Part XIV

MECHANIZATION OF LOGICAL INFERENCE AND PROOF DISCOVERY

44

The Automation of Sound Reasoning and Successful Proof Finding

LARRY WOS AND BRANDEN FITELSON

1 The Cutting Edge

The consideration of careful reasoning can be traced to Aristotle and earlier authors. The possibility of rigorous rules for drawing conclusions can certainly be traced to the Middle Ages when types of syllogism were studied. Shortly after the introduction of computers, the audacious scientist naturally envisioned the *automation* of sound reasoning – reasoning in which conclusions that are drawn follow logically and inevitably from the given hypotheses. Did the idea spring from the intent to emulate Sherlock Holmes and Mr. Spock (of *Star Trek*) in fiction and Hilbert and Tarski and other great minds in nonfiction? Each of them applied logical reasoning to answer questions, solve problems, and find proofs. But can such logical reasoning be fully automated? Can a single computer program be designed to offer sufficient power in the cited contexts?

Indeed, while the use of computers was quickly accepted for numerical calculations and data processing, intense skepticism persisted – even in the early 1960s – regarding the ability of computers to apply effective reasoning. The following simple (but perhaps deceptive) example provides a taste of the type of argument that might have been used to support this skepticism.

If one is given a puzzle concerning who holds which jobs, is told that the job of nurse is held by a male, and is asked about the possible jobs for Roberta, one quickly concludes that she is not the nurse. How could a computer program rapidly draw this correct conclusion? After all, the computer would not know that Roberta is (implicitly) female, and, of greater usefulness, it would not know that being a female implies that one is not a male. In fact, even a person often does not realize that the latter fact is used in drawing the correct conclusion for this puzzle. Since the answering of deep questions and the solving of hard problems require far more lengthy paths of reasoning, where do things stand today regarding the automation of drawing conclusions that are sound and relevant, and what is the contemporary view concerning this effort?

In answer to the latter question, still debated with vigor and fascination is the value of automation both in the context of inference rule application for drawing conclusions and in the context of useful proof finding, whether the area be mathematics, logic, circuit design, program verification, or puzzle solving. This essay may settle the issue

for many. Indeed, proofs that for decades have eluded some of the greatest logicians and mathematicians have recently been obtained with a single program, William McCune's OTTER (McCune 1994). (Various other reasoning programs exist; some offer far less power, while others are special-purpose programs, for example, designed mainly for program verification. A special-purpose program is in the majority of cases not nearly as useful as a general-purpose program is in the context of attacking a wide variety of deep questions, such as offered by logic and mathematics.) In Section 3, we shall list some of the theorems that had remained elusive for many, many years, theorems that were recently proved by an automated reasoning program and, moreover, proved in but a few CPU-hours. For the eager reader, we note that the material that is offered in Section 2 is not required for an appreciation of the significance of the successes.

The material presented here is at the cutting edge, featuring proofs not found in the literature of the masters that include Hilbert and Ackermann, Tarski and Bernays, Rose and Rosser, Łukasiewicz, and Meredith. The proofs concern results that fall mainly into three classes: those proved in a nonaxiomatic manner, where an axiomatic proof is preferred; those announced without proof; and those whose proof eluded all attempts. To be current, we focus mainly on successes from mid-1998 to the present. Our presentation – emphasizing examples rather than formalism – makes the content of this essay equally accessible to student and researcher alike, and we assume no background. Nevertheless, what we discuss offers depth, scope, and challenge. Among the treasure, one finds that – through automation – various theorems have been proved whose proof waited for many, many years. One also finds open questions to consider, questions that might be attacked using the program OTTER offered in the first of two intriguing new books on automated reasoning (Wos and Pieper 1999, 2000).

Immediately one might ask how a computer program was able to extend the work of great scholars in such an impressive manner. Indeed, not much more than 50 years ago, what did the eminent logician Łukasiewicz fail to see when he asserted that a formalized proof cannot be 'discovered mechanically' but can only be 'checked mechanically' (Łukasiewicz 1948)? (His remark would still have held essentially even in the late 1970s.) Surely the execution speed of today's machine cannot be the answer: logic and mathematics are far too deep to admit such a simple solution in the context of proof finding. Nor can the answer rest with overcoming the obstacle of the implementation of sound reasoning (inference rules); this obstacle was not severe. Can it be (as some prophesied in the 1960s) that a means has been found to effectively emulate the problem-solving skills of the gifted? That explanation also misses the mark, misses it widely, for no such means has yet been devised.

Instead, (for us) the key to the discovery of so many long-sought proofs rests with the reliance on diverse strategies, some to restrict a program's reasoning and some to direct it. (Some authors, including M. Fitting, use the word *heuristic* in a manner similar to our use of the word *strategy*; other authors sometimes use *strategy* in the context of a specific problem rather than to problems in general.) The occasional ease of discovery is startling even to us who have used OTTER for years. We shall illustrate a powerful strategy that restricts reasoning and an effective strategy that directs it. The nature of both strategies, as well as that of numerous others that are offered by OTTER, permits their embedding in many unrelated reasoning programs.

