Logical Basis for the Automation of Reasoning: Case Studies[1]

Larry Wos, Robert Veroff, and Gail W. Pieper

Contents

1 Introduction

With the availability of computers in the late 1950s, researchers began to entertain the possibility of automating reasoning. Immediately, two distinctly different approaches were considered as the possible basis for that automation. In one approach, the intention was to study and then emulate people; in the other, the plan was to rely on mathematical logic.

[1]This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and in part by National Science Foundation grant no. CDA-9503064.

In this chapter, we focus mainly on the latter, for logic has proved to provide an excellent basis for the automation of reasoning.

Among the various paradigms for automated reasoning that rely on logic, we concentrate on the *clause language paradigm*. As used in this chapter, the clause language paradigm is characterized by four elements, the *combination* of which distinguishes the clause language paradigm from other paradigms: (1) use of clauses to represent information, (2) retention of clauses deduced during an attack on a question or problem, (3) reliance on strategies to restrict and to direct the reasoning, and (4) emphasis on methods for canonicalization and simplification. When the assignment involves proving a theorem (from mathematics, logic, or some other discipline), the program is given a statement or statements that correspond to assuming the theorem false. The program then begins its attack with the goal of finding a *proof by contradiction*. In other words, the objective for proving a theorem is to establish the unsatisfiability of a set of clauses. The clause language paradigm is discussed in detail in Section 2; proof by contradiction is described in Section 2.2.

The clause language paradigm has evolved and continues to evolve principally as a result of experiments with real problems in mathematics and logic. To provide a perspective for measuring the progress that has occurred and to illustrate graphically the value of experimentation, we present in Section 3 experiments that directly influenced the evolution of the clause language paradigm.

The applications of automated reasoning include research in mathematics and logic, program verification and synthesis, circuit design and validation, and, in general, tasks that depend on logical reasoning. Substantial evidence exists that automated reasoning now occupies a significant position in science, for its use has led to answering open questions in finite semigroups [Winker *et al.*, 1981], in logical calculi [Kalman, 1978; Wos *et al.,* 1984; Harris and Fitelson, 2001], in combinatory logic [Wos and McCune, 1988], and in algebra [McCune and Padmanabhan, 1996; McCune *et al.*, 2001]. To assess the scope and significance of the various contributions, we discuss in Sections 4 and 5 a number of the questions that have been answered by relying heavily on an automated reasoning program.

To provide insight into the possible reasons for the cited successes and also provide an appreciation for how a program based on the clause language paradigm complements the mechanisms on which people rely, we compare in Section 6 the clause language paradigm with both logic programming and person-oriented reasoning.

Despite the marked advances in the field, there remain many obstacles to overcome before automated reasoning reaches its full potential. Far more experimentation is required for the continued evolution of the clause language paradigm and for establishing a metatheory for its most effective use. To encourage such experimentation, we focus in Section 7 on challenge problems for testing, comparing, and evaluating programs, approaches, and ideas. Some of the included problems remain open at the time of this writing, but we conjecture that these are amenable to attack with an existing automated reasoning program. Since a frequent request concerns research topics—suitable for a doctoral dissertation, for example—we present, in Section 8, basic research problems and discuss the current state of automated reasoning.

To further encourage and support research, we include in Section 9 information on appropriate books and various automated reasoning programs that can assist in systemati-

cally seeking shorter proofs, identifying dependent axioms, formulating conjectures, checking proofs, finding new axiom systems, and proving theorems.

We close the chapter by summarizing what we have covered and focusing on the future. The long-term objectives for automated reasoning all focus on providing an automated reasoning assistant, one able to function as a colleague, self-analytically choosing inference rules and strategies, modifying an attack based on current performance, and examining results to highlight those that offer the most significance. This assistant will offer interactive mode, batch mode, and 'collaborative mode'—the last referring to the style that is present when working with a research colleague in which interaction occurs, but only at the highest level.

2 The clause language paradigm

The *clause language paradigm*, as presented in this chapter and as practiced at Argonne National Laboratory since the early 1960s, is one approach to the automation of reasoning. The motivation for developing the clause language paradigm was to prove theorems from various areas of mathematics. The language of clauses was chosen because it seemed a convenient way to convey both specific and general information. Following are three statements about people, written first in everyday language and then in the clause language:

```
Nan is pretty
PRETTY(Nan).

Sheila is not a boxer
-HASAJOB(Sheila, boxer).

All people are female or male but not both
FEMALE(x)  | MALE(x).
-FEMALE(x) | -MALE(x).
```

As the statements show, in the clause language **not** is represented with '`-`' and **or** with '|'. Such a language may, at first, seem to be cumbersome. Why not simply use a natural language? The answer rests in part with the fact that the use of formal mathematical languages eliminates ambiguities inherent in natural language. Of the formal mathematical languages, the clause language has proven to be particularly well suited for automation.

The clause language paradigm includes far more, however, than merely using clauses to present an assignment. Indeed, the clause language paradigm relies on the retention of deduced clauses, and it relies heavily on the use of inference rules to draw conclusions from a problem description, strategies to restrict and to direct the application of the inference rules, and procedures to canonicalize and simplify conclusions or to purge an important class of conclusions that are logically weaker than others present in the set of retained conclusions. To provide the basis for an appreciation of the clause language paradigm and the need for its complexity, we describe in this section the basic components and their interplay.

2.1 Components of the clause language paradigm

During the proof of a theorem, mathematicians typically prove intermediate lemmas. Similarly, early researchers in automated reasoning tacitly assumed that, during the completion of a proof, the best approach is to retain deduced conclusions. They were correct: indeed, to do otherwise sharply interferes with the likelihood of success, at least when the assignment concerns proving an even moderately interesting theorem. (We include evidence when we turn in Section 4 to the topic of answering open questions.)

The decision to retain deduced clauses has an important consequence, however. It virtually forces the use of procedures to manage the database of retained clauses. In particular, two procedures, *subsumption* [Robinson, 1965b] and (when equality is present) *demodulation* [Wos *et al.,* 1967], are essential components of the clause language paradigm. Subsumption is used to purge the clause B in the presence of the clause A when a (not necessarily proper) subclause of B can be obtained from A by instantiation, the uniform replacement of variables by terms. For example, the fact that 'Kim's daughter is female' is a trivial consequence and instance of the fact that 'everybody's daughter is female'; if the latter information is present and the former deduced, the procedure subsumption will purge the less general bit of information. Demodulation is used to canonicalize and simplify clauses by applying demodulators (rewrite rules) chosen by the researcher or found by the program. For example, with the appropriate information present, the program will replace $0 + x$ by x, $-(-x)$ by x, $(xy)z$ by $x(yz)$, and $xy + xz$ by $x(y + z)$.

To see that subsumption is required, one need only examine the results of various experiments in which this procedure is absent or suppressed. One immediately finds clauses that are identical copies of other clauses, and one also finds many clauses that can be obtained from other retained conclusions by instantiation. Such redundancy, which can dramatically decrease the effectiveness of the automated reasoning program, is not caused by the use of the clause language; instead, it is simply a property of reasoning in general, if not appropriately controlled. In contrast to the clause language paradigm, in which proper instances of existing information are avoided, one frequently encounters in mathematics books a sequence of steps containing a pair of assertions, the second of which is a proper instance of the first; we expand on this point when we discuss person-oriented reasoning versus computer-oriented reasoning in Section 6.

The need for the use of demodulation to canonicalize and simplify expressions is less obvious than the need for using subsumption to purge trivial corollaries. However, when one notes that the canonicalization and simplification of expressions often map many different expressions to a single expression (for example, with applications of the demodulator $sum(x, 0) = x$), then one quickly sees how demodulation meshes well with subsumption to decrease the amount of redundancy in the database of retained clauses.

Demodulation also meshes well with the *weighting* strategy, discussed at the end of this subsection. Of course, more directly, one sees the value of using demodulation in a manner somewhat reminiscent of mathematics and logic; various significant differences are discussed when we turn to person-oriented reasoning.

Among the inference rules on which the clause language paradigm relies are binary resolution [Robinson, 1965b], hyperresolution [Robinson, 1965a], UR-resolution [McCharen

et al., 1976], paramodulation [Robinson and Wos, 1969; Wos and Robinson, 1973], and various linked inference rules [Veroff and Wos, 1990; Veroff, 2001a]. Where binary resolution requires the consideration of exactly two (parent) clauses, applications of hyperresolution and UR-resolution may consider several clauses simultaneously. Clauses deduced by applications of hyperresolution consist of positive literals only (literals that do not have the negation symbol). Clauses deduced by applications of UR-resolution are unit clauses—consisting of a single literal, either positive or negative. Paramodulation is an inference rule for equality-oriented reasoning and relies on the built-in treatment of equality. Linked inference rules are designed to generalize and extend well-known inference rules. Currently, no formal metarules exist for dictating an effective choice of inference rule(s). Nevertheless, some guidelines are given in [Wos and Pieper, 1999a].

In contrast to most paradigms, including the majority of dialects of logic programming, the clause language paradigm offers a built-in treatment of equality. Specifically, the predicate EQUAL (or certain abbreviations of that form) and the infix operator '=' are automatically treated as representing an equality relation.

Therefore, if the appropriate inference rules (for instance, paramodulation) are chosen, one need not include the usual axioms for equality—with the exception of reflexivity, $x = x$.

The use of *strategy*, one type designed to restrict reasoning and one type to direct it, is vital to the effectiveness of the clause language paradigm. Other paradigms for the automation of reasoning that rely on only one or two very limited strategies are appealing in that they can prove simple theorems extremely quickly. Nevertheless, such paradigms generally fall short when it comes to proving even moderately interesting theorems. (The Prolog programs designed and implemented by Mark Stickel [1988] are an exception; they have been used to obtain a proof of a deep theorem in equivalential calculus, as discussed in [Wos and McCune, 1992].)

Of the strategies on which the clause language paradigm relies, the two most heavily used are the *set of support strategy* [Wos *et al.,* 1965] to restrict the application of inference rules and *weighting* [McCharen *et al.,* 1976] to direct their application. Intuitively, the set of support strategy requires one to choose a nonempty subset of the input clauses and restrict the reasoning to lines of inquiry that begin with a clause in the chosen subset. To use the set of support strategy, one partitions the set S of input clauses into two subsets, a nonempty subset T (called the initial set of support) and $S - T$; the latter may be empty. The set of support strategy restricts the application of an inference rule by disallowing application of the rule to sets of clauses consisting solely of clauses from the set $S - T$. Put another way, all newly retained clauses are added to T (the set of support), and each set of clauses to which an inference rule is applied must contain at least one clause in T.

In contrast to restricting the application of an inference rule, the weighting strategy is used to direct its application. To use this strategy, one assigns priorities, called *weights*, to the various concepts, relations, and terms. The weights can be based simply on symbol count, or they can be based on a set of elaborate patterns. The intention is to give the researcher the opportunity of guiding the program's search by giving it hints based on knowledge and intuition. By setting a maximum on the weight of a retained clause, the strategy also permits one to impose a cutoff point on complexity; any generated clause

whose weight exceeds the maximum will immediately be purged. The weighting strategy meshes well with demodulation, for consideration of a fact in its simplified form contributes to the ease of determining its significance.

