Computers and the Sociology of Mathematical Proof

Donald MacKenzie

In this paper, I shall explore the relationship between mathematical proof and the digital computer. This relationship is at the heart of the six scientific and technological activities shown in figure 1. Proofs are conducted *about* computers in at least three areas: those systems upon which human lives depend; key aspects of some microprocessors; and those systems upon which national security depends. These proofs about computers are themselves normally conducted *using* computer programs that prove theorems: automated theorem provers. But mathematicians themselves have also turned to the computer for assistance in proofs of great complication, and automated theorem provers are of considerable interest and importance within artificial intelligence. They raise, for example, the question of whether a computer can be an "artificial mathematician."

I have explored in other papers aspects of the evolution of these six areas since the 1950s: for an outline chronology, see appendix.¹ Here, I shall explore the questions how knowledge *about* computers, and knowledge produced *using* computers, become seen as trustworthy? In particular, I shall address the question of variation: the way in which demonstrations that are convincing to some actors are unconvincing to others; the way in which "mathematical proof" can mean different things to different people. In Galison's terminology,² we are dealing here with a "trading zone": an arena within which different communities, with different traditions of demonstration, achieve a measure of practical coordination of their activities. I have focused upon differences in usage of "proof," but in a sense this is an

arbitrary choice: it would also be possible to examine how they have developed what Galison calls a “pidgin,” which allows fruitful communication between, for example, the National Security Agency, mathematicians, philosophers, and computer hardware designers.

Before turning to variation in the meaning of “proof,” however, it is first necessary for me to explain briefly why proof *about* computers, and proof *using* computers, is thought necessary by many in these interacting communities.

Computers, inductive testing and deductive proof

Why might deductive, mathematical, proofs *about* computers be thought to be necessary? Computer systems can, of course, simply be tested empirically, for example by selecting sets of input data and checking that the corresponding output is correct. Surely that is a clear-cut, inductive way to demonstrate that computer systems will behave as intended?

For testing to yield certainty it would, at a minimum,³ have to be exhaustive: otherwise, errors clearly could lurk in the untested aspects of computer systems. Computer scientists do not agree on whether exhaustive testing of computer hardware or software (I intend the generic term “computer systems” to encompass both) is possible.⁴ Not surprisingly, the answer given tends to depend upon what “exhaustive” is taken to mean. If it is taken to mean coverage of all possible sets of inputs, exhaustive testing will normally be out of the question because the number of possible inputs is intractably large.⁵ On the other hand, “exhaustive” can have more restrictive meanings: for example, that the test cases have exercised all possible

execution paths through a program. In that meaning of exhaustive, then as Myers's standard textbook of program testing puts it (rather equivocally), "possibly the program can be said to be completely tested."⁶

Unfortunately, the practicalities of exhaustive testing, even in the more restrictive meanings, are typically daunting. Myers's text, for example, shows that the number of possible execution paths through even simple programs - ten to twenty lines long - can be enormously large, of the order of as much as 10^{14} , or 100 trillion.⁷ It takes only modest complexity in the system being tested to render exhaustive testing infeasible in practice, even with the fastest and most highly automated testing techniques currently available.

There are two broad responses to the practical limitations of exhaustive testing. On the one hand, proponents of testing point out that it can nevertheless be a very powerful way to discover errors, if enough resources are devoted to it, if the systems in question are designed to facilitate it, if it is systematic, and if it is planned and conducted by suitably skilled and suitably motivated people (especially people not directly involved in the development of the system in question). The world of commercial computing - of mainframe computers, microprocessors, packaged software and the like - has almost always accepted that (in combination, perhaps, with review of designs and programs by persons other than their authors) testing, however imperfect, is the best practical means of gaining knowledge of the properties of computer systems in advance of their use. In an environment where priorities are shaped by the need to control development costs and to move products to the market quickly, the general feeling has been that there is no alternative but to live with the limitations of testing. Intriguingly, there is evidence that the practical efficacy of the testing of deployed systems is considerably greater than its abstract statistical analysis might suggest.⁸ One

possible explanation is that programmers develop a good understanding of where mistakes are likely to lie, and choose test cases with this in mind.⁹

Since the late 1960s, however, a significant strand of thought - originating with academic computer scientists, but influential also in government (especially, in the U.S., in the national security community) and in limited sectors of industry - has begun to take a different approach.¹⁰ Practitioners of this approach seek to verify programs and hardware designs mathematically, rather than just by empirical testing. The key practical appeal of this approach is that a deductive, mathematical analysis can claim to cover *all* cases, not merely the finite number that can be subject to empirical testing. A mathematician who wants to demonstrate the truth of a theorem about, for example, triangles does not embark on the endless task of drawing and checking all possible triangles. He or she looks for a compelling argument - a mathematical proof - that the result must be true for every triangle. The proponents of “formal verification” in computer science have argued since the 1960s that the way to get round the limitations of empirical testing is to proceed as the geometer does: to prove theorems about computer systems.

