EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

NOTES FOR CHAPTER 12:

The Principle of Computational Equivalence

NOTES FOR CHAPTER 12

The Principle of Computational Equivalence

Basic Framework

■ **All is computation.** The early history of science includes many examples of attempts to treat all aspects of the universe in a uniform way. Some were more successful than others. "All is fire" was never definite enough to lead to much, but "all is number" can be viewed as an antecedent to the whole application of mathematics to science, and "all is atoms" to the atomic theory of matter and quantum mechanics. My "all is computation" will, I believe, form the basis for a fruitful new direction in science. It should be pointed out, however, that it is wrong to think that once one has described everything as, say, computation, then there is nothing more to do. Indeed, the phenomenon of computational irreducibility discussed in this chapter specifically implies that in many cases irreducible work has to be done in order to find out how any particular system will behave.

Outline of the Principle

■ **Note for mathematicians.** The way I discuss the Principle of Computational Equivalence is in a sense opposite to what would be typical in modern mathematics. For rather than starting with very specific definitions and then expanding from these, I start from general intuition and then use this to come up with more specific results. In the years to come there will no doubt be many attempts to formulate parts of the Principle of Computational Equivalence in ways that are closer to the traditions of modern mathematics. But at least at first, I suspect that huge simplifications will be made, with the result that all sorts of misleading conclusions will probably be reached, perhaps in some cases even seemingly contradicting the principle.

■ **History.** As I discuss elsewhere, aspects of the Principle of Computational Equivalence have many antecedents. But the complete principle is presented for the first time in this book, and is the result of thinking I did in the late 1980s and early 1990s.

■ **Page 717 · Church's Thesis.** The idea that any computation that can be done at all can be done by a universal system such as a universal Turing machine is often referred to as Church's Thesis. Following the introduction of so-called primitive recursive functions (see page 907) in the 1880s, there had by the 1920s emerged the idea that perhaps any reasonable function could be computed using the small set of operations on which primitive recursive functions are based. This notion was supported by the fact that certain modifications to these operations were found to allow only the exact same set of functions. But the discovery of the Ackermann function in the late 1920s (see page 906) showed that there are reasonable functions that are not primitive recursive. The proof of Gödel's Theorem in 1931 made use of so-called general recursive functions (see page 1121) as a way to represent possible functions in arithmetic. And in the early 1930s the two basic idealizations used in foundational studies of mathematical processes were then general recursive functions and lambda calculus (see page 1121). By 1934 these were known to be equivalent, and in 1935 Alonzo Church suggested that either of them could be used to do any mathematical calculation which could effectively be done. (It had been noted that many specific kinds of calculations could be done within such systems—and that processes like diagonalization led to operations of a seemingly rather different character.) In 1936 Alan Turing then introduced the idea of Turing machines, and argued that any mathematical process that could be carried out in practice, say by a person, could be carried out by a Turing machine. Turing proved that his machines were exactly equivalent in their computational capabilities to lambda calculus. By the 1940s Emil Post had shown that the string rewriting systems he had studied were also equivalent, and as electronic computers began to be developed it became quite firmly established that Turing machines provided an appropriate idealization for what computations could be done. From the 1940s to 1960s many different types of systems—almost all mentioned at some point or another in this book—were shown to be equivalent in their computational capabilities. (Starting in the 1970s, as discussed on page 1143, emphasis shifted to studies not of overall equivalence but instead equivalence with respect to classes of transformations such as polynomial time.)

When textbooks of computer science began to be written some confusion developed about the character of Church's Thesis: was it something that could somehow be deduced, or was it instead essentially just a definition of computability? Turing and Post seem to have thought of Church's Thesis as characterizing the "mathematicizing power" of humans, and Turing at least seems to have thought that it might not apply to continuous processes in physics. Kurt Gödel privately discussed the question of whether the universe could be viewed as following Church's Thesis and being "mechanical". And starting in the 1950s a few physicists, notably Richard Feynman, asked about fundamental comparisons between computational and physical processes. But it was not until the 1980s—perhaps particularly following some of my work—that it began to be more widely realized that Church's Thesis should best be considered a statement about nature and about the kinds of computations that can be done in our universe. The validity of Church's Thesis has long been taken more or less for granted by computer scientists, but among physicists there are still nagging doubts, mostly revolving around the perfect continua assumed in space and quantum mechanics in the traditional formalism of theoretical physics (see page 730). Such doubts will in the end only be put to rest by the explicit construction of a discrete fundamental theory along the lines I discuss in Chapter 9.

The Content of the Principle

▪ **Page 719 · Character of principles.** Examples of principles that can be viewed in several ways include the Principle of Entropy Increase (Second Law of Thermodynamics), the Principle of Relativity, Newton's Laws, the Uncertainty Principle and the Principle of Natural Selection. The Principle of Entropy Increase, for example, is partly a law of nature relating to properties of heat, partly an abstract fact about ensembles of dynamical systems, and partly a foundation for the definition of entropy. In this case and in others, however, the most important role of a principle is as a guide to intuition and understanding.

▪ **Page 720 · Oracles.** Following his introduction of Turing machines Alan Turing tried in 1937 to develop models that would somehow allow the ultimate result of absolutely every conceivable computation to be determined. And as a step towards this, he introduced the idea of oracles which would give results of computations that could not be found by any Turing machine in any limited number of steps. He then noted, for example, that if an oracle were set up that could answer the question for a particular universal system of whether that system would ever halt when given any specific input, then with an appropriate transformation of input this same oracle could also answer the question for any other system that can be emulated by the universal system. But it turns out that this is no longer true if one allows systems which themselves can access the oracle in the course of their evolution. Yet one can then imagine a higher-level oracle for these systems, and indeed a whole hierarchy of levels of oracles—as studied in the theory of degrees of unsolvability. (Note that for example to answer the question of whether or not a given Turing machine always halts can require a second-order oracle, since it is a Π_2 question in the sense of page 1139.)

▪ **Initial conditions.** Oracles are usually imagined as being included in the internal rules for a system. But if there are an infinite number of elements that can be specified in the initial condition—as in a cellular automaton—then a table for an oracle could also be given in the initial conditions.

▪ **Page 722 · Criteria for universality.** To be universal a system must in effect be able to emulate any feature of any system. So at some level any feature can be thought of as a criterion for universality. Some features—like the possibility of information transmission—may be more obvious than others, but despite occasional assertions to the contrary in the scientific literature none is ever the whole story. Since any given universal system must be able to emulate any other universal system it follows that within any such system it must in a sense be possible to find any known universal system. But inevitably the encoding will sometimes be very complicated. And in practice if there are many simple rules that are universal they cannot all be related by simple encodings. (See also the end of Chapter 11.)

▪ **Page 722 · Encodings.** One can prevent an encoding from itself introducing universality by insisting, for example, that it be primitive recursive (see page 907) or always involve only a bounded number of steps. One can also do this—as in the rule 110 proof in the previous chapter—by having programs and data be encoded separately, and appear, say, as distinct parts of the initial conditions for the system one is studying. (See also page 1118.)

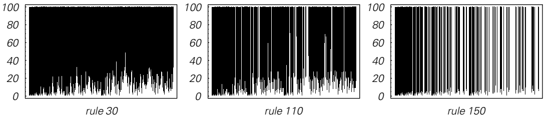
▪ **Density of universal systems.** One might imagine that it would be possible to make estimates of the overall density of universal systems, perhaps using arguments like those for the density of primes, or for the density of algorithmically random sequences. But as it turns out I know of no way to make any such estimates. If one has shown that various simple rules are universal, then it follows that rules which generalize these must also be universal. But even from this I do not know, for example, how to prove that the density of universal rules cannot decrease when rules become more complicated.

▪ **Page 723 · Proving universality.** The question of whether a system is universal is in general undecidable. Using a specific mathematical axiom system such as Peano arithmetic or set theory it may also be that there is no proof that can be given. (It is straightforward to construct complicated examples where this is the case.) In practice it seems to get more difficult to prove universality when the structure of a system gets simpler. Current proofs of universality all work by showing how to emulate a known universal system. Some level of checking can be done by tracing the emulation of random initial conditions for the universal system. In the future it seems likely that automated theorem-proving methods should help in finding proofs of universality.

▪ **Page 724 · History.** There are various precedents in philosophy and mysticism for the idea of encoding all possible knowledge of some kind in a single object. An example in computation theory is the concept emphasized by Gregory Chaitin of a number whose n^{th} digit specifies whether a computation with initial condition n in a particular system will ever halt. This particular number is far from being computable (see page 1128), as a result of the undecidability of the halting problem (see page 754). But a finite version in which one looks at results after a limited number of steps is similar to my concept of a universal object. (See also page 1067.)

▪ **Page 725 · Universal objects.** A more direct way to create a universal object is to set up, say, a 4D array in which two of the dimensions range respectively over possible 1D cellular automaton rules and over possible initial conditions, while the other two dimensions correspond to space and time in the evolution of each cellular automaton from each initial condition. (Compare the parameter space sets of page 1006.)

▪ **Page 725 · Block occurrences.** The pictures below show at which step each successive block of length up to 8 first appears in evolution according to various cellular automaton rules starting from a single black cell. For rule 30, the numbers of steps needed for each block of lengths 1 through 10 to appear at least once is *{1, 2, 4, 12, 22, 24, 33, 59, 69, 113}*. (See also page 871.)



The Validity of the Principle

▪ **Page 729 · Continuum and cardinality.** Some notion of a distinction between continuous and discrete systems has existed since antiquity. But in the 1870s the distinction became more precise with Georg Cantor's characterization of the total numbers of possible objects of various types in terms of different orders of infinity (see page 1162). The total number of possible integers corresponds to the smallest level of infinity, usually denoted $\aleph_0$. The total number of possible lists of integers of given finite length—and thus the number of possible rational numbers—turns out also to be $\aleph_0$. The reason is that it is always possible to encode any finite list of integers as a single integer, as discussed on page 1120. (A way to do this for pairs of non-negative integers is to use *σ[{x_, y_}] := 1/2 (x + y) (x + y + 1) + x*.) But for real numbers the story is different. Any real number x can be represented as a set of integers using for example

```
Rest[FoldList[Plus, 1, ContinuedFraction[x]]]
```

but except when x is rational this list is not finite. Since the number of possible subsets of a set with k elements is 2^k, the number of possible real numbers is $2^{\aleph_0}$. And using Cantor's diagonal argument (see note below) one can then show that this must be larger than $\aleph_0$. (The claim that there are no sets intermediate in size between $\aleph_0$ and $2^{\aleph_0}$ is the so-called continuum hypothesis, which is known to be independent of the standard axioms of set theory, as discussed on page 1155.) Much as for integers, finite lists of real numbers can be encoded as single real numbers—using for example roughly *FromDigits[Flatten[Transpose[RealDigits[list]]]]*—so that the number of such lists is $2^{\aleph_0}$. (Space-filling curves yield a more continuous version of such an encoding.) But unlike for integers the same turns out to be true even for infinite lists of real numbers. (The function σ above can for example be used to specify the order in which to sample elements in *RealDigits[list]*). The total number of possible functions of real numbers is $2^{2^{\aleph_0}}$; the number of continuous such functions (which can always be represented by a list of coefficients for a series) is however only $2^{\aleph_0}$.

In systems like cellular automata, finite arrangements of black cells on a background of white cells can readily be specified by single integers, so the number of them is $\aleph_0$. But infinite configurations of cells are like digit sequences of real numbers (as discussed on page 869 they correspond more precisely to elements in a Cantor set), so the number of them is $2^{\aleph_0}$. Continuous cellular automata (see page 155) also have $2^{\aleph_0}$ possible states.

▪ **Computable reals.** The stated purpose of Alan Turing's original 1936 paper on computation was to introduce the notion of computable real numbers, whose n^{th} digit for any n could be found by a Turing machine in a finite number of steps. Real numbers used in any explicit way in traditional mathematics are always computable in this sense. But as Turing pointed out, the overwhelming majority of all possible real numbers are not computable. For certainly there can be no more computable real numbers than there are possible Turing machines. But with his discovery of universality, Turing established that any Turing machine can be emulated by a single universal Turing machine with suitable initial conditions. And the point is that any such initial conditions can always be encoded as an integer.