2.2 Interplay of the components

The clause language paradigm functions according to the following constraints. All information must be represented as clauses, and one or more inference rules may be chosen. Among the input clauses, the user must designate a nonempty subset to serve as the initial set of support. (Although technically the set of support strategy must be used, when the chosen subset T is in fact all of the input set S, in the obvious sense one is not using the set of support strategy.) The weighting strategy must also be used; if no weights are assigned by the user, the program implicitly assumes that the weight of any term is its symbol count. The inclusion in the input of a set of demodulators is optional; whichever the choice, one can instruct the program to seek additional demodulators to be used.

A distinction is made in the clause language paradigm between *forward subsumption* and *back subsumption*. In forward subsumption, a newly deduced clause may be subsumed by a clause in the database of retained clauses; the subsumed clause is not added to the database. In back subsumption, a newly deduced clause may subsume a retained clause, causing that clause to be purged from the current database. In many cases, back subsumption is not needed; on the other hand, forward subsumption is virtually a requirement to prevent the automated reasoning program from drowning in redundant information.

Two primary activities are involved in completing a given assignment: processing newly deduced clauses and controlling the deduction of new clauses by selecting the clauses to which to apply the chosen inference rule(s). With regard to clause processing, each clause that is deduced is first rewritten (in the spirit of simplification and canonicalization) as much as possible with demodulation; a positive unit equality that is designated to be used with demodulation is called a demodulator. The demodulated form then is weighed to see whether its weight exceeds the maximum allowed and tested for forward and back subsumption. In the special case that the clause is a unit clause (consisting of a single literal) and accepted for retention in the database, the clause is considered for *unit conflict*—considered with each existing unit clause that is alike in predicate but opposite in sign to see whether a contradiction has been obtained. If a deduced clause in its fully demodulated form is not subsumed and if its weight does not exceed the maximum weight allowed, then the clause is added to the database of retained clauses.

With regard to the second activity, clause selection for inference rule application, two lists of clauses are involved: the 'set of support' list and the 'usable' list. The set of support list consists of clauses in the set of support that have not yet been chosen as the focus of attention. The usable list consists of clauses that either were input but not in the set of support, or are in the set of support but have already been chosen as the focus of attention. Initially, the set of support list contains just those input clauses chosen to be in the set of support.

The main loop for deducing new clauses consists of the following steps:

1. The program chooses from the set of support list the clause with the smallest weight and immediately moves it to the usable list.

2. Each of the inference rules that is in use is then applied to every set consisting of the chosen clause together with some subset of clauses from the usable list. When a clause is deduced, processed, and retained, it is placed on the set of support list—available to be chosen later as the focus of attention.

The program continues to choose clauses from the set of support list, stopping when a particular user-specified limit has been reached: for example, the assigned maximum CPU time is exceeded, the maximum allowed memory is exceeded, the maximum number of clauses permitted to initiate reasoning is exceeded, the set of support list becomes empty, or a proof by contradiction is found. A *proof by contradiction* is found either when two unit clauses have been deduced that are opposite in sign but that unify (unit conflict) or when the empty clause (consisting of no literals) has been deduced.

Reliance on the clause language paradigm does not preclude access to other features—for example, the option of searching for more than one proof in a single run; the choice of various options governing the use of subsumption; and the use of an ANSWER literal to accumulate answers [Green, 1969], capture constructed objects, and otherwise label proofs. Such features are offered in the automated reasoning program OTTER [McCune, 1990], which will be discussed later in this chapter and which relies on the clause language paradigm featured here.

3 A fragment of the history of automated reasoning 1960–2002

A history of the field of automated reasoning would show how far the field has advanced since its beginning, which we mark as approximately 1960. Such a history also would show the importance of experimentation to this advancement—specifically, how the field has gradually evolved as a direct consequence of attempts to solve problems with automated reasoning programs. Rather than presenting a complete history, which would require an entire volume, we focus on a few selected occurrences. These occurrences concern individual questions, problems, and theorems, and the successes and failures resulting from their consideration by some automated reasoning program. Indeed, occurrences of the type we present provided the impetus for the development of the procedures that are the foundation of the powerful reasoning programs that exist today.

The first problem we identify with the field, the Davis-Putnam example [Davis and Putnam, 1960], is not very impressive by today's standards. The Davis-Putnam example asks for a proof of the unsatisfiability of the following set of clauses:

```
P(x,y).
-P(y,f(x,y)) | -P(f(x,y),f(x,y)) | Q(x,y).
-P(y,f(x,y)) | -P(f(x,y),f(x,y)) | -Q(x,f(x,y))
             | -Q(f(x,y),f(x,y)).
```

When one considers that the field began with a study of this problem and that the original approach was simply to accrue, without direction or restriction, ever-increasing sets of variable-free clauses to be tested for truth-functional unsatisfiability, one might naturally wonder why any of the early researchers believed success with interesting questions was even remotely possible. Indeed, could anything substantial grow from success with this small example coupled with an apparently naive approach? The items we take from history and the examples we provide throughout this chapter will in fact show that the field has grown impressively.

Since the Davis-Putnam problem had been considered a challenge for the reasoning programs in the early 1960s, its immediate solution with the use of binary resolution [Robinson, 1965b] provided the impetus for the study of theorems taken from various areas of mathematics. For the key to the development of the clause language paradigm featured in this chapter, as the following occurrences show, we need only look to experimentation—experimentation frequently resulting in (temporary) failure. Such failures provided the force for the formulation and introduction of various types of strategy, of demodulation (to rewrite conclusions by simplifying and canonicalizing them), and of other important procedures.

The first experiment to be discussed concerns proving that groups of exponent 2—those in which the square of every element x is the identity e—are commutative. Even with binary resolution as the inference rule, the attempt to prove the theorem failed; after 2,000 clauses were retained, memory was exhausted (on an IBM 704 computer). An analysis of the experiment revealed that the lack of success rested *not* with properties of binary resolution itself but, rather, with its unrestricted application. Indeed, the program had exhausted the allotted memory by retaining conclusions that, rather than having relevance to the specific theorem under study, were pertinent to groups in general. The analysis of the failure led to the formulation of the set of support strategy (discussed in Section 2.1), the use of which nicely solved the simple exercise. Since then, the set of support strategy has been successfully applied to numerous challenging problems (see, for example, [Wos *et al.,* 1991]).

The second experiment in our discussion of so-called failures in the early 1960s involved an attempt to prove that subgroups of index 2 are normal. An analysis of the failed experiment showed that the proof naturally separates into two cases—one for elements in the subgroup, and another for elements not in the subgroup—and that the two cases were being 'mixed' rather than being considered separately. Such mixing of cases is frequently caused by applying an inference rule to a subset of clauses, two of which are *nonunit clauses*—consisting of more than one literal [Wos *et al.,* 1964]. To succeed with this problem, researchers formulated and used a primitive but natural form of case analysis. By introducing new unit clauses—for example, by splitting over some variable-free tautology—the likelihood of mixing cases is significantly reduced. Obtaining a computer proof of the index 2 theorem marked a clear advance for the field, since the theorem offers some difficulty for a person to prove and is somewhat significant from the viewpoint of mathematics. The index 2 problem has added appeal because the attack that succeeded is representative of the way case analysis should be treated, rather than, say, by automatically splitting every variable-free clause into its individual literals.

The third and final experiment to be discussed in this section concerns an attempt to prove a version of the theorem asserting that the square root of a prime number is

irrational—another experiment that failed initially. A study of the 2,000-clause output from the failed experiment revealed the presence of numerous clauses that exhibited an annoying triviality. Specifically, although the deduction that $0 + a = a$ for some constant a can be interesting, far less inspiring is the deduction of additional trivial variations such as $0+0+a = a$ and $0+a+0 = a$. Although the presence of such unwanted clauses is itself not serious, an analysis showed that the removal of those unwanted clauses and their various descendants would reduce the total number of clauses from 2,000 to 61. The formulation of demodulation (discussed in Section 2.1) followed shortly as a direct result of the analysis. With this procedure, by using the appropriate *demodulators*, various unwanted clauses exhibiting certain triviality are immediately rewritten to more interesting clauses. Frequently, the more interesting clauses are already present, with the result that the rewritten clauses are *subsumed* rather than being added to the database of retained clauses.

The strategies and procedures inspired by the cited examples, as well as the other components of the clause language paradigm, have proven time and time again to be important advances in the field of automated reasoning. The development of most of these features can be traced, historically, to experimentation—more often than not, to experiments that initially failed.

4 Answering open questions

By including the following material—various questions that have been answered with the assistance of an automated reasoning program relying on the clause language paradigm—we address four issues. First, examination of the material shows that the clause language paradigm with all of its complexity and corresponding intricacy of implementation offers impressive power. Second, a study of the successes establishes that automated reasoning has indeed contributed to various fields of mathematics and logic. Third, a comparison of problems solved with today's reasoning programs with those solved in the early 1960s graphically demonstrates that the field has advanced markedly. Fourth, the evidence we present strongly suggests logic to be a powerful, and perhaps superior, basis for the automation of reasoning.

We begin with a brief summary of open questions answered principally in the 1980s. The researchers who posed the questions include I. Kaplansky, J. Kalman, R. McFadden, A. Grau, and R. Smullyan. All of the posed questions were answered by relying heavily on programs based on the clause language paradigm.

1. The Kaplansky question concerned the possible existence of a finite semigroup admitting a nontrivial antiautomorphism but admitting no nontrivial involutions. By definition, a nontrivial antiautomorphism for the semigroup H is a one-to-one mapping f from H onto H such that $f(xy) = f(y)f(x)$ and such that f is not the identity; a nontrivial involution is a nontrivial antiautomorphism whose square is the identity. The assembly language program AURA [Smith, 1988] was used to answer this question, resulting in the discovery of such a semigroup of order 83 [Winker *et al.,* 1981]. Later experiments with AURA showed that the smallest semigroup with the desired properties has order 7 and that four such semigroups exist.

2. The Kalman question [Peterson, 1977; Kalman, 1978] concerned the possibility of seven formulas being single axioms for equivalential calculus. With AURA's assistance, each of the four formulas *XJL*, *XKE*, *XAK*, and *BXO* was shown to be too weak [Wos *et al.,* 1984], and each of the formulas *XHK* and *XHN* was shown to be strong enough to serve as a complete axiomatization [Wos *et al.,* 1991]. The status of the formula *XCB* remains undetermined.

   Kalman also posed a question concerning the possible existence of a shorter proof for the formula *XGK*; he had found a 44-step proof [Kalman, 1978]. This question was answered with AURA by finding a 23-step proof [Wos *et al.,* 1984]. Several years later, using Stickel's [1988] program based on logic programming, Marien [private communication, 1989] discovered a 10-step proof, which, of course, is a more satisfying answer to the question.

3. The McFadden questions concerned the sizes of various finite semigroups, when given their respective generators and rules for determining the value of various products. These questions were answered with the use of OTTER [Lusk and McFadden, 1987]. The largest semigroup of interest has order 102,303.

4. Grau's question concerned the possible independence of the first three of the five axioms for a ternary Boolean algebra. This question was answered with AURA's assistance by finding appropriate models [Winker, 1982].