To verify mathematically the properties of a computer system requires the drawing up of an exact specification of its intended behaviour. Because of their many ambiguities, natural languages such as English are not considered adequate: the specification has to be expressed in a mathematical formalism. From this specification one can attempt to derive deductively a detailed design that would implement it. More commonly, a program is written, or a hardware design produced, by more traditional methods, and then a mathematical proof is constructed to show that the program or design is a correct implementation of the specification.

Proofs *Using* Computers

In the first academic work in the 1960s on formal, mathematical, program verification, computer scientists used the traditional mathematician's tools of pencil and paper to prove that some simple examples of computer programs corresponded to their specifications. Mainstream opinion in this field, however, quickly came to the conclusion that hand-proof is of limited use: the reasoning involved is too tedious, intricate and error-prone, especially for real-world systems rather than "toy" examples. Since the 1970s, therefore, the field has turned to specially-written computer programs for assistance in proof construction: either proof checkers ("book-keeping" programs, intended to store proofs and to detect errors in them) or automated theorem provers (which are programs with a limited capacity to find proofs for themselves).¹¹

Computer scientists interested in formal verification were, however, not the only people to use computer systems to conduct proofs. The earliest automated theorem provers were developed by those who wanted to construct "thinking machines" to demonstrate the feasibility of artificial intelligence: by common consent, the first AI program was Newell, Shaw, and Simon's "Logic Theory Machine," designed to prove simple theorems in the propositional calculus. However, early work on automated theorem proof was also done by philosophers and logicians such as Martin Davis, Hao Wang, and Hilary Putnam. Wang, in particular, seemed to see the artificial intelligence work as incompetent intrusion into the terrain of the logician, and there certainly was tension between the two approaches; tension replicated *within* artificial intelligence as the latter field bifurcated in the 1970s into contending camps of logicist "neats" and proceduralist "scruffies."¹²

Mathematicians, too, also turned to the computer for proof, though not, in general, to automated theorem-provers: their use of the computer was rather for ad hoc assistance in particular parts of complicated proofs. The best-known, and most debated, instance is the use of computer analysis by Kenneth Appel and Wolfgang Haken of the University of Illinois in their 1976 proof of the four-colour conjecture (that four colours suffice to colour in any map drawn upon a plane in such a way that countries which share a border are given different colours). First put forward in 1852, the conjecture had become perhaps the most famous unsolved problem in mathematics, resisting a multitude of efforts at proof for over a century. Appel and Haken's demonstration rested upon computerized analysis, occupying 1,200 hours of computer time, of over 1,400 graphs. The analysis of even one of those graphs typically went beyond what an unaided human being could plausibly do: the ensemble of their demonstration certainly could not be checked in detail by human beings. In consequence, whether that demonstration constituted "proof" was deeply controversial.¹³ (It will be of great interest to discover whether the computerized proof of the Kepler conjecture announced in August 1998 by Thomas Hales of the University of Michigan will be equally controversial, since that is the first instance since the four-colour theorem of computer proof of a "famous" conjecture.)

A prediction and its (near) confirmation

That there might be variation in the kinds of argument that are taken to constitute mathematical proof should come as no surprise to either the sociologist of scientific knowledge or the historian of mathematics. Although nearly all subsequent empirical work in the sociology of scientific knowledge has concerned the natural sciences, rather than mathematics or logic, David

Bloor clearly intended his "strong programme" to be applied to those latter fields as well.¹⁴ In the work of Bloor, and also, for example, that of Eric Livingston, at least schematic arguments can be found to suggest that a sociology of mathematical proof should be possible.¹⁵

The history of mathematics is even more unequivocal in revealing variation in the kinds of argument that have been taken as constituting mathematical proof.¹⁶ Eighteenth-century work in calculus, for example, often relied upon manipulating infinitesimally small quantities or infinite series in ways that became unacceptable in the nineteenth century. Early twentieth-century mathematics was riven by dispute over the acceptability, in proofs involving infinite sets, of the law of the excluded middle (the principle that if a proposition is meaningful, either it or its negation must be true).

