1

Knowing Computers

Of all the technologies bequeathed to us by the twentieth century, one above all saturates our lives and our imaginations: the digital computer. In little more than half a century, the computer has moved from rarity to ubiquity. In the rich, Euro-American, world—and to a growing extent beyond it as well—computers now play an essential part in work, education, travel, communication, leisure, finance, retail, health care, and the preparation and the conduct of war. Increasingly, computing is to be found in devices that do not look like computers as one ordinarily thinks of them: in engines, consumer products, mobile telephones, and in the very fabric of buildings.

The benefits brought by all this computerization are clear and compelling, but it also brings with it dependence. Human safety, the integrity of the financial system, the functioning of utilities and other services, and even national security all depend upon computing. Fittingly, the twentieth century's end was marked both by an upsurge of enthusiasm for computing and by a wave of fear about it: the huge soon-to-be-reversed rise in the prices of the stock of the "dotcom" Internet companies and the widespread worries about the "millennium bug," the Y2K (year 2000) date change problem.

How can we know that the computing upon which we depend is dependable? This is one aspect of a question of some generality: how do we know the properties of artifacts? The academic field of the social studies of science and technology (a diverse set of specialisms that examine the social nature of the content of science and technology and their relations to the wider society) has for several decades been asking how we know the properties of the natural world.[1] The corresponding question for the properties of artifacts is much less often asked; surprisingly so, given that a good part of recent sociological interest in technology derives ultimately from the sociology of scientific knowledge.[2]

Asking the question specifically for computers—how do we know the properties of computers and of the programs that run on them?—is of particular interest because it highlights another issue that sociological work on science has not addressed as much as it might: deductive knowledge.

Sources of knowledge, whether of the properties of the natural world or of those of artifacts, can usefully be classified into three broad categories:

- induction—we learn the properties by observation, experiment, and (especially in the case of artifacts) testing and use;
- authority—people whom we trust tell us what the properties are; and
- deduction—we infer the properties from other beliefs, for example by deducing them from theories or models.[3]

Social studies of science have focused primarily on the first of these processes, induction, and on its relations to the other two: on the dependence of induction upon communal authority[4] and interpersonal trust;[5] and on the interweaving of induction and deductive, theoretical knowledge.[6]

Deduction itself has seldom been the focus of attention: the sociology of mathematics is sparse by comparison with the sociology of the natural sciences; the sociology of formal logic is almost nonexistent.[7] At the core of mathematics and formal logic is deductive proof. That propositions in these fields can be proved, not simply justified empirically, is at the heart of their claim to provide "harder," more secure, knowledge than the natural sciences. Yet deductive proof, for all its consequent centrality, has attracted remarkably little detailed attention from the sociology of science, the work of David Bloor and Eric Livingston aside.[8] In the social studies of science more widely, the single best treatment of proof remains one that is now forty years old, by the philosopher Imre Lakatos in the 1961 Ph.D. thesis that became the book *Proofs and Refutations* (see chapter 4).[9] Indeed, it has often been assumed that there is nothing sociological that can be said about proof, which is ordinarily taken to be an absolute matter. I encountered that presumption more than once at the start of this research. Even the founder of the modern sociology of knowledge, Karl Mannheim, excluded mathematics and logic from the potential scope of its analyses.[10]

Asking how we know the properties of computer systems (of hardware and/or of software) directly raises the question of deductive knowledge

and of proof. An influential current of thinking within computer science has argued that inductive knowledge of computer systems is inadequate, especially in contexts in which those systems are critical to human safety or security. The number of combinations of possible inputs and internal states of a computer system of any complexity is enormously large. In consequence it will seldom be feasible, even with the most highly automated testing, to exercise each and every state of a system to check for errors and the underlying design faults or "bugs" that may have caused them.[11] As computer scientist Edsger W. Dijkstra famously put it in 1969, "Program testing can be used to show the presence of bugs, but never to show their absence!"[12] Even extensive computer use can offer no guarantees, because bugs may lurk for years before becoming manifest as a system failure.

Authority on its own is also a problematic source of knowledge of the properties of computer systems. In complex, modern societies trust is typically not just an interpersonal matter, but a structural, occupational one. Certain occupations, for example, are designated "professions," with formal controls over membership (typically dependent upon passing professional examinations), norms requiring consideration of the public good (not merely of self-interest), mechanisms for the disciplining and possible expulsion of incompetent or deviant members, the capacity to take legal action against outsiders who claim professional status, and so on. Although many computer hardware designers are members of the established profession of electrical and electronic engineering, software development is only partially professionalized. Since the late 1960s, there have been calls for "software engineering" (see chapter 2), but by the 1990s it was still illegal, in forty-eight states of the United States, to describe oneself as a "software engineer," because it was not a recognized engineering profession.[13]

