• *ABSTRACT*

*The first part of this paper discusses floating-point arithmetic, as performed by computers and advanced digital calculators. It shows that different computer arithmetics have been proposed, and that the nearest approximation to a consensual computer arithmetic – the Institute of Electrical and Electronics Engineers' standard for floating-point arithmetic – had to be negotiated, rather than deduced from existing human arithmetic.*

*The second part of the paper discusses mathematical proof of the correctness of programs and hardware designs, which is increasingly being demanded for systems crucial to safety or security. It argues that this extension of the domain of application of mathematical proof involves the negotiation of what a proof consists in. In 1987, this argument led the author and colleagues to predict a legal case concerning the nature of mathematical proof. Such litigation took place in 1991; the key point at issue is described. More general disagreement about proof as applied to computer systems is also discussed. A further prediction is made: that there will be litigation concerning not just what kind of argument counts as a proof, but over the internal structure of formal proofs.*

# Negotiating Arithmetic, Constructing Proof: The Sociology of Mathematics and Information Technology

## Donald MacKenzie

**Since the 1960s**, computer systems have been a subject of considerable interest to social scientists. Their diffusion, their likely effects on organizations, on employment levels and on society at large, the evolution of the computer industry – these and other topics have received considerable attention. Computer systems as *mathematical* entities have, however, remained almost entirely the province of technical specialists. This paper seeks to redress that balance, by arguing that computer systems offer interesting, counter-intuitive, case studies in the sociology of mathematics.

The paper discusses two different aspects of computer systems as mathematical entities. First is the computer (and also the advanced

*Social Studies of Science* (SAGE, London, Newbury Park and New Delhi), Vol. 23 (1993), 37–65

digital calculator) as an arithmetical tool. Our intuitions might suggest that arithmetic done by calculator or computer would be a wholly unproblematic area. Arithmetic, after all, is the very paradigm of secure, consensual knowledge, and surely the calculator or computer simply removes the tedium and error-proneness of arithmetic performed by human beings? Not so. Not only is considerable skill, normally taken entirely for granted, needed in order reliably to perform arithmetic even on simple calculators, as Harry Collins has recently discussed,[1] but there has also been significant dispute as to the nature of the arithmetic to be implemented, at least when the numbers to be worked with are not integers. Different computer arithmetics have been proposed, and the nearest approximation to a consensual computer arithmetic, the Institute of Electrical and Electronics Engineers' standard for floating-point arithmetic, had to be negotiated, rather than deduced from existing human arithmetic.

The second aspect of computer systems discussed here is mathematical proof of the correctness of a program or hardware design. As computer systems are used more and more in situations where their failure could have disastrous consequences, there have been increasing demands for such mathematical proofs in place of, or in addition to, empirical testing. This is of interest from the point of view of the sociology of knowledge, because it involves a move of 'proof' from the world of mathematicians and logicians to that of engineers, corporations and lawyers.

Although mathematical proof is being sought precisely because of the certainty it is ordinarily held to grant, constructing proofs of computer system correctness again turns out to be no simple 'application' of mathematics. It involves negotiating what proof consists in. Five years ago, colleagues and I predicted that the demand for proofs of computer system correctness would inevitably lead to a court of law having to rule on the nature of mathematical proof.[2] The paper summarizes the controversy that led to this prediction having already come close to confirmation, and explores more general issues of the 'sociology of proof' in the context of computer systems.

## Negotiating Arithmetic

Although past societies have used different number systems, human arithmetic in advanced industrial societies is now consensual. Typically, agreement can be found on the correct outcome of any

calculation. There are, for example, to my knowledge no scientific disputes where different sides have disagreed at the level of the arithmetic itself. Furthermore, there are certain 'ideal' laws that we agree must hold, independent of any particular calculation. For example, we agree that the addition ( + ) and multiplication ( × ) of real numbers should be both associative and commutative – that is to say, that:

$$x + y = y + x \qquad x \times y = y \times x$$
$$(x + y) + z = x + (y + z) \qquad (x \times y) \times z = x \times (y \times z)$$

whatever the values of x, y and z.

The consensual status of arithmetic has indeed been taken as indicating a self-evident limit on the scope of the sociology of knowledge.[3] It might seem to make implementing arithmetic on a calculator or computer a straightforward business. Yet that has not been the case.

The difficulties are most striking in the form of arithmetic used in the kind of computations typically found in scientific work: floating-point arithmetic. This is closely analogous to the well known 'scientific notation' for expressing numbers. In the latter, a number is expressed in three parts:

— a positive or negative sign
— a set of decimal digits, known as the significand or mantissa, including a decimal point.
— a further set of digits, known as the exponent, which is a power of ten.

Thus 1,245,000 could be expressed in 'scientific notation' as

$$+ 0.1245 \text{ times } 10^{7}$$

and − 0.0006734 as

$$- 0.6734 \text{ times } 10^{-3}$$

The advantage of scientific notation is that allowing the decimal point to 'float' in this way leads to a more economical and easily manipulable format than the standard representation, where the decimal point is 'fixed' in its position.

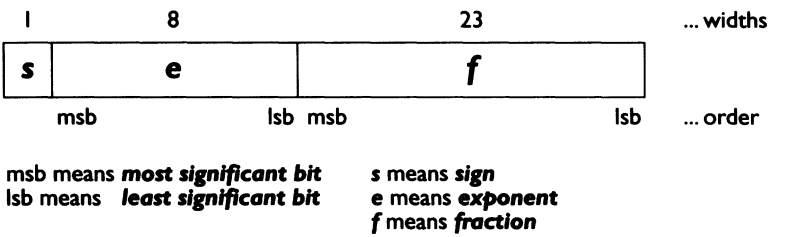
Computer floating-point arithmetic carries over this flexibility, and is quite similar, except in the following respects:

1. Decimal (base ten) representation is now unusual, with hexadecimal (base sixteen) or binary (base two) more common. Since the episode I am about to discuss concerns binary arithmetic, let us

concentrate on that. All digits are one or zero: the decimal 3, for example, is expressed as 11, the decimal 4 as 100, and so on. The exponent is a power of two, and the equivalent of the decimal point is known as the binary point. The sign, similarly, is expressed as a binary digit, typically with a zero for positive numbers and a one for negative numbers.[4]

2. A firm decision has to be taken as to the number of binary digits (bits) in the significand and exponent. In the arithmetic to be discussed, for example, the basic format[5] has one binary digit to express the sign, eight to express the exponent,[6] and twenty-three to express the significand, adding to a total of thirty-two.

**FIGURE 1**
**Number Representation in IEEE Floating-Point Arithmetic**




*Source*: *IEEE Standard for Binary Floating-Point Arithmetic: American National Standards Institute/Institute of Electrical and Electronics Engineers Standard 754–186* (New York: Institute of Electrical and Electronics Engineers, 12 August 1985), 9.

The total number of bits (sometimes called word-length) is of considerable importance, since most modern computers seek to process all the bits making up a number in parallel, rather than one after the other. The more bits to be processed simultaneously, the greater the complexity of the hardware needed.

3. Any floating-point system, unless constrained, allows multiple representations of the same number. For example, $-0.0006734$ could be expressed by $-67.34$ times $10^{-5}$, as well as by $-6.734$ times $10^{-4}$, or $-0.6734$ times $10^{-3}$. Computer floating-point systems, however, typically employ a unique 'normal' representation of each non-zero number. In that representation, there is always exactly one non-zero bit to the left of the binary point. Since that bit must be a one, it need not be stored explicitly, and only the parts of the

significand to the right of the binary point (known for obvious reasons as the 'fraction') are explicit.

There are several decisions, then, to be taken in implementing computer floating-point arithmetic. What base shall be used? What word-length, size of significand and size of exponent? Shall a sign-bit of one represent negative numbers or positive ones? How shall zero be represented? What methods of rounding shall be used? What should be done if the result of a calculation exceeds the largest absolute value expressible in the chosen system (that is, if it 'overflows'), or if it falls below the smallest (that is, if it 'underflows')? What should be done if a user attempts an arithmetically meaningless operation, such as dividing zero by zero?