We shall shed much light on the texture of the strategies that are employed, and thus show why the automation of proof finding is often so successful. To address the concerns of many, and to complete much of the picture, among the pressing questions, this essay answers the following. What, if any, are the important differences between a program's reasoning and that of a person? What are the advantages and disadvantages of reliance on a program that applies logical reasoning? What (if any) means are employed to enable a program to reason effectively, in contrast to merely accruing new conclusions until by accident the goal is reached? How does such a program 'know' when an assignment has been completed, in particular, that a proof has been found? To what practical uses can such a program be put? Which significant open questions have been answered by such a program, and how much guidance was provided by the researcher?

2 Automated Reasoning, Principles and Elements

The breakthrough leading to the more successful mechanization (automation) of inference rule application and proof finding can perhaps be traced to the formulation of and adherence to a few principles. The first of these principles asserts that more general statements are preferred over less general. The second (which overlaps the first) concerns the avoidance of what might be termed 'person-oriented reasoning.' To illustrate the two principles, a single example taken from everyday language suffices; it also provides a taste of the language typically used by the more powerful reasoning programs and a glimpse of the typical test (discovery of a proof by contradiction) used to determine assignment completion.

Consider the following two statements, innocently uttered by someone in casual conversation. "Plato likes everybody" and "Nobody likes Plato." A casual interpretation of the given two utterances perhaps leads to mere acceptance and to no conclusion. However, closer inspection shows that the two statements contradict each other. Indeed, formally, the first can be written, for all *x*, LIKES(Plato,x). Where '–' denotes logical **not**, the second can be formally written, for all *y*, –LIKES(y,Plato). The contradiction is quickly made transparent by substituting Plato for both *x* and *y* in the respective two statements. In other words, overlooked in the casual interpretation is the fact that the first statement includes the case that Plato likes himself, where the second says that he does not. Indeed, where everyday language typically would have permitted the two statements to be accepted simultaneously without blanching, logically the two form a contradictory pair.

If the explicit use of 'for all' (*universal quantification*) is removed with the corresponding variables treated as implicitly meaning 'for all,' then one has two examples of a *clause* and a small taste of the basic linguistic unit used to present information to an automated reasoning program. (For OTTER, a variable is denoted by an expression beginning with a letter between lower-case *u* and *z* inclusive.) The example also provides a glimpse of the typical test for assignment completion that an automated reasoning program relies upon.

Regarding both the principle of generality preference and that of person-oriented reasoning avoidance, the detection of contradiction (inconsistency) by the program

does not require the application of the cited substitution to explicitly produce, respectively, the two variable-free statements LIKES(Plato,Plato) and –LIKES(Plato,Plato). Indeed, the program prefers the two (original) statements as uttered, in their given generality, *without* making the substitution that would emulate what a person would most likely do. To further clarify the picture focusing on both the preference for generality and the avoidance of person-oriented reasoning, two additional examples are in order, examples offering more depth.

When a researcher, such as an algebraist, is producing a proof, the conclusions that are presented are often influenced by their intended use. Therefore, although the conclusion in the middle of the presentation that the square of x is the identity e may be justified, instead one might find the conclusion $(yz)\,(yz) = e$. Because of the basic mechanisms relied upon (which will be illustrated), the type of reasoning program under discussion would prefer the conclusion EQUAL(prod(x,x),e) and would avoid EQUAL(prod(prod(y,z), prod(y,z)),e). (Each equality is a clause, more examples of the language used in automated reasoning.) The first equality offers more generality; the second emulates the kind of reasoning more typical of the researcher not relying on a reasoning program.

Although most likely far from obvious, the preference for generality contributes markedly to effectiveness. (For a pertinent example of the type of general reasoning, in the context of equality, applied by a reasoning program but not ordinarily by a person, see the discussion of *paramodulation* in the section entitled "Inference Rules" below.) Also far from obvious, no effective automated technique is known for wisely choosing which of the myriad of less general conclusions to draw, indeed, how to effectively emulate that aspect of person-oriented reasoning. In other words, automated reasoning programs do not offer the type of reasoning called *instantiation*, which can be used to yield the second equality from the first by replacing (instantiating) x by yz. Although instantiation serves logicians and mathematicians well, unless an effective strategy is discovered to control its use, instantiation is unneeded and even unwanted in the context of mechanizing inference rule application and proof finding. Indeed, its use (in effect) conflicts with a reasoning program's preference for generality that in turn contributes to effectiveness.

