On the desirability of mechanizing calculational proofs

Panagiotis Manolios [1], J Strother Moore [2]

Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712-1188, USA

Abstract

Dijkstra argues that calculational proofs are preferable to traditional pictorial and/or verbal proofs. First, due to the calculational proof format, incorrect proofs are less likely. Second, syntactic considerations (letting the "symbols do the work") have led to an impressive array of techniques for elegant proof construction. However, calculational proofs are *not* formal and are *not* flawless. Why not make them formal and check them mechanically?

Key words: Mechanical reasoning, Automatic theorem proving, Formal methods, Calculational proofs

0 Introduction

At the symposium in honor of Professor Edsger Wybe Dijkstra (at the University of Texas at Austin, May 2000), Natarajan Shankar expressed disappointment that Dijkstra has not seriously considered mechanical theorem proving. It seems that the calculational proof [3] community at large has not seriously considered mechanical theorem proving either. This is surprising, especially since the close relationship between calculational and mechanical proof is widely recognized. For example, in EWD1268 [12] [4], Dijkstra writes "Calculational arguments are praised for their rigor, for their amenability to mechanical verification ... ".

[1] Corresponding author. E-mail: pete@cs.utexas.edu.
[2] E-mail: moore@cs.utexas.edu.
[3] See the introduction to this special issue.
[4] The EWDs are available on the Web [13].

Furthermore, the construction of calculational proofs is often portrayed by their authors as being almost algorithmic. For example, on page 5 of EWD932a [5] Dijkstra writes:

> Independently, we and Boyer/Moore have found cases in which a syntactic analysis of the formal statement of the demonstrandum gave strong heuristic guidance, occasionally to the point of almost dictating how the proof had to be conducted. The time seems ripe to estimate whether our joint experiences warrant a more conscious exploration in that direction.

The obvious way to test the hypothesis that calculational proofs are amenable to mechanical verification and that proofs often follow from syntactic analysis is to produce a program with these capabilities and to present it to the academic community for judgment. Therefore, we are throwing down a gauntlet to the calculational proof community: If your proofs are so rigorous and so amenable to mechanization, stop just saying so and do it. Build a program that will check calculational proofs at the level they are written and assert your belief that any formula that can be proved by your program is valid.

In the next section, we answer a question of Dijkstra's by pointing out flaws in some of the EWDs, thus supporting the case for mechanized checking. In the section after that we discuss some of the issues that the community should address in order to move toward mechanized checking.

1 Some Erroneous Proofs in EWDs

On page 0 of EWD1227 [11] Dijkstra writes:

> If they [the logicians] can help and correct me by pointing out where my precautions have been insufficient and where my own discipline has led or will lead me astray, they are most welcome: I love to be corrected. (Besides being a most instructive experience, being corrected shows that the other one cares about you.)

Consider page 21 of EWD1130 [7], on which Dijkstra discusses "lifting". Here X, Y, and Z are understood to be predicates. He writes:

> rewrite
>
> $$\langle \forall s : s \in S : X.s \quad \equiv \quad Y.s \wedge Z.s \rangle$$
>
> as
>
> $$[X \quad \equiv \quad Y \wedge S]$$

Note that the last "S" above should be a "Z". This is almost certainly a calligraphic error. But if the symbols are to do the work, the symbols must be right. Furthermore, the corrected transformation is valid only under some unstated assumptions, *e.g.*, that S is the domain of X, Y, and Z. Making sure that the symbols are right and that assumptions are made explicit when designing and verifying a complicated system can be daunting without mechanical assistance.

Next, we turn our attention to page 5 of EWD1185 [9].

> Consider now the following transformations
> α: $x \leftrightarrow y$, *i.e.* an interchange of x and y .
> ...
> Note that the transformations $\alpha, \beta, \gamma, \delta$ commute and that each is its own inverse.
> Because (8) $\Rightarrow$ (11) and the combination (α, β, γ) leaves (8) unchanged, (8) implies (11) transformed by (α, β, γ) ...

