cube of S(C(Bx)). In the context of Problem 1 and its given solutions, we can immediately pose one of the promised open questions. Does the second solution contain the shortest expression for a y satisfying the weak fixed point property, where y is expressed purely in terms of S and K?

Having finished with Problem 1, we can turn in the next section to the second problem we suggest for testing and evaluating theorem-proving programs. However, in contrast to the treatment we have just given Problem 1, we shall be far briefer from here on, confining the discussion in most cases to the statement of the problem and a short proof in logical or mathematical terms. The copious details we have given regarding Theorem C1.1 can be used as a guide for interpreting the clauses obtained when attempting to solve later problems. They are included also to provide an example of how one can map a logical or mathematical proof into clause notation.

2.2. Problem 2

The second problem we pose asks one to use an automated theorem-proving program to find a proof of Theorem C2, which we state after introducing another new concept. For this problem, we need the definition of the *strong fixed point property*.

Definition. If P is a given set of combinators, then the *strong fixed point property* holds for P if and only if there exists a combinator y such that, for all combinators x, $yx = x(yx)$.

Theorem C2. The strong fixed point property holds for the set P consisting of the combinators B and W alone, where ((Bx)y)z = x(yz) and (Wx)y = (xy)y.

To add to the background for studying problems in combinatory logic, note that the presence for P of the strong fixed point property implies the presence of the weak fixed point property. The converse is not true, as one can see by considering the combinator L with

(Lu)v = u(vv),

noting that the expression (Lx)(Lx) is a y that satisfies the equation for the weak fixed point property which establishes that the weak fixed point property holds for the set P consisting of L alone, and using Theorem C3 to show that the strong fixed point property does not hold for this set P.

The following clauses can be used in the attempt to have a theorem-proving program solve Problem 2.

(1) x = x
(2) a(a(a(B,x),y),z) = a(x,a(y,z))
(3) a(a(W,x),y) = a(a(x,y),y)
(4) a(y,f(y)) ≠ a(f(y),a(y,f(y)))

Even though we recommend this set of clauses for studying Problem 2, we now give a proof more in the style that an algebraist might give.

A Proof of Theorem C2

Let N = ((B((B((B(WW))W))B))B). Since, with the following sequence of equalities, we can show that Nx = x(Nx) for all x, we can set y equal to N to complete our proof. To obtain the sequence, we begin with Nx, occasionally abbreviate (W(B(Bx))) to R, apply the reduction corresponding to B or that corresponding to W depending on the leading symbol of the expression under consideration, and substitute from an intermediate result to deduce the final step.

Nx = ((B((B((B(WW))W))B))B)x = ((B((B(WW))W))B)(Bx) =
((B(WW))W)(B(Bx)) = (WW)(W(B(Bx))) =
(W(W(B(Bx))))(W(B(Bx))) = ((W(B(Bx)))(W(B(Bx))))(W(B(Bx))) = (RR)R =
(((B(Bx))R)R)R = ((Bx)(RR))R = x((RR)R) = x(Nx)

Here we have an example of how automated theorem proving differs sharply from mathematics. Specifically, a theorem-proving program has no way to magically offer an expression, such as N, for use in completing a proof. The mathematician, on the other hand, often exhibits the disarming capacity to make such offers; the offers are based on experience, intuition, and who knows what else. This dichotomy between the approach that apparently must be taken by a theorem-proving program and that which is frequently taken by a mathematician is precisely why the two make a powerful team for solving problems and answering open questions. Indeed, the combinator N, which answered a question that was once open, is just such an example of effective teamwork. This combinator was discovered while we were studying B and W with the assistance of various theorem-proving programs designed and implemented by members of our group.

That same study also led us to formulate a new strategy, mentioned earlier, for sharply restricting the application of paramodulation. The strategy restricts paramodulation to considering an *into* term only if its position vector, to be defined immediately, consists of all 1's, which we express by saying that all paramodulation steps must satisfy the 1's rule. The position vector of a term gives the position of the term within a literal. For example, the position vector [2,3,1] says that the corresponding term is the first subterm of the third subterm of the second argument. For a concrete illustration of the use of position vectors, the third occurrence of the constant W in the equality

a(a(B,a(a(B,a(a(B,a(W,W)),W)),B)),B) = N

has the position vector [1,1,2,1,2,2]. (The fixed point combinator N played a vital role in our discovery of what turned out to be an astoundingly large family of combinators.) We find this strategy of restricting *into* terms to be chosen from those whose position vector consists of all 1's to be very effective for our studies in combinatory logic. The strategy, as we discovered some time after its formulation, focuses on a generalization of what is known in combinatory logic as a *head reduction*.