For the promised second example, an aspect of logic suffices. If one browses with some care in the literature focusing on *implication* (denoted here by i), one finds within proofs the deduction of formulas such as $i(i(x,y),i(x,y))$, where the deduction of the formula $i(z,z)$ would have been justified and sound. Generality was not the choice; rather, the choice was based on what was deemed more convenient for subsequent steps. Similar to the preceding discussion of the two equalities, the automated reasoning program would have deduced the latter formula, because of its generality, and would have avoided the former even though its use emulates the mind of an expert. Because of this practice, with reliance on the program, occasionally more general proofs are found and more general theorems are proved (which we shall cite).

The basic elements of automated reasoning

The paradigm (for the automation of logical reasoning) featured in this essay rests on six elements: a language for presenting the question or problem under study; types of

reasoning (inference rules) for drawing conclusions some of which are adjoined to the supplied information; strategies for controlling the reasoning; a means for simplifying and canonicalizing information; a means for purging types of redundant information; and a means for determining assignment completion (most often, proof finding). Regarding other paradigms, some differ by addition, some by subtraction. Specifically (in the spirit of addition), some offer induction, where such is not the case for the paradigm in focus here. As for subtraction, some paradigms do not retain new conclusions, which (to us) accounts in part for their lack of power compared with that which (for example) OTTER offers by accruing sometimes a vast number of new conclusions. Equally serious, but of a different nature, many paradigms do not emphasize the use of types of strategy, indispensable for attacking deep questions and hard problems in our view. Regarding another crucial omission, some paradigms do not offer a built-in treatment of equality.

Language

For presenting a question or problem for study by an automated reasoning program, the *clause language* (a dialect of first-order predicate calculus) serves nicely. Its lack of richness is an asset, not a liability. Indeed, rich languages offer more obstacles for formulating effective strategies for reasoning within them. However, the nature of the clause language does present at least an annoyance for one who wishes to enlist the aid of a reasoning program.

In the clause language, only two logical connectives are explicitly permitted, **not** (denoted by '–') and **or** (denoted by '|'). Between each pair of clauses logical **and** is present implicitly. In place of logical **if-then** (logical **implies**), **not** and **or** suffice; one simply replaces **if** *P* **then** *Q* with **not** *P* **or** *Q*. This replacement rule dictates what must be done for the logical operator **equivalent.**

Regarding variables, every variable within a clause is implicitly treated as meaning 'for all,' universally quantified. Existentially quantified variables are replaced with appropriate functions, Skolem functions and constants. Explicit quantification is not permitted. The scope of a variable is limited to the clause in which it occurs. Therefore, if a variable, say *x*, appears in two different clauses, it is treated as merely a coincidence, as if the two names are distinct. A few examples illustrate how it works.

For the assertion that Nan **and** Larry like cats, one writes two clauses, (1) LIKES(Nan,cats) and (2) LIKES(Larry,cats). The clause language implicitly assumes (logical) **and** between every pair of clauses. If one prefers to be more formal and be more precise, one writes –IS(x,cat)|LIKES(Nan,x) and its counterpart.

Since programs such as OTTER offer a built-in treatment of equality, one can write for the equality of *x* and minus(minus(*x*)) the clause EQUAL(x,minus(minus(x))). For the statement that for all *x* there exists a *y* with *y* greater than *x*, one writes GREATER(x,f(x)), where the function *f* is a Skolem function introduced for the existentially quantified variable *y*. The clause exhibits the dependence of *y*, in the form of *f*(*x*), on *x*.

Inference rules

At the heart of all of the inference rules that are used by the type of program featured here (of which OTTER is but one example) is a procedure called *unification*, a

procedure that looks for substitutions that modify variables as little as need be to find a common expression. For example, from the clause IS(Snowflake,cat) and the clause –IS(x,cat)|LIKES(Nan,x), a program can deduce LIKES(Nan,Snowflake) by replacing *x* by the constant Snowflake and applying *modus ponens*, the rule that asserts the deducibility of *Q* from the pair *P* and *P* **implies** *Q*. (Recall that logical **if-then**, **implies**, can be replaced by using logical **not** and logical **or**.) Similarly, for the two clauses cited earlier focusing on Plato, (as noted) a substitution into each was possible that yielded a contradiction.

In contrast, if one considers the clause Q(x,x) and the clause –Q(y,f(y)), no contradiction can be found because no appropriate substitution exists. The general rule when applying unification (in the context of the preceding example) asserts that one is not allowed to substitute a term containing as a subterm a variable for that variable. As the following illustrates, most-general substitutions are always what the program seeks to find, which is not always the case in the literature of logic and mathematics (as discussed somewhat differently earlier). In the spirit of syllogism, from the clause –P(x)|Q(x,a,u) and the clause –Q(b,y,v)|R(y,v), the program can deduce the clause –P(b)|R(a,u). Although the reasoning would be sound, the program would not, for example, deduce –P(b)|R(a,a).

To determine whether two expressions are unifiable, one seeks a table of substitutions of terms for variables. An effective approach is to, first, rename all the variables so that no variable name appears in common in the two expressions and, second, proceed left to right, continually updating the table. Unification can fail for a variety of reasons, such as when one finds a term containing a variable opposite that same variable.