Now, at first sight, this seems like an interesting technique. In Dijkstra's words (page 7 of EWD1185):

> When—what perhaps we should not do—we consider transformations α, β, γ, δ "semantically meaningless", it could be argued that we have pushed the notion of calculation to a next level of abstraction.

However, consider the following similarly worded argument:

> Consider now the following transformation
> θ : $\underline{\text{false}} \leftrightarrow \underline{\text{true}}$, *i.e.* an interchange of $\underline{\text{true}}$ and $\underline{\text{false}}$.
>
> Note that the transformation θ commutes and is its own inverse.
> Because $p \Rightarrow \underline{\text{true}}$ and the combination (θ) leaves p unchanged, p implies $\underline{\text{true}}$ transformed by (θ).

Unfortunately, we have just "proved" $\neg p$.

Notice that the argument given by Dijkstra is calculational. The proof of (8) $\Rightarrow$ (11) on page 5 is based on the proof format that characterizes calculational proofs, but there is often a natural language component to calculational proofs, *e.g.*, the proof contains "we observe for any x, y" (implicit is that x, y are of the appropriate type). Contrary to page 9 of EWD1268 [12]:

> in mathematics ... the woolly quality criterion of "convincing enough of your fellow mathematicians" has been replaced by the requirement that the result be derived, according to the rules of an explicitly stated logic, from explicitly stated postulates. Logic and postulates act as the firewall, shield-

> ing the proof-designing mathematician from the psychological preferences of his colleagues.

the calculational proofs we have found in the EWDs are not "derived, according to the rules of an explicitly stated logic, from explicitly stated postulates." Instead, we characterize them as rigorous arguments in a strict format (helping to avoid errors) which are often very revealing by exposing the syntactic nature of the argument, but where the notion of proof is "convincing enough for your fellow mathematicians". The question arises, suppose we tried to mimic this proof in a truly formal system, what would happen? The observation that syntactic transformations of this kind are often useful has led to what are called "meta rules" in ACL2 (see [1]). However, meta rules have to be proved correct. Presumably, one cannot prove the above meta rule correct.

For the third example, we quote from page 0 of EWD1219 [10]:

> Consider the statements a0 and a1:
> (a0) I have a proof that <u>true</u> (holds)
> (a1) <u>true</u> (holds) .
> Then, a0 and a1 are equivalent. (Since—by definition—the proof that <u>true</u> (holds) is empty, it is impossible not to have it.)

We interpret the above as follows. Both a0 and a1 are statements of a metatheory used to analyze a formal system (we are using the terminology found in Kleene [20]). The language of the formal system is well-defined and includes the statement <u>true</u>. The statement that <u>true</u> holds (a1) is a statement in the metatheory that means <u>true</u> is valid (true in all interpretations). We suppose that there is a proof of <u>true</u> in the formal system that is empty. Whether or not "I" has it seems like a philosophical question. Therefore, we interpret a0 as saying that <u>true</u> is provable in the formal system.

(Aside) It is possible to interpret a0 and a1 as being statements in the formal system. This kind of reflection is used in proving Gödel's incompleteness theorems. For our purposes, it is irrelevant which interpretation is used. **(End Aside)**

Dijkstra continues as follows:

> Consider the statements b0 and b1:
> (b0) I have a proof that <u>false</u> (holds)
> (b1) <u>false</u> (holds) .
> Then, b0 and b1 are equivalent. (Since—by definition—the proof that <u>false</u> (holds) does not exist, it is impossible to have it.)

The statement "the proof that <u>false</u> (holds) does not exist" is equivalent to the statement that the formal system is consistent (for a reasonable formal system). One of the fundamental results in logic, Gödel's second incompleteness theorem, shows that if a formal system (powerful enough to represent the natural numbers) is consistent, it is not possible to prove its consistency formally, let alone to posit its consistency "by definition". Therefore, we have to reject the "proof" that b0 and b1 are equivalent. In EWD1268 [12] Dijkstra characterizes "formalists" as those who "do mathematics by manipulating uninterpreted formulae accordingly to explicitly states rules" and "informalists" (mathematicians who are not formalists) as those who "constantly interpret formulae and 'reason' in terms of the model underlying that interpretation." But it is this type of informality, interpretation, and reasoning about the underlying model that leads to the claim that false is not provable, by definition.