These are clues that mathematical proof is a less straightforward, absolute matter than it is often taken to be. In 1987, colleagues and I drew upon this evidence to make a prediction about the effort (by then widely pursued) to apply mathematical proof to computer systems. We noted that this effort involved moving mathematical proof into a commercial and regulatory arena. We speculated that the pressures of that arena would force potential variation in the meaning of proof out into the open, but that disputes about proof would no longer simply be academic controversies. We suggested that it might not be long before a court of law had to rule on what a mathematical proof is.¹⁷

That prediction was nearly borne out in 1991, when litigation broke out in Britain over the application of mathematical proof to a microprocessor chip called VIPER (Verifiable Integrated Processor for Enhanced Reliability), which had been developed by computer scientists working for the Ministry of

Defence's Royal Signals and Radar Establishment. At stake was whether the chain of mathematical reasoning connecting the detailed design of VIPER to its specification was strong enough and complete enough to be deemed a proof. Some members of the computer-system verification community denied that it was,¹⁸ and (largely for unconnected reasons) sales of VIPER were disappointing. Charter Technologies Ltd., a firm which had licensed aspects of VIPER technology from the Ministry of Defence, took legal action against the Ministry, alleging, amongst other things, that VIPER's design had not been proven to be a correct implementation of its specification.

No "bug" had been found in the VIPER chips; indeed, their design had been subjected to an unprecedented amount of testing, simulation, checking and mathematical analysis. At issue was whether or not this process, as it stood immediately prior to the litigation,¹⁹ amounted to a mathematical proof. Matters of fact about what had or had not been done were not central; the key questions that had been raised by critics were about the status, adequacy and completeness, from the viewpoint of mathematical proof, of particular kinds of argument. With the Ministry of Defence vigorously contesting Charter's allegations, the case failed to come to court only because Charter became bankrupt before the High Court heard it. Had it come to court, it is hard to see how the issue of what, in this context, mathematical proof consists in could have been avoided.

The VIPER controversy has been reported elsewhere,²⁰ and a single episode has inevitable idiosyncrasies. Let me turn, therefore, to wider issues about mathematical proof raised by the attempt to apply it to computer systems.

Mathematical proof, intentions and the material world

The first issue is not contentious amongst specialists in the mathematical verification of computer systems, but it is important to highlight it nonetheless because it reveals a potential gap between what these specialists mean by "proof" and what the layperson might take them as meaning. Specialists agree that human intentions, as psychological entities, and computer systems, as physical artefacts, cannot be the objects of mathematical proof. For example, one specialist interviewed for the research upon which this paper is based observed that:

When the President of the United States says, "Shoot down all incoming Russky missiles," that's not quite in predicate logic. ... To translate the requirement into a formal specification introduces a gulf which is not verifiable in any formal system.

Specialists in the field believe that "proof" applied to a computer program or computer hardware can only be proof of correspondence between two mathematical models: a model (the formal specification) of the intended behaviour of the program or hardware; and a model of the program or of the detailed design of the hardware.

To specialists, therefore, "proof," or "formal verification," refers to the correspondence between two mathematical models, not to the relationship between these models and either psychological or physical reality. This immediately opens up a potential discrepancy between specialist and lay views of the assurance conveyed by terms such as "proven" or "verified." To the specialist, they do not mean that systems are correctly designed, or cannot fail (as the lay person might infer), merely that a mathematical model of their detailed design corresponds to their formal specification.²¹

The potential discrepancy matters because lay people's views of the trustworthiness of computer systems can be of considerable importance. The safety of many systems depends on the interaction between computers and human users (who will normally be lay people in terms of their knowledge of computer system verification). Computer-related fatal accidents are caused more commonly by problems in this interaction than by failures of the computer system itself, and users' undue trust in computer systems plays a significant role in these interaction problems.²²

Formal proof and rigorous argument

Substantively important though it is, the potential discrepancy between lay and specialist views of "proof" is less interesting intellectually than variation amongst specialist views. This variation is caused less by the fact that proof is applied *to* computer systems than by the fact that proof is conducted *using* computer systems; but it also goes beyond the latter.

If we examine proofs within the disciplines (mathematics, logic and computer science) where deductive proof is central, we find a large variety of types of argument. Much of this variety is, of course, simply the result of variation in the subject-matter being reasoned about. However, it is also possible to class proofs into two more over-arching categories:

(1) *Formal proof.* A formal proof is a finite sequence of formulae, in which each formula is either an axiom or derived from previous formulae by application of rules of logical inference such as *modus ponens*.²³ The steps in a formal proof are "mechanical" or syntactic applications of inference rules;

their correctness can therefore be checked without understanding the meaning of the formulae involved.