The arsenal of inference rules that is offered is not restricted to those that consider hypotheses taken two at a time. Indeed, one of those rules (*hyperresolution*) serves perfectly for the study of many areas of logic, as the following shows. First, consider a mundane example concerning relations among people. From the clause –PARENT(x,y)|–FEMALE(x)|MOTHER(x,y) and the clause PARENT(G,K) and the clause FEMALE(G), an application of hyperresolution yields the clause MOTHER(G,K). This inference rule, which by definition is required to yield clauses free of logical **not**, considers the three clauses simultaneously.

Not far removed from this mundane example is the following incarnation of the inference rule *condensed detachment*, frequently used in logic. Indeed, consider the clause –P(i(x,y))|–P(x)|P(y), which is quite reminiscent of *modus ponens*, asserting that the presence of *x* **implies** *y* and *x* justifies the conclusion of *y*. In the given three-literal clause, the expression unified with the first literal is called the *major premise*, and that unified with the second literal the *minor premise.* If P(i(i(x,y),i(i(y,z),i(x,z)))) is the major premise and P(i(i(u,v),i(v,u))) is the minor premise, the use of condensed detachment yields P(i(i(v,u),z),i(i(u,v),z)) as the conclusion. For the conclusion, no substitution is required for the variables in the minor premise.

Far more complicated (and clearly not easily seen) is the case (taken from equivalential calculus, with the function *i* replaced by the function *e*) in which both the major and minor premises are P(e(e(e(e(x,e(y,z)),e(y,x)),e(z,u)),u)) and condensed detachment is applied. The conclusion that is yielded is P(e(x,x)), requiring a nontrivial substitution for the variables in both the major and the minor premise. Such complicated unifica-

tions are in no way difficult for an automated reasoning program, but they can be tiresome (or worse) for an unaided researcher.

Of the various inference rules, one of the more complicated is *paramodulation*, which enables an automated reasoning program to treat equality as if it is 'understood.' Paramodulation – which is the best example of a computer-oriented inference rule, and one that a person probably should not apply by hand – generalizes the usual notion of equality substitution. The following example illustrates the cited generalization and demonstrates that paramodulation is indeed computer oriented. Paramodulating *from* the equation $x + (-x) = 0$ *into* the equation $y + (-y + z) = z$ yields in a single step the conclusion $y + 0 = -(-y)$.

Strategy

Because the space of deducible conclusions is so huge (many, many millions), without the use of strategy to restrict and strategy to direct the reasoning, an attempt to find significant proofs would be doomed. In contrast to reasoning programs, researchers succeed because of much knowledge, intuition, and experience. But often proofs that are desired escape even the masters. In Section 3, we give examples of such proofs – proofs that were missing for decades, but that were found through automation.

Two strategies provide a fair taste of what is needed and of what has made the difference. The first strategy, the *set of support strategy*, was formulated to restrict a program's reasoning. For this strategy, the term 'special hypothesis' was introduced, referring to that part of the problem presentation that is outside of the set of axioms and conclusion to be proved. In one of the two strongly recommended uses, the strategy allows a program to draw a conclusion only if it can be recursively traceable to either the special hypothesis or the denial of the conclusion to be proved. For example, if one is asked to prove that rings in which the cube of x equals x (for all x) are commutative, the special hypothesis consists of the property that $xxx = x$. The denial of the conclusion, in the preceding, consists of the assumption that such rings are *not* commutative, that there exist two elements a and b with ab not equal to ba.

In general, when one asks a reasoning program to attempt to find a proof, one supplies a set of statements that include those that correspond to assuming that the conclusion of the theorem under study is false. As indicated earlier, the test that is used for assignment completion, especially for the determination that a proof has been completed, is the detection of a contradiction. For the set of support strategy in its purest form, (put another way) the program is restricted from applying the chosen inference rules to sets of hypotheses all of which are members of the axioms. By imposing such a restriction, the program is prevented from exploring the underlying theory and, instead, is forced to key recursively on the special hypothesis and the denial of the conclusion. Often, our preference is to instruct the program to recursively key on the special hypothesis alone, using the denial of the conclusion solely to detect that the assignment has been completed. The following simple syntactic example illustrates the use of the set of support strategy.

Let the axioms consist of three clauses: P|Q; –Q|R; –R|S. Let the special hypothesis consist of the single clause –P, and let the conclusion to be proved consist of the single clause S. The denial of the conclusion is, therefore, –S. The search for a proof can begin by focusing mainly on the axioms until the clause P|S is deduced, and then hyperreso-

lution can be used to consider that clause with the special hypothesis and the denial of the conclusion to show that a contradiction has been found. However, if one imagines the case in which the set of axioms is far, far richer, one can easily conjecture that the program might get lost (among a huge set of deduced-and-retained conclusions) and never find a proof. Instead, with the set of support strategy, keying on the special hypothesis, in succession, Q is deduced, then R, then S, which with –S provides the sought-after contradiction.