Different computer manufacturers (and then, as it became possible to implement floating-point arithmetic on electronic calculators, different calculator manufacturers) answered these questions differently. This was patently a source of some practical difficulty, since it led to problems in using a numerical program written for one computer on another, even when standard programming languages such as FORTRAN were used. But did it matter more profoundly than that? Were the results of the different decisions really different arithmetics? Or were they simply different, but essentially equivalent, ways of implementing the one true arithmetic?

The answer to these questions depends on how one reacts to what might be called 'anomalous calculations'. Under some circumstances, different machines yield substantially different results for the same calculation. In certain cases it is possible to check machine results against the results of human arithmetic, and in some of the cases the machine can be judged wrong. In other cases, machine arithmetic violates consensual laws, such as the commutative and associative nature of multiplication. Examples are given in Figure 2.

Different reactions to 'anomalous calculations' can be categorized according to the schema developed by Lakatos in his celebrated analysis of the evolution of Euler's theorem concerning the relationship between the number of faces (F), edges (E) and vertices (V) of a polyhedron. Lakatos showed that attempts to prove the relationship

$$V - E + F = 2$$

were plagued by counter-examples – figures which could be claimed to be tetrahedra, but which did not obey the law.[7]

**FIGURE 2**

Multiplication is neither commutative nor monotonic on the TI [Texas Instruments] 59; try $e\pi\text{-}\pi e$. Division on the TI Business Analyst gets a different quotient for 1/3 than for 9/27. Double precision in BASIC on the IBM PC alleges often that $X/1 \neq X$, and $1.000\ldots0000/1.000\ldots0001 \geqslant 1$.

---

A bank retains a legal consultant whose thoughts are so valuable that she is paid for them at the rate of a penny per second, day and night. Lest the sound of pennies dropping distract her, they are deposited into her account to accrete with interest at the rate of 10% per annum compounded every second. How much will have accumulated after a year (365 days)?

Enter data:
n [number of periods]: = 60 × 60 × 24 × 365 = 31,536,000 sec. per year.
i [rate of interest]: = 10/n = 0.000 000 317 097 9198 % per sec.
PV [Principal Value at start] = 0
PMT [amount of periodic PayMenTs] = −0.01 = one cent per sec. to the bank.

Pressing {FV} [Final Value] should display one year's accretion but different financial calculators display different amounts:

| *Calculators* | *FV displayed* |
|---|---|
| 27, 92, 37, 38, 12 | $331 667.00 |
| BA | 293 539.16 |
| MBA | 334 858.18 |
| 58, 58C, 59 | 331 559.38 |

*Source*: W. Kahan, 'Mathematics written in Sand – the hp-15C, Intel 8087, etc.', *Proceedings of the American Statistical Association, 1983 Statistical Computing Section*, 12–26, at 14 and 16.

One response to these 'anomalous figures' was what Lakatos calls 'primitive exception barring', simple indifference to their existence. That characterizes well what seems to have been the majority response to anomalous computer or calculator calculations. Most users have probably been either unaware of the possibility of anomalous calculations, or alternatively unconcerned about them, in the same sort of sense that we continue happily to cross bridges even though we are aware that some bridges have collapsed. For many computer designers, too, anomalous calculations seem to have been well down the list of matters needing attention, if they were on it at all.

A more sophisticated 'exception barring' strategy was to cite the vast bulk of calculations that were performed perfectly satisfactorily, and to argue that anomalous calculations were instances of problems that were not 'well posed'. A well posed problem was one in which a slight change of data caused only a slight change of result; the solution employed an algorithm that was in this sense 'numerically stable'. The newly developed technique of 'backward error analysis' was used, in justification of this response, to discriminate between well posed problems and those that were not well posed. Computers and calculators worked reliably on well posed problems. If 'pathological' calculations and 'degenerate' problems were avoided (use of these terms might be taken as indicating that exception barring was sliding into what Lakatos calls 'monster barring'), no difficulties would arise.[8]

A small number of computer scientists, however, positively sought these 'monsters'.[9] Leading among them was Professor W. Kahan, who holds a joint appointment in the Mathematics Department and Department of Electrical Engineering and Computer Science at the University of California, Berkeley. The examples of anomalous calculations in Figure 2 are drawn from his writing. Kahan's approach is an example of what Lakatos calls the 'dialectical' strategy: 'anomalies and irregularities are welcomed and seen as the justification for new approaches, new concepts and new methods'. [10] Kahan has been a reformer, not content with the current state of computer and calculator floating-point arithmetic, and constantly seeking to devise, and build support for, ways of improving it. He has quite deliberately sought to discover and publicize anomalies which can be used to show that differences between computer arithmetics are serious and consequential.

What first gave Kahan the opportunity to reform arithmetic in the direction he desired was competition between two leading manufacturers of sophisticated calculators, Texas Instruments and Hewlett Packard. The former attacked the latter's calculators as the less accurate, while Hewlett Packard responded by claiming that calculation on its competitor's machines manifested more anomalies. A Hewlett Packard executive, Dennis Harms, saw advantage in attempting to strengthen Hewlett Packard's position in this respect: he employed Kahan as a consultant on the design of the arithmetic of the corporation's new generation calculators, thus enabling Kahan to get his ideas incorporated into them.[11]

Kahan's next opening came around 1977, when the leading microprocessor firm Intel started to develop a silicon chip specifically to

perform floating-point arithmetic. Existing microcomputers implemented floating-point arithmetic in their software, rather than hardware, while the hardware floating-point units in large computers were multi-chip. The Intel i8087, as the chip was eventually christened, was intended as a 'floating-point coprocessor', working alongside the main processing chip in a microcomputer to improve its arithmetic performance.

John Palmer, the engineer leading the design of the i8087, had attended lectures by Kahan as an undergraduate, and turned to him for advice.[12] Palmer rejected the idea of adopting 'IBM arithmetic', despite its widespread use; Kahan believed this arithmetic to be inferior. The arithmetic of the leading minicomputer maker, the Digital Equipment Corporation, was also rejected. Palmer was, however, not simply seeking 'product differentiation'. He was worried that if the wrong decisions were made it would be impossible to share some programs between 'different boxes all bearing the Intel logo', and he wanted to avoid for floating-point arithmetic on microprocessors 'the chaotic situation that now exists in the mainframe and minicomputer environments'.[13]

Intel and other Silicon Valley chip manufacturers supported the establishment of a committee to consider standards for floating-point arithmetic. The initiative for the committee had come from an independent consultant, Dr Robert G. Stewart, who was active in the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE), under whose aegis the committee was established. Stewart recruited to the committee Kahan and representatives of Intel, other chip manufacturers, and minicomputer makers (see Table 1). Richard Delp was appointed by Stewart as the first chair of the working group.[14] Intel – which was hard at work on other projects – agreed to delay finalizing the arithmetic of the i8087 while the committee deliberated, even though Intel clearly hoped that the final standard would be similar to what it had already developed internally.

Negotiating arithmetic proved to be a lengthy process. The committee started work in September 1977, and the IEEE Standard 754, Binary Floating-Point Arithmetic, was adopted only in 1985.[15] The general nature of the vested interests involved is clear. Unless the committee took the easy route of writing a general standard that would 'grandfather' all widely used existing arithmetics (an option that was considered but rejected), or unless it opted for an arithmetic radically different from any in existence, whatever it decided would be

**TABLE 1**

*Voting Members of Floating-Point Working Group, Microprocessor Standards Committee, IEEE Computer Society, at time of adoption of Draft Standard in 1980.*