5. Smullyan [1985] posed a number of questions, all concerning aspects of combinatory logic [Barendregt, 1981]. One such question asked whether the fragment whose basis consists of L alone can satisfy the strong fixed point property. Since L is a *regular* combinator and since one cannot construct a fixed point combinator from a single regular combinator [Statman, 1986; McCune and Wos, 1989], the fragment under consideration cannot satisfy the strong fixed point property.

Turning from this brief summary of results in the 1980s, we now present a slightly more detailed discussion of more recent successes. As one will see, both the nature of the questions and the fields from which they are taken vary widely.

4.1 Robbins algebra

The following basis (expressed here in clause form) was presented for Boolean algebra by E. V. Huntington in the early 1930s.

```
(H1)  x + y = y +x  % commutativity
(H2)  (x + y) + z = x + (y + z)  % associativity
(H3)  n(n(x) + y) + n(n(x) + n(y)) = x  % Huntington equation.
```

Shortly thereafter, H. Robbins conjectured that H3 could be replaced by the following clause:

```
(R3)  n(n(x + y) + n(x + n(y))) = x  % Robbins equation
```

Algebras satisfying H1, H2, and R3 are called Robbins algebras. Since every Boolean algebra is a Robbins algebra, the question arose: Is every Robbins algebra Boolean?

The question intrigued some of the finest minds, including the famed logician A. Tarski and his students [Henkin *et al.*, 1971; Tarski, private communication 1980]—to no avail. Subsequently, S. Winker attacked the problem using a combination of individual insight and an automated reasoning program. He was able to prove that certain conditions suffice to make a Robbins algebra Boolean [Winker, 1990, 1992]. But, with the reasoning programs available, he was unable to prove that any of these conditions follow from the axioms.

Then, in 1996, with the development of a new theorem prover called EQP [McCune, 1997b], the problem was cracked. The 133-step solution, which relied on a technique known as associative-commutative unification [Stickel, 1981], required 20 hours using 18 megabytes on a Unix workstation. The answer—all Robbins algebras *are* Boolean—took the world by storm [McCune, 1997a]. It was covered by the *New York Times* [Kolata, 1996] and was cited as one of five major accomplishments in artificial intelligence by the NEC Research Institute and Computing Research Association [Waltz, 1997]. Perhaps this success will motivate researchers to attack one of the open questions or challenge problems discussed in Section 7.

4.2 Boolean algebra

As further evidence of the importance of having available in the clause language paradigm an arsenal of weapons—strategies, methodologies, and inference rules—to attack deep questions in mathematics and logic, we cite several recent successes in Boolean algebra.

Many of these successes focus on equational systems in terms of the Sheffer stroke. In 1913, Sheffer presented the following three-axiom basis for Boolean algebra in terms of a binary connective now known as the Sheffer stroke, or NAND, that is, $x|y = x' + y'$ [Sheffer, 1913]:

```
(x | x) | (x | x) = x                % Sheffer 1
x | (y | (y | y)) = x | x            % Sheffer 2
(x | (y | z)) | (x | (y | z)) =
     ((y | y) | x) | ((z | z) | x)   % Sheffer 3
```

Using a new technique called 'proof sketches'—a method for systematically generating sequences of clauses that provide valuable guidance and direction in finding a proof [Veroff, 2001b]—R. Veroff proved several new two-axiom systems for Boolean algebra in terms of the Sheffer stroke [Veroff, 2000a, 2000b], including a proof that the following two equations comprise a two-basis for the same theory. (This pair is from a list—consisting of 25 single equations and 2 pairs of equations—presented by Stephen Wolfram in an e-mail February 4, 2000, asking whether OTTER could prove any to be bases.)

```
(x | y) | (x | (y | z)) = x
x | y = y | x
```

This 2-basis result in turn led to proofs that

```
(x | ((y | x) | x)) | (y | (z | x)) = y
```

and

```
((x | y) | z) | (x | ((x | z) | x)) = z
```

(and their corresponding mirror images) are each a 1-basis for Boolean algebra in terms of the Sheffer stroke [McCune *et al.,* 2001]. (The equations were members of Wolfram's cited list of conjectures.)

In addition to these recent successes with Boolean algebra in terms of the Sheffer stroke, McCune found ten short single equational axioms for Boolean algebra in terms of disjunction (**or**, denoted by +) and negation (**not**, denoted by ').

```
(((x+y)'+z')'+((u'+u)'+(z'+x))')=z
(((x+y)'+z)'+(x+(z'+(z+u)')')')'=z
(((x+y)'+z')'+(x+(z+(z'+u)')')')=z
((x+y)'+((x+z)'+(y'+(y+u)')')')'=y
((x+y')'+((x+z)'+(y+(y'+u)')')')=y
((x+y)'+((y'+(z+y)')'+(x+u)')')'=y
((x+y')'+((y+(z+y')')'+(x+u)')')=y
((x+y)'+((y'+(z+y)')'+(u+x)')')'=y
((x+y')'+((y+(z+y')')'+(u+x)')')=y
(((x+y)'+z)'+((z'+(u+z)')'+y)')'=z
```

Each has length 22 and four variables [McCune, 2000]—a distinct improvement over the shortest previously known single axiom of length 131 with six variables.

4.3 Logical calculi

The various logical calculi—including equivalential calculus, sentential calculus, and implicational calculus—have provided a wealth of open questions that, especially in the past five years, have been successfully attacked with automated reasoning programs.

Within the area of equivalential calculus, for example, one question that intrigued researchers in this area was the following: What are the shortest single axioms for the equivalential calculus when the inference rules condensed detachment and reversed condensed detachment both are available? The answer was obtained by Hodgson [1998], using the automated reasoning program OTTER and the model generation program MACE [McCune, 2001].

In a related area, known as two-valued sentential calculus, a complete axiomatization (provided by Łukasiewicz) is given by the following three formulas expressed in clause notation.

```
(L1)  P(i(i(x,y),i(i(y,z),i(x,z)))).
(L2)  P(i(i(n(x),x),x)).
(L3)  P(i(x,i(n(x),y))).
```

The first of the three can be thought of as transitivity, $(x \rightarrow y) \rightarrow ((y \rightarrow z) \rightarrow (x \rightarrow z))$. The second can be read as $(x' \rightarrow x) \rightarrow x$, with $'$ denoting negation (complement), and the third as $x \rightarrow (x' \rightarrow y)$. An axiom set is *complete* for sentential calculus if it is strong enough to permit the deduction of any formula that is true under all assignments of **true** and **false**, with the usual notions of implication and negation. The theorem in question asserts that the Łukasiewicz system is complete for sentential calculus. The strategy for proving completeness is to deduce another system, such as *FL* (Church's), that is known to be complete.

```
(FL1)  P(i(x,i(y,x))).
(FL2)  P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
(FL3)  P(i(i(n(x),n(y)),i(y,x))).
```

Until the early 1990s, the shortest known proof for deducing the three axioms of *FL* was 33 steps in length. Was it possible to find a shorter proof? This question first was answered by OTTER's producing a 29-step proof; the proof was obtained after retaining approximately 103,000 clauses. More impressive, OTTER later found a 21-step proof.

Another startling success in two-valued sentential calculus involved the Łukasiewicz 23-letter formula, the following (where the function i denotes implication; the function n denotes negation; and the variables x, y, z, u, and v are universally quantified and mean 'for all').

```
P(i(i(i(x,y),i(i(i(n(z),n(u)),v),z)),i(w,i(i(z,x),i(u,x))))).
```

Łukasiewicz had presented the formula—without proof—as a single axiom for two-valued sentential calculus; and for more than six decades, a proof remained missing. In the year 1999, however, OTTER found the desired proof [Fitelson and Wos, 2000]. Not counting the axiom under study, the proof has length 200.

Most recently, several new researchers—Z. Ernst, B. Fitelson, and K. Harris–have embraced the use of automated reasoning programs in studies of other logical calculi. Among their numerous successes are the following:

- In RM→ logic (the implicational fragment of Dunn's classical relevance logic, and equivalent to the implicational fragment of Sobociński's three-valued logic S) – provided a 3-basis of size 31 to replace the well-known 4-basis of size 48
- In $C4$ – provided the first known single axiom

  ```
  P(i(i(x,i(i(y,i(z,z)),i(x,u))),i(i(u,v),i(w,i(x,v))))).
  ```

- In $C5$ logic (the implicational fragment of $S5$) – found a basis of size 18, and proved that no smaller basis for $C5$ can exist

4.4 Combinatory logic

Barendregt [1981] defines combinatory logic as an equational system satisfying the combinators S and K, where the respective actions of the constants S and K are given by the following two clauses.

```
EQUAL(a(a(a(S,x),y),z),a(a(x,z),a(y,z))).
EQUAL(a(a(K,x),y),x).
```

The function a in this representation can be interpreted as 'applying' one combinator to another. The result is a combinator.

Rather than focusing on combinatory logic as a whole, here we study various *fragments* of the logic. A fragment is determined by considering sets of (proper) combinators other than S and K; the chosen set is called the *basis* of the fragment under consideration. Given a basis **B** and a combinator P, an expression (solely in terms of the combinators in **B**) that satisfies the equation for P is said to be a *combination* from **B**.

Motivated by correspondence with R. Smullyan, we considered problems concerning various combinations of combinators, including the following (a convention in combinatory logic is that the equations are left associated unless otherwise indicated).

(B) $Bxyz = x(yz)$	(T) $Txy = yx$
(Q) $Qxyz = y(xz)$	(V) $Vxyz = zxy$

Our approach to finding appropriate combinations relied heavily on paramodulation and the set of support strategy. In each case, we used the ANSWER literal to extract (from a proof) the desired combination. We give here two examples; details of the study, including the precise clauses and parameter settings, are in [Wos *et al.,* 1991].

- *From the basis consisting of B and T, can one find a combination that satisfies the equation for Q?* The solution rests with an examination of either of the following two combinators. No shorter combination exists with the desired properties, nor does any other exist of length 6.

  $B(TB)(BBT)$ $\qquad$ $B(B(TB)B)T$

- *From the basis consisting of B and T, can one find a combination that satisfies the equation for V?* The solution rests with an examination of any of the following ten combinations, all found in less than 250 CPU seconds.

$B(T(BBT))(BB(BBT))$	$B(T(BBT))(BB(BTT))$
$B(B(T(BBT))B)(BBT)$	$B(B(T(BBT))B)(BTT)$
$B(T(BBT))(B(BBB)T)$	$B(T(BBT))(B(BBT)T)$
$B(B(T(BBT))(BBB))T$	$B(B(T(BBT))(BBT))T$
$B(B(B(T(BBT))B)B)T$	$B(B(B(T(BBT))B)T)T$

In addition, no shorter combination exists with the desired properties, nor does any other exist of length 10.

Another open question posed by Smullyan [1985] was whether there exists a fixed point combinator constructible from B and W alone; the equation for W is

$$Wxy = xyy$$

The question was answered by Statman [1986] with the construction of $B(WW)(BW(BBB))$. When this question was attacked with the assistance of an automated reasoning program, rather than merely duplicating Statman's success OTTER found four additional combinators of the desired type [McCune and Wos, 1987]. The solution took 20 CPU-hours. Later, after the formulation of the *kernel strategy* [Wos and McCune, 1988; McCune and Wos, 1989], OTTER found all five fixed point combinators in less than 3 CPU-seconds. A startling discovery also was made by Wos: with the kernel strategy, one can construct an infinite class of infinite sets of fixed-point combinators; the smallest has length 8, and the number of that length is five.

$$\Theta_1 = B(B(B(WW)W)B)B$$
$$\Theta_2 = B(B(WW)W)(BBB)$$
$$\Theta_3 = B(B(WW)(BWB))B$$
$$\Theta_4 = B(WW)(BW(BBB))$$
$$\Theta_5 = B(WW)(B(BWB)B)$$

4.5 Ortholattices

Quantum computing has received increased interest in the past few years, as researchers have sought new ways to surmount the limitations of classical computers. One aspect of quantum computing involves *quantum logic,* which describes the probability of events in quantum dynamics.