In contrast to the preceding strategy (which restricts the reasoning of a program), the *resonance strategy* directs the reasoning. With this strategy, the researcher supplies formulas or equations (resonators) that are deemed attractive in the sense that any formula or equation that is similar to a resonator is given preference for driving the program's reasoning, where 'similar' means that there is an exact match if all variables are treated as indistinguishable. To illustrate the use of the resonance strategy, let us consider the following clauses that axiomatize two-valued sentential (or propositional) calculus, where the function *i* denotes **implication**, the function *n* denotes **negation**, and the predicate *P* denotes 'provable.'

```
%  Łukasiewicz 1 2 3.
P (i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(n(x),x),x)).
P(i(x,i(n(x),y))).
```

If the researcher conjectures that any formula that is similar (in the sense just given) to one of the axioms merits immediate attention, then the three axioms are placed in an appropriate list. Because of being similar to the first of the three axioms – with the use of the resonance strategy – the formula P(i(i(x,x),i(i(y,u),i(x,v)))) will be given prompt consideration for initiating applications of, say, condensed detachment when and if it is deduced and retained.

With either of the two given strategies, the researcher can provide substantial aid to a reasoning program by a judicious choice of, respectively, which clauses to recursively key upon and which to consider heavily and immediately.

Canonicalization and redundancy

Another aspect of automated reasoning that contributes to effectiveness is its ability, when given the appropriate equalities, to automatically canonicalize and simplify information. For example, in the presence of the equality EQUAL(sum(0,x),x), if the program is instructed to do so, new conclusions are automatically rewritten, with subterms of the form sum(0,t) for terms *t* replaced by *t*. The procedure is called *demodulation*. With its use as described, a class of redundant information is purged. In particular, quite similar items are not kept in the many forms that might otherwise be kept. Such is the case for laws that include associativity, where, if so instructed, the program will not retain the many associated forms of a given expression.

Another mechanism, called *subsumption*, is relied upon to purge different identical copies of the same conclusion and, perhaps more important, to purge proper instances of retained conclusions. For example, if a program has retained EQUAL(prod(x,x),e) (for the identity *e*, as in the study of group theory), it will immediately purge through the

use of subsumption items such as EQUAL(prod(prod(y,z),prod(y,z)),e). By taking such an action and others of a more complicated nature, a reasoning program focuses again on generality and avoids emulation (in the sense that a person might retain less general information in the presence of more general).

An intriguing proof

We close this section with an impressively short proof that nicely illustrates: (1) that which an automated reasoning program does well and (2) that which might have eluded fine minds for a long time. The proof is also of value to logic because of focusing on two three-axiom systems for two-valued sentential (or propositional) calculus. The first system (consisting of what Łukasiewicz denotes as theses 19, 37, and 60) was found through automation; the second (consisting of theses 19, 37, and 59) is due to Łukasiewicz himself. Because the two axiom systems share in common two members (theses 19 and 37), what is required (to prove that the set of formulas consisting of 19, 37, and 60 is an axiom system) is a deduction of thesis 59 (whose negation is found as the input clause numbered 6) from theses 19, 37, and 60 (respectively, the input clauses numbered 7 through 9).

When the eminent logician Dana Scott was notified of the following four-step proof, his reaction, by e-mail, was that it might indeed be "a very neat proof that would not be obvious to a human investigator." Scott explained that it is not particularly easy to do unification in one's head – and is he ever right!

A neat proof focusing on the Wos axiom system for two-valued sentential calculus

```
5    [ ]   –P(i(x,y))|–P(x)|P(y).
6    [ ]   –P(i(i(n(p),r),i(i(q,r),i(i(p,q),r))))|$ANS(negation_thesis_59).
7    [ ]   P(i(i(i(x,y),z),i(y,z)))   # label(thesis_19).
8    [ ]   P(i(i(i(x,y),z),i(n(x),z)))   # label(thesis_37).
9    [ ]   P(i(i(u,i(n(x),z)),i(u,i(i(y,z),i(i(x,y),z)))))   # label(thesis_60).
-------------------------------
16   [hyper,5,9,8]   P(i(i(i(x,y),z),i(i(u,z),i(i(x,u),z)))).
23   [hyper,5,16,7]   P(i(i(x,i(y,z)),i(i(i(u,y),x),i(y,z)))).
30   [hyper,5,23,7]   P(i(i(i(x,y),i(i(z,y),u)),i(y,u))).
34   [hyper,5,30,9]   P(i(i(n(x),y),i(i(z,y),i(i(x,z),y))))   # label(thesis_59).
```

Clause (34) contradicts clause (6), and the proof is complete.