A proof of Gödel's second incompleteness theorem can be found in textbooks on mathematical logic (*e.g.*, [20], [25], [15], and [16]). A study of formal provability via modal logic is conducted by Boolos [0], where on page xvii, Boolos makes the point that although for *every* theorem S of a sufficiently strong formal system F, there is also a theorem of F to the effect that S is a theorem of F, for *no* non-theorem S' of F is there a theorem of F to the effect that S' is a non-theorem of F (if F is consistent).

There are more problems with this EWD, *e.g.*, it is claimed that c0 (I have a proof that I have a proof that A (holds)) is equivalent to c1 (I have a proof that A (holds)), but this is not the case (see [26]). The "proof" of their equivalence (hinted at, but not given) uses: if ϕ and ξ have the same truth value, then ϕ is provable iff ξ is. But truth and provability are different, *e.g.*, replace ϕ by true and ξ by CON_F (the sentence asserting the consistency of F, the sufficiently strong formal theory used).

One of the advantages of working in a mechanized logic is that even if tempted to think about the above problem, once you try to render it formally, you realize there is more to it than was apparent at the outset. We are not implying that deep mathematical results cannot be proved formally: Shankar [24] presents a mechanical proof (using the Boyer-Moore theorem prover) of Gödel's first incompleteness theorem.

We suspect there are more proof errors in EWDs. It would be surprising if there were not, in such a large and remarkable body of mathematical work. But given the level of rigor and attention to detail in the calculational proof community, it seems entirely appropriate to try to eliminate as many errors as we can.

2 Mechanical Theorem Proving

According to page 13 of EWD1073 [6], "calculational proofs ... tend to be shorter, more explicit, and so complete that they can be read and verified without pencil and paper". This is suggestive, but too ambiguous to be of technical use. More promising is that calculational proofs are described by Dijkstra as "formal calculations" (*e.g.*, on page 7 of [12]). However, even a cursory look at the EWDs reveals that Dijkstra's modus operandi is for the most part informal. In a formal system the notion of "proof" must be recursive (cf. page 132 of [25]). (By "recursive" we mean, as in the theory of computability, that there is a Turing machine that can decide whether or not its input is a proof.) This requirement embodies the "finitistic" requirements of Hilbert and allows us to "calculate ... to see who is right", a partial fulfillment of Leibniz's dream. We have not found any EWD that contains an algorithm that can be used to check the correctness of the proofs appearing in the EWDs. That calculational proofs as described by Dijkstra are not formal is—we expect—well known, *e.g.*, Lifschitz [21], referring to [14], points out that "what Dijkstra and Scholten call formal proofs are not really formal proofs in the usual sense; they are informal proofs written in a relatively strict format."

Formal systems have been studied extensively by logicians who have traditionally been interested in the power and limits of formal systems and have therefore (and rightly so) considered the simplest such systems. As tools for the working computing scientist, such systems are woefully inadequate. We do not expect this will surprise the logicians: on page 47 of [3], Church writes "our interest in formalized languages being less often in their actual and practical use as languages than in the general theory of such use and in its possibilities in principle." We believe that the work in calculational proofs can lead to formal systems that suit the needs of the working computing scientist, but extreme care must be taken to insure that we do not embrace pseudo-formal methods that seem convenient but are not well-defined and/or inconsistent.

What about elegance? Could it be that any attempt to render calculational proofs suitable for mechanical checking will result in ugly, hard to comprehend proofs? Will the mechanized proof checker necessarily be hard to learn and difficult to use?