(2) *Rigorous argument.* This broader category includes all those arguments that are accepted by mathematicians (or other relevant specialists) as constituting mathematical proofs, but that are not formal proofs in the above sense. The proofs of ordinary Euclidean geometry, for example, are rigorous arguments, not formal proofs: even if they involve deducing a theorem from axioms (and some involve reasoning that it is not, at least directly, of this form), the steps in the deduction are typically not simply applications of rules of logical inference. This is not simply a reflection of the antiquity of Euclidean geometry: articles in modern mathematics journals, whatever their subjects, almost never contain formal proofs.²⁴

The distinction between formal proof and rigorous argument bears a close - though not one-to-one - relation to the computerization of proof. Formal proof has been relatively easy to automate. The application of rules of inference to formulae, considered simply as strings of symbols, can be implemented on a digital computer using syntactic pattern matching.²⁵ The automation of rigorous argument, on the other hand, has been a far more difficult problem. There are some parts of specific rigorous arguments that can be reduced to calculation or algorithmic checking, and there the potential for automation is high. Furthermore, there are now widely-used commercial programs that automate symbol manipulation in fields like algebra and calculus.²⁶ There are, however, as yet no "artificial mathematicians," in the sense of automated systems capable of handling the full spectrum of rigorous arguments used in different fields of mathematics. The proof checkers and theorem provers used in computer system verification automate formal proof, not - in general - rigorous argument.

Formal proof and rigorous argument are not necessarily *inherently* opposed. It is widely held, for example, that rigorous arguments are "sketches" or "outlines" of formal proofs: arguments with gaps that could, in principle, usually be filled by sequences of applications of rules of logical inference.²⁷ Yet formal proof and rigorous argument remain *available* to be counterposed, for there is a sense in which they have complementary virtues. Because the steps in formal proof do not depend, at least directly, upon the meaning of the formulae being manipulated, they avoid appeals, often implicit, to intuitions of meaning - appeals that can contain subtle, deep pitfalls. That formal proofs can be checked mechanically is a great advantage in a field, like computer system verification, where proofs are typically not "deep" (in the mathematicians' sense of involving profound concepts) but are large and intricate, and where it is particularly desirable to have a corrective to human wishful thinking.

Rigorous arguments, on the other hand, typically have the virtue of surveyability. They are nearly always very much shorter than corresponding formal proofs, and are thus easier for human beings to read and to understand. By virtue of their appeal to the meaning of formulae, rigorous-argument proofs can produce a psychological sense of conviction ("it *must* be so") that is hard to achieve with formal proof, and they have a certain robustness. Typographical mistakes in them, for example, are commonplace, but what carries conviction is the overall structure of the argument, which is more than an aggregate of individual steps. Because this overall structure can be grasped mentally, it can be checked by others, used by them in other contexts, and rejected or improved upon as necessary.

At its most extreme, defenders of one variety of proof can deny the label "proof" to the other. That, for example, was the basis of a famous attack on the formal verification of computer programs mounted in the late 1970s by the American computer scientists Richard DeMillo, Richard Lipton and Alan Perlis. Program verifications, they argued, were mere formal manipulations, incapable of being read, understood and assessed like "real" mathematical proofs. A proof, they said "is not ... [an] abstract object with an independent existence," independent of these "social processes." Since program verifications could not, they claimed, be subject to these social processes, they should not be seen as proofs.²⁸ A similar, albeit less sweeping, contrast between formal proof and rigorous argument also underpinned the most explicit defence of the VIPER proof, which was mounted by a leading figure in the U.K. software industry, Martyn Thomas, head of the software house Praxis. "We must beware," he wrote, "of having the term 'proof' restricted to one, extremely formal, approach to verification. If proof can only mean axiomatic verification with theorem provers, most of mathematics is unproven and unprovable. The 'social' processes of proof are good enough for engineers and other disciplines, good enough for mathematicians, and good enough for me. ... If we reserve the word 'proof' for the activities of the followers of Hilbert [David Hilbert, 1862-1943, leader of 'formalism' within mathematics], we waste a useful word, and we are in danger of overselling the results of their activities."²⁹

Proof and disciplinary authority

Over and above particular contexts in which formal proof and rigorous argument are counterposed is something of a disciplinary divide. Although logic and mathematics are often thought of as similar enterprises, and there is in a general sense a considerable overlap between their subject-matter, they

are, to a degree, socially distinct. Logic's origins as a discipline are closer to philosophy rather to mathematics, and even now logicians distinguish between mathematical logic and philosophical logic. Even the former is sometimes not seen by university mathematics departments as a genuine part of their province, and mathematics undergraduates often learn no more than the most elementary formal logic. "I was going to be a mathematician, so I didn't learn any logic," said one interviewee.