With induction criticized and authority weak, the key alternative or supplementary source of knowledge of the properties of computer systems has therefore often been seen as being deductive proof. Consider the analogy with geometry.[14] A mathematician does not seek to demonstrate the correctness of Pythagoras's theorem[15] by drawing triangles and measuring them: since there are infinitely many possible right-angled triangles, the task would be endless, just as the exhaustive testing of a computer system is, effectively, infeasible. Instead, the mathematician seeks a proof: an argument that demonstrates that the theorem must hold in all cases. If one could, first, construct a faithful mathematical representation of what a program or hardware design was intended to

do (in the terminology of computing, construct a "formal specification"), and, second, construct a mathematical representation of the actual program or hardware design, then perhaps one could prove deductively that the program or design was a correct implementation of its specification. It would not be a proof that the program or design was in an absolute sense "correct" or "dependable," because the specification might not capture what was required for safety, security, or some other desired property, and because an actual physical computer system might not behave in accordance with even the most detailed mathematical model of it. All of those involved in the field would acknowledge that these are questions beyond the scope of purely deductive reasoning.[16] Nevertheless, "formal verification" (as the application of deductive proof to programs or to hardware designs is called) has appeared to many within computer science to be potentially a vital way of determining whether specifications have been implemented correctly, and thus a vital source of knowledge of the properties of computer systems.

The Computer and Proof

Formal verification is proof about computers. Closely related, but distinct, is the use of computers in proof. There are at least three motivations for this use. First, proofs about computer hardware designs or about programs are often highly intricate. Many of those who sought such proofs therefore turned to the computer to help conduct them. Mechanized proofs, they reasoned, would be easier, more dependable, and less subject to human wishful thinking than proofs conducted using the traditional mathematicians' tools of pencil and paper. This would be especially true in an area where proofs might be complicated rather than conceptually deep and in which mistakes could have fatal real-world consequences. Second, some mathematicians also found the computer to be a necessary proof tool in pure mathematics. Chapter 4 examines the most famous instance of such use: the 1976 computer-assisted solution of the long unsolved four-color problem: how to prove that four colors suffice to color in any map drawn upon a plane such that countries that share a border are given different colors. A third source of interest in using computers in proofs is artificial intelligence. What is often claimed to be the first artificial-intelligence program was designed to prove theorems in propositional logic (see chapter 3), and automated theorem proving has remained an important technical topic in artificial intelligence ever since. Underneath the more glamorous

applications of artificial intelligence, such as robotics, are often "inference engines" based upon theorem proving.

Although the theorems they prove would not be regarded by mathematicians as interesting or challenging, more sophisticated mechanized provers (far broader in their scope than the ad hoc programs used in the four-color proof) have been used to solve problems that human mathematicians found intractable, such as the 1996 solution of the Robbins problem, an open question in Boolean algebra (see chapter 3). Automated theorem proving, therefore, raises the question of whether there is a sense in which a computer can be an artificial mathematician. To the proponents of artificial intelligence this has been an attractive possibility; to its opponents it is a repugnant one.

Historical Sociology

The topic of this book is the interrelations of computing, risk, and proof. It is neither a technical nor a philosophical treatment of them, but rather a historical sociology. The historical aspect is the more straightforward. What this book seeks to do is to describe salient features of the interrelations between computing and proof as they have evolved from the 1950s onward. Relevant archival material is still sparse, so I have used two main bodies of evidence. First is the technical literature of computer science and of artificial intelligence, which offers the central documentary record of the evolving thinking of the relevant technical communities. Second is an extensive series of "oral history" interviews with the main participants. Interview data must be treated with caution: interviewees' memories are fallible, and they may wish a particular version of events to be accepted. As far as possible, therefore, I have tried to use written sources to document factual assertions, using interviews primarily as evidence of interviewees' opinions and beliefs.

Even in this role, interviews are an imperfect source (interviewees' opinions may change through time in ways they may not wish to acknowledge or of which they may even be unaware). Nevertheless, they add an important dimension to what would otherwise be the overabstract and disembodied history that would result from the use of the technical literature alone. The interviews were conducted under conditions of confidentiality, but they were tape-recorded and transcribed, and intended quotations were sent to interviewees for review and permission to publish.[17] In many cases, points made in interview were then elaborated in further exchanges, especially by electronic mail.

The result, I hope, is accurate narrative history. Nevertheless, it is selective history: I have not attempted comprehensive accounts of the development of the various technical specialisms bearing upon computing, risk, and proof.[18] I have focused upon episodes, issues, and debates that appeared to be particularly interesting from the viewpoint of my twin concerns: the nature of knowledge of the properties of computer systems and the nature of deductive proof. I see these not just as technical matters (though plainly they are that) but as sociological questions, issues for the social studies of science and technology.

Since the social studies of science (less so those of technology) have recently become controversial in the debate called the "science wars,"[19] some preliminary remarks on the sociological aspect of the approach taken here are also necessary. Although some of the diverse strands that make up social studies of science can rightly be classified as criticism of science (not just in the literary, but in the everyday meaning of "criticism"), that is emphatically not true of the kind of historical sociology pursued here. To investigate the variety of meanings of mathematical proof, for example, is simply to inquire, not to denigrate. The fundamental point—almost always missed in "science wars" debates—is that the analysis of science, technology, and mathematics is not a zero-sum game in which the greater the weight of social influence the less the weight of empirical input or other aspects of what used to be called "internal" factors, such as intellectual consistency and rigor.[20] For example, that modern mathematicians do not usually work in isolation, but as members of specialist mathematical communities, deeply affecting each other's work, has surely the typical effect of improving the mathematics they generate. That kind of social influence, which arises from the way in which science and mathematics are conducted within communities, is in no sense in tension with empirical input or with matters such as consistency. In an appropriate community with an appropriate orientation, social processes surely generate rigor rather than undermine it.