As examples of non-computable reals that can readily be defined, Turing considered numbers whose successive digits are determined by the eventual behavior after an infinitely long time of a universal system with successive possible initial conditions (compare page 964). With two possible forms of behavior *h[i] = 0* or *1* for initial condition *i*, an example of such a number is $Sum[2^{-i} h[i], \{i, \infty\}]$. Closely related is the total probability for each form of behavior, given for example by *Sum[2^(-Ceiling[Log[2, i]]) h[i], {i, ∞}]*. I suspect that many limiting properties of systems like cellular automata in general correspond to non-computable reals. An example is the average density of black cells after an arbitrarily long time. For many rules, this converges rapidly to a definite value; but for some rules it will wiggle forever as more and more initial conditions are included in the average.

▪ **Diagonal arguments.** Similar arguments were used by Georg Cantor in 1891 to show that there must be more real numbers than integers and by Alan Turing in 1936 to show that the problem of enumerating computable real numbers is unsolvable. One might imagine that it should be possible to set up a function *f[i, n]* which if given successive integers *i* would give the n^{th} base 2 digit in every possible real number. But what about the number whose n^{th} digit is *1 - f[n, n]*? This is still a real number, yet it cannot be generated by *f[i, n]* for any *i*—thus showing that there are more real numbers than integers. Analogously, one might imagine that it should be possible to have a function *f[i, n]* which enumerates all possible programs that always halt, and specifies a digit in their output when given input *n*. But what about the program with output *1 - f[n, n]*? This program always halts, yet it does not correspond to any possible value of *i*—even though universality implies that any program should be encodable by a single integer *i*. And the only possible conclusion from this is that *f[i, n]* cannot in fact be implemented as a program that always halts—thus demonstrating that the computable real numbers cannot explicitly be enumerated. (Closely related is the undecidability of the problem discussed on page 1137 of whether a system halts given any particular input.) (See also pages 907 and 1162.)

▪ **Continuous computation.** Various models of computation that involve continuous elements have been proposed since the 1930s, and unlike those with discrete elements they have often not proved ultimately equivalent. One general class of models based on the work of Alan Turing in 1936 follow the operation of standard digital computers, and involve looking at real numbers in terms of digits, and using discrete processes to generate these digits. Such models inevitably handle only computable reals (in the sense defined above), and can never do computations beyond those possible in ordinary discrete systems. Functions are usually considered computable in such models if one can take the procedure for finding the digits of *x* and get a procedure for finding the digits of *f[x]*. And with this definition all standard mathematical functions are computable—even those from chaos theory that excavate digits rapidly. (It seems possible however to construct functions computable in this sense whose derivatives are not computable.) The same basic approach can be used whenever numbers are represented by constructs with discrete elements (see page 143), including for example symbolic formulas.

Several times since the 1940s it has been suggested that models of computation should be closer to traditional continuous mathematics, and should look at real numbers as a whole, not in terms of their digit or other representations. In a typical case, what is done is to generalize the register machines of page 97 to have registers that hold arbitrary real numbers. It is then usually assumed, however, that the primitive operations performed on these registers are just those of ordinary arithmetic, with the result that only a very limited set of functions (not including for example the exponential function) can be computed in a finite number of steps. Introducing other standard mathematical functions as primitives does not usually help much, unless one somehow gives the system the capability to solve any equation immediately (see below). (Other appropriate primitives may conceivably be related to the solubility of Hilbert's Thirteenth Problem and the fact that any continuous function with any number of arguments can be written as a one-argument function of a sum of a handful of fixed one-argument functions applied to the arguments of the original function.)

Most of the types of programs that I have discussed in this book can be generalized to allow continuous data, often just by having a continuous range of values for their elements (see e.g. page 155). But the programs themselves normally remain discrete, typically involving discrete choices made at discrete steps. If one has a table of choices, one can imagine generalizing this to a function of a real number. But to specify this function one normally has no choice but to use some type of finite formula. And to set up any kind of continuous evolution, the most obvious approach is to use traditional mathematical ideas of calculus and differential equations (see page 161). This leads to models in which possible computations are assumed, say, to correspond to combinations of differential equations—as in Claude Shannon's 1941 general-purpose analog computer. And if one assumes—as is usually implicitly done in traditional mathematics—that any solutions that exist to these equations can somehow always be found then at least in principle this allows computations impossible for discrete systems to be done.

■ **Initial conditions.** Traditional mathematics tends to assume that real numbers with absolutely any digit sequence can be set up. And if this were the case, then the digits of an initial condition could for example be the table for an oracle of the kind discussed on page 1126—and even a simple shift mapping could then yield output that is computationally more sophisticated than any standard discrete system. But just as in my discussion of chaos theory in Chapter 7, any reasonably complete theory must address how such an initial condition could have been constructed. And presumably the only way is to have another system that already violates the Principle of Computational Equivalence.

■ **Constructible reals.** Instead of finding successive digits using systems like Turing machines, one can imagine constructing complete real numbers using idealizations of mechanical processes. An example studied since antiquity involves finding lengths or angles using a ruler and compass (i.e. as intersections between lines and circles). However, as was shown in the 1800s, this method can yield only numbers formed by operating on rationals with combinations of *Plus*, *Times* and *Sqrt*. (Thus it is impossible with ruler and compass to construct π and "square the circle" but it is possible to construct 17-gons or other *n*-gons for which *FunctionExpand[Sin[π/n]]* contains only *Plus*, *Times* and *Sqrt*.) Linkages consisting of rods of integer lengths always trace out algebraic curves (or algebraic surfaces in 3D) and in general allow any algebraic number (as represented by *Root*) to be constructed. (Linkages were used by the late 1800s not only in machines such as steam engines, but also in devices for analog computation. More recently they have appeared in robotics.) Note that above degree 4, algebraic numbers cannot in general be expressed in radicals involving only *Plus*, *Times* and *Power* (see page 945).

■ **Page 732 · Equations.** For any purely algebraic equation involving real numbers it is possible to find a bound on the size of any isolated solutions it has, and then to home in on their actual values. But as discussed on page 786, nothing similar is true for equations involving only integers, and in this case finding solutions can in effect require following the evolution of a system like a cellular automaton for infinitely many steps. If one allows trigonometric functions, any equation for integers can be converted to one for real numbers; for example $x^2 + y^2 == z^2$ for integers is equivalent to $Sin[\pi x]^2 + Sin[\pi y]^2 + Sin[\pi z]^2 + (x^2 + y^2 - z^2)^2 == 0$ for real numbers.

■ **Page 732 · ODEs.** The method of compressing time using algebraic transformations works not only in partial but also in ordinary differential equations.

■ **Emulating discrete systems.** Despite it often being assumed that continuous systems are computationally more sophisticated than discrete ones, it has in practice proved surprisingly difficult to make continuous systems emulate discrete ones. Some integer functions can readily be obtained by supplying integer arguments to continuous functions, so that for example *Mod[x, 2]* corresponds to $Sin[\pi x/2]^2$ or *(1 - Cos[π x])/2*,

```
Mod[x, 3]↔1 + 2/3 (Cos[2/3 π (x - 2)] - Cos[2 π x/3])
Mod[x, 4]↔(3 - 2 Cos[π x/2] - Cos[π x] - 2 Sin[π x/2])/2
Mod[x, n]↔Sum[j Product[(Sin[π (x - i - j)/n]/
              Sin[π i/n])^2, {i, n - 1}], {j, n - 1}]
```

(As another example, *If[x > 0, 1, 0]* corresponds to *1 - 1/Gamma[1 - x]*.) And in this way the discrete system *x → If[EvenQ[x], 3 x/2, 3 (x + 1)/2]* from page 122 can be emulated by the continuous iterated map *x → (3 + 6 x - 3 Cos[π x])/4*. This approach can then be applied to the universal arithmetic system on page 673, establishing that continuous iterated maps can in principle emulate discrete universal systems. A similar result presumably holds for ordinary and therefore also partial differential equations (PDEs). One might expect, however, that it should be possible to construct a PDE that quite directly emulates a system like a cellular automaton. And to do this approximately is not difficult. For as suggested by the bottom row of pictures on page 732 one can imagine having localized structures whose interactions emulate the rules of the cellular automaton. And one can set things up so that these structures exhibit the analog of attractors, and evolve towards one of a few discrete states. But the problem is that in finite time one cannot expect that they will precisely reach such states. (This is somewhat analogous to the issue of asymptotic particle states in the foundations of quantum field theory.) And this means that the overall state of the system will not be properly prepared for the next step of cellular automaton evolution.

Generating repetitive patterns with continuous systems is straightforward, but generating even nested ones is not. Page 147 showed how $Sin[x] + Sin[\sqrt{2}\ x]$ has nested features, and these are reflected in the distribution of eigenvalues for ODEs containing such functions. Strange attractors for many continuous systems also show various forms of Cantor sets and nesting.

▪ **Page 732 · Time and gravity.** General relativity implies that time can be affected by gravitational fields—and that for example a process in a lower gravitational field will seem to be going faster if it is looked at by an observer in a higher gravitational field. (Related phenomena associated with motion in special relativity are more difficult to interpret in a static way.) But presumably there are effects that prevent infinite speedups. For if, say, energy were coming from a process at a constant rate, then an infinite speedup would lead to infinite energy density, and thus presumably to infinite gravitational fields that would change the system.

At least formally, general relativity does nevertheless suggest infinite transformations of time in various cases. For example, to a distant observer, an object falling into a black hole will seem to take an infinite time to cross the event horizon—even though to the object itself only a finite time will seem to have passed. One might have thought that this would imply in reverse that to an observer moving with the object the whole infinite future of the outside universe would in effect seem to go by in a finite time. But in the simplest case of a non-rotating black hole (Schwarzschild metric), it turns out that an object will always hit the singularity at the center before this can happen. In a rotating but perfectly spherical black hole (Kerr metric), the situation is nevertheless different, and in this case the whole infinite future of the outside universe can indeed in principle be seen in the finite time between crossing the outer and inner event horizons. But for the reasons mentioned above, this very fact presumably implies instability, and the whole effect disappears if there is any deviation from perfect spherical symmetry.

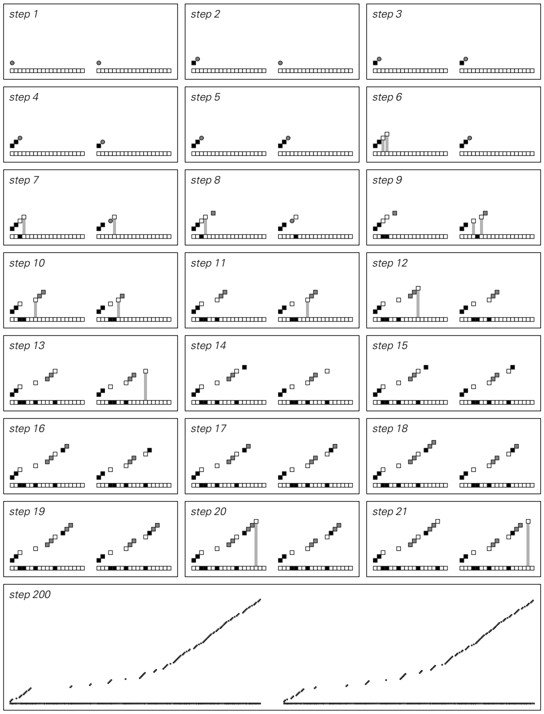
Even without general relativity there are already issues with time and gravity. For example, it was shown in 1990 that close encounters in a system of 5 idealized point masses can lead to infinite accelerations which cause one mass to be able to go infinitely far in a finite time.

▪ **Page 733 · Human thinking.** The discovery in this book that even extremely simple programs can give rise to behavior vastly more complex than expected casts suspicion on any claim that programs are fundamentally unable to reproduce features of human thinking. But complete evidence that human thinking follows the Principle of Computational Equivalence will presumably come only gradually as practical computer systems manage to emulate more and more aspects of human thinking. (See page 628.)

▪ **Page 734 · Intermediate degrees.** As discussed on page 753, an important indication of computational sophistication in a system is for its ultimate behavior to be undecidable, in the sense that a limited number of steps in a standard universal system cannot determine in general what the system will do after an infinite number of steps, and whether, for example, it will ever in some sense halt. Such undecidability is inevitable in any system that is universal. But what about other systems? So long as one only ever looks at the original input and final output it turns out that one can construct a system that exhibits undecidability but is not universal. One trivial way to do so is to take a universal system but modify it so that if it ever halts its output is discarded and, say, replaced by its original input. The lack of meaningful output prevents such a system from being universal, but the question of whether the system halts is still undecidable. Nevertheless, the pattern of this undecidability is just the same as for the underlying universal system. So one can then ask whether it is possible to have a system which exhibits undecidability, but with a pattern that does not correspond to that of any universal system.