The two general notions of proof - formal proof and rigorous argument - are, to an extent, underpinned by this social divide. Logic has provided the notation and conceptual apparatus that make formal proof possible, and provides a viewpoint from which proof as conducted by mathematicians can be seen as unsatisfactory. One logician, Peter Nidditch, wrote, in 1957, that "in the whole literature of mathematics there is not a single valid proof in the logical sense."³⁰ An interviewee reported that even as an undergraduate:

I knew, from the point of view of formal logic, what a proof was ...
[and was] already annoyed by the vagueness of what constitutes a
mathematics proof.

Mathematics, on the other hand, provides the exemplars of proof as rigorous argument. The main resource for criticism of formal proof is, as we have seen, appeal to the practice of mathematics.

Note, however, that attitudes to proof are not in any simple sense determined by disciplinary background. Disciplines are not homogeneous, biographies are complex, and other factors are at work. Many of those who work in the formal aspects of computer science have training in both mathematics and logic, and so have three disciplinary identities potentially

open to them. When logicians themselves do proofs about formal logical systems, these proofs are typically rigorous arguments, not formal proofs. Those who have wished to computerize proof have had little alternative but to automate formal proof, whatever their disciplinary background. Rather, the connection between discipline and proof is more a matter of the disciplinary authorities of mathematics and, in a weaker sense, logic being *available* to those who, whatever their backgrounds (Martyn Thomas, for example, was trained as a biochemist before entering the computer industry), defend rigorous argument against enthusiasts for formal proof, or vice versa.

Logics, bugs and certainty

Those who perform computer system verifications are often, in practice, adherents of formal proof,³¹ and the regulatory standard in this area that most sharply distinguishes formal proof from rigorous argument, the U.K. Ministry of Defence's interim standard governing safety-critical software, came down in favour of the former, at least for those situations where the highest level of assurance is desirable.³² (One reason this area is fascinating is that a defence procurement standard is forced onto philosophical terrain such as this!) There is, therefore, perhaps potential consensus - at the heart of this "trading zone," within the field of computer system verification, not more widely - that "proof" should mean formal, mechanized proof. Would such consensus lead to complete agreement on "proof," and remove the possibility for future dispute and litigation?

Agreement on the formal notion of proof still leaves open the question of the precise nature of the logical system to be used to manipulate formulae. There are different formal logics. The early twentieth century dispute over the law of the excluded middle has echoes in a continuing divide between

classical logic (which allows one to prove that a mathematical object exists by showing that its non-existence would imply a contradiction) and constructive logic (which requires demonstration of how to construct of the object in question). Computer science has, furthermore, been an important spur to work on "non-standard" logics, such as modal and temporal logics.

"Relevance logic" - long thought to be a philosophers' plaything - is being automated with a view to practical applications in inference from databases.³³

There is even a vogue, and several practical applications, for "fuzzy logic," which permits degrees of simultaneous adherence to both a proposition and its negation, so violating what some have felt to be a cultural invariant.³⁴

This zoo of diverse, sometimes exotic, formal logics is currently a less potent source of dispute than might be imagined. Not all of them are regarded as suitable for mathematical reasoning or computer system verification. Furthermore, there has been a subtle shift in attitudes to formal logic. Nowadays, few specialists take a unitary view of logic in which there is only one true logic and all else is error. It is much more common (especially in computer science) to find a pluralist, pragmatic attitude, in which different logics are seen as technical tools appropriate in different circumstance. Direct clashes between the proponents of different logical systems are far less common than might be expected by extrapolation from, for example, the bitter earlier disputes between classical logic and constructivism. One interviewee, for example, told me that the decision about whether or not to include the law of excluded middle in his theorem prover, a modern version of the issue that had caused great philosophical angst early in the twentieth century, had been simply "a marketing decision"!

Nevertheless, the possibility remains that a pluralist, pragmatic attitude to different logics is a product of the early, exploratory, academic

phase of the application of formal logic to computer systems.³⁵ It cannot be guaranteed that it would remain intact in a situation where there are major financial or political interests in the validity or invalidity of a particular chain of formal reasoning. Specialists who are called upon to *justify* their formal logic, under courtroom cross-examination by a well-briefed lawyer, may face difficulties, for that process of justification is fraught with philosophical problems.³⁶

At least equal in importance to the diversity of formal logics as a source of potential dispute is the simple fact that the tools of mechanized formal proof (proof checkers and automated theorem provers) are themselves computer programs. As such, they - especially automated theorem provers, which are quite complicated programs - may contain errors. In interviews for this research, designers of automated theorem provers often reported experience of bugs in their systems that would have allowed “theorems” that they knew to be false nevertheless to be proven. Such bugs were not large in number, they were corrected whenever they were discovered, and no interviewee reported such a bug causing a false result whose falsity had not been detected readily. But no designer seemed able to give an unequivocal guarantee that no such bugs remained.