Using the clause language paradigm, McCune attacked the following open question: Are the following three identities, known to hold for orthomodular lattices, also true for ortholattices? (Here, $\wedge$ is meet, $\vee$ is join, and ' is complement.)

```
(E1) ((x ^ y') v x')' v ((x ^ y') v ((x' ^ ((x v y') ^ (x v y)))
       v (x' ^ ((x v y') ^ (x v y))'))) = 1.
(E2) (x v ((x' ^ ((x v y') ^ (x v y))) v
       (x' ^ ((x' ^ y) v (x' ^ y'))))) = 1.
(E3) ((((x' ^ y) v (x' ^ y')) v (x ^ (x' v y)))' v (x' v y)) = 1.
```

Using the program EQP, McCune proved that E2 and E3 are ortholattice identities; and using the model generation program MACE (which searches for counterexamples), McCune showed that E1 is not an ortholattice identity (MACE found a 10-element ortholattice countermodel) [McCune, 1998]. More recently, Veroff discovered the following generalization of E3:

```
(((x ^ y) v (z ^ u)) v (w ^ (z v y)))' v (z v y) = 1.
```

Answering open questions of this type may someday affect the computational efficiency of systems based on quantum logic.

5 Missing proofs and simpler proofs

The preceding section focused on open questions. As long as a question remains open, the answer to such a question—the proof—is obviously *missing*.

Proofs may, however, be missing for a variety of other reasons. For example, the only known proof may be based on metaargument rather than the explicit use of axioms, as logicians and mathematicians prefer. A proof may exist that uses certain lemmas or terms that the researcher would like to avoid. Or, one may suspect that a proof exists that is shorter—perhaps far shorter—than one given in the literature.

The clause language paradigm, with its numerous and diverse strategies, provides an excellent means for finding missing proofs. For example, while no method exists for *avoiding* the deduction of conclusions relying on a certain lemma or term, demodulation (as well as other techniques) serves well to *discard* unwanted conclusions. If the goal is to find a proof whose longest deduced step is less complex than that in hand, one may use weighting, that is, may instruct the program to 'weigh' each deduced conclusion and discard those that exceed a user-assigned value. If a proof is simply outlined (or, as many mathematics papers declare, 'the proof is obvious'—but missing), one may use proof sketches [Veroff, 2001b]. If no steps are available to be borrowed from an existing proof, one may use the hints strategy [Veroff, 1996] (discussed in Section 6.2). Or one may use the resonance strategy coupled with lemma adjunction [Wos, 2001]; the approach involves using various theorems from a related logic as resonators (described in more detail later in this section), proving lemmas that might (or might not) be pertinent to a proof, and then, in a succeeding run, adjoining them and possibly their proof steps.

No algorithm exists for finding missing proofs. Nevertheless, remarkable strides have been made in this area, as the following selected accomplishments illustrate:

- A proof that derives the three-axiom system of Łukasiewicz from the Meredith single axiom (the following)

  ```
  P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).
  ```

  and relies on no more than five distinct variables.

- A proof of Łukasiewicz's 23-letter single axiom (the following) that is free of double negation; no term of the form $n(n(t))$ for any term t occurs.

  ```
  P(i(i(i(x,y),i(i(i(n(z),n(u)),v),z)),i(w,i(i(z,x),i(u,x))))).
  ```

- A proof of the Wajsberg system [Wajsberg, 1977] for the implicational fragment of two-valued sentential logic:

```
P(c1).
P(i(x,i(c1,x))).
P(i(i(i(x,y),x),x)).
P(i(i(x,y),i(i(y,z),i(x,z)))).
```

Wajsberg's system is apparently weaker than the Tarski-Bernays three-axiom system (the following) for the same logic.

```
P(i(x,i(y,x))).
P(i(i(i(x,y),x),x)).
P(i(i(x,y),i(i(y,z),i(x,z)))).
```

The words *apparently weaker* are used because Wajsberg's system seems to depend on weaker assumptions than the Tarski-Bernays axioms; in fact, however, the two systems are provably equivalent. Wajsberg's proof relies on mathematical induction. In contrast, the OTTER proof is based solely on condensed detachment.

- Condensed detachment proofs of distributive laws from the axioms of Łukasiewicz for infinite-valued sentential logic.

```
(MV1) P(i(x,i(y,x))).
(MV2) P(i(i(x,y),i(i(y,z),i(x,z)))).
(MV3) P(i(i(i(x,y),y),i(i(y,x),x))).
(MV4) P(i(i(n(x),n(y)),i(y,x))).
(MV5) P(i(i(i(x,y),i(y,x)),i(y,x))).
```

(We note that the first four axioms suffice, since the fifth axiom is dependent on the first four [Łukasiewicz, 1970].)

- A proof of the sufficiency of the axiom XCB (the following) plus symmetry for equivalential calculus.

```
P(e(e(x,y),e(y,x))).
```

(It remains an open question whether XCB is a single axiom for equivalential calculus; the cited proof reduces the open question to determining whether symmetry is deducible from XCB.)

A special category of missing proofs focuses on *shorter proofs*—proofs with the fewest deduced steps. Interest in proof length is not a new phenomenon: the logicians Meredith, Prior, and Thomas each studied and published proofs that are shorter than those of their predecessors. And Hilbert apparently also was interested; his recently discovered twenty-fourth problem [Thiele, 2001] focuses on finding simpler proofs.

In part, this interest stems from the elegance of such proofs. But aside from the aesthetic appeal, shorter proofs may be of substantial interest to other disciplines, such as circuit design and computer programming. For example, a proof with fewer steps may provide the key to designing a more efficient component or the key to sharply reducing the CPU time for a subroutine.

Two different approaches exist for systematically searching for shorter proofs with an automated reasoning program. Neither involves what one might expect: a breadth-first search. The reason rests with the fact that subsumption may get in the way. Thus, while such a search may lead to a shorter *subproof*, unfortunately, shorter subproofs do not necessarily lead to a shorter *proof*. Instead, one must seek a way to search for shorter proofs without the interference of subsumption.

One approach relies heavily on the use of the resonance strategy. Resonators, selected from various proofs of problems in related logics, are used to direct the program's reasoning. The resonators are supplemented by the use of demodulators to block retention of conclusions that are known to interfere with obtaining the desired abridgment. In an iterative technique, resonators corresponding to proofs obtained in an initial run are then used for one or more subsequent runs [Wos, 2001].

A different approach relies on applications of linked UR-resolution [Veroff, 2001a] and uses demodulation to categorize derivations. With linked UR-resolution, condensed detachment is applied exhaustively. This has the advantage of extending the search space, of making available for consideration a clause that might otherwise never be considered. The approach has the disadvantage, however, that it is necessarily restricted in practice to relatively small problems.

Both approaches have been used with startling success. Here we limit the list of selected accomplishments to logical systems based on the inference rule condensed detachment.

- In two-valued sentential calculus, the discovery of a 32-step proof for the Łukasiewicz shortest single axiom for the implicational fragment of this logic. That proof is one step shorter than the so-called abridgment published by Meredith and Prior.
- In infinite-valued (or many-valued) sentential calculus, the discovery of a dozen proofs of length 30 that the fifth of Łukasiewicz's five axioms (MV5) is dependent on the other four; the OTTER proofs cut seven steps from the 37-step proof found in the literature.
- In equivalential calculus, the discovery of a 19-step proof that XHN is a single axiom—a startling improvement over the original 159-step proof!

6 The clause language paradigm and other types of reasoning

Because logic programming (Prolog in particular) has attracted the attention of various researchers as a possible basis for automated reasoning programs, we present in this section a summary of various similarities and differences between the logic programming and clause language paradigms. In addition, because the manner in which individuals reason—mathematicians and logicians in particular—also is of interest, we include a brief discussion of person-oriented reasoning. Our primary motivation is to address some common misconceptions and to permit a better understanding and appreciation of the power of the clause language paradigm.

6.1 Logic programming

A comparison of logic programming and the clause language paradigm may seem odd because they address fundamentally different issues. In the strictest sense (with various exceptions for applications to deductive databases, expert systems, and limited aspects of theorem proving), logic programming is concerned with *programming,* designed to easily implement specific algorithms, especially those in which logical reasoning plays an important role. In contrast, the clause language paradigm is concerned with *general-purpose deduction;* the paradigm comprises a collection of procedures that together form a reasoning system designed to attack problems for which one does not know of an effective algorithm to apply. Nevertheless, a comparison is valuable, especially for those who are not familiar with logic programming or with the clause language paradigm discussed in this chapter. The comparison may remove certain points of confusion regarding the possible advantages and disadvantages of each.

The most obvious similarity between logic programming and the clause language paradigm is that both rely on the use of clauses to present an assignment to be completed. The second important similarity is that the type of reasoning used in logic programming is one of the types of reasoning that can be used by an automated reasoning program relying on the clause language paradigm. At the most general level, that reasoning corresponds to ordered input resolution, in which inference rules are restricted to apply to sets of clauses containing at least one input clause and in which the search is ordered in such a way that one proof is constructed at a time. We note that the reasoning also can be viewed as an application of linked hyperresolution [Veroff and Wos, 1990]. At a lower level, every reasoning step of each path that solves a subgoal is a successful application of binary resolution.

In addition, logic programming and the clause language paradigm both rely heavily on the use of the set of support strategy, although logic programming uses a very restricted version of this strategy. Both paradigms also rely on proof by contradiction, and both can use proof by contradiction to construct or discover an object satisfying certain stated properties.

At least four crucial differences between logic programming and the clause language paradigm immediately come to mind, and a number of lesser ones can be given. First, although both paradigms rely on the clause language, classical logic programming languages are restricted to the use of Horn clauses, clauses containing at most one positive literal. (In fact, program clauses are required to be definite, containing exactly one positive literal; only the goal clause is permitted to be all negative.) In the clause language paradigm, however, no restriction is placed on the clauses to be used in presenting a question or a problem. A notable consequence of the restriction to definite Horn clauses is that relationships such as **exclusive or** cannot be expressed naturally in a logic programming language. Some logic programming languages mitigate this restriction, in part, by including a negation operator and interpreting negation as *negation as failure*—an appropriate interpretation for several important applications.

Second, in logic programming—and in theorem-proving programs based on a similar paradigm, for example [Stickel, 1988]—no new clauses are retained during an attack on a question or problem. This lack of retention clearly is one important source of the impressive speed offered by logic programming. The clause language paradigm, on the other hand,

relies heavily on the retention of new information. Although such retention (when all other considerations are ignored) can be expensive both in execution time and in memory usage, we conjecture that—especially for deep questions and hard problems—it is necessary for effectiveness. For one who doubts this necessity, we recommend an attempt to duplicate, by means of logic programming, the successes discussed in Sections 4 and 5. In addition, further insight and appreciation for the clause language paradigm can be gained by accepting the challenges we present in Section 7.