As I discuss on page 1137, almost all known proofs of undecidability in practice work by reduction to the halting problem for some universal system—this is, by showing that if one could resolve whatever is supposed to be undecidable then one could also solve the halting problem for a universal system. But in 1956 Richard Friedberg and Albert Muchnik both gave an intricate and abstract construction of a system that has a halting problem which is undecidable but is not reducible to the halting problem of any universal system.

The pictures at the top of the facing page show successive steps in the evolution of an analog of their system. The input is an integer that gives a position in either of the two rows of cells at the bottom of each picture. All these cells are initially white, but some eventually become black—and the system is considered to halt for a particular input if the corresponding cell ever becomes black.



The rules for the system are quite complicated, and in essence work by progressively implementing a generalization of a diagonal argument of the kind discussed on page 1128. Note first that the configuration of cells in the rows at the bottom of each picture can be thought of as successive finite approximations to tables for an oracle (see page 1126) which gives the solution to the halting problem for each possible input to the system. To set up the generalized diagonal argument one needs a way to list all possible programs. Any type of program that supports universality can be used for this purpose; the pictures shown use essentially the register machines from page 97. Each row above the bottom one corresponds in effect to a successive register machine—and shows, if relevant, its output when given as input the integer corresponding to that position in the row, together with the complete bottom row of cells found so far. (A dot indicates that the register machine does not halt.) The way the system works is to put down new black cells in the bottom row in just such a way as to arrange that for any register machine at least the output shown will ultimately not agree with the cells in the bottom row. As indicated by vertical gray lines, there is sometimes temporary agreement, but this is always removed within a finite number of steps.

The fact that no register machine can ever ultimately give output that agrees everywhere with the bottom row of cells then demonstrates that the halting problem for the system—whose results appear in the bottom row—must be undecidable. Yet if this halting problem were reducible to a halting problem for a universal system, then by using its results one should ultimately be able to solve the halting problem for any system. However, even using the complete bottom row of cells on the left it turns out that the construction is such that no register machine can ever yield results after any finite number of steps that agree everywhere with the row of cells on the right—thus demonstrating that the halting problem for the system is not reducible to the halting problem for a universal system.

Note however that this result is extremely specific to looking only at what is considered output from the system, and that inside the system there are all sorts of components that are definitely universal.

Explaining the Phenomenon of Complexity

▪ **Definition of complexity.** See page 557.

▪ **Ingredients for complexity.** With its emphasis on breaking systems down to find their underlying elements traditional science tends to make one think that any important overall property of a system must be a consequence of some specific feature of its underlying construction. But the results of this section imply that for complexity this is not the case. For as discussed on page 1126 there is no direct structural criterion for sophisticated computation and universality. And indeed most ways of ensuring that these do not occur are in essence equivalent just to saying that the overall behavior exhibits some specific regularity and is therefore not complex.

▪ **Relativism and equivalence.** Although the notion has been discussed since antiquity, it has become particularly common in the academic humanities in the past few decades to believe that there can be no valid absolute conclusions about the world—only statements made relative to particular cultural contexts. My emphasis of the importance of perception and analysis might seem to support this view, and to some extent it does. But the Principle of Computational Equivalence implies that in the end essentially any method of perception and analysis that can actually be implemented in our universe must have a certain computational equivalence, and must therefore at least in some respects come to the same absolute conclusions.

Computational Irreducibility

▪ **History.** The notion that there could be fundamental limits to knowledge or predictability has been discussed repeatedly since antiquity. But most often it has been assumed that the origin of this must be inadequacy in models, not difficulty in working out their consequences. And indeed already in the 1500s with the introduction of symbolic algebra and the discovery of formulas for solving cubic and quartic equations the expectation began to develop that with sufficient cleverness it should be possible to derive a formula for the solution to any purely mathematical problem. Infinitesimals were sometimes thought to get in the way of finite understanding—but this was believed to be overcome by calculus. And when mathematical models for natural systems became widespread in the late 1600s it was generally assumed that their basic consequences could always be found in terms of formulas or geometrical theorems, perhaps with fairly straightforward numerical calculations required for connection to practical situations. In discussing gravitational interactions between many planets Isaac Newton did however comment in 1684 that "to define these motions by exact laws admitting of easy calculation exceeds, if I am not mistaken, the force of any human mind". But in the course of the 1700s and 1800s formulas were successfully found for solutions to a great many problems in mathematical physics (see note below)—at least when suitable special functions (see page 1091) were introduced. The three-body problem (see page 972) nevertheless continued to resist efforts at general solution. In the 1820s it was shown that quintic equations cannot in general be solved in terms of radicals (see page 1137), and by the 1890s it was known that degree 7 equations cannot in general be solved even if elliptic functions are allowed. Around 1890 it was then shown that the three-body problem could not be solved in general in terms of ordinary algebraic functions and integrals (see page 972). However, perhaps in part because of a shift towards probabilistic theories such as quantum and statistical mechanics there remained the conviction that for relevant aspects of behavior formulas should still exist. The difficulty for example of finding more than a few exact solutions to the equations of general relativity was noted—but a steady stream of results (see note below) maintained the belief that with sufficient cleverness a formula could be found for behavior according to any model.

In the 1950s computers began to be used to work out numerical solutions to equations—but this was seen mostly as a convenience for applications, not as a reflection of any basic necessity. A few computer experiments were done on systems with simple underlying rules, but partly because Monte Carlo methods were sometimes used, it was typically assumed that their results were just approximations to what could in principle be represented by exact formulas. And this view was strengthened in the 1960s when solitons given by simple formulas were found in some of these systems.

The difficulty of solving equations for numerical weather prediction was noted even in the 1920s. And by the 1950s and 1960s the question of whether computer calculations would be able to outrun actual weather was often discussed. But it was normally assumed that the issue was just getting a better approximation to the underlying equations—or better initial measurements—not something more fundamental.

Particularly in the context of game theory and cybernetics the idea had developed in the 1940s that it should be possible to make mathematical predictions even about complex human situations. And for example starting in the early 1950s government control of economies based on predictions from linear models became common. By the early 1970s, however, such approaches were generally seen as unsuccessful, but it was usually assumed that the reason was not fundamental, but was just that there were too many disparate elements to handle in practice.

The notions of universality and undecidability that underlie computational irreducibility emerged in the 1930s, but they were not seen as relevant to questions arising in natural science. Starting in the 1940s they were presumably the basis for a few arguments made about free will and fundamental unpredictability of human behavior (see page 1135), particularly in the context of economics. And in the late 1950s there was brief interest among philosophers in connecting results like Gödel's Theorem to questions of determinism—though mostly there was just confusion centered around the difficulty of finding countable proofs for statements about the continuous processes assumed to occur in physics.

The development of algorithmic information theory in the 1960s led to discussion of objects whose information content cannot be compressed or derived from anything shorter. But as indicated on page 1067 this is rather different from what I call computational irreducibility. In the 1970s computational complexity theory began to address questions about overall resources needed to perform computations, but concentrated on computations that perform fairly specific known practical tasks. At the beginning of the 1980s, however, it was noted that certain problems about models of spin glasses were NP-complete. But there was no immediate realization that this was connected to any underlying general phenomenon.

Starting in the late 1970s there was increasing interest in issues of predictability in models of physical systems. And it was emphasized that when the equations in such models are nonlinear it often becomes difficult to find their solutions. But usually this was at some level assumed to be associated with sensitive dependence on initial conditions and the chaos phenomenon—even though as we saw on page 1098 this alone does not even prevent there from being formulas.

By the early 1980s it had become popular to use computers to study various models of natural systems. Sometimes the idea was to simulate a large collection of disparate elements, say as involved in a nuclear explosion. Sometimes instead the idea was to get a numerical approximation to some fairly simple partial differential equation, say for fluid flow. Sometimes the idea was to use randomized methods to get a statistical approximation to properties say of spin systems or lattice gauge theories. And sometimes the idea was to work out terms in a symbolic perturbation series approximation, say in quantum field theory or celestial mechanics. With any of these approaches huge amounts of computer time were often used. But it was almost always implicitly assumed that this was necessary in order to overcome the approximations being used, and not for some more fundamental reason.

Particularly in physics, there has been some awareness of examples such as quark confinement in QCD where it seems especially difficult to deduce the consequences of a theory—but no general significance has been attached to this.

When I started studying cellular automata in the early 1980s I was quickly struck by the difficulty of finding formulas for their behavior. In traditional models based for example on continuous numbers or approximations to them there was usually no obvious correspondence between a model and computations that might be done about it. But the evolution of a cellular automaton was immediately reminiscent of other computational processes—leading me by 1984 to formulate explicitly the concept of computational irreducibility.

No doubt an important reason computational irreducibility was not identified before is that for more than two centuries students had been led to think that basic theoretical science could somehow always be done with convenient formulas. For almost all textbooks tend to discuss only those cases that happen to come out this way. Starting in earnest in the 1990s, however, the influence of *Mathematica* has gradually led to broader ranges of examples. But there still remains a very widespread belief that if a theoretical result about the behavior of a system is truly fundamental then it must be possible to state it in terms of a simple mathematical formula.

▪ **Exact solutions.** Some notable cases where closed-form analytical results have been found in terms of standard mathematical functions include: quadratic equations (~2000 BC) (*Sqrt*); cubic, quartic equations (1530s) ($x^{1/n}$); 2-body problem (1687) (*Cos*); catenary (1690) (*Cosh*); brachistochrone (1696) (*Sin*); spinning top (1849; 1888; 1888) (*JacobiSN*; *WeierstrassP*; hyperelliptic functions); quintic equations (1858) (*EllipticTheta*); half-plane diffraction (1896) (*FresnelC*); Mie scattering (1908) (*BesselJ*, *BesselY*, *LegendreP*); Einstein equations (Schwarzschild (1916), Reissner-Nordström (1916), Kerr (1963) solutions) (rational and trigonometric functions); quantum hydrogen atom and harmonic oscillator (1927) (*LaguerreL*, *HermiteH*); 2D Ising model (1944) (*Sinh*, *EllipticK*); various Feynman diagrams (1960s–1980s) (*PolyLog*); KdV equation (1967) (*Sech* etc.); Toda lattice (1967) (*Sech*); six-vertex spin model (1967) (*Sinh* integrals); Calogero-Moser model (1971) (*Hypergeometric1F1*); Yang-Mills instantons (1975) (rational functions); hard-hexagon spin model (1979) (*EllipticTheta*); additive cellular automata (1984) (*MultiplicativeOrder*); Seiberg-Witten supersymmetric theory (1994) (*Hypergeometric2F1*). When problems are originally stated as differential equations, results in terms of integrals (“quadrature”) are sometimes considered exact solutions—as occasionally are convergent series. When one exact solution is found, there often end up being a whole family—with much investigation going into the symmetries that relate them. It is notable that when many of the examples above were discovered they were at first expected to have broad significance in their fields. But the fact that few actually did can be seen as further evidence of how narrow the scope of computational reducibility usually is. Notable examples of systems that have been much investigated, but where no exact solutions have been found include the 3D Ising model, quantum anharmonic oscillator and quantum helium atom.

▪ **Amount of computation.** Computational irreducibility suggests that it might be possible to define “amount of computation” as an independently meaningful quantity—perhaps vaguely like entropy or amount of information. And such a quantity might satisfy laws vaguely analogous to the laws of thermodynamics that would for example determine what processes are possible and what are not. If one knew the fundamental rules for the universe then one way in principle to define the amount of computation associated with a given process would be to find the minimum number of applications of the rules for the universe that are needed to reproduce the process at some level of description.

▪ **Page 743 · More complicated rules.** The standard rule for a cellular automaton specifies how every possible block of cells of a certain size should be updated at every step. One can imagine finding the outcome of evolution more efficiently by adding rules that specify what happens to larger blocks of cells after more steps. And as a practical matter, one can look up different blocks using a method like hashing. But much as one would expect from data compression this will only in the end work more efficiently if there are some large blocks that are sufficiently common. Note that dealing with blocks of different sizes requires going beyond an ordinary cellular automaton rule. But in a sequential substitution system—and especially in a multiway system (see page 776)—this can be done just as part of an ordinary rule.