No automated theorem prover has, to date, itself been subject in its entirety to formal verification.³⁷ Even if one had been verified - or even if the results of an unverified prover were checked by an independent proof checking program (something several interviewees advocated, but which is rare in practice) - many specialists in the field would still not regard this as guaranteeing complete certainty. The reasons given by them ranged from “Gödelian” concerns about the consistency of formal systems³⁸ to the

possibility of coincident errors in different automated theorem provers or proof checkers.

Reactions to the difficulty of achieving certainty with automated theorem provers varied amongst those interviewed. One interviewee suggested that it indicates that the overall enterprise of formal verification is flawed, perhaps because of what he felt to be the impoverishment of its notion of proof:

You've got to prove the theorem-proving program correct. You're in a regression, aren't you? ... That's what people don't seem to realise when they get into verification. They have a hairy great thing they're in doubt about, so they produce another hairy great thing which is the proof that this one's OK. Now what about this one which you've just [used to perform the proof]? ... I say that serves them jolly well right.

That is not the response of program and hardware verification "insiders." While paying considerable attention to soundness, they feel that theorem-prover bugs are not important practical worries compared to ensuring that the specification of a system expresses what, intuitively, is intended:

If you ... ask where the risks are, and what are the magnitudes of the risks, the soundness of the logic is a tiny bit, a really tiny bit, and the correctness of the proof tool implementing the logic is slightly larger [but] actually ... quite a small risk.

Insiders almost all share the perception that the risk of a serious mistake in computer system verification being caused by a bug in an automated prover is small, but they are also wary of claiming that the risk can

ever be shown to be *zero*. Their judgments differ as to what measures are necessary to allow claims of proof safely to be made. Some would not be happy without a full formal proof checked by an independent proof checking program; others feel that this would be a waste of effort, compared to the need for attention to more likely dangers, notably deficient specifications.

One potentially contentious issue is the use of decision procedures: procedures that can decide, in a wholly deterministic, algorithmic way, whether or not mathematical statements in particular domains are true. Typically, decision procedures return simply the judgement "true" or "false," not a formal proof of truth or falsity. To some interviewees, decision procedures are a necessary and harmless part of the "real world" of computer system verification. They offer the great practical advantage that their use does not require the time-consuming skill that a generic theorem prover typically demands. To other practitioners, however, decision procedures must themselves be verified formally (which is currently far from universal). Otherwise, using them is "like selling your soul to the Devil - you get this enormous power, but what have you lost? You've lost proof, in some sense."

Conclusion

That there is variation in what proof is taken as consisting in, and that insiders do not believe absolute certainty is achievable, are not arguments against seeking to apply mathematical proof to computer systems. Even in its current state, the effort to do so has shown its practical worth in finding errors not found by conventional methods.³⁹ Furthermore, the field is not static, and some of the issues I have discussed (such as the verification of decision procedures) are active research topics. Even the overall divide between formal proof and rigorous argument is not necessarily unbridgeable,

with many insiders believing it to be possible to construct proofs that are formal in their detailed steps but that still have a humanly surveyable overall structure. It is true that the end of the Cold War, and subsequent reductions in defence budgets, have caused a precipitous decline in the traditional chief funding source for formal verification: computer security. Nevertheless, a well-established tradition of academic research, and the emergence of new spheres of application for formal methods (especially to hardware), give grounds for at least modest confidence that the field will survive its short-term difficulties.

Historian and sociologists of science certainly have grounds for hoping that it does so! The automation of proof, largely fuelled as a practical activity by the desire to do proofs about computer systems, is a fascinating intellectual experiment. It highlights the fact that our culture contains not one, but two, ideals of mathematical proof: rigorous argument and formal proof. The distinction, of course, predates the digital computer. But when David Hilbert, for example, put forward the notion of formal proof in the 1920s, he did not intend that mathematicians should actually do their proofs in this way. Instead, formal proof was a tool of his "metamathematics," a way of making mathematical proof the subject of mathematical analysis by providing a simple, syntactic, model of it. Nidditch was still able, as we have seen, to claim in 1957 that the "whole literature of mathematics" contained not a single formal proof.