The third important difference focuses on the use of strategy. Compared with the rich use of strategy in the clause language paradigm, logic programming confines its use of strategy to a depth-first and left-to-right search and to a very limited form of the set of support strategy. While restriction to a single search strategy contributes to performance, it can severely limit the effectiveness of the paradigm for solving difficult problems (and can prove disastrous when unbounded searches are permitted).

The fourth significant difference between logic programming and the clause language paradigm is their treatment of equality. ¿From what we know, no satisfactory and fully general treatment currently exists in logic programming. Although we prefer that proofs based solely on equality-oriented reasoning rely solely on paramodulation, we recognize that this inference rule does not by itself solve all of the problems that accompany a natural treatment of equality. As a result, we are still actively searching for ever more powerful strategies to control the application of paramodulation.

In contrast to using paramodulation, some researchers in logic programming have considered building equality into the unification algorithm [DeGroot and Lindstrom, 1986], more or less in the style of E-resolution [Morris, 1969]. Although initial results proved disappointing, researchers continue to explore variants such as rigid E-unification [Degtyarev and Voronkov, 1998]. One of the more promising approaches to building in equality is associative-commutative (AC) unification; see "Opening the AC-Unification Race" [Bürckert *et al.*, 1988]. We note that AC unification together with 'basic' paramodulation (which prevents paramodulation into terms that arise solely by instantiation [Bachmair *et al.*, 1992]) was key to EQP's solving the Robbins problem (see Section 4.1).

Among the other noteworthy differences between logic programming and the clause language paradigm is the absence of the 'occurs check' (for unification in most logic programming languages), in contrast to its required presence in the clause language paradigm. The absence of the occurs check probably contributes to the speed of logic programming, but that speed comes at a cost: soundness is not guaranteed (see [Plaisted, 1984]).

In summary, we conjecture that it will take more than minor extensions to the logic programming paradigm to produce the power needed for effective automated reasoning. After all, if one considers how impressively mathematicians and logicians apply logical reasoning to questions and problems, one should expect that no shortcut exists to producing a computer program that can prove even moderately interesting theorems. ¿From the evidence cited earlier, however, such programs do exist—programs relying on the clause language paradigm.

6.2 Person-oriented reasoning

The clause language paradigm was not intended to imitate a mathematician or logician; indeed, the paradigm exhibits significant differences from the type of reasoning applied by people.

For example, the conclusions drawn by a person are often far less general than those drawn by a program based on the clause language paradigm. An excellent example of this contrast in generality is provided by a person's use of *instantiation* (the uniform substitution of terms for variables) and the computer's use of paramodulation (an inference rule that generalizes the usual notion of equality-oriented reasoning). Clearly, instantiation is used in mathematics and logic, and used wisely. For example, a mathematician might use instantiation to deduce that $(yz)(yz) = e$ (the identity in a group) from the hypothesis that $(xx) = e$, and then use the new conclusion as a key step in proving that the type of group under study is commutative. A typical automated reasoning program would not draw such a conclusion, since instantiation is ordinarily *not* an inference rule offered by such a program. No known strategy exists for wisely choosing appropriate instances, even though mathematicians and logicians do have that ability.

On the other hand, a person finds the use of paramodulation—certainly, by hand—difficult. Perhaps the explanation rests in part with the fact that this inference rule combines the selecting of instances of the appropriate type—those found by unification of an argument and a term—with the use of the usual notion of equality substitution.

With regard to the use of strategy, the analysis is more complicated. Indeed, although a person's reasoning is seldom restricted or directed by the explicit use of strategy, one can find (in such reasoning) examples analogous to the use of strategy by a computer program. For example, the practice in mathematics of restricting the reasoning by keying on the assumed falseness of a theorem when seeking a proof by contradiction is clearly reminiscent of the use of the set of support strategy. As a different example, the preference by mathematicians of one class of expression over another class reminds one of assigning weights (priorities) when using the weighting strategy in the clause language paradigm; assigning weights is somewhat like giving hints for solving a challenging exercise.

While weighting does provide some ability to give hints, however, it is no substitute for a person's intuition. In this regard, person-oriented reasoning has seemed to many to have an advantage over automated reasoning—at least until recently. In the past few years, automated reasoning has closed the gap, with the introduction of three strategies: the resonance strategy, the hints strategy, and the hot list strategy. With resonators [Wos, 1995], the user can include the steps of other proofs, steps conjectured to merit imitation or preference; the objective is to focus on term structure, independent of the names of the variables appearing in the included steps. With hints [Veroff, 1996], the user focuses on individual formulas or equations conjectured to be key milestones on the way to proving a theorem. With the hot list strategy [Wos and Pieper, 1999b], the user can conjecture that certain statements in the problem description merit immediate visiting and even immediate revisiting when conclusions are to be drawn. All three strategies enable the user to impart experience, knowledge, and intuition to an automated reasoning program. All three also have proved to be powerful weapons in finding new proofs and more elegant (simpler) proofs.

7 Challenge problems for testing, comparing, and evaluating

We present in this section various challenge problems, including questions that are still open but that are amenable to attack with an automated reasoning program. Five factors motivate our including these problems. First, no more profitable path than experimentation exists for causing advances in automated reasoning to occur. Second, to accurately measure the value of programs, procedures, techniques, inference rules, strategies, and the like, the nature of the field requires testing and comparing on real questions and problems. Third, for a claim of progress to be valid, it must be substantiated with evidence focusing on reducing the CPU time required to complete an assignment or focusing on expanding the number of successes. Fourth, new ideas in theory, implementation, and application often spring from accepting a challenge; among the more exhilarating experiences is that of answering an open question. Fifth, at least one program—OTTER—offers an autonomous mode for automatically setting options and assigning parameter values; indeed, McCune and Padmanabhan [1996] cite numerous successes using the autonomous mode to answer previous open questions. Even new users of OTTER, therefore, can attack challenging problems.

7.1 Combinatory logic

Combinatory logic presents a number of intriguing open questions with respect to various combinators, including the following.

(B) $Bxyz = y(xz)$
(L) $Lxy = x(yy)$
(S) $Sxyz = xz(yz)$
(Q) $Qxyz = y(xz)$
(N_1) $N_1xyz = xyyz$

Open question. Does the fragment with a basis consisting of the combinators B and S alone satisfy the strong fixed point property? The following clauses capture most of what is needed.

```
% basis of interest
EQUAL(a(a(a(B,x),y),z),a(x,a(y,z))).
EQUAL(a(a(a(S,x),y),z),a(a(x,z),a(y,z))).

% denial of the theorem that there exists a combinator
% satisfying the strong fixed point property
-EQUAL(a(y,f(y)),a(f(y),a(y,f(y)))).
```

If paramodulation is the inference rule, then these clauses, together with reflexivity of equality, suffice. If the inference rule of choice is one of the resolution-based rules, hyperresolution for example, then one must include clauses for reflexivity, symmetry, and transitivity of equality and for substitutivity (of equality) into each of the arguments of the function a.

Open question. Does the fragment with a basis consisting of the combinators B and N_1 satisfy the strong fixed point property? The behavior of the combinator N_1 is given by the following clause.

```
EQUAL(a(a(a(N1,x),y),z),a(a(a(x,y),y),z)).
```

To further encourage research focusing on the automation of model generation, we present here two additional open questions.

Open question. Does there exist a finite model satisfying the combinators B and L in which the strong fixed point property fails to hold?

Open question. Does there exist a finite model satisfying the combinators Q and L in which the strong fixed point property fails to hold?

That some (possibly infinite) model of this type exists (regarding each of the two preceding open questions) follows from results of McCune and Wos [1989].

7.2 Logical calculi

The availability of powerful reasoning programs coupled with the recent formulation of powerful methodologies has made it possible for those with even a cursory knowledge of logic to successfully address challenging problems such as the following.

Open question. Does there exist a proof of the Meredith single axiom

```
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).
```

that requires no more than six variables and that is also free of double negation?

Open question. Does there exist a single axiom for infinite-valued sentential calculus of length less than 69, where (tacitly) the only rule of inference is condensed detachment?

Challenging exercise. As noted in Section 4.3, OTTER has found numerous 30-step proofs that the fifth of Łukasiewicz's five axioms (which he conjectured to suffice for infinite-valued sentential calculus) is in fact dependent on the other four. Those proofs rely solely on condensed detachment. Somewhat easier to consider (hence, the use of the word 'exercise') is proving the dependency of the fifth axiom of Łukasiewicz in an *equality* form (Font *et al.*, 1984), rather than with condensed detachment. For such a study, the following clauses are appropriate, where `T` can be thought of as **true**.

```
(EL1)  EQUAL(i(T,x),x).
(EL2)  EQUAL(i(i(x,y),i(i(y,z),i(x,z))),T).
(EL3)  EQUAL(i(i(x,y),y),i(i(y,x),x)).
(EL4)  EQUAL(i(i(n(x),n(y)),i(y,x)),T).
```

The theorem to prove asserts that the following clause is deducible from EL1 through EL4.

```
(EL5)  EQUAL(i(i(i(x,y),i(y,x)),i(y,x)),T).
```

7.3 Algebra

A glance at the *Journal of Automated Reasoning* or the 1993 special issue on automated reasoning in *Computers and Mathematics with Applications* shows that abstract algebra offers a wealth of problems that can be and have been successfully attacked with an automated reasoning program. Here we present a few open questions from various areas of abstract algebra.

Open question. Does there exist a nice single axiom for the variety of groups of exponent 6, 8, 10, and the like? A variety of groups refers to the set of all groups that satisfy the usual axioms for a group and also satisfy one or more additional equations. Researchers have provided nice single axioms for groups of odd exponent greater than or equal to 3 [McCune and Wos, 1992; Hart and Kunen, 1995] and for the variety of groups of exponent 4 [Kunen, 1995].

Open question. Must every nontrivial conjugacy closed (CC-) loop have a nontrivial (middle) nucleus? To be nontrivial, the CC-loop must contain more than one element. The (middle) nucleus is trivial if and only if every element, other than 1, of the CC-loop is absent from the nucleus. Note also that, in every extra loop (which is a stronger condition than being a CC-loop), the square of every element is in the nucleus. The study of CC-loops originated in [Goodaire and Robinson, 1982], which is where the open question was posed; also of interest is Kunen's recent study of the structure of conjugacy loops [Kunen, 2000].

Challenging problem. Find a first-order proof that the following equation can be derived from HBCK.

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

The quasivariety HBCK is defined by the following set of clauses.

```
      x = x.
(M3)  x * 1 = 1.
(M4)  1 * x = x.
(M5)  (x * y) * ((z * x) * (z * y)) = 1.
(M7)  x * y != 1 | y * x != 1 | x=y.
(M8)  x * x = 1.
(M9)  x * (y * z) = y * (x * z).
(H)   (x * y) * (x * z) = (y * x) * (y * z).
```

Blok and Ferreirim (see [McCune and Padmanabhan, 1996]) indicated that such a proof could be found, but the result relied on a model-theoretic argument. This question remained open until January 2002, when Veroff, using the method of proof sketches, found the desired first-order proof.