Nor should one assume a priori the existence of a zero-sum trade-off when social influence arises from outside the scientific community. In a sense, most of this book is a study of social influence of that second kind, of the effects upon deductive knowledge of the desire to be able reliably to predict the behavior of computer systems upon which human life and security depend. Those effects, I would argue and I think this book demonstrates, have in general been beneficial intellectually as well as practically. Concern for safety or security does not diminish concern for rigor or for intellectual consistency; it increases it.

Another cross-disciplinary matter is raised by the historical sociology of science or technology: technical accuracy. Writing about disciplines that are not one's own is an error-prone activity, and its difficulty is increased when these disciplines are mathematical and one is trying to describe developments in them in a way that is accessible to the non-mathematical reader. The healthiest attitude is to accept that one will make mistakes and to seek the help of one's technical colleagues in eradicating them. Although I am extremely grateful to those who have assisted in this, I have no illusions that what remains will be beyond criticism. I welcome any errors here of any kind being pointed out; I only ask the "science wars" critic not simply to indulge in facile cross-disciplinary point scoring, but to concentrate attention on the main themes and arguments of the book.

Risk, Trust, and the Sociology of Proof

From a sociological viewpoint, the question of one's knowledge of computer-system dependability blends in to a more general issue: risk and trust in the societies of "high modernity." (The term is drawn from the sociologist Anthony Giddens.[21] It is preferable to the more voguish "postmodernity," which tends to exaggerate the discontinuities between the present and the recent past, sometimes on the basis of sloppily impressionistic analyses.) One of the most striking features of the politics of high modernity is the salient place in it of issues of technological risk.

Measured by life expectancy, today's Euro-American world is uniquely safe. We have learned to protect ourselves against natural hazards such as earthquake, flood and storm; famine is for us a distant memory; and epidemic infections (the great killers of premodernity) have largely been eliminated or, at worst, controlled. Patently, however, members of these societies do not always feel safe. The issue of whether the cattle disease BSE (bovine spongiform encephalopathy, or "mad cow disease") could be transmitted to human beings played a significant part in British politics in the 1990s. Debates over the safety of chemical pesticides, of nuclear power, and, most recently, of genetically modified organisms have raged internationally, often involving political direct action and significant impact upon the industries involved.

Among the most influential commentators on the politics of risk in high modernity has been the German sociologist Ulrich Beck.[22] "[S]ooner or later in the continuity of modernization," wrote Beck, "the social positions and conflicts of a 'wealth-distributing' society begin to

be joined by those of a 'risk-distributing' society." Previous forms of societies had their dangers, but the "hazards in those days assaulted the nose or the eyes and were thus perceptible to the senses, while the risks of civilization today typically escape perception." The layperson's senses are no guide to whether BSE poses risks to human beings, or whether genetically modified foodstuffs are safe. On questions such as these, the layperson must turn to others, and in the process risks can "be changed, magnified, dramatized or minimized within knowledge. . . . [T]he mass media and the scientific and legal professions in charge of defining risks become key social and political positions. . . . Knowledge gains a new political significance. Accordingly the political potential of the risk society must be elaborated and analyzed in a sociological theory of the origin and diffusion of knowledge about risks."[23]

Despite the pervasive high-modern concern with technological risk, in only one episode has fear of the computer become widespread: the Y2K problem. In the United States especially there were many predictions and a substantial amount of popular fear that the "millennium bug" would seriously disrupt utilities and other essential parts of the infrastructure of high modernity. These concerns turned out to be misplaced, and in retrospect it is easy to mock them and to question whether the large sums spent on checking for and eliminating the "bug" were justified (worldwide spending is said to have totaled as much as $400 billion, though much of this sum represents system replacement and upgrading desirable on other grounds).[24]

The episode highlights, however, both the dependence of high-modern societies upon computing and the difficulty of forming a judgment on the risks posed by that dependence. However small or large Y2K dangers might have been in the absence of the effort to detect and to correct them, political and business leaders throughout most of the Euro-American world judged the effort essential: not knowing how computer systems would behave was intolerable. The Y2K problem was, furthermore, in a sense an easy case: one in which the possible underlying fault (the vulnerability of two-digit representations of the year to a century change) was clear, even though immense effort was needed to be sure what its effects would be. More typically, those involved in issues of computer-system dependability will be trying to judge the trustworthiness of a system that may be vulnerable to any number of faults, including design faults or "bugs" in its software. Even though this is an issue scarcely ever discussed in the burgeoning literature on "risk society,"[25] it is an archetypal case of the phenomenon described by Beck: an invisible

potential danger, the seriousness of which the layperson's unaided senses cannot judge.