Andrew Allison, Los Altos Hills, California
William Ames, Hewlett-Packard Data Systems
Mike Arya, Cupertino, Calif.
Janis Baron, Intel
Dileep Bhandarkar, Digital Equipment Corporation
Joel Boney, Motorola
Jim Bunch, University of California, La Jolla
Ed Burdick, National Semiconductor
Paul Clemente, Prime Computer
W. J. Cody, Argonne National Laboratory
Jerome T. Coonen, University of California, Berkeley
Jim Crapuchettes, Menlo Computer Associates
Richard H. Delp, Four-Phase Systems
Alvin Despain, University of California, Berkeley
Tom Eggers, Digital Equipment Corporation
Dick Fateman, University of California, Berkeley
Don Feinberg, Digital Equipment Corporation
Stuart Feldman, Bell Laboratories
Eugene Fisher, Lawrence Livermore National Laboratory
Paul F. Flanagan, Analytical Mechanics
Gordon Force, Xylex
Lloyd Fosdick, University of Colorado
Robert Fraley, Hewlett-Packard Laboratories
Howard Fullmer, Parasitic Engineering
Daniel D. Gajski, University of Illinois, Urbana
David Gay, Massachusetts Institute of Technology
C.W. Gear, University of Illinois, Urbana
Martin Graham, University of California, Berkeley
David Gustavson, Stanford Linear Accelerator Center
Guy Haas, Datapoint
Chuck Hastings, Data General
David Hough, Apple Computer
John E. Howe, Intel
Thomas E. Hull, University of Toronto
Suren Irukulla, Prime Computer
Richard James III, Santa Clara, California
Paul S. Jensen, Lockheed Research Laboratory
William Kahan, University of California, Berkeley
Howard Kaikow, Nashua, New Hampshire
Dick Karpinski, University of California, San Francisco
Virginia Klema, Massachusetts Institute of Technology
Les Kohn, National Semiconductor
Dan Kuyper, Sperry Univac
M. Dundee Maples, M & E Associates
John Markiel, Westmont, New Jersey
Roy Martin, Apple Computer
Dean Miller, Motorola
Webb Miller, University of California, Santa Barbara
John C. Nash, Vanier, Ontario, Canada
Dan O'Dowd, National Semiconductor
Cash Olsen, Signetics
John F. Palmer, Intel
Beresford Parlett, University of California, Berkeley
Dave Patterson, University of California, Berkeley
Mary Payne, Digital Equipment Corporation
Tom Pittman, Itty Bitty Computers
Lewis Randall, Apple Computer
Robert Reid, Dunstable, Massachusetts
Christian Reinsch, Leibniz-Rech/Bay. Akad. Wiss.
Roger Stafford, Beckman Instruments

*cont.*

TABLE 1—*cont.*

David Stevenson, Zilog
G. W. Stewart, University of Maryland
Robert G. Stewart, Stewart Research Enterprises
Harold Stone, University of Massachusetts
William D. Strecker, Digital Equipment Corporation
Robert Swarz, Digital Equipment Corporation
George Taylor, University of California, Berkeley
Dar-Sun Tsien, Intel
Greg Walker, Motorola
John Stephen Walther, Hewlett Packard Laboratories
P.C. Waterman, Burlington, Massachusetts

*Source*: 'A Proposed Standard for Binary Floating-Point Arithmetic', *Computer* (March 1981), 51–62, at 62.

bound to advantage those companies whose existing practice was closest to the standard, and to disadvantage those whose practice differed widely from it. The latter would be forced into an unpleasant choice. If they retained their existing arithmetic, their market could diminish as a result of users preferring machines implementing the standard. If they changed, considerable investment of time and money would have to be written off, and there might be troublesome incompatibilities between their new machines and their old ones.

Ultimately the choice came down to two arithmetics closely aligned with major corporate interests. One was essentially the arithmetic employed by the Digital Equipment Corporation (DEC), the leading manufacturer of minicomputers. Its VAX machines were very widely used in scientific computing, and their arithmetic was admitted even by its critics to be 'sound' and 'respectable'.[16] The other was a proposal written by Kahan, his graduate student Jerome Coonen, and Dr Harold Stone, Manager of Advanced Architectures at IBM's Yorktown Heights Laboratory. Not surprisingly, given the collaboration between Kahan and Intel's Palmer, that proposal was very similar to what Intel was already well on the way to implementing.[17]

The Kahan-Coonen-Stone scheme has several interesting features, such as the handling of zero. In their basic format they opted for what is called a 'normalized zero'. Zero is expressed only by a zero significand and zero exponent (0 times $2^0$). The combination of zero significand and non-zero exponent (0 times $2^1$, 0 times $2^2$, and so on) is not permitted. But they permitted the sign bit to take both values, and allowed its value to be meaningful. In other words, unlike consensual human arithmetic, which contains only one zero, their arithmetic contains both a positive and a negative zero, with, for example, the rule that the square root of $-0$ is $-0$.[18]

This and other features of their arithmetic were, however, relatively uncontroversial. The battleground between their proposal and the main alternative arithmetic was underflow. Unlike the arithmetic of real numbers, where there is no number 'next-to-zero' and indefinitely small numbers are possible, computer arithmetics contain a lower bound, albeit tiny, on the size of number that can be represented. For example, $2^{-126}$, or roughly $10^{-38}$, is the smallest number possible in normal representation in the Kahan-Coonen-Stone scheme's basic format. Like the majority of existing computer arithmetics, DEC's arithmetic simply represented all numbers as precisely as possible until the number next to zero was reached. Should a calculation yield a result smaller than that, very small, number, the result was stored as zero. 'Flush-to-zero underflow' is what this scheme is generally called.

Kahan and his colleagues advocated the different principle of 'gradual underflow'.[19] They introduced a special set of 'denormalized numbers' smaller in size than the normal-format number next-to-zero. As noted above, in normal floating-point format the digit to the left of the binary point is always one. In a denormalized number it is zero. Denormalized numbers are created by right-shifting the significand so that the exponent always remains within the expressible range. So, in a system where the smallest normal number is $2^{-126}$, $2^{-127}$ could be given denormalized expression as a half (0.1 in binary) times $2^{-126}$; $2^{-128}$ as a quarter (0.01 in binary) times $2^{-126}$; and so on.

Of course, this meant that accuracy would usually be lost, as one or more significant digits would have to be discarded in right-shifting the significand. But this, to its proponents, seemed an acceptable price to pay for a more gradual approach to zero. Their argument against what many took to be the 'obvious' DEC procedure was that, using the latter, as one approached zero the differences between successive numbers diminished, until one reached the number next-to-zero, whose distance from zero would be much greater than its distance from the next larger number. In gradual underflow the differences between successive numbers diminished monotonically all the way down to zero (see Figure 3).

Gradual underflow became the focus of attacks on Kahan, Coonen and Stone's proposed arithmetic:

> The interconnectedness of the proposed standard's basic features complicated attempts to oppose it. Early challenges within the subcommittee were not easily

focused on single aspects of the proposed number system and its encoding, since so many of the design choices were interconnected. These challenges ultimately addressed the proposal as a whole and, quite naturally, tended to drift to its points of least resistance. Thus it was possible for gradual underflow – one of the system's less compelling features – to become its most contentious.[20]

There was no wholly compelling way in which one scheme could be proved superior to the other. Proponents of the Kahan-Coonen-Stone scheme could point to anomalous calculations caused, they argued, by flush-to-zero underflow, anomalies that would be corrected by gradual underflow:

Consider the simple computation

$$(Y-X)+X$$

where $Y-X$ underflows. Then gradual underflow always returns Y exactly, flush to zero returns X . . . . We could look at this as another isolated example, but I prefer to look at it as the preservation of the associative law of addition to within rounding error. That is, under gradual underflow we always have

$$(Y-X)+X = Y+(-X+X)$$

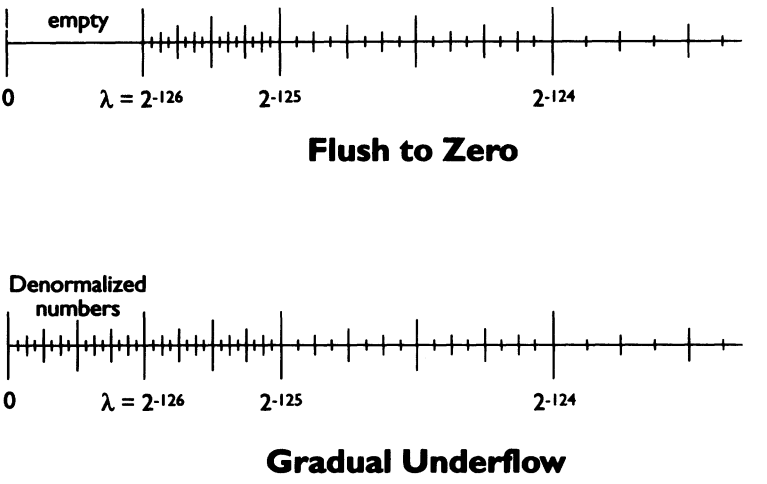
to within rounding error. This is compelling, in my opinion.[21]