2.3. Problem 3

For the third problem we suggest for experimentation, we focus on theorem C3, a theorem we proved to answer a question that had been open. The question concerns the possible constructibility, from the combinators B and L, of a fixed point combinator. For the problem under discussion, we depart rather sharply from our usual practice of focusing on proof by contradiction by suggesting instead that an automated theorem-proving program be used to find a model pertinent to Theorem C3. If the program succeeds in finding such a model, then, in a sense that will become obvious upon reading the statement of Theorem C3 which we give immediately, the program will have found a proof of that theorem.

Theorem C3. The strong fixed point property fails to hold for the set P consisting of the combinators B and L alone, where ((Bx)y)z = x(yz) and (Lx)y = x(yy). Equivalently, from B and L alone, one cannot construct a Q such that Qx = x(Qx) for all x.

A Proof of Theorem C3

Assume, by way of contradiction, that the strong fixed point property holds for B and L. Then there exists a combinator Q, which is constructed from B and L alone, such that, for an arbitrary combinator f, Qf = f(Qf). (We use the constant f rather than F to be consistent with our notational convention when denying some theorem is true.) By the Church-Rosser property for combinatory logic, there exists a combinator E such that Qf reduces to E and f(Qf) also reduces to E. (The reductions that are used are paramodulations from the left sides of B and L.) Since f(Qf) reduces to E, and since the first occurrence of f cannot be affected by any reduction with B or L, E must be of the form fT for some combinator T. Therefore, Qf reduces to fT for that same T. The combination Qf obviously has the form CD, where C contains no occurrences of f. Let us consider a one-step reduction of CD, and show by case analysis that the result C′D′ is such that C′ contains no occurrences of f.

Case 1. The reduction involves C only or D only. Obvious.

Case 2. The reduction is with B and involves both C and D. Then C must unify with (Bx)y, D must unify with z, C′ must be the image of x, and D′ must be the image of yz. Therefore, C′ must be a subterm of C, which implies that C′ contains no occurrences of f.

Case 3. The reduction is with L and involves both C and D. Then C must unify with Lx, D must unify with y, C′ must be the image of x, and D′ must be the image of yy. Therefore, C′ must be a subterm of C, which implies that C′ contains no occurrences of f.

We can conclude, therefore, that, regardless of the number of reductions we apply starting with CD = Qf, we can never obtain a combinator of the form C*D* with C* containing an occurrence of f. In particular, we can never reduce Qf to fT, and we have arrived at a contradiction. In other words, the strong fixed point property fails to hold for the set P consisting of B and L alone.

The object of Problem 3 is to find a model that satisfies B and L but fails to satisfy the strong fixed point property. Of course, such a model would show that indeed the strong fixed point property does not hold for the set consisting of B and L alone. Problem 3 has added interest since, as far as we know, no one has yet succeeded in finding such a model—in other words, Problem 3 is an open problem. Of course, since we have just proved Theorem C3, a model with the desired properties must exist. Before we had proved Theorem C3, as commented earlier, the question focusing on the constructibility of a fixed point combinator from B and L alone was open.

An alternative to Problem 3 asks for an automated theorem-proving program to find a proof of Theorem C3 directly, starting with its denial and proceeding in the standard fashion in our field. Such an achievement would be of great interest since the proof we give is outside first-order predicate calculus. However, various researchers in the field have discussed the possibility of using an automated theorem-proving program to prove theorems of this type—theorems whose proof depends on properties of unification, and theorems about unification.

2.4. Problem 4

For Problem 4, we focus on one of the systems of combinators that is known to be complete. As commented earlier, the set P consisting of the combinators S and K is one of those systems—given a combinator E and an equation that characterizes its behavior, one can construct from S and K alone a combinator that behaves as E does. For such a construction, one can apply the well-known algorithm found in Smullyan's book. Alternatively—as is standard in our field—one could have a theorem-proving program attempt such a construction by denying that any expression exists satisfying the equation given for the combinator E under study. If the program succeeds with this approach, then—as occurred in the various proofs of Theorem CI.1—the desired construction is obtained by analyzing the unifications upon which the proof rests.

The object of Problem 4 is to construct from S and K alone, by following the standard approach in automated theorem proving rather than by applying the well-known algorithm for such constructions, a combinator that behaves as the combinator U does, where the equation

(Ux)y = y((xx)y)

gives the behavior of U for all x and all y. The idea is to proceed as we illustrated in Section 2.1 and extract the construction from a proof by contradiction. One can use the following clauses.