■ **Page 744 · Reducible systems.** The color of a cell at step *t* and position *x* can be found by starting with initial condition

```
Flatten[With[{w = Max[Ceiling[Log[2, {t, x}]]]},
    {2 Reverse[IntegerDigits[t, 2, w]] + 1,
      5, 2 IntegerDigits[x, 2, w] + 2}]]
```

then for rule 188 running the cellular automaton with rule

```
{{a : (1 | 3), 1 | 3, _} → a, {_, 2 | 4, a : (2 | 4)} → a,
  {3, 5 | 10, 2} → 6, {1, 5 | 7, 4} → 0, {3, 5, 4} → 7,
  {1, 6, 2} → 10, {1, 6 | 11, 4} → 8, {3, 6 | 8 | 10 | 11, 4} → 9,
  {3, 7 | 9, 2} → 11, {1, 8 | 11, 2} → 9, {3, 11, 2} → 8,
  {1, 9 | 10, 4} → 11, {_, a_ /; a > 4, _} → a, {_, _, _} → 0}
```

and for rule 60 running the cellular automaton with rule

```
{{a : (1 | 3), 1 | 3, _} → a, {_, 2 | 4, a : (2 | 4)} → a,
  {1, 5, 4} → 0, {_, 5, _} → 5, {_, _, _} → 0}
```

■ **Speed-up theorems.** That there exist computations that are arbitrarily computationally reducible was noted in work on the theory of computation in the mid-1960s.

■ **Page 745 · Mathematical functions.** The number of bit operations needed to add two *n*-digit numbers is of order *n*. The number of operations *m[n]* needed to multiply them increases just slightly more rapidly than *n* (see page 1093). (Even if one can do operations on all digits in parallel it still takes of order *n* steps in a system like a cellular automaton for the effects of different digits to mix together—though see also page 1149.) The number of operations to evaluate *Mod[a, b]* is of order *n* if *a* has *n* digits and *b* is small. Many standard continuous mathematical functions just increase or decrease smoothly at large *x* (see page 917). The main issue in evaluating those that exhibit regular oscillations at large *x* is to find their oscillation period with sufficient precision. Thus for example if *x* is an integer with *n* digits then evaluating *Sin[x]* or *FractionalPart[x c]* requires respectively finding π or *c* to *n*-digit precision. It is known how to evaluate π (see page 912) and all standard elementary functions to *n*-digit precision using about *Log[n] m[n]* operations. (This can be done by repeatedly making use of functional relations such as *Exp[2 x] == Exp[x]*^{2} which express *f[2 x]* as a polynomial in *f[x]*; such an approach is known to work for elementary, elliptic, modular and other functions associated with *ArithmeticGeometricMean* and for example *DedekindEta*.) Known methods for high-precision evaluation of special functions—usually based in the end on series representations—typically require of order $n^{1/s}$ *m[n]* operations, where *s* is often 2 or 3. (Examples of more difficult cases include *HypergeometricPFQ[a, b, 1]* and *StieltjesGamma[k]*, where logarithmic series can require an exponential number of terms. Evaluation of *BernoulliB[x]* is also difficult.) Any iterative procedure (such as *FindRoot*) that yields a constant multiple more digits at each step will take about *Log[n]* steps to get *n* digits. Roots of polynomials can thus almost always be found with *NSolve* in about *Log[n] m[n]* operations. If one evaluates *NIntegrate* or *NDSolve* by effectively fitting functions to order *s* polynomials the difficulty of getting results with *n*-digit precision typically increases like $2^{n/s}$. An adaptive algorithm such as Romberg integration reduces this to about *2^*$\sqrt{n}$. The best-known algorithms for evaluating *Zeta[1/2 + ⅈ x]* (see page 918) to fixed precision take roughly $\sqrt{x}$ operations—or $2^{n/2}$ operations if *x* is an *n*-digit integer. (The evaluation is based on the Riemann-Siegel formula, which involves sums of about $\sqrt{x}$ cosines.) Unlike for continuous mathematical functions, known algorithms for number theoretical functions such as *FactorInteger[x]* or *MoebiusMu[x]* typically seem to require a number of operations that grows faster with the number of digits *n* in *x* than any power of *n* (see page 1090).

■ **Formulas.** It is always in principle possible to build up some kind of formula for the outcome of any process of evolution, say of a cellular automaton (see page 618). But for there to be computational reducibility this formula needs to be simple and easy to evaluate—as it is if it consists just of a few standard mathematical functions (see note above; page 1098).

■ **Page 747 · Short computations.** Some properties include:

(a) The regions are bounded by the hyperbolas *x y == Exp[n/2]* for successive integers *n*.

(d) There is approximate repetition associated with rational approximations to π (for example with period 22), but never precise repetition.

(e) The pattern essentially shows which *x* are divisors of *y*, just as on pages 132 and 909.

(h) *Mod[Quotient[s,* 2^n*], 2]* extracts the digit associated with 2^n in the base 2 digit sequence of *s*.

(i) Like (e), except that colors at neighboring positions alternate.

(l) See page 613.

(m) The pattern can be generated by a 2D substitution system with rule *{1 -> {{0, 0}, {0, 1}}, 0 -> {{1, 1}, {1, 0}}}* (see page 583).

(See also page 870.)

Even though standard mathematical functions are used, few of the pictures can readily be generalized to continuous values of x and y.

▪ **Intrinsic limits in science.** Before computational irreducibility other sources of limits to science that have been discussed include: measurement in quantum mechanics, prediction in chaos theory and singularities in gravitation theory. As it happens, in each of these cases I suspect that the supposed limits are actually just associated with a lack of correct analysis of all elements of the relevant systems. In mathematics, however, more valid intrinsic limits—much closer to computational irreducibility—follow for example from Gödel's Theorem.

The Phenomenon of Free Will

▪ **History.** Early in history it seems to have generally been assumed that everything about humans must ultimately be determined by unchangeable fate—which it was sometimes thought could be foretold by astrology or other forms of divination. Most Greek philosophers seem to have believed that their various mechanical or moral theories implied rigid determination of human actions. But especially with the advent of the Christian religion the notion that humans can at some level make free choices—particularly about whether to do good or not—emerged as a foundational idea. (The idea had also arisen in Persian and Hebrew religions and legal systems, and was supported by Roman lawyers such as Cicero.) How this could be consistent with God having infinite power was not clear, although around 420 AD Augustine suggested that while God might have infinite knowledge of the future we as humans could not—yielding what can be viewed as a very rough analog of my explanation for free will. In the 1500s some early Protestants made theological arguments against free will—and indeed issues of free will remain a feature of controversy between Christian denominations even today.

In the mid-1600s philosophers such as Thomas Hobbes asserted that minds operate according to definite mechanisms and therefore cannot exhibit free will. In the late 1700s philosophers such as Immanuel Kant—agreeing with earlier work by Gottfried Leibniz—claimed instead that at least some parts of our minds are free and not determined by definite laws. But soon thereafter scientists like Pierre-Simon Laplace began to argue for determinism throughout the universe based on mathematical laws. And with the increasing success of science in the 1800s it came to be widely believed that there must be definite laws for all human actions—providing a foundation for the development of psychology and the social sciences.

In the early 1900s historians and economists emphasized that there were at least not simple laws for various aspects of human behavior. But it was nevertheless typically assumed that methods based on physics would eventually yield deterministic laws for human behavior—and this was for example part of the inspiration for the behaviorist movement in psychology in the mid-1900s. The advent of quantum mechanics in the 1920s, however, showed that even physics might not be entirely deterministic—and by the 1940s the possibility that this might lead to human free will was being discussed by physicists, philosophers and historians. Around this time Karl Popper used both quantum mechanics and sensitive dependence on initial conditions (see also page 971) to argue for fundamental indeterminism. And also around this time Friedrich Hayek (following ideas of Ludwig Mises in the early 1900s) suggested—presumably influenced by work in mathematical logic—that human behavior might be fundamentally unpredictable because in effect brains can explain only systems simpler than themselves, and can thus never explain their own operation. But while this has some similarity to the ideas of computational irreducibility in this book it appears never to have been widely studied.

Questions of free will and responsibility have been widely discussed in criminal and other law since at least the 1800s (see note below). In the 1960s and 1970s ideas from popular psychology tended to diminish the importance of free will relative to physiology or environment and experiences. In the 1980s, however, free will was increasingly attributed to animals other than humans. Free will for computers and robots was discussed in the 1950s in science fiction and to some extent in the field of cybernetics. But following lack of success in artificial intelligence it has for the most part not been seriously studied. Sometimes it is claimed that Gödel's Theorem shows that humans cannot follow definite rules—but I argue on page 1158 that this is not correct.

▪ **Determinism in brains.** Early investigations of internal functioning in the brain tended to suggest considerable randomness—say in the sequence of electrical pulses from a nerve cell. But in recent years, with more extensive measurement methods, there has been increasing evidence for precise deterministic underlying rules. (See pages 976 and 1011.)

▪ **Amounts of free will.** In my theory the amount of free will associated with a particular decision is in effect related to the amount of computation required to arrive at it. In conscious thinking we can to some extent scrutinize the processes we use, and assess how much computation they involve. But in unconscious thinking we cannot. And probably often these just involve memory lookups with rather little computation. But other unconscious abilities like intuition presumably involve more sophisticated computation.

■ **Responsibility.** It is often assumed that if there are definite underlying rules for our brains then it cannot be meaningful to say that we have any ultimate moral or legal responsibility for our actions. For traditional ideas lead to the notion that in this case all our actions must somehow be thought of as the direct result of whatever external causes (over which we have no control) are responsible for the underlying rules in our brains and the environment in which we find ourselves. But if the processes in our brains are computationally irreducible then as discussed in the main text their outcome can seem in many respects free of underlying rules, making it reasonable to view the processes themselves as what is really responsible for our actions. And since these processes are intrinsic to us, it makes sense to treat us as responsible for their effects.

Several different theories are used in practical legal systems. The theory popular from the behavioral sciences tends to assume that human actions can be understood from underlying rules for the brain, and that people should be dealt with according to the rules they have—which can perhaps be modified by some form of treatment. But computational irreducibility can make it essentially impossible to find what general behavior will arise from particular rules—making it difficult to apply this theory. The alternative pragmatic theory popular in rational philosophy and economics suggests that behavior in legal matters is determined through calculations based on laws and the deterrents they provide. But here there is the issue that computational irreducibility can make it impossible to foresee what consequences a given law will have. Western systems of law tend to be dominated by the moral theory that people should somehow get what they deserve for choices they made with free will—and my explanation now makes this consistent with the existence of definite underlying rules for the brain.

Young children, animals and the insane are typically held less responsible for their actions. And in a moral theory of law this can be understood in my approach as a consequence of the computations they do being less sophisticated—so that their outcome is less free of the environment and of their underlying rules. (In a pragmatic theory the explanation would presumably be that less sophisticated computations would not be up to the task of handling the elaborate system of incentives that laws had defined.)

■ **Will and purpose.** Things that are too predictable do not normally seem free. But things that are too random also do not normally seem to be associated with the exercise of a will. Thus for example continual random twitching in our muscles is not normally thought to be a matter of human will, even though some of it is the result of signals from our brains. For typically one imagines that if something is to be a genuine reflection of human will then there must be some purpose to it. In general it is very difficult to assess whether something has a purpose (see page 829). But in capturing the most obvious aspects of human will what seems to be most important is at least short-term coherence and consistency of action—as often exists in class 4, but not class 3, systems.

■ **Source of will.** Damage to a human brain can lead to apparent disappearance of the will to act, and there is some evidence that one small part of the brain is what is crucial.

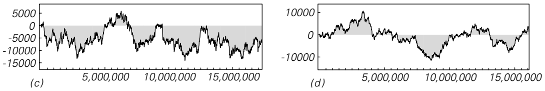
Undecidability and Intractability

■ **History.** In the early 1900s, particularly in the context of the ideas of David Hilbert, it was commonly believed that there should be a finite procedure to decide the truth of any mathematical statement. That this is not the case in the standard theory of arithmetic was in effect established by Kurt Gödel in 1931 (see page 1158). Alonzo Church gave the first explicit example of an undecidable problem in 1935 when he showed that no finite procedure in lambda calculus could guarantee to determine the equivalence of two lambda expressions. (A corollary to Gödel's proof had in fact already supplied another explicit undecidable problem by implying that no finite procedure based on recursive functions could decide whether a given primitive recursive function is identically 0.) In 1936 Alan Turing then showed that the halting problem for Turing machines could not be solved in general in a finite number of steps by any Turing machine. Some similar issues had already been considered by Emil Post in the context of tag and multiway systems starting in the 1920s, and in 1947 Post and Andrei Markov were able to establish that an existing mathematical question—the word problem for semigroups (see page 1141)—was undecidable. By the 1960s undecidability was being found in all sorts of systems, but most of the examples were too complicated to seem of much relevance in practical mathematics or computing. And apart from a few vague mentions in fields like psychology, undecidability was viewed mainly as a highly abstract curiosity of no importance to ordinary science. But in the early 1980s my experiments on cellular automata convinced me that undecidability is vastly more common than had been assumed, and in my 1984 paper

"Undecidability and intractability in theoretical physics" I argued that it should be important in many issues in physics and elsewhere.