8 Current state of the art: Basic research problems to solve

In this section, we present some obstacles to the further advancement of automated reasoning. In the obvious sense, each obstacle is a source of significant research in the field. Although some of the following obstacles do not directly apply to every paradigm for the automation of reasoning, each in some manner merits study.

1. **Clause Generation**: The reasoning program draws far too many conclusions, many of which are redundant and many of which are irrelevant even though they are not redundant.
2. **Clause Retention**: The program keeps too many deduced clauses (too many conclusions) in its database of information.
3. **Size of the Deduction Steps**: The inference rules do not take deduction steps (steps of reasoning) of the appropriate size.
4. **Inadequate Focus**: The program gets lost too easily.
5. **Metarules**: No adequate guidelines exist for selecting the appropriate representation, inference rule(s), strategy or strategies, or the appropriate order to apply procedures such as subsumption, demodulation, weighting, and unit conflict.

A far more extensive and thorough treatment of these obstacles is given in the book *Automated Reasoning: 33 Basic Research Problems* [Wos, 1988]. That book also offers problems for testing possible solutions to the 33 research problems discussed there and provides axiom systems, with the corresponding clauses, for diverse areas of mathematics and logic. (The book, although out of print, is offered on the CD-ROM included in [Wos and Pieper, 1999a].)

In addition to these general obstacles, we discuss here in more detail five problems whose solution would markedly advance the field of automated reasoning. Among the more significant problems to solve is that concerning a built-in treatment of set theory. Next to reasoning focusing on equality, perhaps the most common and important—at least in the context of mathematics—is that focusing on set theory. The problem to solve concerns some means—for example, through the use of inference rules or strategy—to permit an automated reasoning program to treat set theory as if it is 'understood' in the sense that paramodulation permits a program to treat equality as 'understood'. Often, the set-theoretic aspects of a question present a serious obstacle with regard to efficient reasoning by a program. Eventually, a breakthrough will occur: the groundbreaking work of A. Quaife [1992] has recently been extended by J. Belinfante [1999a, 1999b], and the Mizar group (see Section 10) is also active in the study of set theory.

Strikingly different from the preceding problem is that concerning the automated generation of models and counterexamples. Two programs, MACE [McCune, 2001] and SEM [Zhang and Zhang, 1995], have proved useful in finding appropriate counterexamples when a researcher suspects or conjectures that a purported theorem is in fact not provable. Recently,

for example, the programs were used to answer several questions concering ortholattices and orthomodular lattices [McCune, 1998; Rose and Wilkinson, 2001]. Nevertheless, far more progress in model generation is needed to match the impressive progress that has occurred in theorem proving.

In the area of implementation, two problems, if solved, would add, respectively, to the desire to use reasoning programs and to the efficiency with which they complete assignments. The first problem focuses on designing and implementing a sophisticated user interface to permit those unfamiliar with the intricacies of automated reasoning to use reasoning programs. If such an interface were to provide substantial assistance in making wise choices for representation, inference rules, and strategies, mathematicians and logicians would be more likely to seek the assistance of a program to answer a wide spectrum of open questions.

The second problem for implementation concerns the discovery of an efficient means to test nonunit clauses (those containing more than one literal) for forward subsumption. Specifically, given a nonunit clause, the problem is to efficiently retrieve from the database of retained clauses a clause that subsumes the given clause, if such a clause exists. The corresponding problem for unit clauses has been solved with McCune's [1992] adaptation of discrimination trees. Such trees, however, have not yet been adapted to the application of forward subsumption of nonunit clauses; this problem is particularly difficult because it requires finding a clause that subsumes any subset of the literals of the nonunit clause. With various experiments in which nonunit clauses are generated in abundance, one can quickly see the importance of this problem. If one chooses to use subsumption, there is a risk that an increasing fraction of the CPU time will be spent in testing for the subsumption of nonunit clauses. Alternatively, if one chooses to suppress the use of subsumption, there is a risk of a sharp degradation in performance because of the presence of clauses that could be subsumed, sometimes including numerous copies of a single clause. Either choice can be disappointing.

We conjecture that the most significant contributions to automated reasoning will be in the area of strategy—specifically, strategy for restricting the actions of a program. For example, the set of support strategy restricts the application of an inference rule by preventing it from being applied to various subsets of clauses. That strategy, still considered one of the most powerful strategies yet formulated, has the added advantage of requiring no CPU time to exercise. Nevertheless, the effect of the set of support strategy is limited. For a specific research problem to solve, one might study the possibility of extending or generalizing the set of support strategy. For a second specific research problem to solve, one might study possible strategies to control the actions of paramodulation; uncontrolled, the inference rule produces far too many conclusions.

Regardless of the choice of problem, aspect, or area—almost without exception—what is required is experimentation. A sharp increase in experimentation was in fact one of the major forces for the rapid advance of automated reasoning. As evident from the material in the next section, the conditions for easy and substantial experimentation are far better than at any other time in the brief history of the field.

9 Programs, books, and a database: A world of information about a field in its infancy

Various programs and books are available to help the person who wishes to conduct research in automated reasoning, to pursue an application of automated reasoning, or simply to explore what is possible.

9.1 Reasoning programs for diverse applications

The Boyer-Moore theorem prover Nqthm has been used successfully to solve real problems in a variety of fields. Among the theorems proved by Nqthm are various theorems in metamathematics [Shankar, 1985], the invertibility of the Rivest, Shamir, and Adleman public key encryption algorithm [Boyer and Moore, 1984], the soundness and completeness of a propositional calculus decision procedure, the soundness of an arithmetic simplifier (actually used in their program), the termination of Takeuchi's function, and the correctness of many elementary list and tree-processing functions. The Boyer-Moore theorem-proving program has also been used successfully to prove properties of many recursive functions and to prove the correctness (more precisely, the verification conditions) of various Fortran programs, including a fast string-searching algorithm, an integer square root algorithm using Newton's method, and a linear-time majority vote algorithm. These Fortran programs are each relatively small, requiring no more than a page of code. Nevertheless, the correctness arguments are fairly deep.

If one is seriously interested in using the Boyer-Moore program, or if one wishes to gain a full appreciation of their impressive achievement, the book *A Computational Logic Handbook* [Boyer and Moore, 1998] serves well. The book gives a cursory overview of the structure of the prover, describes the logic of the prover (both formally and informally), describes the commands for using it (both abstractly and with examples), and includes an installation guide. Also of interest is a brief tutorial on interaction with the Boyer-Moore theorem prover [Kaufmann and Pecchairi, 1996].

Whereas the Boyer-Moore program sets the standard for applications focusing on proving properties of programs and algorithms, McCune's program OTTER [McCune, 1990] has become the benchmark for proving theorems from various fields of mathematics and logic. OTTER is written in C and runs on various workstations, personal computers, and the Macintosh. Unlike the Boyer-Moore program, OTTER does not offer induction; nor does it offer an interactive facility. It does, however, offer a number of inference rules, various strategies, demodulation, subsumption, the ANSWER literal, and a host of options and special procedures. OTTER's arsenal of weapons has enabled the program to play a key role in answering open questions from various fields, including combinatory logic, finite semigroups, and a number of logical calculi. Not only does this program perform at impressive speed, but its performance barely changes with time or the size of the database of retained information. Indeed, after 19 CPU-hours or after the retention of 900,000 clauses or with the use of 20,000 demodulators (rewrite rules), OTTER's performance is not substantially less than with far smaller numbers. If one is interested in seeing how OTTER is used in research, *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial* [Wos,

1996] is an excellent resource. The book present both successes and failures from numerous experiments with OTTER.

Two other reasoning programs deserve mention. The first is ACL2 (A Computational Logic for Applicative Common Lisp), which incorporates an 'industrial strength' version of the Boyer-Moore theorem prover. Just as with Nqthm, models of all kinds of computing systems can be built in ACL2. Moreover, once an ACL2 model has been built, ACL2 can be used as a tool to help prove properties of that model. Such proofs are generally *not* automatic; ACL2 is intended as an interactive theorem prover, benefiting from conjectures and advice from the user. Users of the ACL2 system have modeled and proved properties of hardware designs, microprocessors, microcode programs, and software; in one recent example, researchers have developed a theorem prover that is coded in ACL2 and makes calls to the reasoning program OTTER to search for proofs [McCune and Shumsky, 2000]. In addition, ACL2 has been used to prove many theorems in mathematics and metamathematics [Kaufmann *et al.*, 2000; Kaufmann and Moore, 2001]. Currently (in 2002), ACL2 is being used by the chip manufacturer AMD in the context of verification.

The second program that has received considerable attention recently is the equational prover Waldmeister [Hillenbrand *et al.*, 1999], which has won a number of theorem-proving contests. Waldmeister is an implementation of unfailing Knuth-Bendix completion. The system architecture is built on a stringent distinction between active facts (selected rewrite rules) and passive facts (critical pairs). This distinction helps speed the most costly operations; an indexing method known as 'refined perfect discrimination tree indexing' is used. While Waldmeister can handle problems in many areas, for difficult proof tasks the user must classify the problem with respect to its algebraic structure and then use a reduction ordering and heuristic assessment of passive facts.

Of a strikingly different nature from the cited programs is the program of Shang-Ching Chou [1988]. Chou's program is remarkable in its ability to prove one theorem after another from classical geometry. Indeed, Chou and his colleagues have used their prover to produce short and readable proofs for numerous theorems (not including inequalities) in Euclidean geometry [Chou *et al.*, 1996].

9.2 Books for background and further research

For those interested in starting with the basics of automated reasoning, several books are useful. More than a quarter of a century ago, before the term 'automated reasoning' had been introduced, Chang and Lee [1973] wrote a book called *Symbolic Logic and Mechanical Theorem Proving*. This book provides a thorough treatment of the formalism on which the clause language paradigm is based. It also includes proofs of the theorems that establish the needed logical properties of various inference rules and strategies. Another good choice for a discussion of logic oriented toward automated reasoning is the book by D. Loveland entitled *Automated Theorem Proving: A Logical Basis* [Loveland, 1978].

Most recently, J. Kalman [2001] published a book entitled *Automated Reasoning with Otter*. This scholarly work covers in fine detail such topics as how to convey a problem to an automated reasoning program, how to find a proof by contradiction, and how to reason about equality. It is a unique blend of theoretical and experimental materials, with myriad

examples.

For an introduction to automated reasoning with a focus on successful applications of the field, the book *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning* [Wos and Pieper, 1999a], is recommended. Indeed, various chapters are devoted to applications in circuit design, circuit validation, mathematics, and logic. Included with the book is a CD-ROM containing a wealth of material: a copy of the reasoning program OTTER, numerous input files, output files to study for new ideas, proofs to examine, many unpublished papers about the field, and a copy of the out-of-print book *Automated Reasoning: 33 Basic Research Problems*—itself a treasury of ideas for testing and evaluating proposed solutions to the research problems it presents, axioms and clauses for studying various areas of mathematics and logic, and the axioms and clauses for studying all of set theory within first-order logic by using the Gödel approach.