In Figure 0 we present one of many lemmas taken from the article by Russinoff and Flatau in [18]. In the article, the authors describe a mechanically checked proof of the correctness of the RTL (register transfer language) description of a floating-point multiplier.

As with our extracts from EWDs, we have omitted many definitions and no-

Lemma 13.3.7 (`expo-xy`)

$$expo(\hat{x}\hat{y}) = expo(\hat{x}) + expo(\hat{y}) + overflow$$

Proof: By Lemmas 13.1.15 and 13.3.6,

$$\begin{aligned}\hat{x}\hat{y} &= sgn(\hat{x})sig(\hat{x})2^{expo(\hat{x})}sgn(\hat{y})sig(\hat{y})2^{expo(\hat{y})}\\ &= sgn(\hat{x}\hat{y})\left[sig(\hat{x})sig(\hat{y})/2^{overflow}\right]2^{expo(\hat{x})+expo(\hat{y})+overflow}\\ &= sgn(\hat{x}\hat{y})sig(prod)2^{expo(\hat{x})+expo(\hat{y})+overflow}.\end{aligned}$$

The result now follows from Lemma 13.1.18. □

Fig. 0. A Mechanically Checked Lemma

tational conventions necessary to understand precisely what is being said; but the basic style and flow of the argument should be understandable. In Figure 0 x and y are understood to be floating point numbers in a certain format and $\hat{x}$ and $\hat{y}$ are the corresponding rationals. As the reader may well guess, *sgn*, *sig*, and *expo* are functions that produce the sign, significand, and exponent, respectively, of a rational. Lemma 13.1.15 states that every rational can be represented in a certain normal form in terms of *sgn*, *sig*, and *expo*. Lemma 13.3.6 is $sig(\hat{x})sig(\hat{y})/2^{overflow} = sig(prod)$, where *prod* is the product of the significand fields of floating point numbers x and y. Lemma 13.1.18 states that, if $|x| = 2^n y$, then $expo(x) = n$, under suitable conditions on n and y.

The proof in Figure 0 was formalized in the ACL2 logic [19] and proved with the ACL2 theorem prover. The formalization of the proof requires about as many lines as shown in the Figure. The relevant lemmas (named, rather than numbered) are cited explicitly. The formula is presented in the Lisp syntax of ACL2

```
(equal (expo (* (hat (x*)) (hat (y*))))
       (+ (expo (hat (x*))) (expo (hat (y*))) (overflow*)))
```

and the proof is formalized similarly. The authors of this proof use a formal device to introduce "implicit" hypotheses on their "variables": they use constrained constants, such as `(x*)`, instead of variables. This allows them to introduce names for commonly occurring subexpressions and use the names as expressions, as is done with *prod* above, without carrying around the hypotheses syntactically. One might question the clarity of the authors' style, but one should note that the formalism is used in the service of their chosen style, not that the style is in the service of the adopted formalism.

No doubt many readers find the syntax burdensome and could think of a more pleasant one. But the syntax is not particularly important here. Our point is that even with "the relevant" lemmas noted explicitly, completing the proof requires a non-trivial amount of work. For example, the proof implicitly uses the commutativity and associativity of multiplication and various properties of exponentiation and absolute value. What is shown in Figure 0 is not a formal proof but a sketch of one. Filling in the gaps to "check" such a proof, even one described so carefully, can involve many more decisions than many people at first realize. For example, a computer program can find an infinite sequence of things to do with commutativity alone: repeatedly swap the arguments to multiplication. Of course, this soon becomes boringly repetitive and pointless, to a person. The traps are less obvious when the usual arithmetic identities are added to the mix. Sometimes it really is productive to make explicit a factor of 1, to multiply and divide by the same quantity (as here) or to otherwise violate an informally understood "normal form". Many such traps are encountered in the attempt to check simple proofs and the author of a proof checker or theorem proving methodology must make explicit the algorithms or heuristics for avoiding them, otherwise, it is difficult to believe that "many steps that may seem surprising at first sight, are, in fact, (almost) dictated" (page 0 of [8]).