■ **Mathematical impossibilities.** It is sometimes said that in the 1800s problems such as trisecting angles, squaring the circle, solving quintics, and integrating functions like $Exp[x^2]$ were proved mathematically impossible. But what was actually done was just to show that these problems could not be solved in terms of particular levels of mathematical constructs—say square roots (as in ruler and compass constructions discussed on page 1129), arbitrary roots, or elementary transcendental functions. And in each case higher mathematical constructs that seem in some sense no less implementable immediately allow the problems to be solved. Yet with undecidability one believes that there is absolutely no construct that can explicitly exist in our universe that allows the problem to be solved in any finite way. And unlike traditional mathematical impossibilities, undecidability is normally formulated purely in terms of ordinary integers—making it in a sense necessary to collapse basic distinctions between finite and infinite quantities if any higher-level constructs are to be included.

■ **Page 755 · Code 1004600.** In cases (c) and (d) steady growth at about 0.035 and 0.039 cells per step (of which 28% on average are non-white) is seen up to at least 20 million steps, though there continue to be fluctuations as shown below.



■ **Halting problems.** A classic example of a problem that is known in general to be undecidable is whether a given Turing machine will ever halt when started from a given initial condition. Halting is usually defined by the head of the Turing machine reaching a special halt state. But other criteria can equally well be used—say the head reaching a particular position (see page 759), or a certain pattern of colors being formed on the tape. And in a system like a cellular automaton a halting problem can be set up by asking whether a cell at a particular position ever turns a particular color, or whether, more globally, the complete state of the system ever reaches a fixed point and no longer changes.

In practical computing, one usually thinks of computational programs as being set up much like the register machines of page 896 and halting when they have finished executing their instructions. User interface and operating system programs are not normally intended to halt in an explicit sense, although without external input they often reach states that do not change. *Mathematica* works by taking its input and repeatedly applying transformation rules—a process which normally reaches a fixed point that is returned as the answer, but with definitions like $x = x + 1$ (x having no value) formally does not.

■ **Proofs of undecidability.** Essentially the same argument due to Alan Turing used on page 1128 to show that most numbers cannot be computable can also be used to show that most problems cannot be decidable. For a problem can be thought of as an infinite list of solutions for successive possible inputs. But this is analogous to a digit sequence of a real number. And since any program for a universal system can be specified by an integer it follows that there must be many problems for which no such program can be given.

To show that a particular problem like the halting problem is undecidable one typically argues by contradiction, setting up analogs of self-referential logic paradoxes such as "this statement is false". Suppose that one had a Turing machine m that could solve the halting problem, in the sense that it itself would always halt after a finite number of steps, but it would determine whether any Turing machine whose description it was given as input would ever halt. One way to see that this is not possible is to imagine modifying m to make a machine m' that halts if its input corresponds to a machine that does not halt, but otherwise goes into an infinite loop and does not itself halt. For if one considers feeding m' as input to itself there is immediately no consistent answer to the question of whether m' halts—leading to the conclusion that in fact no machine m could ever exist in the first place. (To make the proof rigorous one must add another level of self-reference, say setting up m' to ask m whether a Turing machine will halt when fed its own description as input.) In the main text I argued that undecidability is a consequence of universality. In the proof above universality is what guarantees that any Turing machine can successfully be described in a way that can be fed as input to another Turing machine.

■ **Page 756 · Examples of undecidability.** Once universality exists in a system it is known from Gordon Rice's 1953 theorem and its generalizations that most questions about ultimate behavior will be undecidable unless their answers are always trivially the same. Undecidability has been demonstrated in various seemingly rather different types of systems, most often by reduction to halting (termination) problems for multiway systems.

In formal language theory, questions about regular languages are always decidable, but ones about context-free languages (see page 1103) are already often not. It is decidable whether such a language is finite, but not whether it contains every possible string, is regular, is unambiguous, or is equivalent to a language with a different grammar.

In mathematical logic, it can be undecidable whether statements are provable from a given axiom system—say predicate logic or Peano arithmetic (see page 782). It is also undecidable whether one axiom system is equivalent to another—even for basic logic (see page 1170).

In algebra and related areas of mathematics problems of equivalence between objects built up from elements that satisfy relations are often in general undecidable. Examples are word problems for groups and semigroups (see page 1141), and equivalence of finitely specified 4D manifolds (see page 1051). (Equivalence for 3D manifolds is thought to be decidable.) A related undecidable problem is whether two integer matrices can be multiplied together in some sequence to yield the zero matrix. It is also undecidable whether two sets of relations specify the same group or semigroup.

In combinatorics it is known in general to be undecidable whether a given set of tiles can cover the plane (see page 1139). And from this follows the undecidability of various problems about 2D cellular automata (see note below) and spin systems. Also undecidable are many questions about whether strings exist that satisfy particular constraints (see below).

In number theory it is known to be undecidable whether Diophantine equations have solutions (i.e. whether algebraic equations have integer solutions) (see page 786). And this means for example that it is in general undecidable whether expressions that involve both algebraic and trigonometric functions can be zero for real values of variables, or what the values of integrals are in which such expressions appear as denominators (compare page 916).

In computer science, general problems about verifying the possible behavior of programs tend to be undecidable, usually being directly related to halting problems. It is also for example undecidable whether a given program is the shortest one that produces particular output (see page 1067).

It is in general undecidable whether a given system exhibits universality—or undecidability.

■ **Undecidability in cellular automata.** For 1D cellular automata, almost all questions about ultimate limiting behavior are undecidable, even ones that ask about average properties such as density and entropy. (This results in undecidability in classification schemes, as mentioned on page 948.) Questions about behavior after a finite number of steps, even with infinite initial conditions, tend to be decidable for 1D cellular automata, and related to regular languages (see page 957). In 2D cellular automata, however, even questions about a single step are often undecidable. Examples include whether any configurations are invariant under the cellular automaton evolution (see page 942), and, as established by Jarkko Kari in the late 1980s, whether the evolution is reversible, or can generate every possible configuration (see page 959).

■ **Natural systems.** Undecidable questions arise even in some traditional classes of models for natural systems. For example, in a generalized Ising model (see page 944) for a spin system the undecidability of the tiling problem implies that it is undecidable whether a given energy function leads to a phase transition in the infinite size limit. Somewhat similarly, the undecidability of equivalence of 4-manifolds implies undecidability of questions about quantum gravity models. In models based both on equations and other kinds of rules the existence of formulas for conserved quantities is in general undecidable. In models that involve continuous quantities it can be more difficult to formulate undecidability. But I strongly suspect that with appropriate definitions there is often undecidability in for example the three-body problem, so that the questions such as whether one of the bodies in a particular scattering process will ever escape to infinity are in general undecidable. In biology formal models for neural processes often involve undecidability, so that in principle it can be undecidable whether, say, there is any particular stimulus that will lead to a given response. Formal models for morphogenesis can also involve undecidability, so that for example it can in principle be undecidable whether a particular organism will ever stop growing, or whether a given structure can ever be formed in some class of organisms. (Compare page 407.)

■ **Undecidability in *Mathematica*.** In choosing functions to build into *Mathematica* I tried to avoid ones that would often encounter undecidability. And this is why for example there is no built-in function in *Mathematica* that tries to predict whether a given program will terminate. But inevitably functions like *FixedPoint*, *ReplaceRepeated* and *FullSimplify* can run into undecidability—so that ultimately they have to be limited by constructs such as *$IterationLimit* and *TimeConstraint*.

■ **Undecidability and sets.** Functions that can be computed in finite time by systems like Turing machines are often called recursive (or effectively computable). Sets are called recursive if there is a recursive function that can test whether or not any given element is in them. Sets are called recursively enumerable if there is a recursive function that can eventually generate any element in them.

The set of initial conditions for which a given Turing machine halts is thus not recursive. But it turns out that this set is recursively enumerable. And the reason is that one can generate the elements in it by effectively maintaining a copy of the Turing machine for each possible initial condition, then following a procedure where for example at step n one updates the one for initial condition *IntegerExponent[n, 2]*, and watches to see if it halts. Note that while the complement of a recursive set is always recursive, the complement of a recursively enumerable set may not be recursively enumerable. (An example is the set of initial conditions for which a Turing machines does not halt.) Recursively enumerable sets are characteristically associated with so-called Σ_1 statements of the form $\exists_t\ \phi[t]$ (where ϕ is recursive). (Asking whether a system ever halts is equivalent to asking whether there exists a number of steps t at which the system can be determined to be in its halting state.) Complements of recursively enumerable sets are characteristically associated with Π_1 statements of the form $\forall_t\ \phi[t]$—an example being whether a given system never halts. (Π_1 and Σ_1 statements are such that if they can be shown to be undecidable, then respectively they must be true or false, as discussed on page 1167.) If a statement in minimal form involves n alternations of $\exists$ and $\forall$ it is Σ_{n+1} if it starts with $\exists$ and Π_{n+1} if it starts with $\forall$. The Π_n and Σ_n form the so-called arithmetic hierarchy in which statements with larger n can be constructed by allowing ϕ to access an oracle for statements with smaller n (see page 1126). (Showing that a statement with $n \geq 1$ is undecidable does not establish that it is always true or always false.)

■ **Undecidability in tiling problems.** The question of whether a particular set of constraints like those on page 220 can be satisfied over the whole 2D plane is in general undecidable. For much as on page 943, one can imagine setting up a 1D cellular automaton with the property that, say, the absence of a particular color of cell throughout the 2D pattern formed by its evolution signifies satisfaction of the constraints. But even starting from a fixed line of cells, the question of whether a given color will ever occur in the evolution of a 1D cellular automaton is in general undecidable, as discussed in the main text. And although it is somewhat more difficult to show, this question remains undecidable even if one allows any possible configuration of cells on the starting line. (There are several different detailed formulations; the first explicit proof of undecidability in a tiling problem was given by Hao Wang in 1960; the version with no fixed cells by Robert Berger in 1966 by setting up an elaborate emulation of a register machine.) (See also page 943.)

■ **Page 757 · Correspondence systems.** Given a list of pairs *p* with *{u, v} = Transpose[p]* the constraint to be satisfied is

StringJoin[u⟦s⟧] == StringJoin[v⟦s⟧]

Thus for example *p = {{"ABB", "B"}, {"B", "BA"}, {"A", "B"}}* has shortest solution *s = {2, 3, 2, 2, 3, 2, 1, 1}*. (One can have lists instead of strings, replacing *StringJoin* by *Flatten*.)

Correspondence systems were introduced by Emil Post in 1945 to give simple examples of undecidability; he showed that the so-called Post Correspondence Problem (PCP) of satisfying their constraints is in general undecidable (see below). With 2 string pairs PCP was shown to be decidable in 1981. It is known to be undecidable when 9 pairs are used, but I strongly suspect that it is also undecidable with just 3 pairs. The undecidability of PCP has been used to establish undecidability of many problems related to groups, context-free languages, and other objects defined by relations (see page 1141). Finding PCP solutions shorter than a given length is known to be an NP-complete problem.

With r string pairs and *n = StringLength[StringJoin[p]]* there are 2^n *Binomial[n - 1, 2 r - 1]* possible constraints (assuming no strings of zero length), each being related to at most *8 r!* others by straightforward symmetries (or altogether 4^{n-1} for given n). The number of constraints which yield solutions of specified lengths *Length[s]* for $r = 2$ and $r = 3$ are as follows (the boxes at the end give the number of cases with no solution):

r = 2, n = 4	1:12	4							
r = 2, n = 5	1:64	64							
r = 2, n = 6	1:208	2:28	404						
r = 2, n = 7	1:640	3:32	1888						
r = 2, n = 8	1:1680	2:176	4:48	7056					
r = 2, n = 9	1:4352	3:112	5:56	24152					
r = 2, n = 10	1:10496	2:744	3:80	4:168	6:64	74464			
r = 3, n = 6	1:56	8							
r = 3, n = 7	1:576	192							
r = 3, n = 8	1:3312	2:168	3:84	1812					
r = 3, n = 9	1:14592	2:1140	3:192	4:288	5:96	8:48	30:48	44:48	12220
r = 3, n = 10	1:55296	2:4752	3:2712	4:372	5:492	6:264	7:216		
	12:24	18:48	24:48	36:48	75:48	78:48	64656		

With $r = 2$, as n increases an exponentially decreasing fraction of possible constraints have solutions; with $r = 3$ it appears that a fraction more than 1/4 continue to do so. With $r = 2$, it appears that if a solution exists, it must have length $n + 4$ or less. With $r \leq 3$, the longest minimal solution lengths for $n \leq 10$ are given above. (Allowing $r > 3$ yields no greater lengths for these values of n.) With $n = 11$, example (l) yields a solution of length 112. The only possible longer $n = 11$ case is *{{"AAB", "B"}, {"B", "A"}, {"A", "AABB"}}*, for which any possible solution must be longer than 200. With n = 12, {{"AABAAB", "B"}, {"B", "A"}, {"A", "AB"}} has minimal solution length 120 and {{"A", "AABB"}, {"AAB", "B"}, {"B", "AA"}} has minimal solution length 132.