From the point of view of the history, philosophy, and sociology of science, the significance of the automation of proof is that it has turned formal proof from a tool of metamathematical inquiry into a practical possibility. To some extent at least, a *choice* between formal proof and rigorous argument has been opened up. The mainstream mathematical community remains firmly

wedded to rigorous argument and sceptical of computer "proof," as was demonstrated by reactions to the Appel and Haken proof of the four colour theorem, or, more recently, by the indifference that greeted "QED," an ambitious proposal to create a giant computerized encyclopaedia of formally, mechanically proven mathematics. Nevertheless, formal, mechanized proof is now well entrenched in a variety of niches outside of pure mathematics. Although, to be sure, there are many areas of mathematics where such proof is not, or at least is not yet, a viable alternative to conventional forms of demonstration, its very existence indicates something of the contingency of the latter.

Furthermore, the automation of proof throws interesting light upon formal proof itself. Just as a choice between formal proof and rigorous argument has opened up, so those committed to formal, mechanized proof face decisions. Which logic shall they adopt? How shall they ensure that a putative formal proof is indeed a proof? Is an independent mechanized proof checker a necessity? Should a theorem prover itself be subject to formal verification? Going beyond the current practical state of the art, further questions can be anticipated: must the compiler for the prover's language be verified? What about the machine upon which it runs?

Questions such as these are not merely of academic interest. Formal verification is, as I have emphasized, a practical activity, often conducted in a commercial and regulatory arena. As the VIPER controversy showed, epistemological questions can, in that context, take on a sharp "real-world" significance, as practitioners have to decide what forms of demonstration constitute mathematical proof. The VIPER episode is as yet *sui generis*: no other dispute about formal verification or automated proof has led to litigation. If,

however, formal verification prospers, it will be of great interest to discover whether (in Galison's terminology) a stable creole, with a stable meaning of "proof," will emerge in this trading zone, or whether the VIPER episode becomes the first of many disputes over what proof consists in.

Appendix: An Outline Chronology

1955/56	Newell, Shaw, Simon: Logic Theory Machine
1959	Herbert Gelernter: Geometry Machine
1960	Hao Wang champions algorithmic against heuristic theorem provers
1962	John McCarthy: "Towards a Mathematical Science of Computation"
1963	Alan Robinson: Resolution, a machine-oriented inference rule
1966	Peter Naur: "Proof of Algorithms by General Snapshots"
1967	R.W. Floyd: "Assigning Meanings to Programs"
1967	First public discussion of computer security vulnerabilities of US defence systems
1968	NATO conference on Software Engineering at Garmisch: diagnosis of "software crisis"
1969	C. A. R. Hoare: "An Axiomatic Basis for Computer Programming"
1971	Robert Boyer and J. Strother Moore begin collaboration that leads to their theorem prover
1973-80	SRI PSOS project. Provably Secure Operating System
1976	Appel and Haken prove four-colour conjecture, using 1200 hours computer time
1978-87	SRI proof work on SIFT fault-tolerant avionics computer
1979	DeMillo, Lipton and Perlis attack program proof in <i>Communications of the ACM</i>
1983	"Orange Book." Design proof required for A1 secure systems

1983	SIFT Peer Review
1984-85	VIPER proof effort begins at Royal Signals and Radar Establishment
1985	Honeywell Secure Communications Processor: first system to achieve A1 rating.
1986-90	Disputes over Trip Computer Software delay licensing of Darlington, Ontario, nuclear power station
1986	First clear-cut software error deaths: overdoses from Therac-25 radiation therapy machines
1987	Inmos T800 transputer: floating-point unit microcode formally verified
1987-88	Commercialisation of VIPER begins
1988-89	Entry into service of Airbus A320, first fly-by-wire airliner
1988	Crash of A320 at Habsheim air show triggers fierce controversy in France
1988	James Fetzner attacks program proof in <i>Communications of the ACM</i>
1989	Formally verified SACEM train protection software enters service on RER, Paris
1991	UK Interim Defence Standard 00-55: proof required for systems most critical to safety
1991	VIPER law suit
1994	Hackers seize control of computer network at USAF Rome (N.Y.) Lab
1994	Pentium divide bug
1995	Entry into service of Boeing 777, most computer-intensive airliner to date

1996   New computer-assisted proof of four-colour theorem by Robertson et al.

---

¹ The research reported on here was funded by the UK Economic and Social Research Council under the Program in Information and Communication Technologies (A35250006) and research grants R000234031 and R00029008; also by the Engineering and Physical Sciences Research Council under grants GR/J58619, GR/H74452 and GR/L37953. More detailed aspects are discussed in MacKenzie (1991, 1994, 1995, forthcoming a, forthcoming b), MacKenzie & Pottinger (1997), and MacKenzie & Tierney (1996). This paper is a development of three earlier papers attempting to summarize the main results of this work: MacKenzie (1993a, 1993b, 1996).

²Gallison (1997).