Reliance upon experts is an inevitable aspect of high modernity. Particular experts, including those regarded by the scientific community as the appropriate experts, may be disbelieved (a sense that the public authority of science has declined may well be one factor fueling the passions of the science wars), but, without turning one's back entirely on high modernity, that distrust cannot be generalized to rejection of cognitive authority of all kinds. Disbelief in one set of experts' knowledge claims is often belief in those of others, in those of environmental groups such as Greenpeace, for example, rather than in those of biotechnology companies such as Monsanto. The sociologist Brian Wynne and others have pointed out that lay thinking about scientific and technological matters is often more sophisticated and less ignorant than commonly believed,[26] but one should not exaggerate the possibility of cognitive "direct democracy." In a complex society, engaged in many complex activities, no one can be an expert in everything. Outside the necessarily narrow domains in which one has expertise, one can at best choose wisely in whom or in what to trust and find an appropriate balance between blind faith and self-defeating scepticism.

Among those within the social studies of science who have written most interestingly on the problems of trust are the historians Theodore Porter and Steven Shapin. The question addressed by Porter is how "to account for the prestige and power of quantitative methods in the modern world," not just in science but in areas such as actuarial work, accountancy, cost-benefit analysis, and social policy more generally. Although formal verification in computer science differs in important ways from these applications of quantitative methods, Porter's analysis of them is worth considering because it suggests a trade-off between mathematicization and trust. In traditional communities "face-to-face interactions typically obviated the need for formal structures of objectivity. . . . [U]ntil the eighteenth century, measurements even of land or grain volume were never intended to be purely mechanical, but normally involved an explicit judgment of quality. In a small-scale and unstandardized world, bargaining over measures caused no more inconvenience than bargaining over prices."

As *Gemeinschaft* (community) becomes modernity's *Gesellschaft* (society), however, face-to-face solutions are no longer viable. Modernity's "trust in numbers" is a substitute for absent trust in people, argues Porter: "reliance on numbers and quantitative manipulation minimizes the

need for intimate knowledge and personal trust." Mathematics is suited to this role because it is "highly structured and rule-bound," a "language of rules, the kind of language even a thing as stupid as a computer can use."[27] Porter's analysis nicely captures the vastly increased importance of quantification and of standardization in modernity. (Indeed, one reason for scepticism about broad-brush notions of "post modernity" is that apparent departures from uniformity often rest upon deeper standardization, and quantification's significance has grown rather than declined in recent years.) Furthermore, Porter's is an account wholly compatible with standard sociological analyses of modernity such as that of Anthony Giddens, who sees modernity as involving "disembedding mechanisms" in which trust shifts from local, face-to-face relationships to delocalized, abstract systems. In driving, flying or even simply entering a building, one implicitly trusts that others—automobile engineers, pilots, air traffic controllers, structural engineers, building inspectors—have done or will do their job properly. But this trust is not confidence in specific, individual people: those on whose expertise we depend are frequently strangers to us. Instead, says Giddens, what we trust are "systems of technical accomplishment or professional expertise,"[28] not particular, personally known people.

Steven Shapin, however, offers a perspective subtly different from Porter's or Giddens's. He does not deny that much of the everyday experience of high modernity involves trust in anonymous, abstract "systems of expertise," but he also argues that "within communities of practitioners, for example within the communities of scientific knowledge-producers . . . it is far from obvious that the world of familiarity, face-to-face interaction, and virtue is indeed lost." Gentlemanly codes "linking truth to honor . . . were adapted and transferred to provide substantial practical solutions to problems of credibility in seventeenth-century English science." Nor should one assume, Shapin argues, that they offered only a temporary solution: even the scientists of high modernity "know so much about the natural world by knowing so much about whom they can trust."[29] Although given its sharpest formulation by Shapin, this is a widely-shared conclusion in social studies of science. Porter, for example, acknowledges that, at least within secure, high-status scientific communities, knowledge is still often produced and assessed in informal, nonstandardized ways.[30]

The subjects of Shapin's historical work, however, would not have agreed fully with his conclusion that "no practice has accomplished the rejection of testimony and authority and . . . no cultural practice recog-

nizable as such could do so. . . . In securing our knowledge we rely upon others, and we cannot dispense with that reliance."[31] The early modern scientific thinkers discussed by Shapin were sharply aware of the necessary reliance in the natural sciences upon honest testimony and upon skillful experimentation. Thomas Hobbes, one of the two key protagonists of Shapin and Schaffer's earlier *Leviathan and the Air-Pump,* saw empirical, experimental knowledge as inadequate on those and other grounds. Hobbes saw in mathematics, specifically in geometry, however, a model of reasoning that was in contrast sure, incontrovertible and not dependent upon the dangerous testimony of others. Although Hobbes's opponent, Robert Boyle, had a quite different, much more positive view of the knowledge produced by experiment and by trustworthy testimony, he too noted that "In pure mathematicks, he, that can demonstrate well, may be sure of the truth of a conclusion, without consulting experience about it." Even Boyle, the great founder of the modern "experimental life," thus conceded that mathematics was, in Shapin's words, "the highest grade of knowledge." If one's goal were "uncontroverted certainty and confidence in one's knowledge, the culture of pure mathematics possessed the means to satisfy that goal."[32]