A given constraint can fail to have a solution either because the colors of cells at some point cannot be made to match, or because the two strings can never have the same finite length (as in {{"A", "AA"}}). To know that a solution exists in a particular case, it is sufficient just to exhibit it. To know that no solution is possible of any length, one must in effect have a proof.

In general, one condition for a solution to exist is that integer numbers of pairs can yield strings of the same length, so that given the length differences d = Map[StringLength, p, {2}] . {1, -1} there is a vector v of non-negative integers such that v . d == 0. If only one color of element ever appears this is the complete condition for a solution—and for r = 2 solutions exist if Apply[Times, d] < 0 and are then of length at least Apply[Plus[##]/GCD[##] &, Abs[d]]. With two colors of elements additional conditions can be constructed involving counting elements of each color, or various blocks of elements.

The undecidability of PCP can be seen to follow from the undecidability of the halting problem through the fact that the question of whether a tag system of the kind on page 93 with initial sequence s ever reaches a halting state (where none of its rules apply) is equivalent to the question of whether there is a way to satisfy the PCP constraint

```
TSToPCP[{n_, rule_}, s_] :=
 Map[Flatten[IntegerDigits[#, 2, 2]] &, Module[{f}, f[u_] :=
    Flatten[Map[{1, #} &, 3 u]]; Join[Map[{f[Last[#]],
       RotateLeft[f[First[#]]]} &, rule], {{f[s], {1}}}, Flatten[
      Table[{{1, 2}, Append[RotateLeft[f[IntegerDigits[j, 2,
           i]]], 2]}, {i, 0, n - 1}, {j, 0, 2^i - 1}], 1]]], {2}]
```

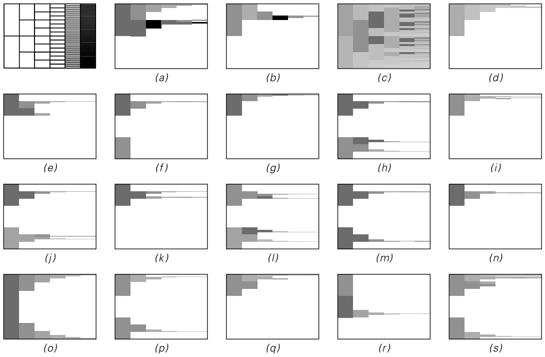
Any PCP constraint can also immediately be related to the evolution of a multiway tag system of the kind discussed in the note below. Assuming that the upper string is never shorter than the lower one, the rules for the relevant tag system are given simply by

```
Apply[Append[#2, s___] → Prepend[#1, s] &, p, {1}]
```

In the case of example (e) the existence of a solution of length 24 can then be seen to follow from the fact that MWTSEvolve[rule, {{"B"}}, 22] contains {"B", "A"}.

This correspondence with tag systems can be used in practice to search for PCP solutions, though it is usually most efficient to run tag systems that correspond both to moving forward and backward in the string, and to see whether their results ever agree. (In most PCP systems, including all the examples shown except (a) and (g), one string is always systematically longer than the other.) The tag system approach is normally limited by the number of intermediate strings that may need to be kept.

The pictures below show which possible sequences of up to 6 blocks yield upper and lower strings that agree in each of the PCP systems in the main text. As indicated in the first picture for the case of two blocks, each possible successively longer sequence corresponds to a rectangle in the picture (compare page 594). When a sequence of blocks leads to upper and lower strings that disagree, the rectangle is left white. If the strings agree so far, then the rectangle is colored with a gray that is darker if the strings are closer in length. Rectangles that are black (as visible in cases (a) and (b)) correspond to actual PCP solutions where the strings are the same length. Note that in case (c) the presence of only one color in either block means that strings will always agree so far. In cases (m) through (s) there is ultimately no solution, but as the pictures indicate, in these specific PCP systems there are always strings that agree as far as they have gone—it is just that they never end up the same length.



As one example of how one proves that a PCP constraint cannot be satisfied, consider case (s). From looking at the structure of the individual pairs one can see that if there is a solution it must begin with pair 1 or pair 3, and end with pair 1. But in fact it cannot begin with pair 1 because this would mean that the upper string would have to start off being longer, then at some point cross over to being shorter. However, the only way that such a crossover can occur is by pair 3 appearing with its upper A aligned with its second lower A. Yet starting with pair 1, the upper string is longer by 2 As, and the pairs are such that the length difference must always remain even—preventing the crossover from occurring. This means that any solution must begin with pair 3. But this pair must then be followed by another pair 3, which leaves BAAB sticking out on the bottom. So how can this *BAAB* be removed? The only way is to use the sequence of pairs 2, 3, 3, 2—yet doing this will just produce another *BAAB* further on. And thus one concludes that there is no way to satisfy these particular PCP constraints.

One can generalize PCP to allow any number of colors, and to require correspondence among any number of strings—though it is fairly easy to translate any such generalization to the 2-string 2-color case.

▪ **Multiway tag systems.** As an extension of ordinary multiway systems one can generalize tag systems from page 93 to allow a list of strings at each step. Representing the strings by lists, one can write rules in the form

```
{{1, 1, s___} → {s, 1, 0}, {1, s___} → {s, 1, 0, 1}}
```

so that the evolution is given by

```
MWTSEvolve[rule_, list_, t_] :=
  Nest[Flatten[Map[ReplaceList[#, rule] &, #], 1] &, list, t]
```

▪ **Word problems.** The question of whether a particular string can be generated in a given multiway system is an example of a so-called word problem. An original more specialized version of this was posed by Max Dehn in 1911 for groups and by Axel Thue in 1914 for semigroups. As discussed on page 938 a finitely presented group or semigroup can be viewed as a special case of a multiway system, in which the rules of the multiway system are obtained from relations between strings consisting of products of generators. The word problem then asks if a given product of such generators is equal to the identity element. Following work by Alan Turing in the mid-1930s, it was shown in 1947 by Emil Post from the undecidability of PCP that the word problem for semigroups is in general undecidable. Andrei Markov gave a specific example of this for a semigroup with 13 generators and 33 relations, and by 1966 Gennadií Makanin had found the simpler example

```
{"CCBB" ↔ "BBCC", "BCCCBB" ↔ "CBBBCC", "ACCBB" ↔ "BBA",
  "ABCCCBB" ↔ "CBBA", "BBCCBBBBCC" ↔ "BBCCBBBBCCA"}
```

Using these relations as rules for a multiway system most initial strings yield behavior that either dies out or becomes repetitive. The shortest initial strings that give unbounded growth are *"BBBBABB"* and *"BBBBBBA"*—though both of these still eventually yield just exponentially increasing numbers of distinct strings. In 1967 Yuri Matiyasevich constructed a semigroup with 3 complicated relations that has an undecidable word problem. It is not yet known whether undecidability can occur in a semigroup with a single relation. The word problem is known to be decidable for commutative semigroups.

The word problem for groups was shown to be undecidable in the mid-1950s by Petr Novikov and William Boone. There are however various classes of groups for which it is decidable. Abelian groups are one example. Another are so-called automatic groups, studied particularly in the 1980s, in which equivalence of words can be recognized by a finite automaton. (Such groups turn out to have definite geometrical properties, and are associated with spaces of negative curvature.) Even if a group ultimately has only a finite number of distinct elements, its word problem (with elements specified as products of generators) may still be undecidable. Constructions of groups with undecidable word problems have been based on setting up relations that correspond to the rules in a universal Turing machine. With the simplest such machine known in the past (see page 706) one gets a group with 32 generators and 142 relations. But with the universal Turing machine from page 707 one gets a group with 14 generators and 52 relations. (In general $s k + 4$ generators and $5 s k + 2$ relations are needed.) From the results in this book it seems likely that there are still much simpler examples—some of which could perhaps be found by setting up groups to emulate rule 110. Note that groups with just one relation were shown always to have decidable word problems by Wilhelm Magnus in 1932.

For ordinary multiway (semi-Thue) systems, an example with an undecidable word problem is known with 2 types of elements and 5 very complicated rules—but I am quite certain that much simpler examples are possible. (1-rule multiway systems always have decidable word problems.)

▪ **Sequence equations.** One can ask whether by replacing variables by sequences one can satisfy so-called word or string equations such as

```
Flatten[{x, 0, x, 0, y}] == Flatten[{y, x, 0, y, 1, 0, 1, 0, 0}]
```

(with shortest solution *x = {1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0}*, *y = {1, 0, 1, 0, 0, 1, 0, 1, 0, 0}*). Knowing about PCP and Diophantine equations one might expect that in general this would be undecidable. But in 1977 Gennadií Makanin gave a complicated algorithm that solves the problem completely in a finite number of steps (though in general triple exponential in the length of the equation).

▪ **Fast algorithms.** Most of the fast algorithms now known seem to fall into a few general classes. The most common are ones based on repetition or iteration, classic examples being Euclid's algorithm for *GCD* (page 915), Newton's method for *FindRoot* and the Gaussian elimination method for *LinearSolve*. Starting in the 1960s it began to be realized that fast algorithms could be based on nested or recursive processes, and such algorithms became increasingly popular in the 1980s. In most cases, the idea is recursively to divide data into parts, then to do operations on these parts, and finally reassemble the results. An example is the algorithm of Anatolii Karatsuba from 1961 for finding products of n-digit numbers (with $n = 2^s$) by operating on their digits in the nested pattern of page 608 (see also page 1093) according to

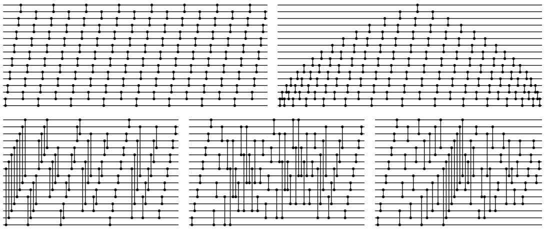
```
First[f[IntegerDigits[x, 2, n], IntegerDigits[y, 2, n], n/2]]

f[x_, y_, n_] :=
  If[n < 1, x y, g[Partition[x, n], Partition[y, n], n]]

g[{x1_, x0_}, {y1_, y0_}, n_] :=
  With[{z1 = f[x1, y1, n/2], z0 = f[x0, y0, n/2]},
    z1 2^(2 n) + (f[x0 + x1, y0 + y1, n/2] - z1 - z0) 2^n + z0]
```

Other examples include the fast Fourier transform (page 1074) and related algorithms for *ListConvolve*, the quicksort algorithm for *Sort*, and many algorithms in fields such as computational geometry. Starting in the 1980s fast algorithms based on randomized methods (see page 1192) have also become popular. But particularly from the discoveries in this book, it seems likely that the very fastest algorithms for many kinds of problems will not in the end have the type of regular structure that characterizes almost all algorithms currently used.

▪ Sorting networks. Any list can be sorted using *Fold[PairSort, list, pairs]* by doing a fixed sequence of comparisons of pairs

```
PairSort[a_, p : {_, _}] := Block[{t = a}, t[[p]] = Sort[t[[p]]]; t]
```

(Different comparisons often do not interfere and so can be done in parallel.) The pictures below show a few sequences of pair comparisons that sort lists of length $n = 16$.