For a history of automated reasoning—from its earliest stages as mechanical theorem proving to its broad base now as automated reasoning—the two-volume set *Collected Works of Larry Wos* [Wos with Pieper, 2000] is of interest. The book illustrates some of the remarkable successes automated reasoning programs have had in tackling challenging problems in mathematics, logic, and program verification. Of equal interest is the planned book *Automated Reasoning and the Discovery of Missing and Elegant Proofs* by L. Wos and B. Fitelson, which offers numerous examples of the use of automated reasoning to find missing proofs, particularly in the fields of logic calculi and areas of algebra. Both books include open questions and challenge problems, along with input files on a CD-ROM.

With these books, one is well prepared for gaining a full understanding of the basics of automated reasoning, its possible applications, and its successes.

9.3 A database of test problems

G. Sutcliffe and his associates have organized and maintain a fine database of test problems for those who wish to evaluate the capabilities of an automated reasoning program [Sutcliffe and Suttner, 1998]. The collection, called TPTP (Thousands of Problems for Theorem Provers), is a library for system evaluation and comparison. The library contains more than 4,000 test problems, which can be downloaded from the Web The library has been used for several CADE contests in which automated reasoning programs compete in problem solving (see, for example, [Sutcliffe, 2000, 2001]).

10 Summary and the future

In this chapter, we have shown how automated reasoning—the field whose objective is to design and implement computer programs that address the reasoning aspects of problem solving—is effectively based on logic. Indeed, the field often relies on a subset of first-order predicate calculus for the representation of information and on the use of formal inference rules. Key to the success of the clause language paradigm featured in this chapter is the explicit use of types of strategy, among which are those to restrict the actions of the inference rules and those to direct their action. In addition to the use of strategy,

the use of subsumption is required to avoid the retention of a class of information that is logically weaker than existing information. And, especially when equality is present, the use of demodulation is required to canonicalize and simplify information.

The material we present in this chapter provides evidence for the long-standing debate between Minsky and McCarthy, the former asserting that a general emphasis on logic and a specific emphasis on resolution are essentially a waste of time, and the latter endorsing the role of logic and the attempt to axiomatize various areas. McCarthy is winning, as the evidence shows, and winning decisively—at least for the application of automated reasoning. Indeed, in every regard—in number of successes, variety of successes, and significance of successes—automated reasoning programs with logic as the basis have achieved dramatically more than any program based on an approach taken by people. Of such logic-based programs, those based on what has come to be known as the clause language paradigm have received the most recognition. In this chapter, we focus on that paradigm and on various successes with its use.

Because two other approaches to the automation of reasoning—that based on logic programming and that based on person-oriented reasoning—continue to receive attention, we include in this chapter a discussion of each in relation to the clause language paradigm. Logic programming, which grew out of the study of automated theorem proving, has in fact contributed to the advance of automated reasoning. Nevertheless, because logic programming typically lacks the complexity found in the clause language paradigm—in particular, logic programming does not provide access to an arsenal of strategies and inference rules—it seems doomed, at least if the application is mathematics or logic or program verification. Similarly doomed is an approach based strictly on person-oriented reasoning, as characterized by the classical work in artificial intelligence. Indeed, key to the power of an automated reasoning program is its ability to venture where a human would fear to tread—for example, a problem whose proof was retained after the retention of hundreds of thousands of conclusions.

As for the future, the picture is more promising now than at any other time in the history of the field, which some mark as beginning in approximately 1960. Although the progress was far slower in the first two decades than it might have been—too little experimentation in that period accounts for this fact—the past two decades have seen impressive advances, in theory, in implementation, and in application. Included among the significant advances are the research and implementation of associative-commutative unification algorithms [Stickel, 1981; Bürckert *et al.*, 1988; Siekmann, 1989; McCune, 1997b], the development of new strategies [Wos and McCune 1988; Wos, 1995; Wos and Pieper, 1999b; Veroff, 1996, 2001b], the formulation and use of linked inference rules [Veroff and Wos, 1990; Veroff, 2001a], and the design and implementation of powerful programs such as the Boyer-Moore [1988] theorem prover Nqthm and McCune's [1990] OTTER. Of equal significance is the use of reasoning programs by chip manufacturers that include Intel and AMD; Boyer and Moore pioneered this important subfield of program verification.

Two ambitious projects reflect this spirit of optimism about use of automated reasoning for mathematics. The first project, *Mizar*, has as a primary objective the development of software for working mathematicians. The Mizar database includes more than two thousand definitions of mathematical concepts and about twenty thousand theorems; and the Mizar mathematical library consists of more than 650 articles formalizing the fundamentals of

introductory mathematics (www.mizar.org/library). Each of the articles is based on the axioms of the Tarski-Grothendieck set theory; all articles are verified by Mizar to be correct consequences of those axioms.

The second project, the *QED Project,* has as its objective to build a single, distributed, computerized repository that rigorously represents all important, established mathematical knowledge (www.mcs.anl.gov/qed). One recent study on mechanizing set theory [Paulson and Grabczewski, 1996], done in the spirit of the QED Project, provides evidence for the feasibility of this ambitious project. The authors, L. Paulson and K. Grabczewski, conclude that it is possible to mechanize difficult mathematics but that considerable effort may be required, for example, to formalize an intuitive observation or to handle incompatible definitions in standard mathematical concepts.

That mathematicians, logicians, and computer scientists worldwide are engaging in such projects is encouraging. That they are beginning to experiment with the use of automated reasoning programs to attack deep questions is exciting. As more people begin using a reasoning program, progress certainly will accelerate rapidly. Researchers in automated reasoning will benefit by having valuable feedback, more accurately identifying where new strategies are needed or what areas merit attention. And mathematicians and logicians will benefit by having a flawless, tireless automated reasoning assistant.

References

[Bachmair *et al.*, 1992] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *Proceedings of the 11th international conference on automated deduction*, pages 462–76. *Lecture notes in artificial intelligence*, vol. 607, Springer-Verlag, Berlin, 1992.

[Barendregt, 1981] H. P. Barendregt. *The lambda calculus: Its syntax and semantics.* North-Holland, Amsterdam, 1981.

[Belinfante, 1999a] J. G. F. Belinfante. Computer proofs in Gödel's class theory with equational definitions for composite and cross. *J. Automated Reasoning,* 22(3):311–39, 1999.

[Belinfante, 1999b] J. G. F. Belinfante. On computer-assisted proofs in ordinal number theory. *J. Automated Reasoning,* 22(3):341–78, 1999.

[Boyer and Moore, 1984] R. Boyer and J Moore. Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly,* 91:181–9, 1984.

[Boyer and Moore, 1998] R. Boyer and J Moore. *A computational logic handbook*, 2nd ed. Academic Press, San Diego, 1998.

[Bürckert *et al.,* 1988] H. Bürckert, A. Herold, D. Kapur, J. Siekmann, M. Stickel, M. Tepp, and H. Zhang. Opening the AC-unification race. *J. Automated Reasoning,* 4(4):465–74, 1988.

[Chang and Lee, 1973] C. Chang and R. Lee, R. *Symbolic logic and mechanical theorem proving.* Academic Press, New York, 1973.

[Chou, 1988] S.-C. Chou. *Mechanical geometry theorem proving.* D. Reidel, Dordrecht, 1988.

[Chou *et al.*, 1996] S.-C. Chou, X.-S. Gao, and J.-Z. Zhang. Automated generation of readable proofs with geometric invariants, I and II. *J. Automated Reasoning* 17(3): 325–47, 349–70, 1996.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–15, 1960.

[DeGroot and Lindstrom, 1986] D. DeGroot and G. Lindstrom, editors. *Logic programming: Functions, relations, and equations*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Degtyarev and Voronkov, 1998] A. Degtyarev and A. Voronkov. What you always wanted to know about rigid E-unification. *J. Automated Reasoning,* 20(1–2):47–80, 1998.

[Fitelson and Wos, 2000] B. Fitelson and L. Wos. Axiomatic proofs through automated reasoning. *Bulletin of the Section of Logic,* 29(3):125–36, 2000.

[Font *et al.,* 1984] J. M. Font, A. J. Rodriguez, and A. Torrens. Wajsberg algebras. *Stochastica,* 8(1):5–31, 1984.

[Goodaire and Robinson, 1982] E. G. Goodaire and D. A. Robinson. All loop isotopes. *Canadian J. Math.,* 34:662–72, 1982.

[Green, 1969] C. Green. Theorem-proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Michie, editors, *Machine intelligence 4,* pages 183–205. Edinburgh University Press, Edinburgh, 1969.

[Harris and Fitelson, 2001] K. Harris and B. Fitelson. Distributivity in $Ł_{\aleph_0}$ and other sentential logics. *J. Automated Reasoning*, 27(2):141–56, 2001.

[Hart and Kunen, 1995] J. Hart and K. Kunen. Single axioms for odd exponent groups. *J. Automated Reasoning,* 14(3):383–412, 1995.

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

[Hillenbrand *et al.,* 1999] T. Hillenbrand, A. Jaeger, and B. Löchner. Waldmeister—improvements in performance and ease of use. In H. Ganzinger, editor, *Proceedings of the 16th international conference on automated deduction,* pages 232–6. Springer-Verlag, Berlin, 1999.

[Hodgson, 1998] K. Hodgson. Shortest single axioms for the equivalential calculus with CD and RCD. *J. Automated Reasoning,* 20(3):283–316, 1998.

[Huntington, 1933] E. Huntington. New sets of independent postulates for the algebra of logic, with special reference to Whitehead and Russell's *Principia mathematica. Trans. of AMS,* 35:274–304, 1933.

[Kalman, 1978] J. Kalman. A shortest single axiom for the classical equivalential calculus. *Notre Dame J. Formal Logic,* 19(1):141–4, 1978.

[Kalman, 2001] J. Kalman. *Automated reasoning with Otter.* Rinton Press, Princeton, 2001.

[Kaufmann and Moore, 2001] M. Kaufmann and J Strother Moore. Structured theory development for a mechanized logic. *J. Automated Reasoning,* 26(2):161–203, 2001.

[Kaufmann and Pecchairi, 1996] M. Kaufmann and P. Pecchairi. Interaction with the Boyer-Moore theorem prover: A tutorial study using the arithmetic-geometric mean theorem. *J. Automated Reasoning,* 16(1–2):181–222, 1996.

[Kaufmann *et al.*, 2000] M. Kaufmann, P. Pecchairi, and J Strother Moore, editors. *Computer-aided reasoning: ACL2 case studies.* Kluwer Academic Publishers, Dordrecht, 2000.

[Kolata, 1996] G. Kolata. Computer math proof shows reasoning power. *New York Times,* http://www.nytimes.com/library/cyber/week/1210math.html, December 10, 1996.

[Kunen, 1995] K. Kunen. The shortest single axioms for groups of exponent 4. *Computers and Mathematics with Applications,* 29:1–12, 1995.

[Kunen, 2000] K. Kunen. The structure of conjugacy closed loops. *Trans. AMS,* 352:2889–911, 2000.

[Loveland, 1978] D. W. Loveland. *Automated theorem proving: A logical basis.* North-Holland, Amsterdam, 1978.

[Łukasiewicz, 1970] J. Łukasiewicz. In L. Borkowski, editor, *Jan Łukasiewicz: Selected works*, pages 250-77. North-Holland, Amsterdam, 1970.

[Lusk and McFadden, 1987] E. Lusk and R. McFadden. Using automated reasoning tools: A study of the semigroup F2B2. *Semigroup Forum,* 36(1):75–88, 1987.