It is, indeed, easy to see why deductive, mathematical knowledge was and is granted a status not enjoyed by empirical, inductive, or testimony-based knowledge. Anyone who has been exposed to mathematical proof, at even an elementary level, knows the feeling of compulsion that proof can induce. Given the premisses, a proof shows that the conclusion must follow, and that a proof compels is, at least apparently, not a matter of trust at all. Paraphrasing Hobbes, but largely expressing his own view as well, Porter comments that geometrical, or more generally mathematical, reasoning is "solid demonstration, which brings its own evidence with it and depends on nothing more than writing on paper."[33]

Is deductive proof therefore an exception to Shapin's claim that "the identification of trustworthy agents is necessary to the constitution of any body of knowledge?"[34] Does deductive proof yield a form of knowledge in which the individual is self-sufficient? The problem with this apparently common-sense conclusion is the difficulty for the isolated individual of distinguishing between being right and believing one is right. Although the point has many ramifications,[35] a basic issue was pointed out nearly three centuries ago by David Hume. "There is no Algebraist nor Mathematician so expert in his science, as to place entire confidence in any truth immediately upon his discovery of it," wrote Hume. "Every time he runs over his proof, his confidence encreases; but

still more by the approbation of his friends; and is rais'd to its utmost perfection by the universal assent and applauses of the learned world."[36] Those who have wanted more than personal, subjective conviction of the correctness of a proof have, traditionally, had to turn to others' assessments of it. To anticipate an argument discussed in chapter 6, the "social processes" of the mathematical community sift putative proofs, thereby achieving a rigor that the individual mathematician cannot know for sure he or she possesses.

Yet the human community is now not the only "trustworthy agent" to which to turn: it has been joined by the machine. The mechanization of proof has produced automated systems that check whether a sequence of formulae expressed in sufficient detail in appropriate formalism has the syntactic structure that makes it a formal proof, in other words that check whether each formula in the sequence is generated from previous formulae by application of the rules of inference of the formal system in question. Some systems, furthermore, even have a limited capability to find such proofs for themselves. Modernity's "trust in numbers" can, it appears, lead back to a grounding not in trust in people, but trust in machines.

That the pursuit of objectivity may lead back literally to an object is a conclusion that some may find reassuring, but others disturbing.[37] As Sherry Turkle pointed out, the computer is an "evocative object." Previously, "animals, . . . seemed our nearest neighbors in the known universe. Computers, with their interactivity, their psychology, with whatever fragments of intelligence they have, now bid for this place." The computer is an "object-to-think-with," in particular to think about what it is to be human.[38] As sociologist of science Harry Collins has pointed out, artificial intelligence is in a sense an experiment in the sociology of knowledge.[39] If genuine "artificial experts" are possible, then, since no current machine (not even a neural network) learns by socialization in the way human experts appear to learn, there may be something wrong with the sociological view of expertise. The relationship between the computer and proof has the practical importance outlined above, but it has this intellectual importance as well. If machines can prove, does it mean that proof is not social?

The mechanization of proof is also of interest because of the light it throws upon nonmechanized deduction. Until the advent of formal, machine-implemented mathematics, there was a single dominant ideal type of deductive proof: proof as conducted by skilled human mathematicians. Although even that ideal type was not unitary—the contrast be-

tween different forms of human proving offers fascinating, if still underexplored, material for the social studies of mathematics[40]—it has now been joined by another ideal type: machine proof. There are now two "cultures of proving."[41] Mostly, they exist independently of each other, in different institutional locations, and only seldom is there overt conflict between them. But the existence of each reveals the contingency of the other.

This book examines the historical sociology of machine proof: ordinary, human-conducted, mathematical proof is discussed only in passing (mainly in chapters 4 and 9). I hope, however, that the contrast between the two forms of proving will lead colleagues in the social studies of science to be more curious about ordinary mathematical proof, indeed about mathematics. *Pace* Porter, mathematics, at least the mathematics of human research mathematicians, is not entirely "the kind of language even a thing as stupid as a computer can use." The mechanization of proof, to date, has largely been the mechanization of philosophy's ideal of formal proof (a sequence of formulae in which each formula is deduced from previous formulae by syntactic, "mechanical," inference rules), not proof as it is ordinarily conducted by human mathematicians. Indirectly, therefore, the mechanization of proof shows the need for the development of an analysis of mathematical proof that is better, as an empirical description of what mathematicians actually do, than philosophy's ideal type. As the concluding chapter suggests, there is reason to think that such an analysis will have to have a strong sociological component.[42]

Synopsis of the Book

Although I have tried to keep this book as nontechnical as possible, some of the passages in the chapters that follow will be demanding for the nonspecialist. The reader, therefore, may wish to skip particular passages or even particular chapters while still following the overall narrative. To facilitate this process, the remainder of this chapter gives a reasonably extensive summary of each of the chapters that follow.

