Appendix: Conjectures concerning Proof, Design, and Verification*

Larry Wos

Mathematics and Computer Science Division, Argonne National Laborator y,
Argonne, IL 60439-4801,
wos@mcs.anl.gov

1 Setting the Stage

This article focuses on an esoteric but practical use of automated reasoning that may indeed be new to many, especially those concerned primarily with verification of both hardware and software. Specifically, featured are a discussion and some methodology for taking an existing design—of a circuit, a chip, a program, or the like—and refining and improving it in various ways. (Although the methodology is general and does not require the use of a specific program, McCune's program OTTER does offer what is needed. OTTER has played and continues to play the key role in my research, and an interested person can gain access to this program in various ways, not the least of which is through the included CD-ROM in [3].) When success occurs, the result is a new design that may require fewer components, avoid the use of certain costly components, offer more reliability and ease of verification, and, perhaps most important, be more efficient in the contexts of speed and heat generation. Although I have minimal experience in circuit design, circuit validation, program synthesis, program verification, and similar concerns, (at the encouragement of colleagues based on successes to be cited) I present material that might indeed be of substantial interest to manufacturers and programmers.

I write this article in part prompted by the recent activities of chip designers that include Intel and AMD, activities heavily emphasizing the proving of theorems. As for my research that appears to me to be relevant, I have made an intense and most profitable study of finding proofs that are shorter [2, 3], some that avoid the use of various types of term, some that are far less complex than previously known, and the like. Those results suggest to me a strong possible connection between more appealing proofs (in mathematics and in logic) and enhanced and improved design of both hardware and software. Here I explore diverse conjectures that elucidate some of the possibly fruitful connections.

The strongest argument opposed to what I discuss in this article rests on the great amount of money, time, energy, and expertise that has been devoted to design and related activities. Indeed, one might understandably suspect that such

* 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.

experts already know how to produce superb and often minimal design. (As a counterargument, I note that the proofs found by OTTER, applying the methodology that has been developed as part of my research, often are startlingly unlike those a person might find. Perhaps more important, many of the proofs that are found are in various ways more appealing than the literature ofeers.) However, a test of what is featured here is inexpensive, and, if the result is positive, the reward might be immense. The test consists of some expert first supplying a set of graduated-in-complexity designs and the proofs that they meet specifications. Perhaps the designs supplied would already have been maximized for good properties. Second, if I am to be involved, part of the test requires supplying the proofs in the clause notation, the notation used by OTTER. Perhaps Mathematica could produce the needed translations. Then I would take the proofs (in clause notation) and attempt to shorten them or improve them in some other aspect discussed here. If I found better proofs, which I have in Boolean algebra (quite related to circuit design), I would submit them for evaluation.

If I were not involved, one might consult the book [3], which offers the program and much of what is needed to use it. Such has indeed been the case for areas of mathematics and of logic. Therefore, perhaps a new type of design would emerge. Your cost is that of producing an OTTER input file, and it appears that might require but a few days of a person's time who knew about design; my cost is research time devoted to an attempt to improve a given design. Sometimes such research time and effort lead to a set of solutions, each of which could be evaluated by an expert for its properties.

Also addressed in this article is the concern of design from scratch, that case in which no design exists to be modified, extended, and improved upon. I conjecture that my research and that of colleagues that has culminated in answers to diverse open questions will prove pertinent. After all, producing a design from scratch answers the corresponding open question concerning its existence. Indeed, quite different from the task of finding "nicer" and more desirable proofs is the task of answering open questions.

I shall review without technical details various approaches I and colleagues take for finding "better" proofs and answering open questions, and I claim that many of the approaches will prove useful to manufacturing, at least eventually. The explicit and implicit claims and conjectures should be viewed most critically, in view of my lack of expertise in design and synthesis. I will be content with merely sketching diverse ideas. I will also include observations that might seem too obvious to state, but are included to remove ambiguity.

To complete the stage setting, I give a foretaste of what is to come. Consider the following circuit-design problem (known as the two-inverter puzzle) [3], one that I myself would not have solved, but McCune's program OTTER did solve.

> Using as many AND and OR gates as you like, but using only two NOT gates, can you design a circuit according to the following specification? There are three inputs, i1, i2, and i3, and three outputs, o1, o2, and o3. The outputs are related to the inputs in the following simple way:

```
o1 = not(i1)   o2 = not(i2)   o3 = not(i3).
```

Remember, you can use only two NOT gates!

The fact that an automated reasoning program was able to design the desired circuit hints at what might be possible in the context of synthesizing circuits from scratch; see Section 3.