The defenders of the more traditional DEC scheme could, however, also point to potential problems with gradual underflow:

Multiplication of denormalized numbers by numbers greater than 1 (or division by numbers less than 1) can generate significant inaccuracies. If such a product (or quotient) is in the ordinary range of numbers, it cannot be represented in denormalized form, because of the hidden bit used in KCS [Kahan-Coonen-Stone arithmetic]. However, the denormalized operand has fewer (perhaps many fewer) than the prescribed number of bits for its level of precision. Thus the product (or quotient) could in the worst case contain only one valid bit. KCS specifies two modes to deal with this problem. 'Warning mode' is mandatory: the invalid flag is set, and a symbol NaN (Not a Number) is stored for the result . . . . The other mode, 'normalize', is optional: if present, and selected, the possibly very inaccurate product is stored as an ordinary number, no flag is set, and, of course, further tracking of the effect of the original underflow is impossible.[22]

As of this time, an illegal operation exception is raised when a denormalized number is multiplied by a value 2 or greater. But on closer inspection, there are denormalized numbers which lie close to the normalized range which exhibit more erratic behavior. The denormalized number $(3/4)\times 2^{-126}$, for example, will generate an invalid operation exception when multiplied by 5, but not when multiplied by 6. When multiplied by 8 an exception will again be generated . . . This effect is caused because the exception for the multiplication occurs when attempting to store an unnormalized number into a basic format. When multiplying by $8 = 1\times 2^3$, the result is $(3/4)\times 2^{-123}$, which is unnormalized. But multiplication by $6 = (3/2)\times 2^2$ gives $(9/8)\times 2^{-124}$, which is normalized.[23]

FIGURE 3
**Small Numbers in Flush to Zero and Gradual Underflow**




*Source*: Based upon Jerome T. Coonen, 'Underflow and the Denormalized Numbers', *Computer* (March 1981), 75–87, at 77.

These objections could be dismissed in their turn:

> Like any new tool, it is possible to misuse this facility and to have a malfunction .... I do not believe that the facility introduces malfunctions into processes that previously worked [with flush to zero].[24]

The proneness of the two arithmetics to generating errors and anomalous calculations was not the only issue to be considered. There was, for example, little doubt that gradual underflow was more complicated to implement than flush-to-zero underflow; it would therefore have costs both in money and perhaps in the time taken by arithmetic operations such as multiplication. It might make the proposed standard harder to police, since, given its complication, manufacturers might choose to implement it in software rather than (demonstrably present or absent) hardware.[25] Finally, DEC's scheme simply had the enormous advantage of essentially being that already employed in the world's most popular scientific minicomputers.

Nothing abstract therefore guaranteed that the Kahan-Coonen-Stone scheme would win: in Professor Kahan's words, 'it was not a foregone conclusion'.[26] In its favour were the composition of the

working group, the facts of geography, and its status as the group's original working document. Mary Payne, of the Massachusetts-based DEC, commented:

> The active (and voting) membership of the Working Group is largely from minicomputer and semiconductor manufacturers, Academia, and purveyors of portable software. There is virtually no representation from Mainframe manufacturers and 'ordinary users' – people writing their own programs for their own (or their employers') use. Most of the active membership is from the San Francisco Bay area, and all but one of the meetings have been in this area.[27]

Kahan, on the other hand, believes that his victory is attributable partly to mishandling by DEC's representatives of what he concedes to be their not unreasonable case. They issued what Kahan calls a 'premature rebuttal' of the arithmetic of him and his colleagues, a rebuttal arguing that 'it can't be done, and if it could be done, it would be bad for you'. Unknown to them, gradual underflow had already been implemented on the prototype i8087. According to Kahan, the DEC representatives retreated to the argument that, while gradual underflow could be implemented in the microcode software of a chip like the i8087, it was impossible to implement in the hardware of a minicomputer like the DEC VAX. However, one of Kahan's students, George Taylor, designed a processor board for the VAX that was acknowledged as successfully performing gradual underflow. DEC then employed a well known error analyst, Professor G. W. Stewart, III, of the University of Maryland, to investigate gradual underflow, hoping, according to Kahan, that he would refute it. Stewart, however, gave qualified support to gradual underflow in a verbal report to the committee, and DEC would not release the written form of his report.

In a spring 1980 ballot of the committee, the Kahan-Coonen-Stone scheme received the necessary two-thirds majority support for adoption. The scheme took several more years to pass through higher committees of the Institute of Electrical and Electronics Engineers, but it was finally approved by the IEEE Standards Board in March 1985, and by the American National Standards Institute in July 1985, as ANSI/IEEE Standard 754.

It is not a universal standard. Most supercomputers (such as Crays), mainframes (such as IBM machines) and minicomputers (such as DEC VAXs) are not claimed to comply with it. However, no competing collective standard has been agreed. So a virtuous cycle

exists: as the IEEE 754 Standard becomes more popular, the problems involved in moving numerical programs from one machine to another have diminished, and more and more software is thus written with the 754 Standard in mind, so increasing its popularity. The proponents of new technologies adopt the 754 Standard so that users do not have to rewrite programs to move to these new technologies.[28]

What has happened is thus a version of a form of 'closure' typical within technology. In the words of Brian Arthur:

> Very often, technologies show increasing returns to adoption – the more they are adopted the more they are improved . . . When two or more increasing-returns technologies compete for adopters, insignificant 'chance' events may give one of the technologies an initial adoption advantage. Then more experience is gained with the technology and so it improves; it is then further adopted, and in turn it further improves. Thus the technology that by 'chance' gets off to a good start may eventually 'corner the market' of potential adopters, with the other technologies gradually being shut out.[29]

There are those who deny that what has been institutionalized is the best possible computer arithmetic,[30] and who would indeed attribute the standard's adoption to 'chance events' rather than the intrinsic merits its proponents would claim. That dispute, however, is now in a sense irrelevant: the very process of the institutionalization of the standard gives it practical advantages which make it unlikely that a competitor will be able to overturn it.

## Constructing Proof

Traditionally, such confidence as we have in computer systems has been empirically based. The design of a piece of hardware such as a microprocessor is checked by subjecting it to a large variety of test conditions (often in advance of physical construction, by simulating the design on a computer). As programs are tested and used, errors are detected and corrected, until a sufficiently large amount of error-free running is achieved to feel confident that the program is essentially correct.

It is widely argued that there are limitations to this approach. Corrections to old errors may introduce new ones, and users of large systems often have to accept a relatively stable rate of continuing error. Unless it is exhaustive, testing can show the presence of 'bugs', but cannot demonstrate conclusively their absence; and exhaustive testing, for all possible combinations of inputs and internal states, is

widely held to be impossible in practice for a design or program of any complexity.

These arguments have been deployed to justify a deductive rather than inductive approach to the correctness of designs and programs, at least for those cases where failure would be catastrophic. In 1986, for example, the UK Cabinet Office's Advisory Council for Applied Research and Development argued that the *post hoc* detection of errors was not good enough in applications such as 'industrial process control, nuclear reactors, weapon systems, station-keeping of ships close to oil rigs, aero engines and railway signalling'. For 'disaster-level' software, where failure 'could involve more than ten deaths', the Advisory Council report argued that 'the whole of the software must be checked by formal mathematical proof, which is itself checked by a competent mathematician'.[31]

Among the organizations most influenced by the demand for mathematical proof of correctness has been the UK Ministry of Defence. In April 1991, the ministry issued Interim Defence Standard 00-55, governing 'The Procurement of Safety Critical Software in Defence Equipment'. This demands formal mathematical proof that the most crucial programs correctly implement their specifications.[32]

The demand for proof is not limited to safety-critical aspects of computer systems. Proof has been sought, especially in the United States, that computer systems crucial to national security are indeed designed in such a way as to protect the classified information they contain. In the early 1980s, the US Department of Defense set out its Trusted Computer System Evaluation Criteria, known from the colour of the cover of the document containing them as the 'Orange Book'. The Orange Book criteria demand that in Class A1 ('Verified Design') systems 'a formal model of the security policy must be clearly identified and documented, including a mathematical proof that the model is consistent with its axioms and is sufficient to support the security policy'.[33]

In such documents, with the exception of the most recent (Defence Standard 00-55, discussed below), the notion of 'proof' has typically been used unproblematically. Five years ago, colleagues and I speculated that this unproblematic usage would not survive the entry of proof into the commercial and regulatory domains. We predicted that it might not be long before a 'court of law has to decide what constitutes a mathematical proof procedure'.[34] Our basis for this prediction was the considerable variation, revealed by the history of

mathematics, in the forms of argument that are taken as constituting proofs. Thus Judith Grabiner has shown how arguments that satisfied eighteenth-century mathematicians were rejected as not constituting proofs by their nineteenth-century successors, such as Cauchy.[35] Our prediction rested on the assumption that attempts to prove the correctness of computer systems would bring to light similar disagreement about the nature of proof.