3 Significant Successes

We begin the discussion of successes obtained via automation with a success that is of especial satisfaction and significance. The reasons for assigning such importance to it will become clear almost immediately. Where the function *i* denotes *implication*, the function *n* denotes **negation**, and the predicate *P* denotes 'provable,' the success to be discussed first concerns a 23-letter single axiom (the following, found in 1936 by Łukasiewicz) for two-valued sentential (or propositional) calculus.

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

In his 1936 paper (footnote 10), Łukasiewicz suggests how difficult finding proofs of single axioms is. He laments: "Such research is ... so laborious that it cannot be said when, if ever, it will be completed."

What made finding a proof that the given single axiom suffices for two-valued sentential calculus unusually satisfying was the fact that no proof was given by Łukasiewicz, and, from what we can ascertain, no proof was ever published – until the automated reasoning program OTTER was brought into play in mid-1999. In fact, as far as we know, not even a hint was provided in the literature concerning a method for finding such a proof, nor was a hint provided concerning the target for such a proof – although one might surmise that Łukasiewicz had in mind his three-axiom system rather than, say, the axiom system of Hilbert or some other system.

The finding of the desired proof through mechanization can justly be viewed as a reward for adhering to one of the key principles regarding experimentation. In particular, the formulation of a promising methodology demands its testing on difficult problems that are not required to be related to its wellspring. The genesis of the methodology (whose key aspects will be given shortly) was the study of an even shorter single axiom for two-valued sentential calculus, the following.

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

That axiom was provided by Meredith (1953) 16 years after Łukasiewicz presented his 23-letter axiom and (most likely) in response to an implied Łukasiewicz challenge about finding a single axiom with strictly fewer than 23 letters.

Although Meredith supplied what amounts to a 41-step proof (relying, in effect, on condensed detachment), our goal was to find a means for a reasoning program to produce a proof *without* guidance from the researcher. We had sought such an approach for at least five years, and, in mid-1999, we formulated one that indeed produced a proof, a proof substantially different from Meredith's. When we applied the new methodology to the study of the Łukasiewicz 23-letter axiom, in but four runs, in one afternoon, OTTER produced the first proof we had ever seen, one of length 200 (applications of condensed detachment).

Regarding the methodology and its key aspects, first, it is iterative. It relies on the use of the set of support strategy, adjoining to the appropriate list from run n (to be used in run $n + 1$) results obtained in run n. Although various known axiom systems were admitted as targets to determine that a desired proof had been completed, for our attack on the 23-letter formula, the main target was the Łukasiewicz three-axiom system for this area of logic. The resonance strategy also plays a key role. As for resonators – keeping in mind that we had no clue about the nature of the sought-after proof – we chose to use 68 theses proved by Łukasiewicz, theorems that hold in two-valued sentential calculus. Finally, based on numerous earlier successes, we chose to take an action that one might indeed find counterintuitive, for the action on the surface made the task harder to complete. Specifically, we chose to instruct OTTER to avoid the use of double negation, avoid retaining any new conclusion that contained a term of the form $n(n(t))$ for any term t.

As for properties of the 200-step proof that was found, only eight of its steps are among the 68 theses used as resonators, and only 22 of the 200 steps match one of the 68 resonators (treating all variables as indistinguishable). We include this data in part to address the understandable concern that the researcher may have played an unintentional but key role, in other words, provided much guidance. Such was not the case; we were merely testing the methodology, with, of course, the hope that great fortune would occur; we knew nothing relevant to a possible proof. Double-negation terms are indeed absent. Thus we offer the following open question. Where *P* and *Q* may each be collections of formulas, if **T** is a theorem asserting the deducibility of *Q* from *P* such that *Q* is free of double negation, what conditions guarantee that there exists a proof relying solely on condensed detachment all of whose deduced steps are free of double negation?

Among the other successes we obtained through mechanization – some of which were missing for decades – are the following. In infinite-valued sentential calculus, where logical **or** of x and y can be represented with $i(i(x,y),y)$, one can prove that **or** is associative. This area of logic can be axiomatized with the following five formulas (represented as clauses), where the fifth is dependent on the first four.

```
P(i(x,i(y,x))).
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(i(x,y),(y),i(i(y,x),x))).
P(i(i(n(x),n(y)),i(y,x))).
%   Following is MV5, which is a dependent axiom.
P(i(i(i(x,y),i(y,x)),i(y,x))).
```

As far as we know, until mid-1999, no condensed detachment proof (of associativity) had been reported in the literature. Not only did automation find such a proof – where, before, the only published proofs were not purely axiomatic because they relied partially on reasoning in the metatheory – a more general theorem was proved by OTTER. The generality of the basic mechanisms relied upon by automated reasoning, for example, unification, may have been the primary key to finding the more general result.