In the context of finding better circuits (see Section 2), imagine that a person or a program succeeds in solving the two-inverter puzzle, but the solution absurdly contains as a subexpression the OR of i1 and i1. In other words, assume that the cited subexpression is not needed, that an unneeded OR gate is present. The methodology presented in this article might quickly enable a program, given the unwanted solution, to find a better one, omitting the extra OR gate.

Still with the focus on the two-inverter puzzle, in the context of term avoidance (see Section 4), imagine that the first solution that is offered contains NOT(NOT(i3)). Of course, a canonicalization rule could be applied to replace the apparently unnecessary cited expression with i3. Far better and in the spirit of the corresponding methodology to be touched upon, the program could be instructed to avoid retention of all expressions containing NOT(NOT(t)) for any term t. Possibly not obvious, such avoidance can contribute markedly to program effectiveness; indeed, unwanted conclusions can lead to much wandering—for a program, or for a person.

In contrast to the discussion focusing on combinational circuits, clearly a focus on sequential design in which time and delay are factors presents distinctly different and difficult problems to solve. Although I can at this moment offer little advice in that regard, in that my research has never dealt with this aspect, I nevertheless conjecture that the preceding discussion will, for some, suggest what is more than conceivable and perhaps promising.

2 Shorter Proofs in Relation to Improved Design

Although by no means does a one-to-one correspondence exist, it seems patently clear that (in the following sense) a strong correlation does exist between proof length and simplicity of design. Consider two proofs A and B, source unspecified, each intended to construct the same object (such as a circuit). Assume (in this hypothetical case) that the length of A is moderately to sharply less than the length of B. Finally, assume that the (automated reasoning) program in use offers an ANSWER literal (to display the constructed object) and that the program finds both proofs.

Quite often, although certainly not always, the object displayed when A is completed is preferable to that displayed when B is completed in the sense that it relies on fewer components. Therefore, it seems quite reasonable to conjecture that a methodology for finding shorter proofs might indeed be of interest in the design of circuits or chips or the synthesis of programs. Moreover, a simpler (in the sense under discussion) object in general is easier to verify, less difficult to show that the specifications are met. My research has produced such a methodology, one that has been applied successfully again and again in mathematics and in logic (although quite often no shorter proof is yielded). (Section 6.7 of

[3] discusses the latest methodology I have formulated for systematically seeking shorter proofs.)

In the context of finding shorter proofs in my own research, one of the more satisfying concerned finding a 100-step proof, where I was presented for a start with an 816-step proof. The theorem in focus was one from Boolean algebra, a field relevant in various ways to circuit design. The approach I took did indeed, at the beginning, rely on the supplied 816-step proof. Further, at each stage in the process aimed at finding a shorter and then still shorter proof, the program keyed on the completed proof at an earlier stage.

The notion I suggest that might be of interest asserts that a program could be given a design (circuit, chip, program) whose corresponding proof that the specifications were met was in hand. The cited approach emphasizes the role of the steps of the proof in hand, preferring formulas or equations that are similar to one of the steps. Indeed, with a strategy known as the *resonance strategy*, the proofs that are found along the way—shorter and shorter, if all is going well—play a vital role. Another aspect of the approach concerns blocking the use of various steps of a given proof with the intention that, not only will such blocked steps be absent, but a shorter proof will emerge. (My preferred approach to blocking the use of a step rests on the use of demodulation, a procedure normally used for simplification and canonicalization.)

Also of interest and quite curious is the fact that, occasionally, a shorter proof has the property that all of its steps are among those of the somewhat longer proof being used by the resonance strategy. The explanation rests with the fact that the program finds new ways of connecting already-used items, ignoring others totally, and succeeding in completing a proof. For example, sometimes the program can use the fifth step with the twelfth step to deduce the twentieth step, which in the longer proof was obtained from the eighteenth and nineteenth, and discover that the eighteenth and nineteenth steps can be ignored. The correspondence for design would be the use of some, but not all, of the components of an existing design with (so to speak) a rewiring, without the introduction of new components.

3 New Proofs in Relation to Radically New Designs

In contrast to the preceding section in which the object is to take an existing design and improve upon it, here the focus is on finding the desired object from scratch. In such a case, often, no clue exists concerning the nature of the corresponding proof whose ANSWER literal, if successful, will display the object. Starting from scratch, no surprise, is far more difficult than beginning with an existing object and its corresponding proof. Nevertheless, I and my colleague Branden Fitelson are very encouraged by our various successes with finding a proof where no clue concerning its nature was available [1].