That has turned out to be the case. In the mid and late 1980s, a team of researchers from the UK Ministry of Defence's Royal Signals and Radar Establishment developed a novel microprocessor called VIPER – Verifiable Integrated Processor for Enhanced Reliability. Though VIPER has several other features designed to make it safe (such as simply stopping if it encounters an error state), what was crucial about it was the claimed existence of a mathematical proof of the correctness of its design. VIPER was marketed as 'the first commercially available microprocessor with a proven correct design'.[36]

The claim of proof has been controversial. There has been sharp disagreement over whether the chain of reasoning connecting VIPER's design to its specification can legitimately be called a 'proof'. In January 1991, Charter Technologies Ltd., a small English firm which licensed aspects of the VIPER technology from the Ministry of Defence, began legal action against the Ministry in the High Court.[37] Charter alleged, among other things, that the claim of proof was a misrepresentation, and sought damages under the 1967 Misrepresentation Act. The Ministry vigorously contested Charter's allegations. Charter went into liquidation before the case could come to court, so our prediction, as quoted above, has not come true literally. Nevertheless, the controversy surrounding VIPER, and the aborted litigation, reveal some of the scope for dispute over proof.

The development of VIPER and the construction of its controversial proof are discussed elsewhere.[38] The core of the dispute over the claim of proof is as follows. The critics of the claim of proof – who include Cambridge University computer scientist Avra Cohn, who worked on the proof, and Bishop Brock and Warren Hunt of the Austin, Texas, firm commissioned by NASA to evaluate it – use a definition of formal proof best summarized by Brock and Hunt's colleagues Robert Boyer and J. Strother Moore:

> A formal mathematical proof is a finite sequence of formulas, each element of which is either an axiom or the result of applying one of a fixed set of mechanical rules to previous formulas in the sequence.[39]

By that criterion, there was only a partial proof of the correctness of VIPER's design. This was constructed by Cohn, on contract to the Royal Signals and Radar Establishment, using HOL (Higher Order Logic), an automated system for proof construction, developed by her colleague Mike Gordon. Even though large – her main proof consists of a sequence of over seven million formulae – this work did not encompass all the steps between the top level specification of VIPER's behaviour and the logic-gate level description used to control the automated equipment employed to construct the 'masks' needed to fabricate VIPER chips. (Although work on the VIPER proof is continuing, there is still – to this author's knowledge – no full formal proof, in the above sense, encompassing all these steps.)

Cohn therefore wrote in 1989: 'no formal proofs of Viper (to the author's knowledge) have thus far been obtained at or near the gate level'. Brock and Hunt, likewise, concluded that 'VIPER has been verified in the traditional hardware engineering sense, i.e. extensively simulated and informally checked' but not 'formally verified'.[40]

One can only speculate about precisely how the claim of proof for VIPER would have been defended, if the case had come to court. The ministry's defence is a confidential document. The one published response (known to this author) by a member of the VIPER team to criticism of the claim of proof does not attempt a rebuttal,[41] and, in any case, the defendant in the law suit was the ministry rather than the individual team members, so the line of argument adopted might, therefore, not have been theirs. Nevertheless, it seems clear that a defence of the claim of proof for VIPER would have had to involve challenging the notion of proof underpinning the criticism, so that mathematical arguments not conforming to the model summarized by Boyer and Moore can count as proofs. That would permit gaps in the formal proof to be 'plugged' by other mathematical arguments, so that the entire chain of reasoning could still be defended as a proof.

An attack on the formal notion of proof was indeed the basis of the defence of VIPER mounted, after the litigation halted, by Martyn Thomas, head of the software house Praxis:

> We must beware 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 in 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, we waste a useful word, and we are in danger of overselling the results of their activities.[42]

Thomas's reference is, of course, to the leading formalist mathematician, David Hilbert (1862–1943). Boyer and Moore's definition of proof is in most (though not all) respects similar to Hilbert's.[43]

These competing arguments, as they bear upon the VIPER proof, were never tested in law, while the shadow of litigation has inhibited open discussion of them in the scientific community. However, the reference to Hilbert points us to a wider intellectual context of the dispute. The formalist tradition within mathematics spearheaded by Hilbert sought to break the connection between mathematical symbols and their physical or mental referents. Symbols were marks upon paper devoid of intrinsic meaning.[44] Proofs were constructed by manipulating these symbols according to the rules of transformation of formal logic, rules which took a precise, 'mechanical', form.[45]

However, despite formalism's very considerable influence within mathematics, not all mathematical proofs take this form. Even if it is believed that all valid proofs *could* be translated into such sequences of formulae, many proofs within mathematics are shorter, more 'high-level', more 'informal'. Part of the reason is the sheer tedium of producing formal proofs, and their length; this is also a large part of the attraction of automatic or semi-automatic proof generating systems, such as HOL or the theorem prover developed by Boyer and Moore.

The relatively informal nature of much mathematical proof was a resource for the defence of the claim of proof for VIPER, as we have seen in the quotation from Thomas. It was also the basis for a widely debated general attack on formal verification of programs, a 1979 paper by Richard DeMillo of the Georgia Institute of Technology and Richard Lipton and Alan Perlis of the Yale Department of Computer Science. Proofs of theorems in mathematics and formal verifications of computer programs were radically different entities, they argued:

> A proof is not a beautiful abstract object with an independent existence. No mathematician grasps a proof, sits back, and sighs happily at the knowledge that he can now be certain of the truth of his theorem. He runs out into the hall and looks for someone to listen to it. He bursts into a colleague's office and commandeers the blackboard. He throws aside his scheduled topic and regales a seminar with his new idea. He drags his graduate students away from their dissertations to listen. He gets onto the phone and tells his colleagues in Texas and Toronto . . . .
>
> After enough internalization, enough transformation, enough generalization, enough use, and enough connection, the mathematical community eventually decides that the central concepts in the original theorem, now perhaps greatly changed, have an ultimate stability. If the various proofs feel right and the results

> are examined from enough angles, then the truth of the theorem is eventually considered to be established. The theorem is thought to be true in the classical sense – that is, in the sense that it could be demonstrated by formal deductive logic, although for almost all theorems no such deduction ever took place or ever will . . . .
>
> Mathematical proofs increase our confidence in the truth of mathematical statements only after they have been subjected to the social mechanisms of the mathematical community. These same mechanisms doom the so-called proofs of software, the long formal verifications that correspond, not to the working mathematical proof, but to the imaginary logical structure that the mathematician conjures up to describe his feeling of belief. Verifications are not messages; a person who ran out into the hall to communicate his latest verification would rapidly find himself a social pariah. Verifications cannot readily be read; a reader can flay himself through one of the shorter ones by dint of heroic effort, but that's not reading. Being unreadable and – literally – unspeakable, verifications cannot be internalized, transformed, generalized, used, connected to other disciplines, and eventually incorporated into a community consciousness. They cannot acquire credibility gradually, as a mathematical theorem does; one either believes them blindly, as a pure act of faith, or not at all.[46]