Far from deploring the lack of detail in Figure 0, we applaud it, at least in the setting of a mechanically checked proof. If steps can be filled in by a machine, why put in more than are necessary for the human reader to follow? The calculational proof community well understands the value of presenting a proof at the appropriate level of detail. Mechanical theorem provers, if properly designed and used, can allow one to describe and check proofs at whatever level is deemed appropriate for the argument. This makes it easier to use the theorem prover as an assistant to check one's arguments.

The ACL2 theorem prover is in use today to check proofs about a variety of (mathematical models of) computational artifacts. For example, researchers at Advanced Micro Devices, Inc., which produces microprocessors, used ACL2 to analyze parts of the design of the floating point unit in the latest AMD processor, the Athlon. The RTL implementations of the elementary floating-point operations were proved compliant with the IEEE floating point standard. But that is only one of many applications of ACL2. It has also been used to prove compilers correct, to simulate machine architectures, prove properties of microcode and of other software, and to prove interesting mathematical theorems. See [18] for many case studies.

3 How to Proceed

We do not mean to imply that the calculational proof community would find ACL2 suitable as a proof checking engine. But our familiarity with ACL2 leads us to speculate that it is possible to design a proof checker for the calculational proof community that:

- is sound,
- checks calculational proofs at the level of detail at which they are usually written,
- supports user-defined notations (*i.e.*, an extensible syntax),
- permits the derivation of new rules of inference or metatheorems,
- provides proof structuring tools (*e.g.*, various "scoping" mechanism to control the proliferation of names, definitions, lemmas, proofs, etc.).

As the paragraph above suggests, the design of a useful mechanized proof system is comparable to that of a programming language. It is easy to design something that is theoretically interesting and practically useless.

Behind every good theorem prover is a clear understanding of the logic in which theorems are being proved. Technically speaking, a proof is a non-empty, finite sequence of formulae, each of which is an axiom or follows from the previous formulae by a rule of inference. But a theorem prover or proof checker need not actually construct a formal proof to confirm that one exists. For example, the ACL2 system uses several decision procedures to determine whether a formula is a theorem without actually constructing the derivation. To code a theorem prover correctly, one must ultimately have clearly in mind the formal proof system.

The calculational proof community has spent decades exploring the basic issues, refining proof methods, settling on format, and so on. Our advice to the community then is simple: implement a program to check your proofs.

Four specific recommendations come immediately to mind. First, accept that your first checker will not be satisfactory. But do not fear experimentation in pursuit of experience.

Second, learn to use one or two of the tools already built. These include ACL2 [19], HOL [17], OTTER [22], Mizar (`http://mizar.org`), and PVS [4]. Tools with some support for calculational proofs have already been developed. For example, there is a tool [2] for the refinement calculus based on the theorem prover Ergo [27] that uses a window inference mechanism (see [23] for a discussion and references). A tool that supports calculational proofs of stepwise refinement is described in [28]. None of the above tools will be entirely satisfactory. But, each will be eye opening.

Third, implement a tool, but start with the most straightforward system imaginable: given a chain of purported equalities and justifications, check that the derivation is correct. Use the program to check your proofs and proofs in the literature. Despite its primitive nature, it will probably find omissions in the proofs.

Fourth, submit your tool to the scrutiny of the calculational proof community. The community will evaluate whether they can use the tool to check the proofs that they write (as opposed to being forced to write proofs that the tool can check) and whether your tool satisfies the requirements set out at the beginning of this section.

4 Conclusions

Calculational proofs are an effective way for computing scientists to reason and communicate. But they are not formal. Why not formalize and mechanize them? Doing so can help us avoid missteps. In addition, the notational and proof development insights that have resulted from the calculational experiment can lead to a new class of proof checkers and theorem provers that can make proofs more readable and understandable.

On page 1 of EWD932a [5], Dijkstra writes:

> ... when we learned that formal calculi of all sorts would have an important rôle to play, we postponed the problem of axiomatic foundation of such calculi: the intellectual investment involved would be premature without the "moral certainty" that one is founding the calculi one needs.