As for the word "radically" occurring in the title of this section, it was not used lightly. The proofs yielded by applying the various methodologies relying on OTTER's arsenal of weapons are (so it strongly appears) sharply unlike what a person might produce. For example, in fields of logic, the literature steadfastly offers numerous proofs relying heavily on the use of terms of the form $n(n(t))$ for various terms t, where the function n denotes negation (**not**). In contrast to the literature and the implicit view that such double-negation terms are virtually required, I have found (through heavy use of OTTER) numerous proofs avoiding such terms. More important, the methodology is general—not tuned to any specific type of term, such as that involving negation.

For a second example, where a researcher might understandably shy away from considering a messy and complex formula, equation, or expression, a reasoning program finds little discomfort in its consideration. Indeed, equations with more than 700 symbols present no problem for OTTER. Simply put and without explanation, the attack taken by a powerful automated reasoning program often resembles that taken by an unaided researcher in few if any ways. Rather than a disadvantage, (it seems to me) this divergence in attack accounts for many marked successes. I conjecture (with some trepidation) that, if the goal were a radically new design, an expert might be greatly rewarded by adding as an assistant a program such as OTTER

One key aspect of the methodology OTTER applies when seeking a proof where none is in hand is reliance on the already-cited resonance strategy, but reliance in a slightly different manner. Specifically, what amounts to patterns corresponding to steps that proved useful in related proofs are included in the input. Often very few of those correspondents (resonators) are present in the proof that results when successful, and often not many more of its steps match one of the resonators. Naturally, the question then arises concerning how such inclusions help. With a new proof, I suspect that those few of its steps that are either one of the actual patterns or match a resonator (in a manner where variables are treated as indistinguishable) provide the keys to getting around narrow corners, over wide plateaus, and the like (speaking metaphorically). In other words, without the guidance offered by the included resonators, success would not occur. The idea is similar to the case in which a colleague provides a few vital hints, even if that colleague cannot solve the actual problem.

4 Term-Avoidance Proofs in Relation to Design

The avoidance of terms, such as those in the double-negation class, is somewhat reminiscent of avoiding the use of some component. Sometimes the desire is for minimal but nonzero use of some type of component—as was the case in the two-inverter puzzle—but, often, the intent is to never have present some type of term or component. For example, OR gates might come into play in some fashion during the exploration by person or by program, and yet their actual use might be unwanted. As commented earlier, a program such as OTTER can be instructed to completely avoid retaining any unwanted conclusion, thus reflecting the intent of the user.

5 Complexity of Proofs

In this section, in contrast to the preceding in which I was able to give hints about a concrete relation between properties of proofs and improvements in design, I simply discuss another aspect of my research concerned with proof betterment. In other words, I (at the moment) leave to the expert in design, verification, and synthesis the extrapolation to other areas.

One of the sometimes annoying properties of all proofs in hand is unwanted complexity of various types. The most obvious type concerns the length of the formulas or equations of the deduced proof steps. Simply put, the proofs in hand may each be far messier than preferred. Such messiness is not merely an aesthetic consideration; indeed, its presence can make the proof harder to follow and may suggest that key lemmas (that would reduce the complexity) have as yet not been discovered.

OTTER offers what is needed in the context of deduced-step length, namely, a parameter called max_weight. The user can assign a value to this parameter and can instruct the program to measure deduced-step complexity purely in terms of symbol count. When a new conclusion is drawn whose complexity exceeds the user-assigned value to max_weight, the conclusion is immediately discarded.

Further, by assigning a small value to max_weight and by including as resonators expressions corresponding to the steps of an existing proof with even smaller assigned values, the user can attempt to force the program to find a subproof with an intriguing property (discussed earlier). Specifically, to complete a proof, the program (in the case under discussion) sometimes finds a proof that is shorter than the one in focus such that all of the deduced steps of the shorter proof are among those of the proof whose steps are being used to guide the program's attack. In effect, if successful, the original proof has been (so to speak) rewired in a manner that reduces the number of components needed to achieve the objective.

Of a quite different nature in the context of proof complexity is that concerned with the maximum number of distinct variables found in the deduced steps. In particular, for each formula or equation from among the deduced steps, a number (integer) corresponding to it can be trivially computed that matches the corresponding number of distinct variables present. The formula P(i(x,x)), for example, has the number 1 associated with it, one distinct variable even though two variables (not distinct) are present. The maximum of the assigned numbers to the deduced steps (excluding those that correspond to the input or hypotheses) is the number of maximum distinct variables for the proof. Is that number (as a measure of complexity) in some important manner related to component use or instruction use?