Chapter 2 begins with the issue of dependability. Almost as soon as computers began to be used to control critical systems, the potential for disaster appeared. The chapter documents the main features of an awareness emerging in the 1960s that, despite the enormous strides being made in computer hardware, software production remained slow, expensive, and bug-ridden. This awareness crystallized in October 1968

at a NATO conference held at Garmisch in the Bavarian Alps. A "software crisis" was diagnosed there, and "software engineering" was proposed as its solution.[43] The latter had two broad strands, not always compatible: an emphasis on the "practical disciplines" to be found in other areas of engineering, and an emphasis on the "theoretical foundations" of computer science, especially on its foundations in mathematics and in logic. The chapter discusses the evolution of testing, as an empirical, inductive route to knowledge of the properties of computer systems, but it focuses mainly upon those who sought deductive knowledge of those properties, such as the Dutch computer scientist Edsger W. Dijkstra. Dijkstra hoped that the mountain sunshine of Garmisch represented "the end of the Middle Ages" of programming. He was the most outspoken of a group of computer scientists attempting to subject the computer to the rigor of mathematics and logic: others included the American computer scientist and artificial intelligence pioneer, John McCarthy; the Danish computer scientist, Peter Naur; the American, Robert W. Floyd; and the Briton, Tony Hoare.

Chapter 2 also outlines the first stirrings within computer science of Hume's issue: how does one know that a claimed proof is correct? There were two broad responses to the possibility of erroneous proofs. First was that of Harlan D. Mills, director of software engineering at a key site of the development of critical software: IBM's Federal Systems Division. Mills wanted programmers to prove mathematically that their programs were correct implementations of the specifications they were given, but he saw proof as a human activity that could lead only to "subjective conviction." The second response—the dominant one, at least in academic computer science—was to turn to the computer itself to alleviate the difficulty and error-proneness of the production of proofs. By 1969 the first automated system for applying proof to programs was constructed by Floyd's student, James King.

The automation of mathematical proof, however, did not begin with King's system. Chapter 3 returns to 1956 and to the iconic moment, often taken as the beginning of artificial intelligence, when Herbert Simon declared that he and his colleague Allen Newell had "invented a thinking machine." That machine—first a human simulation of a computer program and then an actual program—proved theorems in propositional logic. Simon's work in artificial intelligence interwove with his social science, especially with his critique of economists' hyperrationalistic view of human beings. In developing the "Logic Theory Machine," Simon studied how human beings like himself found proofs. To the logi-

cians becoming interested in automated theorem proving, however, Simon and Newell's search for human-like "heuristics" was misguided and amateurish. For Hao Wang, one of those logicians, the attraction of computers was not that they were potentially intelligent machines but that they were "persistent plodders." Deep in the history of logic lay the search by the philosopher Gottfried Leibniz for a universal scientific language and logical calculus that would make it possible to "judge immediately whether propositions presented to us are proved . . . with the guidance of symbols alone, by a sure and truly analytical method."[44] In the centuries after Leibniz, human logicians had gradually accumulated the formal systems necessary for machine-like deductive inference; the computer offered them the means of turning these systems into technological reality, and to seek to make their operation human-like was, to the logicians, at best a distraction. In 1963, this strand of automated theorem proving reached its epitome in the development by philosopher and computer scientist Alan Robinson of the single most important technique of automated deduction: resolution, an explicitly "machine-oriented" rather than "human-oriented" form of inference.

Chapter 3 also describes other debates about computers, mathematics and artificial intelligence. Philosopher J. R. Lucas and mathematician Roger Penrose, the first an opponent of "mechanism," the second of artificial intelligence, argued that Gödel's famous incompleteness theorems showed that human minds could know mathematical truths that machines could not prove. From the side of artificial intelligence, Douglas B. Lenat sought to develop an "automated mathematician" (AM), which, he claimed, was able on its own to "rediscover" important parts of human mathematics. Lenat's system not only reopened the divide between artificial intelligence and formal logic (Wang described Lenat's thesis on AM as "baffling"), but also sparked fierce critique from within artificial intelligence.

Chapter 4 turns from computer science and artificial intelligence to mathematics itself. The most celebrated use of a computer to prove a mathematical theorem was the 1976 proof of the four-color conjecture by mathematician Wolfgang Haken and mathematical logician Kenneth Appel. First proposed in 1852, the conjecture can be understood by a young child, but a century of work by human mathematicians had not led to a proof that survived scrutiny. Appel and Haken's solution raised the issue of whether a partially computerized demonstration such as theirs was a proof at all. Opinions on that point differed sharply. The members of what Harry Collins calls the core set,[45] those mathematicians

attempting a similar proof using similar computerized means, accepted the mechanized part of Appel and Haken's analysis as almost certainly correct, and they focused their doubts on the complicated human reasoning involved. Other mathematicians, however, focused their unease on Appel and Haken's use of computers. It was, one commented, as if they had simply consulted an oracle, and oracles were not mathematics.

Chapter 5 returns to the issue of dependability and to the dominant practical concern driving proof about computers: the vulnerability to intrusion of computer systems containing information critical to national security. Here, one encounters the two most important organizations that have influenced the development of computer proof: the U.S. Department of Defense's Advanced Research Projects Agency (ARPA) and the more powerful, but also more secretive, National Security Agency (NSA). For many years, NSA's very existence was not usually acknowledged. From the 1960s on, NSA and some other military agencies began to be concerned about computer security. Early experiments were unfailingly worrying: all serious attempts to circumvent the controls of the "time-sharing" computer systems entering service in the late 1960s succeeded.