This paper by DeMillo, Lipton and Perlis provoked sharp criticism from defenders of the evolving practice of program verification. One wrote: 'I am one of those "classicists" who believe that a theorem either can or cannot be derived from a set of axioms. I don't believe that the correctness of a theorem is to be decided by a general election'.[47] Edsger Dijkstra, one of the leaders of the movement to mathematicize computer science, described the paper as a 'political pamphlet from the middle ages'. Interestingly, though, his defence was of short, elegant, human (rather than machine) proofs of programs. He accepted that 'communication between mathematicians is an essential ingredient of our mathematical culture', and conceded that 'long formal proofs are unconvincing'.[48] Elsewhere, Dijkstra had written:

> To the idea that proofs are so boring that we cannot rely upon them unless they are checked mechanically I have nearly philosophical objections, for I consider mathematical proofs as a reflection of my understanding and 'understanding' is something we cannot delegate, either to another person or to a machine.[49]

At least three positions thus contended in the debate sparked by DeMillo, Lipton and Perlis: the formal, mechanized verification of programs and hardware designs; the denial that verification confers certainty akin to that conferred by proof in mathematics; and the advocacy of human, rather than machine, proof. No wholly definitive closure of the debate within computer science was reached, and the

validity of the analogy between proofs in mathematics and formal verification of computer systems remains controversial.[50]

Within mathematics, too, the status of computer-supported proofs has been the subject of controversy – controversy that crystallized most clearly around the 1976 computer-based proof by Appel and Haken of the four-colour conjecture. The developers of this proof summarized at least some of the objections and their defence thus:

> Most mathematicians who were educated prior to the development of fast computers tend not to think of the computer as a routine tool to be used in conjunction with other older and more theoretical tools in advancing mathematical knowledge. Thus they intuitively feel that if an argument contains parts that are not verifiable by hand calculation it is on rather insecure ground. There is a tendency to feel that the verification of computer results by independent computer programs is not as certain to be correct as independent hand checking of the proof of theorems proved in the standard way.
>
> This point of view is reasonable for those theorems whose proofs are of moderate length and highly theoretical. When proofs are long and highly computational, it may be argued that even when hand checking is possible, the probability of human error is considerably higher than that of machine error.[51]

Although the general issue of the status of computer-generated formal proofs remains a matter of dispute, there are signs that at the level of the setting of standards for safety-critical and security-critical computer systems the dispute is being won in practice by the proponents of formal verification. The demand for verification in the Orange Book represented a victory for this position, albeit a controversial one, since there has been criticism both of the model of 'security' underlying the Orange Book, and of the procedures for certification according to Orange Book criteria.[52] Nor did the Orange Book directly address the question of the nature of proof.

Most recently, however, Defence Standard 00-55, representing official policy of the UK Ministry of Defence, has done so, explicitly tackling the issue of the relative status of different forms of mathematical argument. It differentiates between 'Formal Proof' and 'Rigorous Argument'.

> A Formal Proof is a strictly well-formed sequence of logical formulae such that each one is entailed from formulae appearing earlier in the sequence or as instances of axioms of the logical theory . . .
>
> A Rigorous Argument is at the level of a mathematical argument in the scientific literature that will be subjected to peer review . . . .[53]

According to the ministry, formal proof is to be preferred to rigorous argument:

> Creation of [formal] proofs will . . . consume a considerable amount of the time of skilled staff. The Standard therefore also envisages a lower level of design assurance; this level is known as a Rigorous Argument. A Rigorous Argument is not a Formal Proof and is no substitute for it . . . .[54]

It remains uncertain to what degree software industry practices will be influenced by Defence Standard 00-55, and similar standards for other sectors that may follow – a procedure for granting exceptions to 00-55's stringent demands is embodied in the document. Formal proofs of 'real world' programs or hardware designs are still relatively rare.

If they do indeed become more common, I would predict that a further level of dispute and litigation will emerge. This will concern, not the overall status of computer-generated formal proofs (though that issue will surely be returned to), but an issue that has not hitherto sparked controversy: the internal structure of formal proofs. Even if all are agreed that proofs should consist of the manipulation of formulae according to 'mechanical' rules of logic, it does not follow that all will agree on what these rules should be. The histories of mathematical proof and formal logic reveal the scope for significant disagreement.

The best-known dispute concerns the law of the excluded middle (which asserts that either a proposition or its negation must be true). Formalists like Hilbert did not regard proofs relying on excluded middle as problematic; 'constructivists' and 'intuitionists', notably L.E.J. Brouwer, refused to employ it, at least where infinite sets were concerned.[55]

Other examples are the Lewis principles, named after the logician Clarence Irving Lewis.[56] These are that a contradiction implies any proposition, and that a tautology is implied by any proposition. They follow from intuitively appealing axiomatizations of formal logic, yet have seemed to some to be dubious. Is it sensible, for example, to infer, as the first Lewis principle permits, that 'the moon is made from green cheese' follows from 'John is a man and John is not a man'? In the words of one text:

> Different people react in different ways to the Lewis principles. For some they are welcome guests, whilst for others they are strange and suspect. For some, it is no more objectionable in logic to say that a [contradiction] implies all formulae than it

> is in arithmetic to say that $x^0$ always equals 1 . . . For others, however, the Lewis principles are quite unacceptable because the antecedent formula may have 'nothing to do with' the consequent formula.[57]

Critics have to face the problem that any logical system which gives up the Lewis principles appears to have to give up at least one, more basic, 'intuitively obvious', logical axiom.

These controversial rules of logic are to be found in systems upon which formal proof of programs and hardware depends. The law of the excluded middle is widely used in automated theorem proof – for example, in the HOL system used for the VIPER formal proof. The first Lewis principle – that a contradiction implies any proposition – is, for example, among the basic inference rules of the influential Vienna Development Method.[58]

To date, these rules have not provoked within computer science the kind of controversy that has surrounded them in metamathematics and formal logic. There has been some intellectual skirmishing between the proponents of 'classical' theorem provers, which employ the law of the excluded middle, and 'constructivist' ones, which do not.[59] That skirmishing has not, to date, taken the form of entrenched philosophical dispute, and, to this author's knowledge, no computer-system proof has been objected to because of its reliance on say, the excluded middle, or the Lewis principles. Pragmatic considerations – getting systems to 'work', choosing logics appropriate to particular contexts – have outweighed wider philosophical issues.

Can we assume, however, that a situation of pragmatism and peaceful coexistence between different logical systems will continue? My feeling is that we cannot – that this situation is a product of the experimental, academic phase of the development of proof of computer-system correctness. As formal proofs become of greater commercial and regulatory significance, powerful interests will develop in the defence of, or criticism of, particular proofs. Sometimes, at least, these interests will conflict. In such a situation, the validity of rules of formal logic will inevitably be drawn into the fray, and into the law courts.

## Conclusion

There is an important difference between the cases of computer floating-point arithmetic and the proof of computer systems. In the

former, there was a stable, consensual human arithmetic against which computer arithmetic could often be judged. Human arithmetic was, however, insufficient to *determine* the best form of computer arithmetic. It was indeed a matter of judgement which was best, and contested judgement at that. Human arithmetic provided a resource, drawn on differently by different participants, rather than a set of rules that could simply be applied in computer arithmetic. There is even tentative evidence that social interests, notably the different interests of the Intel and DEC corporations, influenced the judgements made. Similarly, the outcome – 'closure' in favour of the Kahan-Coonen-Stone arithmetic scheme – may have been influenced by contingent factors such as the proximity of the meetings of the relevant committee to Silicon Valley, home to Intel and other semiconductor firms, and to Kahan's Berkeley base.

In the case of the proof of computer systems, pre-existing practices of proof, within mathematics, have been less compelling. The reputation of mathematics for precision and certainty has been an important rhetorical resource for those who sought to move from an empirical to a deductive approach to computer-system correctness. However, critics have argued that proof of computer-system correctness and proof of a mathematical theorem are different in kind.

Already, one dispute over the mathematical proof of a computer system has reached the stage of litigation: the controversy concerning the VIPER microprocessor. The prediction of this paper is that the VIPER case will not be unique. Nor will it be sufficient to reach consensus on the general form to be taken by proofs – for example, to demand that they take the form of finite sequences of symbol-manipulations performed according to the transformation rules of a logical system. For if the position adopted in this paper is correct, that will simply drive dispute 'downwards' from the status of general types of argument to the validity of particular steps in those arguments. Specifically, dispute is to be expected over the logical systems that underpin formal proofs.