³There are other issues here too, for example concerning (a) what it means for output to be correct and (b) the possibility of physical failure of a computer system or its components.

⁴In interviews and other discussions, proponents of testing have told me emphatically that exhaustive testing is possible, while proponents of formal verification frequently deny the possibility of exhaustive empirical testing.

⁵There may be only a finite number of valid inputs (e.g. integers may be restricted to a certain size), but it can be argued that "to be sure of finding all ... errors, one has to test using not only all *valid* inputs, but all *possible* inputs" (Myers, 1979, p. 8). In other situations, the number even of valid inputs will be, effectively, infinite. The valid inputs to a compiler, for example, are all the possible executable programs written in the corresponding language.

⁶Myers (1979, p. 10).

⁷Ibid.

⁸See Hoare (1994).

⁹ It is also possible, however, that there is an element of selection bias here: that systems that come into widespread use are better than those that are not brought to market, or are little used; and widely-used systems are, of course, "debugged" by their many users as well as by their developers.

¹⁰For examples, see MacKenzie (1993).

---

¹¹For an outline history of the latter, see MacKenzie (1995).

¹²For these debates, see MacKenzie (1995). For "neats" and "scruffies" see Hayes (1987) and Crevier (1993).

¹³For a history, see MacKenzie (forthcoming a).

¹⁴Bloor (1973, 1976).

¹⁵See Bloor (1983, 1994, 1997) and Livingston (1986). Barnes, Bloor and Henry (1996, chapter 7).

¹⁶There is a useful survey in Kleiner (1991).

¹⁷Peláez, Fleck and MacKenzie (1987, p. 5).

¹⁸Brock and Hunt (1990); see also Cohn (1989).

¹⁹There has been considerable further proof work since then on VIPER.

²⁰MacKenzie (1991, 1993).

²¹For a trenchant statement of this by a specialist, see Cohn (1989). The celebrated critique of program verification by Fetzer (1988) amounts largely to the same point.

²²MacKenzie (1994).

²³Let p and q be any propositions. If p and p implies q are either axioms or previous formulae in the sequence, then (according to *modus ponens*) we can derive q . The "meaning" of p and q do not enter into the derivation. On the more general conception of formal proof used here, see, for example, Boyer and Moore (1984, p. 181).

²⁴Although those in this field would all recognise something akin to the distinction drawn here, there is no entirely standard terminology for capturing it. Thus many mathematicians would call deductions from axiomatic set theory "formal proofs," even if these deductions are not simply applications of rules of logical inference. What I am calling "rigorous argument" is sometimes called "informal proof", but I avoid that phrase because it carries the connotation of inferiority vis-à-vis formal proof. I draw the term "rigorous argument" from Ministry of Defence (1991, part 2, p. 28): see MacKenzie (1993).

²⁵Many of the theorem-prover bugs discussed below were caused by subtle mistakes in these algorithms. Participants recognize these as "errors," rather than matters of dispute,

---

indicating a degree of consensus about the basic process.

²⁶For a critical discussion of their soundness, see Stoutemyer (1991).

²⁷There is, however, general acceptance of Kurt Gödel's celebrated conclusion (the "incompleteness theorem") that any finite formal system rich enough to encompass arithmetic must contain a true statement for which there is no formal proof within the system.

²⁸DeMillo, Lipton and Perlis (1979, esp. p. 273); see MacKenzie (1993)

²⁹Thomas (1991), quoted in MacKenzie (1993).

³⁰Nidditch (1957, p. v).

³¹The main exception to this generalization concerns decision procedures (see below). There is also variation in attitude to the importance of the surveyability of formal proofs.

³²Ministry of Defence (1991); see Tierney (1992).

³³Thistlewaite, McRobbie, and Meyer (1988). Relevance logic excludes the elementary theorem of standard logic that a contradiction implies any proposition whatsoever (see Bloor 1983, 123-32). The database issue is the possible need to make sensible inferences in the presence of contradictory items of data.

³⁴Q.v. Archer (1987). A major issue about fuzzy logic is whether it is indeed a logic, or a version of probability theory. Claude Rosenthal of the École des Mines has recently completed a PhD thesis dealing with the development of fuzzy logic.

³⁵MacKenzie (1993).

³⁶Best known is Kurt Gödel's theorem that a finite formal system rich enough to express arithmetic cannot be proven consistent without use of a more powerful system, the consistency of which in turn would have to be proven, hence beginning an endless regress. On more general problems of circularity and regress in the justification of deduction, see Haack (1976).

³⁷The Boyer-Moore prover (a leading U.S. system) is being re-engineered with a view to using it to verify its own correctness.

³⁸See note 36 above.

³⁹Glanz (1995).