Next meriting mention are various distributive laws that hold in infinite-valued sentential calculus. Some forms of distributivity have been proved using a combination of axiomatic and metatheoretic reasoning. But some valid forms have eluded proof of any kind. For example, if we define x **or** y as $i(i(x,y),y)$ and x **and** y as $n(i(i(n(x),n(y)),n(y)))$, then **or** and **and** distribute over each other in infinite-valued logic. This can easily be established semantically, but proving these distributivity laws from the complete set of axioms given earlier for infinite-valued logic (together with the rule of condensed detachment) is something that eluded even Rose and Rosser (1959: 12), who wrote the definitive treatise on infinite-valued logic.

Again, mechanization of proof finding met the test, producing the missing proofs based solely on condensed detachment. For the curious who wonder about the appeal of axiomatic proofs and, even more, of proofs relying on a single inference rule, note that they are often more enlightening and often easier to understand.

Of a different nature are questions focusing on possible axiom dependence. Indeed, a book by Epstein (1994) poses several such questions. Automation has quickly settled

some of Epstein's open questions. In one case (Epstein 1994: 85, problem 12), an appropriate dependence proof was found, when OTTER showed how one of the axioms, the axiom $i(x,i(y,x))$ in Epstein's axiomatization of two-valued sentential calculus, could be proven from the others. Then, appropriate models establishing the independence of the remaining set of axioms were found using William McCune's program MACE, which searches for finite models of sets of clauses.

We close this section by turning to questions concerning proof elegance. More than occasionally a theorem has been proved, but the proof is far, far from elegant. It may be much longer than need be, according to experience and intuition. It may rely on formulas that are extremely complex, and, again, educated opinion suggests such is not required. Among the other inelegant features that may be present is that of requiring the use of some unwanted terms. A program such as OTTER has proved useful in all cited areas, often finding a proof offering far more elegance than can be found in the literature. We are content to cite but one example in this essay.

Meredith (1959) proved (in approximately 38 condensed detachment steps, making extensive use of double negation) axiom MV5 of Łukasiewicz's axioms for infinite-valued sentential calculus from the remaining four axioms. Thus he established a dependence within Łukasiewicz's axiomatization. Because of our emphasis on the avoidance of double negation and the conjecture that its avoidance might enable a reasoning program to find shorter proofs, we embarked on an automated search for a shorter, double-negation-free proof of the dependence of axiom MV5. Success was ours: OTTER produced a 32-step proof, and one in which double negation is absent, thus addressing two elements of increased elegance.

For the student or researcher who might enjoy a question focusing on finding a possibly shorter proof, we suggest the Meredith single axiom. Can one find a proof relying on 40 or fewer applications of condensed detachment showing that the Meredith axiom suffices for all of two-valued sentential calculus? To make the open question more precise, the sought-after proof must complete with the deduction of one of the known axiom systems for that area of logic. If one wishes an open question focusing on the need for double negation, the following might be of interest. In infinite-valued sentential calculus, can one find a proof free of double negation that establishes the deducibility from either the four independent or five dependent axioms for that area of logic of the distributive law P(i(i(n(x),n(i(i(n(y),n(z)),n(z)))),n(i(i(n(i(n(x),y)),n(i(n(x),(n(x),z))),n(i(n(x),z)))))))? This question can be rephrased, asking that one prove without the use of double negation that x **or** $(y$ **and** $z)$ **implies** $(x$ **or** $y)$ **and** $(x$ **or** $z)$, where **and** is defined as earlier but **or** is defined as i(n(x),y).

4 Myths, Mechanization, and Mystique

The myths that surround the mechanization of inference rule application and proof finding are many. *Utter pessimism:* effective mechanization is not possible, especially in the context of answering deep, open questions. *Self-worship:* if effective mechanization is possible, emulation of the minds of masters is required. *Uselessness:* one cannot learn from proofs produced from a computer program. The *0/1 myth:* either the program completes the given assignment, or absolutely nothing is produced of value. *Fear:*

reasoning programs will eventually obviate the role of logicians, mathematicians, and the like.

Other than the last given myth (which we shall dispatch shortly), this chapter provides some evidence and some clues that unmask each of the given myths and others unnamed. Indeed, regarding the Utter pessimism myth, the list of open questions (some of which remained open for many decades) that have been answered through heavy reliance on automation is lengthy and continues to grow. As for the Self-worship myth, the more successful and powerful reasoning programs clearly do not emulate person-oriented reasoning. For but two examples, paramodulation (applied so effectively by a computer for equality-oriented reasoning) is the type of inference rule that understandably *is not* used by unaided researchers, and instantiation, which *is* heavily used, is not offered by the type of reasoning program in focus because it appears not to admit effective control.

The Uselessness myth is quickly dispatched. Indeed, although we are far from expert in the areas of logic that we have attacked with OTTER, we do continually learn from the proofs it supplies and, sometimes, from its failures. Even more can, and sometimes is, learned by a master examining the efforts of a reasoning program. Regarding the 0/1 myth, the output file that can be produced may offer a new key lemma and, even better, may contain a proof that the skilled researcher was unable to find unaided. Even if a proof of the desired type is not found, one can study the set of retained conclusions resulting from an unsuccessful attempt and discover precisely what is needed to reach the objective in the next automated attack.