We propose that enough has been done to establish "moral certainty". To the objection that a formal logic is too restrictive, we quote Dijkstra (from page 11 of EWD1268 [12]) who quotes E.T.Bell.

> Mathematicians and scientists of the conservative persuasion may feel that a science constrained by an explicitly formulated set of assumptions has lost some of its freedom and is almost dead. Experience shows that the only loss is denial of the privilege of making avoidable mistakes in reasoning. As is perhaps but humanly natural, each new encroachment of the postulational method is vigorously resisted by some as an invasion of hallowed tradition. Objection to the method is neither more nor less than objection to mathematics.

To which Dijkstra adds "Amen." Amen indeed.

Acknowledgments

We thank the reviewers and John Gunnels for their suggestions.

References

[0] George Boolos. *The Logic of Provability.* Cambridge University Press, 1993.

[1] Robert Stephen Boyer and J Strother Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. In Robert Stephen Boyer and J Strother Moore, editors, *The Correctness Problem in Computer Science*, pages 103–184. Academic Press, 1981.

[2] D. Carrington, I. Hayes, R. Nickson, G. Watson, and J. Welsh. A tool for developing correct programs by refinement. Technical Report 95-49, Software Verification Centre, Department of Computer Science, The University of Queensland, 1995.

[3] Alonzo Church. *Introduction to Mathematical Logic*, volume I. Princeton University Press, 1956.

[4] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A tutorial introduction to PVS. In *Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques.* Boca Raton, FL, April 1995.

[5] Edsger Wybe Dijkstra. The streamlining of the mathematical argument, August 1985. EWD 932a.

[6] Edsger Wybe Dijkstra. How Computing Science created a new mathematical style, March 1990. EWD 1073.

[7] Edsger Wybe Dijkstra. On the economy of doing mathematics, June 1992. EWD 1130.

[8] Edsger Wybe Dijkstra. On the design of calculational proofs, April 1993. EWD 1150.

[9] Edsger Wybe Dijkstra. Lifting, orders, and the Galois connection, Sep 1994. EWD 1185.

[10] Edsger Wybe Dijkstra. "I have a proof that . . . .", October 1995. EWD 1219.

[11] Edsger Wybe Dijkstra. A somewhat open letter to David Gries, January 1996. EWD 1227.

[12] Edsger Wybe Dijkstra. The mathematical divide, December 1997. EWD 1268.

[13] Edsger Wybe Dijkstra. The Writings of Edsger W. Dijkstra, 2000. See URL `http://www.cs.utexas.edu/users/EWD`.

[14] Edsger Wybe Dijkstra and Carel Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, 1990.

[15] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Springer-Verlag, second edition, 1994.

[16] Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[17] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[18] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore, editors.

Computer-Aided Reasoning: ACL2 Case Studies. Kluwer Academic Press, June 2000.

[19] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Press, July 2000.

[20] Stephen Cole Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[21] Vladimir Lifschitz. On calculational proofs, 1998. Unpublished draft.

[22] W. McCune. Otter 3.0 reference manual and guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994. See URL `http://www.mcs.anl.gov/AR/otter/`.

[23] Ray Nickson and Ian Hayes. Program window inference. Technical Report 95-29, Software Verification Centre, Department of Computer Science, The University of Queensland, 1995.

[24] Natarajan Shankar. *Metamathematics, Machines, and Gödel's Proof*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1994.

[25] Joseph Robert Shoenfield. *Mathematical Logic*. Addison Wesley, Reading, Ma., 1967.

[26] C. Smorynski. The incompleteness theorems. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, 1977.

[27] Mark Utting and Keith Whitwell. Ergo user manual. Technical Report 93-19, Software Verification Centre, Department of Computer Science, The University of Queensland, 1993.

[28] Jan L.A. van de Snepscheut. Mechanical support for stepwise refinement. Technical Report CS-TR-94-0, California Institure of Technology, 1994.