Formal proof of computer-system correctness is, therefore, an interesting test case for the sociology of knowledge. For this prediction is contrary to our ordinary intuitions about mathematical certainty. It concerns not informal or semiformal mathematics of the sort that has to date provided most of the empirical material for the sociology of mathematics, but mathematical deduction of the most formal kind: precisely the kind of reasoning which, we might imagine, must simply compel consent. As computer-system proof grows in

significance and moves into the commercial and regulatory worlds, we will have a chance to see whether our ordinary intuitions about mathematics, or the conclusions of the sociology of mathematical knowledge, are correct.

## •NOTES

The research on floating-point arithmetic and on VIPER was supported by the UK Economic and Social Research Council (ESRC) under the Programme on Information and Communication Technologies, grants A35250006 and WA35250006. Current work on the topic of the second part of the paper is being supported by a further grant from the ESRC on 'Studies in the Sociology of Proof' (R000234031).

1. H.M. Collins, *Artificial Experts: Social Knowledge and Intelligent Machines* (Cambridge, MA: MIT Press, 1990), Chapter 5.

2. E. Peláez, J. Fleck and D. MacKenzie, 'Social Research on Software', paper read to National Workshop of Programme in Information and Communication Technologies (Manchester, 16–18 December 1987), 5.

3. K. Mannheim, *Ideology and Utopia* (London: Routledge & Kegan Paul, 1936); see D. Bloor, 'Wittgenstein and Mannheim on the Sociology of Mathematics', *Studies in the History and Philosophy of Science*, Vol. 4 (1973), 173–91.

4. This latter convention is known as 'twos complement arithmetic', since if, in this format, 'the sign is treated as if it were simply another digit, negative numbers are represented by their complements with respect to 2': Robert F. Shaw, 'Arithmetic Operation in a Binary Computer', *Review of Scientific Instrumentation*, Vol. 21 (1950), 687–93, at 687–88. This and other important papers in the area are reprinted in Earl E. Schwartzlander, Jr, *Computer Arithmetic* (Stroudsburg, PA: Dowden, Hutchinson & Ross, 1980).

5. In common with other computer arithmetic, variants on this basic format are permitted – for example, a 64-bit 'double precision' mode. For the sake of simplicity, this issue is ignored in the text.

6. The exponent is typically stored not in its natural binary representation, which would have to include a bit to represent its sign, but in a 'biased' form without a sign.

7. I. Lakatos, *Proofs and Refutations: The Logic of Mathematical Discovery* (Cambridge: Cambridge University Press, 1976). Drawing on the work of Mary Douglas, David Bloor has suggested that different social circumstances may generate different typical reactions to anomalies. Unfortunately, evaluating this suggestion for the case under consideration would demand data which I do not possess. See D. Bloor, 'Polyhedra and the Abominations of Leviticus', *British Journal for the History of Science*, Vol. 11 (1978), 245–72, and M. Douglas, *Natural Symbols: Explorations in Cosmology* (London: Barrie & Rockcliff, 1970).

8. Interview with Professor William Kahan, Berkeley, CA, 19 October 1989; Lakatos, op. cit. note 7. In 'forward error analysis', an upper bound is calculated for the accumulated error at each step, and thus eventually for the error in the final result.

Backward analysis, on the other hand, 'starts with the computed solution and works backward to reconstruct the perturbed problem of which it is an exact solution'. See P.J.L. Wallis, *Improving Floating-Point Programming* (Chichester, Sussex: Wiley, 1990), 12. For some remarks on the history of error analysis, see J.H. Wilkinson, 'Modern Error Analysis', *SIAM Review*, Vol. 13 (1971), 548–68.

9. The term is used in W. Kahan, 'Doubled-Precision IEEE Standard 754 Floating-Point Arithmetic' (typescript, 26 February 1987), 14.

10. Lakatos, op. cit. note 7; D. Bloor, *Wittgenstein: A Social Theory of Knowledge* (London: Macmillan, 1983), 141–42.

11. Kahan interview, op. cit. note 8.

12. Palmer was later to play an important role in the development of parallel computing in the US, when he and others left Intel in 1983 to establish the firm NCube.

13. Kahan interview, op. cit. note 8; J. Palmer, 'The Intel Standard for Floating-Point Arithmetic', *Proceedings of IEEE COMPSAC 1977*, 107–12, at 107.

14. Letter to author from Robert G. Stewart, 18 October 1990.

15. *IEEE Standard for Binary Floating-Point Arithmetic: American National Standards Institute/Institute of Electrical and Electronics Engineers Standard 754-1985* (New York: Institute of Electrical and Electronics Engineers, 12 August 1985).

16. Kahan interview, op. cit. note 8.

17. Intel released the i8087 in 1980, in anticipation of the Standard. For the i8087's arithmetic, see J.F. Palmer and S.P. Morse, *The 8087 Primer* (New York: Wiley, 1984), 7. A third seriously considered proposal was drafted by Robert Fraley and J. Stephen Walther, but the main dispute seems to have been between the DEC and Kahan-Coonen-Stone proposals. The three proposals are compared in W.J. Cody, 'Analysis of Proposals for the Floating-Point Standard', *Computer* (March 1981), 63–66.

18. *IEEE Standard*, op. cit. note 15, 14. The argument for two zeros is that they facilitate the removal of otherwise troublesome singularities in computations common, for example, in the study of fluid flow. With two zeros, 'identities are conserved that would not otherwise be conserved', and capabilities not present in conventional mathematics are provided. An example from the complex numbers concerns the square root of $-1$, which conventionally can have two values, $+i$ and $-i$. Signed zeros make it possible to remove this discontinuity, with the square root of $-1 + 0i$ being $+i$, and the square root of $-1 - 0i$ being $-i$ (Kahan interview, op. cit. note 8).

19. For gradual underflow, see I.B. Goldberg, '27 Bits are not enough for 8-Digit Accuracy', *Communications of the ACM*, Vol. 10 (1967), 105–08. Computer pioneer Konrad Zuse was an early advocate, according to Cody, op. cit. note 17, 63.

20. Jerome T. Coonen, 'Underflow and the Denormalized Numbers', *Computer* (March 1981), 75–87, at 75.

21. Cody, op. cit. note 17, 67.

22. Mary Payne and Dileep Bhandarkar, 'VAX Floating Point: A Solid Foundation for Numerical Computation', *Computer Architecture News*, Vol. 8, No. 4 (June 1980), 22–33, at 28–29.

23. Bob Fraley and Steve Walther, 'Proposal to Eliminate Denormalized Numbers', *ACM Signum Newsletter* (October 1979), 22–23, at 22.

24. Cody, op. cit. note 17, 67.

25. I am drawing these points from my interview with Kahan, op. cit. note 8.

26. Ibid.

27. Mary H. Payne, 'Floating Point Standardization', *COMPCON Proceedings* (Fall 1979), 166–69, at 169.

28. Thus compliance with the standard is spreading into the field of dedicated Digital Signal Processors (DSPs). One executive in the field writes: 'Not being IEEE-compatible turns off users who want to replace an application running on an array processor with a board containing a 32-bit floating-point DSP chip . . . Most array processors are IEEE-compatible': quoted in Jonah McLeod, 'DSP, 32-bit Floating-Point: The Birth Pangs of a New Generation', *Electronics* (April 1989), 71–74, at 73.

29. W. Brian Arthur, 'Competing Technologies and Economic Prediction', *Options* (April 1984), 10–13, at 10.

30. Interview with Chuck Purcell, Minneapolis, 4 April 1990.

31. Cabinet Office, Advisory Council for Applied Research and Development, *Software: A Vital Key to UK Competitiveness* (London: HMSO, 1986), 78, 83.

32. Ministry of Defence, *Interim Defence Standard 00-55: The Procurement of Safety Critical Software in Defence Equipment* (Glasgow: Ministry of Defence Directorate of Standardization, 5 April 1991).

33. Department of Defense, *Trusted Computer System Evaluation Criteria* (Washington, DC: Department of Defense, December 1985, DOD 5200.28-STD).