The top two (both with 120 comparisons) have a repetitive structure and correspond to standard sorting algorithms: transposition sort and insertion sort. (Quicksort does not use a fixed sequence of comparisons.) The first one on the bottom (with 63 comparisons) has a nested structure and uses the method invented by Kenneth Batcher in 1964:

```
Flatten[Reverse[Flatten[With[{m = Ceiling[Log[2, n]] - 1},
    Table[With[{d = If[i == m, 2^t, 2^(i+1) - 2^t]}, Map[
        {0, d} + # &, Select[Range[n - d], BitAnd[# - 1, 2^t] ==
          If[i == m, 0, 2^t] &]]], {t, 0, m}, {i, t, m}]], 1]], 1]
```

The second one on the bottom also uses 63 comparisons, while the last one is the smallest known for $n = 16$: it uses 60 comparisons and was invented by Milton Green in 1969. For $n \le 16$ the smallest numbers of comparisons known to work are *{0, 1, 3, 5, 9, 12, 16, 19, 25, 29, 35, 39, 45, 51, 56, 60}*. (In general all lists will be sorted correctly if lists of just 0's and 1's are sorted correctly; allowing even just one of these 2^n cases to be wrong greatly reduces the number of comparisons needed.) For $n \le 8$ the Batcher method is known to give minimal length sequences of comparisons (for $n \le 5$ the total numbers of minimal sequences that work are *{1, 6, 3, 13866}*). The Batcher method in general requires about *n Log[n]*^{2} comparisons; it is known that in principle *n Log[n]* are sufficient. Various structures such as de Bruijn and Cayley graphs can be used as the basis for sorting networks, though it is my guess that typically the smallest networks for given n will have no obvious regularity. (See also page 832.)

▪ Page 758 · Computational complexity theory. Despite its rather general name, computational complexity theory has for the most part been concerned with the quite specific issue of characterizing how the computational resources needed to solve problems grow with input size. From knowing explicit algorithms many problems can be assigned to such classes as:

- NC: can be solved in a number of steps that increases like a polynomial in the logarithm of the input size if processing is done in parallel on a number of arbitrarily connected processors that increases like a polynomial in the input size. (Examples include addition and multiplication.)
- P (polynomial time): can be solved (with one processor) in a number of steps that increases like a polynomial in the input size. (Examples include evaluating standard mathematical functions and simulating the evolution of cellular automata and Turing machines.)
- NP (non-deterministic polynomial time): solutions can be checked in polynomial time. (Examples include many problems based on constraints as well as simulating the evolution of multiway systems and finding initial conditions that lead to given behavior in a cellular automaton.)
- PSPACE (polynomial space): can be solved with an amount of memory that increases like a polynomial in the input size. (Examples include finding repetition periods in systems of limited size.)

Central to computational complexity theory are a collection of hypotheses that imply that NC, P, NP and PSPACE form a strict hierarchy. At each level there are many problems known that are complete at that level in the sense that all other problems at that level can be translated to instances of that problem using only computations at a lower level. (Thus, for example, all problems in NP can be translated to instances of any given NP-complete problem using computations in P.)

▪ **History.** Ideas of characterizing problems by growth rates in the computational resources needed to solve them were discussed in the 1950s, notably in the context of operation counts for numerical calculations, sizes of circuits for switching and other applications, and theoretical lengths of proofs. In the 1960s such ideas were increasingly formalized, particularly for execution times on Turing machines, and in 1965 the suggestion was made that one should consider computations feasible if they take times that grow like polynomials in their input size. NP completeness (see below) was introduced by Stephen Cook in 1971 and Leonid Levin around the same time. And over the course of the 1970s a great many well-known problems were shown to be NP-complete. A variety of additional classes of computations—notably ones like NC with various kinds of parallelism, ones based on circuits and ones based on algebraic operations—were defined in the 1970s and 1980s, and many detailed results about them were found. In the 1980s much work was also done on the average difficulty of solving NP-complete problems—both exactly and approximately (see page 985). When computational complexity theory was at its height in the early 1980s it was widely believed that if a problem could be shown, for example, to be NP-complete then there was little chance of being able to work with it in a practical situation. But increasingly it became clear that general asymptotic results are often quite irrelevant in typical problems of reasonable size. And certainly pattern matching with __ in *Mathematica*, as well as polynomial manipulation functions like *GroebnerBasis*, routinely deal with problems that are formally NP-complete.

▪ **Lower bounds.** If one could prove for example that $\mathrm{P} \neq \mathrm{NP}$ then one would immediately have lower bounds on all NP-complete problems. But absent such a result most of the general lower bounds known so far are based on fairly straightforward information content arguments. One cannot for example sort n objects in less than about n steps since one must at least look at each object, and one cannot multiply two n-digit numbers in less than about n steps since one must at least look at each digit. (As it happens the fastest known algorithms for these problems require very close to n steps.) And if the output from a computation can be of size 2^n then this will normally take at least 2^n steps to generate. Subtleties in defining how big the input to a computation really is can lead to at least apparently exponential lower bounds. An example is testing whether one can match all possible sequences with a regular expression that involves s-fold repetitions. It is fairly clear that this cannot be done in less than about s steps. But this seems exponentially large if s is specified by its digit sequence in the original input regular expression. Similar issues arise in the problem of determining truth or falsity in Presburger arithmetic (see page 1152).

Diagonalization arguments analogous to those on pages 1128 and 1162 show that in principle there must exist functions that can be evaluated only by computations that exceed any given bound. But actually finding examples of such functions that can readily be described as having some useful purpose has in the past seemed essentially impossible.

If one sufficiently restricts the form of the underlying system then it sometimes becomes possible to establish meaningful lower bounds. For example, with deterministic finite automata (see page 957), there are sequences that can be recognized, but only by having exponentially many states. And with DNF Boolean expressions (see page 1096) functions like *Xor* are known to require exponentially many terms, even—as discovered in the 1980s—if any limited number of levels are allowed (see page 1096).

▪ **Algorithmic complexity theory.** Ordinary computational complexity theory asks about the resources needed to run programs that perform a given computation. But algorithmic complexity theory (compare page 1067) asks instead about how large the programs themselves need to be. The results of this book indicate however that even programs that are very small—and thus have low algorithmic complexity—can nevertheless perform all sorts of complex computations.

▪ **Turing machines.** The Turing machines used here in effect have tapes that extend only to the left, and have no explicit halt states. (They thus differ from the Turing machines which Marvin Minsky and Daniel Bobrow studied in 1961 in the $s = 2$, $k = 2$ case and concluded all had simple behavior.) One can think of each Turing machine as computing a function $f[x]$ of the number x given as its input. The function is total (i.e. defined for all x) if the Turing machine always halts; otherwise it is partial (and undefined for at least some x). Turing machines can be numbered according to the scheme on page 888. The number of steps before a machine with given rule halts can be computed from (see page 888)

```
Module[{s = 1, a, i = 1, d}, a[_] = 0; MapIndexed[a[#2[[1]]] =
    #1 &, Reverse[IntegerDigits[x, 2]]]; Do[{s, a[i], d} =
   {s, a[i]} /. rule; i -= d; If[i == 0, Return[t]], {t, tmax}]]
```

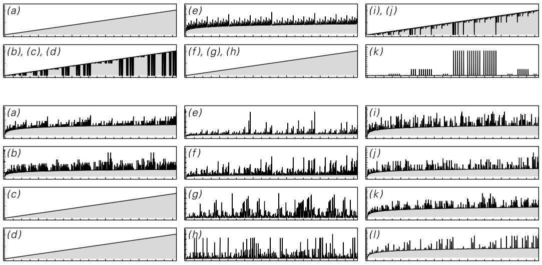
Of the 4096 Turing machines with $s = 2$, $k = 2$, 748 never halt, 3348 sometimes halt and 1683 always halt. (The most rarely halting are ones like machine 3112 that halt only when $x = 4j - 1$.) The number of distinct functions $f[x]$ that can be computed by such machines is 351, of which 149 are total. 17 machines compute $x + 1$; none compute $x + 2$; 17 compute $x - 1$ and do not halt when $x = 0$—an example being 2575. Most machines compute functions that involve digit manipulations without traditional interpretations as mathematical functions. It is quite common to find machines that compute almost the same function: 1507 and 1511 disagree (where 1507 halts) only for $x \geq 35$. If $t[x]$ is the number of steps to compute $f[x]$ then the number of distinct pairs $\{f[x], t[x]\}$ is 492, or 230 for total $f[x]$. In 164 $t[x]$ does not increase with the number of digits n in x, in 295 it increases linearly, in 27 quadratically, and in 6 exponentially. For total $f[x]$ the corresponding numbers are 84, 136, 7, 3; the 3 machines with exponential growth are 378 (example (f) on page 761), 1953 and 2289; all compute trivial functions. Machine 1447 (example (e)) computes the function which takes the digit sequence of x and replaces its first $3 + IntegerExponent[x + 1, 2]$ 0's by 1's.

Among the 2,985,984 Turing machines with $s = 3$, $k = 2$, at least 2,550,972 sometimes halt, and about 1,271,304 always do. The number of distinct functions that can be computed is about 36,392 (or 75,726 for $\{f[x], t[x]\}$ pairs). 8934 machines compute $x + 1$ (by 25 different methods, including ones like machine 164850 that take exponential steps), 14 compute $x + 2$, and none compute $x + 3$. Those machines that take times that grow precisely like 2^n all tend to compute very straightforward functions which can be computed much faster by other machines.

Among the 2,985,984 Turing machines with $s = 2$, $k = 3$, at least 2,760,721 sometimes halt, and about 974,595 always halt. The number of distinct functions that can be computed is about 315,959 (or 457,508 for $\{f[x], t[x]\}$ pairs). (The fact that there are far fewer distinct functions in the $s = 3$, $k = 2$ case is a consequence of equivalences between states but not colors.)

Among the 2^{32} Turing machines with $s = 4$, $k = 2$ about 80% at least sometimes halt, and about 16% always do. Still none compute $x + 3$. And no Turing machine of any size can directly compute a function like x^2, $2x$ or $Mod[x, 2]$ that involves manipulating all digits in x.

■ **Functions.** The plots below show the values of the functions $f[x]$ for x from 0 to 1023 computed by the Turing machines on pages 761 and 763. Many of the plots use logarithmic scales. Rarely are the values close to their absolute maximum $t[x]$.



■ **Machine 1507.** This machine shows in some ways the most complicated behavior of any $s = 2$, $k = 2$ Turing machine. As suggested by picture (k) it fails to halt if and only if its configuration at some step matches *{(0)..., {1, 1}, 1, ___}* (in the alternative form of page 888). For any input x one can test whether the machine will ever halt using

```
u[{Reverse[IntegerDigits[x, 2]], 0}]
u[list_] := v[Split[Flatten[list]]]
v[{a_, b_ : {}, c_ : {}, d_ : {}, e_ : {}, f_ : {}, g___}] :=
  Which[a == {1} || First[a] == 0, True, c == {}, False,
    EvenQ[Length[b]], u[{a, 1 - b, c, d, e, f, g}],
    EvenQ[Length[c]], u[{a, 1 - b, c, 1, Rest[d], e, f, g, 0}],
    e == {} || Length[d] ≥ Length[b] + Length[a] - 2,
    True, EvenQ[Length[e]], u[{a, b, c, d, f, g}],
    True, u[{a, 1 - b, c, 1 - d, e, 1, Rest[f], g, 0}]]
```

This test takes at most $n/3$ recursive steps, even though the original machine can take of order n^2 steps to halt. Among $s = 3$, $k = 2$ machines there are 314 machines that do the same computation as 1507, but none any faster.

■ **Page 763 · Properties.** The maximum numbers of steps increase with input size according to:

(a) $14\; 2\,\hat{}\,Floor[n/2] - 11 + 2\,Mod[n, 2]$

(b) (does not halt for $x = 1$)

(c) $2^n - 1$

(d) $(7\,(1 + Mod[n, 2])\,4\,\hat{}\,Floor[n/2] + 2\,Mod[n, 2] - 7)/3$

(h) (see note below)

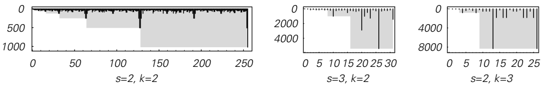
(i) (does not halt for various $x > 53$)

(j) (does not halt for various $x > 39$)

(k) (does not halt for $x = 1$)

(l) $5\,(2^{n-2} - 1)$

■ **Longest halting times.** The pictures below show the largest numbers of steps $t[x]$ that it takes any machine of a particular type to halt when given successive inputs x. For $s = 2$, $k = 2$ the largest results for all inputs of sizes 0 to 4 are $\{7, 17, 31, 49, 71\}$, all obtained with machine 1447. For $n > 4$ the largest results are $2^{n+2} - 3$, achieved for $x = 2^n - 1$ with machines 378 and 1351. For $s = 3$, $k = 2$ the largest results for successive sizes are $\{25, 53, 159, 179, 1021, 5419\}$ (often achieved by machine 600720; see below) and for $s = 2$, $k = 3$ $\{35, 83, 843, 8335\}$ (often achieved by machine 840971). Note the similarity to the busy beaver problem discussed on page 889.