Fortunately, OTTER offers the appropriate parameter, max_distinct_vars. The user can assign a value to this parameter. When a conclusion is deduced, before it is retained, the number of distinct variables in it is compared with the max_distinct_vars and, if it is strictly greater, the new item is immediately purged.

The use of this parameter can have some unexpected consequences for proof betterment and, perhaps, for design enhancement. Indeed, if i is the minimum of the various values of the maximum number of distinct variables for the known proofs, and if j is assigned to max_distinct_vars with j strictly less than i, then the program is forced to pursue a line of study that cannot produce, if successful, any of the known proofs. In other words, reminiscent of Section 3, the program might complete a radically new proof, find a radically new design.

Just as a note, other measures of complexity can be nicely and effectively studied with OTTER. For but one example, a measure of complexity concerns the level of a proof. By definition, the level of the input items that characterize the problem is 0, and the level of a deduced item is 1 greater than the maximum of the levels of the hypotheses from which it is deduced. This parameter is pertinent to tree depth, the tree of the proof.

6 Verification

In this article, I have begun to make a case for the use of automated reasoning in the context of design and verification. Mainly, I have focused on design (implicitly, of circuits, chips, and programs). However, all things being equal, the simpler the design, the greater the ease of verification. Therefore, what has been discussed has some relevance to verification. Explicit is the position that the properties of a proof that constructs some object are reflected in the nature of the object. For example, if the proof is strictly shorter than that in hand, then (quite often) the corresponding object rests on the use of fewer components (whatever they may be). For a second example, if the proof avoids the use of some type of term (such as double negation), then the constructed object avoids the use of some type of component.

As for additional topics that appear to merit mention, perhaps the following are among them. OTTER can be and has been used to show that one of a set of axioms is dependent on the remainder. For design, the parallel might be that of showing that some thought-to-be key property that must be studied, in addition to the rest, in fact is dependent on the rest. If the remaining properties are shown to hold, then the cited key property must, without verifying its presence. Fitelson and I have also succeeded in proving that a weakening of some well-recognized axiom system does the trick, suffices to axiomatize the area of discourse. The analogue might be that of showing that some key property can be replaced by a far weaker property, one that is easier to satisfy and easier to verify.

7 Review and Summary

The approach taken in this article is to merely sketch various notions, to provide hints or clues as to what I conjecture to be more than feasible. Although I claim no expertise in design, synthesis, and verification, my research has yielded some startling results in mathematics and in logic. Some of those results concern the answering of open questions, whose analogue might be the design of a radically new nature. Some of the results focus on proof betterment: shorter, less complex, term-avoidance, and the like. The analogues of those have been discussed, although not in the greatest depth.

The beauty of relying on a program such as McCune's OTTER is that its proofs are most detailed. Another charming and useful aspect of its proofs is that they very often differ sharply from the type of proof an unaided researcher finds. This program offers a veritable arsenal of weapons from which to choose when attacking a question or problem, as well as diverse mechanisms pertinent to powerful reasoning. The program runs with incredible speed and, in contrast to living creatures, tirelessly.

As discussed here, through the use of the resonance strategy, the presence of an actual design can be put to great use when the goal is to refine and improve it in some manner. However, if the various successes in answering open questions points in the right direction, the lack of a design does not prevent the finding of a desired object; indeed, one can start from scratch, as I and my colleague Branden Fitelson have done in areas of logic. Of course, starting from scratch presents a more difficult problem to solve, especially when no clues are offered of any type regarding the nature of a possible proof.

The material sketched here might be timely, in view of the current interest in theorem proving by members of industry that include Intel and AMD. I suspect that (perhaps) many of the items discussed here offer a new notion, even to those familiar with automated reasoning. I cannot measure at this time the practicality. Certainly, one obstacle is sequential design in contrast to combinational, that concerned with time and with delay, for example. Nevertheless, I await (with pleasure and anticipation) your examination of and comment on the ideas presented here. I conjecture that a program such as OTTER will provide a most valuable automated reasoning assistant for design and synthesis—it clearly has for us in mathematics and in logic.

References

1. Fitelson, B., and Wos, L.: Missing Proofs Found, preprint ANL/MCS-P816-0500, Argonne National Laboratory, Argonne, Illinois (2000)
2. Wos, L.: The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial. Academic Press, New York (1996)
3. Wos, L., and Pieper, G. W.: A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning. World Scientific Publishing, Singapore (1999)