[McCharen *et al.,* 1976] J. McCharen, R. Overbeek, and L. Wos. Complexity and related enhancements for automated theorem-proving programs. *Computers and Mathematics with Applications,* 2:1–16, 1976.

[McCune, 1990] W. McCune. OTTER 2.0 users guide. Technical report ANL-90/9. Argonne National Laboratory, Argonne, Illinois, 1990.

[McCune, 1992] W. McCune. Experiments with discrimination tree indexing and path indexing for term retrieval, *J. Automated Reasoning,* 9(2):147–67, 1992.

[McCune, 1997a] W. McCune. Solution of the Robbins problem. *J. Automated Reasoning* 19(3):263–76, 1997.

[McCune, 1997b] W. McCune. 33 basic test problems: A practical evaluation of some paramodulation strategies. In R. Veroff, editor, *Automated reasoning and its applications: Essays in honor of Larry Wos,* pages 71–114. MIT Press, Cambridge, Massachusetts, 1997.

[McCune, 1998] W. McCune. Automatic proofs and counterexamples for some ortholattice identities. *Information Processing Letters,* 65:285–91, 1998.

[McCune, 2000] W. McCune. Short single axioms for Boolean algebra with **or,not**. http://www.mcs.anl.gov/∼mccune/ba/ornot/.

[McCune, 2001] W. McCune. MACE 2.0 reference manual and guide. Technical memo ANL/MCS-TM-249, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 2001.

[McCune and Padmanabhan, 1996] W. McCune and R. Padmanabhan. *Automated deduction in equational logic and cubic curves.* Vol. 1095 in *Lecture notes in computer science.* Springer-Verlag, New York, 1996.

[McCune and Shumsky, 2000] W. McCune and O. Shumsky. System Description: IVY. In D. McAllester, editor. *Proceedings of the 12th international conference on automated deduction,* pages 764–8. *Lecture notes in artificial intelligence*, vol. 814, Springer-Verlag, Berlin, 2000.

[McCune and Wos, 1987] W. McCune and L. Wos. A case study in automated theorem proving: Finding sages in combinatory logic. *J. Automated Reasoning,* 3(1):91–108, 1987.

[McCune and Wos, 1989] W. McCune and L. Wos. The absence and the presence of fixed point combinators. *Theoretical Computer Science,* 87:221–8, 1991.

[McCune and Wos, 1992] W. McCune and L. Wos. Application of automated deduction to the search for single axioms for exponent groups. In A. Voronkov, editor, *Logic programming and automated reasoning,* pages 131–6. *Lecture notes in artificial intelligence,* vol. 624. Springer-Verlag, New York, 1992.

[McCune *et al.*, 2001] W. McCune, R. Veroff, B. Fitelson, K. Harris, A. Feist, and L. Wos. Short single axioms for Boolean algebra. *J. Automated Reasoning*, accepted for publication.

[Morris, 1969] E. Morris. E-resolution: An extension of resolution to include the equality relation. In *Proceedings of the international joint conference on artificial intelligence*, pages 287–94. Washington, D.C., 1969.

[Paulson and Grabczewski, 1996] L. Paulson and K. Grabczewski. Mechanizing set theory. *J. Automated Reasoning,* 17(3):291–323, 1996.

[Peterson, 1977] J. Peterson. The possible shortest single axioms for EC-tautologies. Technical report no. 105. Department of Mathematics Report Series. Auckland University, Auckland, New Zealand, 1977.

[Plaisted, 1984] D. A. Plaisted. The occur-check problem in Prolog. *New Generation Computing*, 2(4):309–22, 1984.

[Quaife, 1992] A. Quaife. *Automated development of fundamental mathematical theories.* Kluwer Academic Publishers, Dordrecht, 1992.

[Robinson, 1965a] J. Robinson. Automatic deduction with hyper-resolution. *International J. Computer Mathematics,* 1:227–34, 1965.

[Robinson, 1965b] J. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM,* 12:23–41, 1965.

[Robinson and Wos, 1969] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In B. Meltzer and D. Michie, editors, *Machine intelligence 4,* pages 135–50. Edinburgh University Press, Edinburgh, 1969.

[Rose and Wilkinson, 2001] M. Rose and K. Wilkinson. Application of model search to lattice theory. AAA Newsletter 52, August 2001. http://www.mcs.anl.gov/AAR/issueaugust01/issueaugust01.html).

[Shankar, 1985] N. Shankar. Towards mechanical metamathematics. *J. Automated Reasoning,* 1(4):407–34, 1985.

[Sheffer, 1913] H. Sheffer. A set of five independent postulates for Boolean algebras, with application to logical constants. *Notre Dame J. Formal Logic,* 10:266–70, 1913.

[Siekmann, 1989] J. Siekmann. Unification theory. *J. Symbolic Computation,* 7:207–74, 1989.

[Smith, 1988] B. Smith. Reference manual for the environmental theorem prover, an incarnation of AURA. Technical report ANL-88-2. Argonne National Laboratory, Argonne, Illinois, 1988.

[Smullyan, 1985] R. Smullyan. *To mock a mockingbird.* Alfred A. Knopf, New York, 1985.

[Statman, 1986] R. Statman. Private correspondence with Raymond Smullyan, 1986.

[Stickel, 1981] M. Stickel. A unification algorithm for associative-commutative functions. *J. ACM,* 28:423–34, 1981.

[Stickel, 1988] M. Stickel. A Prolog technology theorem prover. *J. Automated Reasoning,* 4(4):353–80, 1988.

[Sutcliffe, 2000] G. Sutcliffe. The CADE-16 ATP system competition. *J. Automated Reasoning*, 24(3):371–396, 2000.

[Sutcliffe, 2001] G. Sutcliffe. The CADE-17 system competition. *J. Automated Reasoning*, 27(3):227–250, 2001.

[Sutcliffe and Suttner, 1998] G. Sutcliffe and C. Suttner. The TPTP problem library: The CNF release v.1.2.1. *J. Automated Reasoning*, 21(2):177–203, 1998.

[Thiele, 2001] R. Thiele. On Hilbert's 24th problem: Report on a new source and some remarks. AMS/MAA Joint Mathematics Meeting, New Orleans, January 10–13, 2001.

[Veroff, 1996] R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Automated Reasoning,* 16(3):223–239, 1996.

[Veroff, 2000a] R. Veroff. Axiom systems for Boolean algebra using the Sheffer stroke. Technical report TR-CS-2000-15. Computer Science Department, University of New Mexico, Albuquerque, New Mexico, 2000.

[Veroff, 2000b] R. Veroff. Short 2-bases for Boolean algebra in terms of the Sheffer stroke. Technical report TR-CS-2000-25. Computer Science Department, University of New Mexico, Albuquerque, New Mexico, 2000.

[Veroff, 2001a] R. Veroff. Finding shortest proofs: An application of linked inference rules. *J. Automated Reasoning,* 27(2):123–39, 2001.

[Veroff, 2001b] R. Veroff. Solving open questions and other challenge problems using proof sketches. *J. Automated Reasoning,* 27(2):157–74, 2001.

[Veroff and Wos, 1990] R. Veroff and L. Wos. The linked inference principle, I: The formal treatment. *J. Automated Reasoning,* 10(3):287–343, 1990.

[Wajsberg, 1977] M. Wajsberg. *Logical works.* Polish Academy of Sciences, Wrocław, 1977.

[Waltz, 1997] D. L. Waltz. Artificial intelligence: Realizing the ultimate promises of computing. *AAAI,* 18(3):49–52, 1997.

[Winker, 1982] S. Winker. Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions. *J. ACM,* 29:273–84, 1982.

[Winker, 1990] S. Winker. Robbins algebra: Conditions that make a near-Boolean algebra Boolean. *J. Automated Reasoning,* 6(4):465–89, 1990.

[Winker, 1992] S. Winker, Absorption and idempotency criteria for a problem in near-Boolean algebras. *J. Algebra,* 153(2):414–23, 1992.

[Winker *et al.,* 1981] S. Winker, L. Wos, and E. Lusk. Semigroups, antiautomorphisms, and involutions: A computer solution to an open problem, I. *Mathematics of Computation,* 37:533–45, 1981.

[Wos, 1988] L. Wos. *Automated reasoning: 33 basic research problems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[Wos, 1991] L. Wos. Automated reasoning and Bledsoe's dream for the field. In R. S. Boyer, editor, *Automated reasoning: Essays in honor of Woody Bledsoe,* pages 297–345. Kluwer Academic Publishers, Dordrecht, 1991.

[Wos, 1995] L. Wos. The resonance strategy. *Computers and Mathematics with Applications,* 29(2):133–78, 1995.

[Wos, 1996] L. Wos. *The automation of reasoning: An experimenter's notebook with OTTER tutorial.* Academic Press, New York, 1996.

[Wos, 2001] L. Wos. Conquering the Meredith single axiom. *J. Automated Reasoning,* 27(2):175–99.

[Wos and McCune, 1988] L. Wos and W. McCune. Searching for fixed point combinators by using automated theorem proving: A preliminary report. Technical report ANL-88-10. Argonne National Laboratory, Argonne, Illinois, 1988.

[Wos and McCune, 1992] L. Wos and W. McCune. The application of automated reasoning to questions in mathematics and logic. *Annals of Mathematics and AI,* 5:321–370, 1992.

[Wos and Pieper, 1999a] L. Wos and G. Pieper. *A fascinating country in the world of computing: Your guide to automated reasoning.* World Scientific, Singapore, 1999.

[Wos and Pieper, 1999b] L. Wos and G. Pieper. The hot list strategy. *J. Automated Reasoning,* 22(1):1–44, 1999.

[Wos and Robinson, 1973] L. Wos and G. Robinson. Maximal models and refutation completeness: Semidecision procedures in automatic theorem proving. In W. Boone, F. Cannonito, and R. Lyndon, editors, *Word problems: Decision problems and the Burnside problem in group theory,* pages 609–39. North-Holland, New York, 1963.

[Wos *et al.,* 1964] L. Wos, G. Robinson, and D. Carson. The unit preference strategy in theorem proving. In *AFIPS Proceedings of the fall joint computer conference*, vol. 26, pages 615–21. Spartan Books, Washington, D.C., 1964.

[Wos *et al.,* 1965] L. Wos, G. Robinson, and D. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM,* 12:536–41, 1965.

[Wos *et al.,* 1967] L. Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *J. ACM,* 14:698–709, 1967.

[Wos *et al.,* 1984] L. Wos, S. Winker, R. Veroff, B. Smith, and L. Henschen. A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains. *Artificial Intelligence,* 22:303–56, 1984.

[Wos *et al.,* 1991] L. Wos, S. Winker, W. McCune, R. Overbeek, E. Lusk, R. Stevens, and R. Butler. OTTER experiments pertinent to CADE-10. Technical report ANL-89/36. Argonne National Laboratory, Argonne, Illinois, 1991.

[Wos with Pieper, 2000] L. Wos with G. W. Pieper. *Collected works of Larry Wos,* 2 vols. World Scientific, Singapore, 2000.

[Zhang and Zhang, 1995] J. Zhang and H. Zhang. SEM: A system for enumerating models. In *Proceedings of the international joint conference on artificial intelligence*, pages 298–303. Morgan Kaufmann, San Francisco, 1995.