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(K,x),y) = x
(4) a(a(z,f(z)),g(z)) ≠ a(g(z),a(a(f(z),f(z)),g(z)))

Similar to our earlier approach, we shall simply give two answers to Problem 4, rather than giving a proof relying on these four clauses. If one affixes the variables x and y to either of the following expressions, and if one then reduces with S and K, one can see that both expressions do indeed behave like U.

((S((S(KS))K))((S(K(S((S((SK)K)((SK)K)))))K))

((S(K(S((SK)K))))((S((SK)K))((SK)K)))

The first of the two expressions can be found with the algorithm for using S and K for such constructions; the second can be found by noting that the combination LO behaves like U, and then reducing a combinator that behaves like LO, where (Lx)y = x(yy) and (Ox)y = y(xy). Question: Is there a combinator, expressed purely in terms of S and K, containing fewer than 13 symbols that satisfies the equation for U?

2.5. Problem 5

Problem 5 focuses on the combinators S and W.

((Sx)y)z = (xz)(yz)

(Wx)y = (xy)y

Problem 5, as with Problem 3, has the object of finding a model. The model one is seeking must satisfy S and W and fail to satisfy the weak fixed point property. The following clauses can be used to search for such a model.

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(W,x),y) = a(a(x,y),y)
(4) y ≠ a(f,y)

Rather than giving a complete proof of the theorem that corresponds to Problem 6, we are content with the following outline. To see that the set consisting of S and W alone does not satisfy the weak fixed point property, we again rely on the Church-Rosser property for combinatory logic. In particular, if the weak fixed point property does hold, then three must exist an E and a T such that both T and fT reduce to E, where f is an arbitrary combinator. The number of leading f's in T is one less than the number in fT. Each so-called reduction with S or W does not increase the number of leading f's in T. Therefore, even with different reduction paths, no expression can exist such that both T and fT reduce to it, which contradicts the existence of E, and the proof outline is complete.

Both the proof we have just outlined and the result concerning S and W, as far as we know, represent a new result in combinatory logic.

2.6. Problem 6

Problem 6 focuses on the combinators S and K.

((Sx)y)z = (xz)(yz)
(Kx)y = x

The object of Problem 6 is to prove that the strong fixed point property holds for the set P consisting of S and K alone. An appropriate combinator can be found by obtaining a refutation of the following clauses

(1) x = x
(2) a(a(a(S,x),y),z) = a(a(x,z),a(y,z))
(3) a(a(K,x),y) = x
(4) a(y,f(y)) ≠ a(f(y),a(y,f(y)))

We give three fixed point combinators that are in effect solutions to Problem 6. We know of no shorter combinator than the one we list third. For readability, we use abbreviated notation with the following abbreviations.

I = (SK)K
M = (SI)I
B = (S(KS))K
W = (SS)(KI)

Here are the three solutions.

((S(K((SI)I)))((S(KW))B))

((S(KM))((SB)(KM)))

((S(S(((SS)I)W)))B)

3. Conclusions

One of the most important activities in automated theorem proving is that of experimenting with various problems taken from mathematics and logic. Experimentation is essentially the only way to measure the power of an automated theorem-proving program or the value of a new idea for increasing that power. In this paper, for such experiments, we focused on problems taken from combinatory logic, a field that is unusually amenable to attack with a theorem-proving program. Other areas from which problems can profitably be taken include ring theory (associative and nonassociative), lattice theory, and the algebra of regular expressions. We also have included various open questions since such questions often promote and provoke experimentation. Our emphasis throughout this paper is on equality. Coping effectively with the equality relation is still one of the major obstacles in the field of automated theorem proving.

References

[Barendregt81] Barendregt, H. P., *The Lambda Calculus: It's Syntax and Semantics*, North-Holland, Amsterdam (1981).

[Curry58] Curry, H. B. and Feys, R., *Combinatory Logic I*, North-Holland, Amsterdam (1958).

[Curry72] Curry, H. B., Hindley, J. R., and Seldin, J. P., *Combinatory Logic II*, North-Holland, Amsterdam (1972).

[McCharen76] McCharen, J., Overbeek, R., and Wos, L., "Problems and experiments for and with automated theorem proving programs", *IEEE Transactions on Computers* C-25, pp. 773-782 (1976).

[RobinsonG69] Robinson, G., and Wos, L., "Paramodulation and theorem-proving in first-order theories with equality", pp. 135-150 in *Machine Intelligence 4*, ed. B. Meltzer and D. Michie, Edinburgh University Press, Edinburgh (1969).

[Smullyan85] Smullyan, R., *To Mock a Mockingbird*, Alfred A. Knopf, New York (1985).

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.