In response, key ideas of computer security began to be formulated, such as the notion of a "security kernel" that any security-relevant request by user programs had to invoke. Drawing upon the "general system theory" then current in the work of von Bertalanffy and others, David Elliott Bell and Leonard J. LaPadula of the defense think tank, the MITRE Corporation, developed in the early 1970s the paradigmatic model of what, mathematically, "security" meant. A way forward then appeared clear: develop a security kernel that could be proved to implement the Bell-LaPadula model. Much of the financial support in the United States for the development of automated theorem provers arose from the desire of ARPA and NSA for mechanical proofs of security of this kind, and the 1970s' projects trying to prove the security of operating systems, of kernels, and of computer networks were the first real-world applications of proof to computer systems.

Applying the Bell-LaPadula model in practice was more complicated than it appeared in theory, and system developers found that they had to allow "trusted subjects" to violate the model's rules and to analyze "covert channels" not represented in the model. Bureaucratic turf wars and clashes of culture (such as between ARPA's relative openness and NSA's secrecy) also took their toll. By 1983, a stable set of *Trusted Computer System Evaluation Criteria* (universally known from the color of the

covers of the document containing them as the "orange book") emerged, which demanded deductive proof that the detailed specification of the most highly secure systems (those in the orange book's highest, A1, category) correctly implemented a mathematical model of security like that put forward by Bell and LaPadula. Successful efforts to meet A1 criteria were, however, few. A vicious circle of high costs, limited markets, and long development and evaluation times set in. Furthermore, 1986 brought an apparent bombshell: the claim by computer security specialist John McLean of the U.S. Naval Research Laboratory that the Bell-LaPadula model was flawed: it allowed a hypothetical "System Z" that was patently insecure. The variety of responses to System Z, and the "dialectical" development of models of computer security that sought to block the insecurities permitted by earlier models, are reminiscent of Imre Lakatos's classic account of proving.[46] At stake, however, was not just a mathematical theorem, but arguably the authority of NSA and even the national security of the United States.

Chapter 6 turns to the two most prominent general attacks on the application of computerized proof to software. The first came in the 1970s from two young computer scientists, Richard A. DeMillo and Richard J. Lipton, and from Alan J. Perlis, one of the founders of American computer science. DeMillo, Lipton, and Perlis claimed that "proofs" of computer programs performed by automated theorem provers were quite different from proofs within mathematics. A real proof, they asserted, was not a chain of formal logical inferences of the kind that a computer could perform, but an argument that a human being could understand, check, accept, reject, modify, and use in other contexts. A computer-produced verification, thousands of lines long and readable only by its originator, could not be subject to these social processes and in consequence was not a genuine proof. To practitioners who had felt criticized by advocates of the mathematicization of computing, the attack by DeMillo, Lipton, and Perlis was welcome puncturing of what they took to be a theorists' bubble. To Edsger W. Dijkstra, in contrast, the attack was "a political pamphlet from the Middle Ages." Dijkstra was computer science's great rationalist, a man who describes himself as "a happy victim of the Enlightenment."[47] As the years went by, indeed, Dijkstra's views came to differ even more sharply from those of DeMillo, Lipton, and Perlis. He was no longer prepared to accept proof as ordinarily conducted by mathematicians as the model for computer scientists to emulate. Even mathematicians carried the taint of the Middle Ages: they were a "guild." They "would rather continue to believe that the Dream of

Leibniz is an unrealistic illusion."[48] Computer science, in contrast, showed the way to proofs more rigorous, more formal, than those of ordinary mathematics.

Chapter 6 also describes the quite different attack on the application of proof to computer programs launched in 1988 by the philosopher, James H. Fetzer. Though Fetzer's conclusion was as hostile to "program proof" as DeMillo, Lipton, and Perlis's, his reasoning was quite different. He rejected their sociological view of proof and in so doing returned to philosophical orthodoxy: for Fetzer, the canonical notion of proof was formal proof, and the validity of its logical inferences was not affected by whether they were conducted by a human being or by a machine. Instead, Fetzer argued that "the very idea" of "program proof" was what philosophers call a category mistake. A program was a causal entity that affected the behavior of a computer, which was a physical machine; "proof" was part of the formal, abstract, nonphysical world of mathematics and of logic. The presence or absence of "social processes" was mere contingency: what doomed program proof, claimed Fetzer, was that it was a self-contradictory notion.

Chapter 7 discusses how the relationship between proof and the real, physical world has played out not at the level of philosophical debate but in practice. It begins with the first major project in which proof was applied to a system in which the dominant concern was not security but human safety: an experimental aircraft control system called Software Implemented Fault Tolerance (SIFT). Though now largely forgotten, SIFT was the most prominent verification project in the world in the late 1970s and early 1980s. To SIFT's sophisticated design were applied some of the subtlest ideas of computer science, notably about what computer scientist Leslie Lamport called "Byzantine" faults, most easily conceived of by thinking of a system component, such as a clock, as being actively malicious. SIFT, and the clock convergence algorithm that was embodied in it, represent a kind of historical loop. Historian of physics Peter Galison has suggested that Einstein's theory of relativity may have been inspired in part by reflections on the technological problem of synchronizing geographically separate clocks.[49] Lamport, fascinated by time and its treatment in Einsteinian physics, returned the problem of synchronization from the realm of physics to that of technology. SIFT's ambitious design made the application of proof to it even harder than in the early security projects. A 1983 peer review of the SIFT proof work, intended largely to evaluate DeMillo, Lipton, and Perlis's critique of pro-

gram proving, became, instead, a sharp internecine dispute among those committed to formal verification.