As for the last myth, Fear, it is utter nonsense. The mind of the logician, mathematician, or other scientist will never be replaced, only supplemented! The explanation for the significant contributions to logic and mathematics resulting from the joint effort of program and researcher rests to a great extent with the fact that the general approach (as discussed in this chapter) taken by the more effective reasoning programs differs sharply from that taken by the successful researcher. The two approaches complement each other, and that is the key.

A mystique regarding the automation of reasoning still exists. For but one example, the literature strongly suggests that the proof of numerous deep theorems requires the use of double negation, which in fact is not the case, as shown with the use of OTTER. Is it certain that such success (with the dispensing of double negation) rests with the sharp increase in useful information when compared with total information that is considered? For a second example, who would have thought possible that automated reasoning would yield the answer to a deep question that had remained open since the mid-1930s and that had defied fine minds (that included Tarski)? Specifically, McCune's program EQP – in approximately 10 CPU-days – found a proof showing that every Robbins algebra is a Boolean algebra (McCune 1997).

Part of the mystique, as espoused throughout this chapter, rests with the intense and explicit use of various types of strategy. We strongly conjecture that the successes reported here, as well as numerous others not touched upon, would have been out of reach without the program's reliance on strategy. The formulation of some of the strategies resulted directly from an attempt to answer, through automation, an open question. Because we intend to continue to augment reasoning programs by formulating new strategies and new methodologies, we ask assistance in accruing

new open questions to study. An effective way to convey questions to us is by e-mail: wos@mcs.anl.gov.

Regarding source books and a program that might prove useful and intriguing, two books provide much of what is needed. The first book (Wos and Pieper 1999) serves well as a text, assumes no background, discusses various applications of automated reasoning, offers numerous open questions for consideration, and includes a CD-ROM on which one finds the program OTTER as well as various other useful files. In addition to logic and mathematics, the discussed practical applications include circuit design and validation; the important use of automated reasoning for program verification is not discussed. The second (two-volume) book (Wos and Pieper 2000) consists of reprints of published papers that enable one to follow the development of the field from the early 1960s to the late 1990s. The two books connect in a rather unusual manner: The first contains a long chapter whose subsections each correspond to one of the reprinted papers, giving an overview and appropriate problems. For further information on automated reasoning at Argonne National Laboratory, see http://www.mcs.anl.gov/AR/, which gives all of the needed pointers for new results, for various neat proofs, and for puzzles.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, US Department of Energy, under Contract W-31-109-Eng-38.

References

Epstein, R. (1994) *The Semantic Foundations of Logic: Propositional Logics*, 2nd edn. New York: Oxford University Press.

Łukasiewicz, J. (1948) The shortest axiom of the implicational propositional calculus. *Proceedings of the Royal Irish Academy*, 52A, 3, 25–33.

Łukasiewicz, J. (1970) Logistic and philosophy. In L. Borkowski (ed.), *Jan Łukasiewicz: Selected Works.* Amsterdam: North-Holland (original work of Łukasiewicz published in 1930).

McCune, W. (1994) *OTTER 3.0 Reference Manual and Guide.* Technical report ANL-94/6. Argonne, IL: Argonne National Laboratory.

McCune, W. (1997) Solution of the Robbins problem. *Journal of Automated Reasoning*, 19, 263–76.

Meredith, C. A. (1953) Single axioms for the systems (C,N), (C,O), and (A,N) of the two-valued propositional calculus. *Journal of Computing Systems*, 1, 155–64.

Meredith, C. (1959) The dependence of an axiom of Łukasiewicz. *Transactions of the American Mathematical Society*, 87, 54.

Rose, A. and Rosser, J. B. (1959) Fragments of many-valued statement calculi. *Transactions of the American Mathematical Society*, 87, 1–53.

Wos, L. and Pieper, G. W. (1999) *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning.* Singapore: World Scientific.

Wos, L. and Pieper, G. W. (2000) *Collected Works of Larry Wos.* Singapore: World Scientific.

Further Reading

We recommend the following, listed in order of importance:

Wos, L. (1996) *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial.* New York: Academic Press.

Wos, L. (1993) The kernel strategy and its use for the study of combinatory logic. *Journal of Automated Reasoning*, 10, 287–343.

Wos, L. (1995) Searching for circles of pure proofs. *Journal of Automated Reasoning*, 15, 279–315.

Boyer, R. S. and Moore, J. S. (1998) *A Computational Logic Handbook*, 2nd edn. New York: Academic Press.

McCune, M. and Padmanabhan, R. (1996) *Lecture Notes in Computer Science*, vol. 1095: *Automated Deduction in Equational Logic and Cubic Curves.* New York: Springer.