34. Peláez, Fleck & MacKenzie, op. cit. note 2, 5.

35. J.V. Grabiner, 'Is Mathematical Truth Time-Dependent?', *American Mathematical Monthly*, Vol. 81 (1974), 354–65.

36. N.H. Hughes, foreword to Charter Technologies Ltd, *VIPER Microprocessor Development Tools* (Worcester: Charter Technologies Ltd, 1987; unpaginated). 'VIPER' is a registered trademark of the UK Ministry of Defence.

37. High Court of Justice, Queen's Bench Division, 1991 C No. 691.

38. D. MacKenzie, 'The Fangs of the VIPER', *Nature*, Vol. 352 (8 August 1991), 467–69.

39. R.S. Boyer and J.S. Moore, 'Proof Checking the RSA Public Key Encryption Algorithm', *American Mathematical Monthly*, Vol. 91 (1984), 181–89, at 181.

40. Avra Cohn, 'The Notion of Proof in Hardware Verification', *Journal of Automated Reasoning*, Vol. 5 (1989), 127–39, at 135; Bishop Brock and Warren A. Hunt, Jr, *Report on the Formal Specification and Partial Verification of the VIPER Microprocessor* (Austin, TX: Computational Logic, Inc., Technical Report 46, 15 January 1990), 21.

41. J. Kershaw, foreword to Brock & Hunt, ibid.

42. M. Thomas, 'VIPER Lawsuit Withdrawn', electronic mail communication, 5 June 1991.

43. See Hilbert's 1927 address, 'The Foundations of Mathematics', in Jean van Heijenoort, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931* (Cambridge, MA: Harvard University Press, 1967), 464–79. A potentially significant difference, in view of the arguments by DeMillo et al. discussed below, is that for Hilbert the array of formulae constituting a proof 'must be given as such to our perceptual intuition' (ibid., 465).

44. Formalism's most famous opponent, the intuitionist L.E.J. Brouwer, wrote: 'The question where mathematical exactness does exist, is answered differently by the two sides; the intuitionist says: in the human intellect, the formalist says: on paper'. The quotation is from 'Intuitionism and Formalism', the English translation of Brouwer's 1912 Inaugural Address at the University of Amsterdam, in Brouwer, *Collected Works. Vol. 1. Philosophy and Foundations of Mathematics* (Amsterdam: North-Holland, 1975), 123–38, at 125.

45. For an interesting discussion of the significance of the machine analogy in Hilbert's mathematical programme, see Herbert Breger, 'Machines and Mathematical Styles of Thought', paper read to conference on 'Mathematization of Techniques and Technization of Mathematics', Zentrum für interdisziplinäre Forschung, Universität Bielefeld, 3–7 September 1991.

46. R.A. DeMillo, R.J. Lipton and A.J. Perlis, 'Social Processes and Proofs of Theorems and Programs', *Communications of the Association of Computing Machinery*, Vol. 22 (1979), 271–80, at 273–75. The phrase '"social" processes of proof' in the above quotation from Thomas is probably a reference to the argument of this paper.

47. Leslie Lamport of verification specialists SRI International, as quoted in Stuart S. Shapiro, *Computer Software as Technology: An Examination of Technological Development* (unpublished PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1990), 132. Shapiro's thesis contains a useful account of the 1970s debate surrounding the paper by DeMillo, Lipton and Perlis.

48. Edsger W. Dijkstra, 'On a Political Pamphlet from the Middle Ages', *ACM SIGSOFT, Software Engineering Notes*, Vol. 3, Part 2 (April 1978), 14–16, at 14. Dijkstra was commenting on an earlier version of the DeMillo, Lipton and Perlis paper, published in the *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages* (January 1977), 206–14.

49. Edsger W. Dijkstra, 'Formal Techniques and Sizeable Programs', paper prepared for Symposium on the Mathematical Foundations of Computing Science, Gdansk, 1976, as reprinted in Dijkstra, *Selected Writings on Computing: A Personal Perspective* (New York: Springer, 1982), 205–14, at 212–13. For an interesting analysis of Dijkstra's overall position, see Eloína Peláez, *A Gift from Pandora's Box: The Software Crisis* (unpublished PhD thesis, University of Edinburgh, 1988).

50. The most recent focus of controversy was another article denying, on different grounds from those of DeMillo, Lipton and Perlis, the analogy between verification and mathematical proofs: James H. Fetzer, 'Program Verification: The Very Idea', *Communications of the ACM*, Vol. 31 (1988), 1048–63.

51. Kenneth Appel and Wolfgang Haken, as quoted in Philip J. Davis and Reuben Hersh, 'Why should I believe a Computer?', in Davis and Hersh, *The Mathematical Experience* (Brighton, Sussex: Harvester, 1981), 380–87, at 385–86.

52. National Research Council, System Security Study Committee, *Computers at Risk: Safe Computing in the Information Age* (Washington, DC: National Academy Press, 1991).

53. Ministry of Defence, op. cit. note 32, *Part 2. Guidance*, 28. The history of Def Stan 00-55 is reviewed in Margaret Tierney, 'The Evolution of Def Stan 00-55 and 00-56: An Intensification of the "Formal Methods Debate" in the UK', Edinburgh PICT Working Paper No. 30 (Edinburgh: Programme on Information and Communication Technologies, Research Centre for Social Sciences, University of Edinburgh, 1991).

54. Ministry of Defence, op. cit. note 32, *Part 2. Guidance*, 28.

55. See, for example, J. Van Heijenoort (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931* (Cambridge, MA: Harvard University Press, 1967). Some of Brouwer's reasons for doubting an apparently obvious principle can be seen in the following quotation: 'Now consider the principum *tertii exclusi* [law of excluded middle]: It claims that every supposition is either true or false; in mathematics this means that for every supposed imbedding of a system into another, satisfying certain given conditions, we can either accomplish such an imbedding by a construction, or

arrive by a construction at the arrestment of the process which would lead to the imbedding. It follows that the question of the validity of the principum *tertii exclusi* is equivalent to the question whether unsolvable mathematical problems can exist. There is not a shred of a proof for the conviction, which has sometimes been put forward, that there exist no unsolvable mathematical problems . . . in infinite systems the principum *tertii exclusi* is as yet not reliable . . . . So long as this proposition is unproved, it must be considered as uncertain whether problems like the following are solvable: Is there in the decimal expansion of $[\pi]$ a digit which occurs more often than any other one? . . . And it likewise remains uncertain whether the more general mathematical problem: Does the principum *tertii exclusi* hold in mathematics without exception? is solvable . . . . In mathematics it is uncertain whether the whole logic is admissible and it is uncertain whether the problem of its admissibility is decidable': Brouwer, 'The Unreliability of the Logical Principles', in Brouwer, op. cit. note 44, 107–11, at 109–11, emphases deleted.

56. I am drawing here on Bloor, op. cit. note 10, 124–36.

57. D.C. Makinson, *Topics in Modern Logic* (London: Methuen, 1973), 27–28.

58. Cliff B. Jones, *Systematic Software Development Using VDM*, second edition (Hemel Hempstead, Herts.: Prentice Hall, 1990), 24.

59. 'Unless we have a philosophical commitment to intuitionism, maintaining constructiveness when it is not required can only make a proof system more cumbersome to use. We have seen that certain programs cannot be derived from their specifications in a constructive logic, but can be derived in a classical logic upon which minimal restrictions have been imposed . . . ': Zohar Manna (Computer Science Department, Stanford University) and Richard Waldinger (Artificial Intelligence Center, SRI International), 'Constructive Logic Considered Obstructive' (typescript, n.d.), 8.

*Donald MacKenzie* holds a personal chair in sociology at the University of Edinburgh. He is the author of *Statistics in Britain, 1865–1930: The Social Construction of Scientific Knowledge* (Edinburgh University Press, 1981), *Inventing Accuracy: A Historical Sociology of Nuclear Missile Guidance* (MIT Press, 1990), and co-editor of *The Social Shaping of Technology* (Open University Press, 1985).
*Author's address*: Department of Sociology, University of Edinburgh, 18 Buccleuch Place, Edinburgh EH8 9LN, Scotland, UK.