The overt debate in the peer review concerned the achievements of the SIFT proof work and how these had been reported. The SIFT project, however, also generated a divergence of greater long-term significance. Two participants in the early phases of the SIFT verification, Robert S. Boyer and J Strother Moore, took on what appeared to be a small task: applying proof to the fourteen lines of program that implemented the SIFT "dispatcher," lines written not in a high-level programming language but in assembly language, closer to the operations of the physical machine. Attempting to verify these lines of code, Boyer and Moore began to see a flaw in the entire enterprise of program proof as it was then practiced. It implicitly assumed, they suggested, that it was a "god," not a machine, that implemented programs. They began to believe that proof needed to be driven downwards, towards the hardware: verified programs needed verified machines to implement them. Chapter 7 describes the effort to produce such a machine and also discusses the controversy (described elsewhere)[50] over whether the claim of proof for a different, British-designed, microprocessor was justified. The chapter ends by briefly describing the form in which a version of proof (a highly automated form called model checking) has broken through from research to widespread industrial adoption.

Chapter 8 moves from the applications of mechanized proof to its key tools: automated theorem provers, in particular those systems designed not as exercises in artificial intelligence but to be controlled by a human being and used in practical verification tasks. The development of several of these provers is described to provide the basis for a discussion of what it means to use such a prover to perform a proof. One issue is the fact that different provers embody different formal logics. Another arises because theorem provers are themselves computer programs of some complexity. Alan Robinson, developer of resolution, has taken that latter fact as indicating a vicious regress undermining the entire enterprise of formal verification. Those most centrally involved do not accept that conclusion, and their different responses to the possibility that theorem-proving programs contain bugs are described.

A further matter discussed in chapter 8 is explicitly one of social trust. Among the most popular theorem provers, especially in Europe, is a family of systems that protect the validity of the inferences they perform by building a "firewall" (implemented by what computer scientists call

"type checking") around formulae of a particular type: axioms and those formulae constructed from axioms by the mechanical rules of logical inference that the system implements. To proponents of systems of this kind, the firewall around the type "theorem" makes their use highly rigorous. The most widely used such system, Higher Order Logic (HOL), however, permits a user simply to assert a theorem by using the facility known as mk_thm. In the academic world from which HOL came, users were implicitly trusted not to abuse mk_thm. As HOL entered the world of security-critical computing, however, that trust could no longer be taken for granted. The malicious demon of the philosophy of mathematics became personified in the fear that among those performing an apparent proof might be a hostile agent. In the world of security, mk_thm became not a facility, as it had been in the academic world, but a loophole that had to be closed.

In chapter 9 the threads of the book are pulled together. The history of computer-related accidents is reviewed, and the question is asked why software-based systems appear to have killed relatively few people despite software's dependability problems. That is the most central aspect of what, in recognition of the man who pointed it out, I call the Hoare paradox: the fact that, without use of the proofs that Tony Hoare and his colleagues advocated, the practical record of software systems is better, for example, than had been feared at Garmisch. There are a variety of possible explanations, of which the most interesting from the viewpoint of this book are, first, the successful "moral entrepreneurship" of Hoare and many of his colleagues in alerting technical audiences to software's dangers (and in so doing reducing those dangers), and, second, that inductive and authority-based forms of knowledge of the properties of computer systems appear more effective, in sociotechnical practice, than, for example, the abstract statistical analysis of induction might suggest. In particular, it appears as if they may work to produce knowledge of people—of trustworthy system developers—as well as knowledge of the artifacts those people produce.

Chapter 9 then turns to cultures of proving. In the cleanroom, "proof" is explicitly intersubjective and need not be mathematical in form; for Dijkstra and those influenced by him, it must be formal, but need not be mechanical. The two central cultures of proving, however, are that of formal, mechanized proof and that of ordinary mathematics. Although the intellectual roots of the former lie within logic (a discipline to a significant extent separate from mathematics), its key practical underpinning has become the desire for mechanization, not for formality

per se. The resultant culture, however, is not homogeneous. Different theorem provers implement different formal logics in ways that often differ significantly, and "investments" in developing these often complex programs, and in learning to use them effectively, tend to divide the culture of mechanized proof into distinct, albeit interacting, subcultures.

The book ends by discussing what the proving machines discussed in this book do not do. They simulate, at most, an individual mathematician operating within a given formal system. They neither modify formal systems nor choose between them. Nor, even within a given formal system, do their operations serve as the ultimate criterion of correctness, as the theorem-proving "bugs" discussed in chapter 8 demonstrate: they can be identified as causes of error, rather than as sources of valid, novel deductions, because normativity—the capacity to distinguish between "getting something right or wrong"[51]—remains vested in collective human judgment. It is we who allow, or disallow, machine operations as constituting proofs. Mechanized provers can be a vital aid to the fallible individual computer system developer and, perhaps, eventually to the individual mathematician as well, but they are no substitute for the human collectivity.