The Use and Abuse of Formal Proofs

John R. Thompson

J.L. Kellogg Graduate School of Management
Northwestern University
1725 Orrington Avenue, #722
Evanston, Illinois 60201

K. Culik, in "On formal and informal proofs for program correctness,"[1] notices the apparent resistance of "good programmers and good computer science students" (whatever "good" may mean) to correctness proofs. Culik singles out as a particular cause of this resistance:

> ...a perception by programmers that each program correctness proving [sic] must be formal with a lot of special notation making it incomprehensible.

Culik's proposed solution is

> ...Frequently to present informal program correctness proofs of programs [sic] ... proving program correctness is as easy or as difficult as is actual informal, mathematical proving theorems [sic].

The conclusion is that

> It seems to be much more inportant [sic] to train programmers in informal mathematical proving than in formal logical proving.

I

One dictionary defines "proof" as "evidence sufficient to establish a thing as true or to produce belief in its truth."[2] This definition certainly seems reasonable. Although a well-known computer scientist at my undergraduate institution was once heard to remark that "there are only three kinds of proof: induction, exhaustion, and intimidation," we must certainly include proof by counter-example, proof by picture (e.g. Venn Diagrams), and proof by deduction.

---

1 SIGPLAN Notices, January 1983.
2 American College Dictionary.

The purpose of Culik's article seems to be the separation of "informal" or "mathematical" methods of proof from "formal" or "logical" methods, largely on grounds of convenience. As if to deny the proposition of separate but equal methodologies, Culik ends with a scathing denounciation of proof by logic:

> ... a formal proof in logic was not introduced either to discover theorems or to prove them. A brilliant logician can be a totally sterile mathematician, because he is used to using only some rules on an abstract formal level, while discoveries and inventions are growing out of concrete models and dirty examples.

Indeed, the point Culik tries to make is that

> ...without going into logical formal concepts we can prove program correctness using usual (informal) mathematical proofs.

II

The first point to be made is that, as many a mother has told her child, there is a time and a place for everything. To the best of my (admittedly less than perfect) knowledge, the proponents of formal proofs do not advocate their exclusive use.

For example, E. W. Dijkstra has written:

> I have dealt with the examples in different degrees of formality. This variation was intended, as I would not like to give my readers the impression that a certain, fixed degree of formality is the "right one." I prefer to view formal methods as tools, the use of which might be helpful.3

David Gries adds:

> Our approach to programming is based on proofs of correctness, but be assured that complete attention to formalism is neither necessary nor desirable. Formality alone is inadequate, because it leads to incomprehensible detail; common sense and intuition alone ... are inadequate, because they allow too many errors and bad designs. What is

---

3Edsger W. Dijkstra, A Discipline of Programming, Prentice Hall, Englewood Cliffs, NJ, 1976. Page 215.

needed is a fine balance between the two.4

C.A.R. Hoare anticipated Culik's observations:

> ...program proving, certainly at present, will be difficult even for programmers of high caliber; and may be applicable only to quite simple program designs. As in other areas, reliability can be purchased only at the price of simplicity.5

In fact, Gries cites the following principle:

> Use theory to provide insight; use common sense and intuition where it is suitable, but fall back on the formal theory for support when difficulties and complexities arise.6

Culik's observations are therefore trivial. Of course it is possible to prove programs correct without using formal logic. It is also possible for a shepherd to count his flock without appeal to Peano's Axioms, but that does not imply that they are undesirable.

Culik presents a problem whose solution using formal methods appears quite tedious. The purpose is to show the superiority of informal methods. It should be noted, however, that the above quotations show that advocates of formal proofs have not adopted the dogma "Formal Proofs Are Always Superior." While Culik's counter-example may damage this orphan dogma, one is compelled to note that proof by example only establishes the falsity of a proposition, not its veracity. The example fragment7 proves nothing, except perhaps that poorly commented code (we are told that this was intentional) is often unreadable, a fact that all but diehard COBOL programmers now admit.

In addition, the claim is made that this fragment is correct because it is a faithful translation of an inductive proof, yet without the proof this translation cannot be verified (for whatever proof we might develop, we have not a clue as to whether Culik used the same one). Certainly a poorly commented fragment of PL/1 does not inspire belief in its own truth. Instead, one is reminded of

---

4David Gries, The Science of Programming, Springer-Verlag, New York, 1981. Page 164.

5C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," Communications of the ACM, October, 1969.

6Gries, op. cit., page 165.

7It must be a fragment since the invoked procedures CIRCUT and CONCOMP are nowhere to be found.

8A language Dijkstra called a "fatal disease" in his Turing Award Lecture.

> Principle: A program and its proof should be developed hand-in-hand, with the proof leading the way.[9]

In short, Culik claims that formal proofs are not a universal wrench. This is not surprising since even three of its advocates refuse to claim so. Indeed, it is amazing that anyone would consider one approach suitable for all problems, no matter what their class.

III

Culik ends by asserting two conclusions, the first of which is:

> If there is any actual and general programming methodology at all [sic] then it consists in the conversion of inductive and constructive proofs of existential theorems into corresponding recursive procedures.

The existance of a universal wrench has been treated in the previous section. If Culik does wish to prove the universally superior applicability of proof by existential theorem, then we may expect an existance theorem for the existential theorem method.

Culik's second conclusion is:

> Obviously basic concepts and notation of predicate calculus..., the concept of axiomatic theory, its interpretations and models, should be familiar to each computer science student because without them the concepts of semantics and translation cannot be clarified.

This assertion appears in the last line of the paper. In the context of the final paragraph it seems to confine the application of formal logic only to semantics and translation. This is as absurd as Culik's fear that teaching logic will lead to a sudden increase in the number of sterile mathematicians. Formal logic is a tool and, like any tool, has uses and abuses limited only by the human imagination.

One does well to recall Dijkstra's words:

> After having devoted a considerable number of years of my scientific life to clarifying the programmer's task, with the aim of making it

---

[9]Gries, op. cit., page 164.

intellectually better managable, I found this effort at clarification to my amazement (and annoyance) repeatedly rewarded by the accusation that "I had made programming difficult." But the difficulty has always been there, and only by making it visible can we hope to become able to design programs with a high confidence level ...10

...we shall do a much better programming job, provided that we approach the task with a full appreciation of its tremendous difficulty, provided that we stick to modest and elegant programming languages, provided that we respect the intrinsic limitations of the human mind and approach the task as Very Humble Programmers.11

---

10 A Discipline of Programming, page xvi.

11 "The Humble Programmer," Turing Award Lecture, 14 August, 1972. Reprinted in Gries, D., ed., Programming Methodology: A Collection of Articles by Members of IFIP WG2.3, Springer-Verlag, New York, 1978.