■ **Growth rates.** Some Turing machine can always be found that has halting times that grow at any specified rate. (See page 103 for a symbolic system with halting times that grow like $Nest[2^{\#}\ \&, 0, n]$.) As discussed on page 1162, if the growth rate is too high then it may not be possible to prove that the machines halt using, say, the standard axioms of arithmetic. The maximum halting times above increase faster than the halting times for any specific Turing machine, and are therefore ultimately not computable by any single Turing machine.

■ **Machine 600720.** (Case (h) of page 763.) The maximum halting times for the first few sizes *n* are

```
{5, 159, 161, 1021, 5419, 315391,
   1978213883, 1978213885, 3018415453261}
```

These occur for inputs *{1, 2, 5, 10, 26, 34, 106, 213, 426}* and correspond to outputs (each themselves maximal for given *n*)

```
2^{3, 23, 24, 63, 148, 1148, 91148, 91149, 3560523} - 1
```

Such maxima often seem to occur when the input *x* has the form $(20\,4^s - 2)/3$ (and so has digits *{1, 1, 0, 1, 0, ..., 1, 0}*). The output *f[x]* in such cases is always $2^u - 1$ where

```
u = Nest[(13 + (6# + 8) (5/2)^
                  IntegerExponent[6# + 8, 2])/6 &, 1, s + 1]
```

One then finds that *6 u + 8* has the form *Nest[If[EvenQ[#], 5#/2, # + 21] &, 14, m]* for some *m*, suggesting a connection with the number theory systems of page 122. The corresponding halting time *t[x]* is *Last[Nest[h, {8, 4 s + 24}, s]] - 1* with

$$h[\{i\_, j\_\}] := With[\{e = IntegerExponent[3\,i + 4, 2]\}, \{13/6 + (i + 4/3)\,(5/2)^{e+1}, ((154 + 75\,(i + 4/3)\,(5/2)^{e})^2 - 16321 - 7860\,i - 900\,i^2 + 3360\,e)/3780 + j\}]$$

For $s > 3$ it then turns out that *f[x]* is extremely close to $3560523\,(5/2)^r$, and *t[x]* to $18865098979373\,(5/2)^{2r}$, for some integer *r*.

It is very difficult in general to find traditional formulas for *f[x]* and *t[x]*. But if *IntegerDigits[x, 2]* involves no consecutive 0's then for example *f[x]* can be obtained from

```
2^(b[Join[{1, 1}, #], Length[#]] &)[IntegerDigits[x, 2]] - 1

a[{l_, _}, r_] := ({l + (5r - 3#)/2, #} &)[Mod[r, 2]]

a[{l_, 0}, 0] := {l + 1, 0}

a[{l_, 1}, 0] :=
   ({(13 + # (5/2)^IntegerExponent[#, 2])/6, 0} &)[6 l + 2]

b[list_, i_] := First[Fold[a, {Apply[Plus, Drop[list, -i]], 0},
         Apply[Plus, Split[Take[list, -i], #1 == #2 ≠ 0 &], 1]]]
```

(The corresponding expression for *t[x]* is more complicated.) A few special cases are:

$$f[4\,s] = 4\,s + 3$$

$$f[4\,s + 1] = 2\,f[2\,s] + 1$$

$$f[2^s - 1] = 2^{(10\,s + 5 + 3\,(-1)^s)/4} - 1$$

How the halting times behave for large *n* is not clear. It is certainly possible that they could increase like $NestList[\#^2\ \&, 2, n]$, or 2^{2^n}, although for $x = (20\,4^s - 2)/3$ a better fit for $n \lesssim 200$ is just $2^{2.6\,n}$, with outputs increasing like $2^{2^{1.3\,n}}$.

■ **Page 766 · NP completeness.** Among the hundreds of problems known to be NP-complete are:

- Can a non-deterministic Turing machine reach a certain state in a given number of steps?
- Can a multiway system generate a certain string in a given number of steps?
- Is there an assignment of truth values to variables that makes a given Boolean expression true? (Satisfiability; related to minimal Boolean expressions of page 1095.)
- Will a given sequence of pair comparisons correctly sort any list (see page 1142)?
- Will a given pattern of origami folds yield an object that can be made flat?
- Does a network have any parts that match a given subnetwork (see page 1038)?
- Is there a path shorter than some given length that visits all of some set of points in the plane? (Travelling salesman; related to the network layout problem of page 1031.)
- Is there a solution of a certain size to an integer linear programming problem?
- Is there any $x < a$ such that $Mod[x^2, b] == c$? (See page 1090.)
- Does a matrix have a permanent of given value?
- Is there a way to satisfy tiling constraints in a finite region? (See page 984.)
- Is there a string of some limited length that solves a correspondence problem?
- Is there an initial condition to a cellular automaton that yields particular behavior after a given number of steps?

(In cases where numbers are involved, it is usually crucial that these be represented by base 2 digit sequences, and not, say, in unary.) Many NP-complete problems at first seem quite unrelated. But often their equivalence becomes clear just by straightforward identification of terms. And so for example the equivalence of satisfiability to problems about networks can be seen by identifying variables and clauses in Boolean expressions respectively with connections and nodes in networks.

One can get an idea of the threshold of NP completeness by looking at seemingly similar problems where one is NP-complete but the other is in P. Examples include:

- Finding a Hamiltonian circuit that visits once every connection in a given network is NP-complete, but finding an Euler circuit that visits once every node is in P.
- Finding the longest path between two nodes in a network is NP-complete, but finding the shortest path is in P.
- Determining satisfiability for a Boolean expression with 3 variables in each clause is NP-complete, but for one with 2 variables is in P. (The latter is like a network with only 2 connections at each node.)
- Solving quadratic Diophantine equations $a x^2 + b y == c$ is NP-complete, but solving linear ones $a x + b y == c$ is in P.
- Finding a minimum energy configuration for a 2D Ising spin glass in a magnetic field is NP-complete, but is in P if there is no magnetic field.
- Finding the permanent of a matrix is NP-complete, but finding its determinant is in P.

It is not known whether problems such as integer factoring or equivalence of networks under relabelling of nodes (graph isomorphism) are NP-complete. It is known that in principle there exist NP problems that are not in P, yet are not NP-complete.

■ **Natural systems.** Finding minimum energy configurations is formally NP-complete in standard models of natural systems such as folding protein and DNA molecules (see page 1003), collections of charges on a sphere (compare page 987), and finite regions of spin glasses (see page 944). As discussed on page 351, however, it seems likely that in nature true minima are very rare, and that instead what is usually seen are just the results of actual dynamical processes of evolution.

In quantum field theory and to a lesser extent quantum mechanics and celestial mechanics, approximation schemes based on perturbation series seem to require computations that grow very rapidly with order. But exactly what this implies about the underlying physical processes is not clear.

■ **P versus NP questions.** Most programs that are explicitly constructed to solve specific problems tend at some level to have rather simple behavior—often just repetitive or nested, so long as appropriate number representations are used. And it is this that makes it realistic to estimate asymptotic growth rates using traditional mathematics, and to determine whether the programs operate in polynomial time. But as the pictures on page 761 suggest, arbitrary computational systems—even Turing machines with very simple rules—can exhibit much more complicated behavior with no clear asymptotic growth rate. And indeed the question of whether the halting times for a system grow only like a power of input size is in general undecidable. And if one tries to prove a result about halting times using, say, standard axioms of arithmetic or set theory, one may find that the result is independent of those axioms. So this makes it far from clear that the general P = NP question has a definite answer within standard axiom systems of mathematics. If one day someone were to find a provably polynomial time algorithm that solves an NP-complete problem then this would establish that P = NP. But it could well be that the fastest programs for NP-complete problems behave in ways that are too complicated to prove much about using the standard axioms of mathematics.

■ **Non-deterministic Turing machines.** Generalizing rules from page 888 by making each right-hand side a list of possible outcomes, the list of configurations that can be reached after *t* steps is given by

```
NTMEvolve[rule_, inits_, t_Integer] := Nest[
   Union[Flatten[Map[NTMStep[rule, #] &, #], 1]] &, inits, t]
NTMStep[rule_List, {s_, a_, n_}] /; 1 ≤ n ≤ Length[a] :=
  Apply[{#1, ReplacePart[a, #2, n], n + #3} &,
    Replace[{s, a[[n]]}, rule], {1}]
```

■ **Page 767 · Implementation.** Given a non-deterministic Turing machine with rules in the form above, the rules for a cellular automaton which emulates it can be obtained from

```
NDTMToCA[tm_] := Flatten[{{_, h, _} → h, {s, _c, _} → e, {s,
  _, _} → s, {_, s, c[i_]} → s[i], {_, s, x_} → x, {a[_, _], _s, _} → s,
  {_, a[x_, y_], s[i_]} → a[x, y, i], {x_, _s, _} → x, {_, _, s[i_]} →
  s[i], Map[Table[With[{b = (#[[Min[Length[#], z]]] &)[
   {x, #} /. tm]}, If[Last[b] == -1, {{a[_], a[x, #, z], e} → h, {a[
   _], a[x, #, z], s} → a[x, #, z], {a[_], a[x, #, z], _} → a[b[[2]]],
  {a[x, #, z], a[w_], _} → a[b[[1]], w], {_, a[w_], a[x, #, z]} →
  a[w]}, {{a[_], a[x, #, z], _} → a[b[[2]]], {a[x, #, z], a[w_],
  _} → a[w], {_, a[w_], a[x, #, z]} → a[b[[1]], w]}]], {x,
  Max[Map[#[[1, 1]] &, tm]]}, {z, Max[Map[Length[#[[2]]] &,
  tm]]}] &, Union[Map[#[[1, 2]] &, tm]]], {_, x_, _} → x}]
```

■ **Page 768 · Satisfiability.** Given variables *𝕤[t, s]*, *𝕒[t, x, a]*, *𝕟[t, n]* representing whether at step *t* a non-deterministic Turing machine is in state *s*, the tape square at position *x* has color *a*, and the head is at position *n*, the following CNF expression represents the assertion that a Turing machine with *stot* states and *ktot* possible colors follows the specified rules and halts after at most *t* steps:

```
NDTMToCNF[rules_, {s_, a_, n_}, t_] :=
 {Table[Apply[Or, Table[𝕤[i, j], {j, stot}]], {i, t - 1}],
  Table[! 𝕤[i, j] || ! 𝕤[i, k], {i, 0, t - 1}, {j, stot}, {k, j + 1, stot}],
  Table[Apply[Or, Table[𝕟[i, j], {j, n + i, Max[0, n - i], -2}]],
  {i, 0, t}], Table[! 𝕟[i, j] || ! 𝕟[i, k], {i, 0, t}, {j, n + i, Max[0,
   n - i], -2}, {k, j + 2, n + i}], Table[Apply[Or, Table[𝕒[i, j, k],
   {k, 0, ktot - 1}]], {i, 0, t - 1}, {j, Max[1, n - i], n + i}],
  Table[! 𝕒[i, j, k] || ! 𝕒[i, j, m], {i, 0, t - 1}, {j, Max[1, n - i],
   n + i}, {k, 0, ktot - 1}, {m, k + 1, ktot - 1}], 𝕤[0, s],
Cases[MapIndexed[𝕒[Abs[n - First[#2]], First[#2], #1] &,
 a], 𝕒[x_, _, _] /; x < t], Table[𝕒[Abs[n - i], i, 0],
{i, Length[a] + 1, n + t - 1}], Table[! 𝕒[i, j, k] ||
 If [EvenQ[n + i - j], 𝕟[i, j], False] || 𝕒[i + 1, j, k], {i, 0, t - 2},
{j, Max[1, n - i], n + i}, {k, 0, ktot - 1}], Table[Map[Function[
 z, Outer[! 𝕟[i, j] || ! 𝕤[i, z[[1, 1]]] || ! 𝕒[i, j, z[[1, 2]]] || ## &,
  Apply[Sequence, Map[If [i < t - 1, {𝕤[i + 1, #[[1]]], 𝕟[
     i + 1, j - #[[3]]], 𝕒[i + 1, j, #[[2]]]}, {𝕟[i + 1, j - #[[3]]]}] &,
   z[[2]]]]]], rules], {i, 0, t - 1}, {j, n + i, Max[1, n - i], -2}],
Apply[Or, Table[𝕟[i, 0], {i, n, t, 2}]]]} /. List → And
```

▪ **Density of difficult problems.** There are arguments that in an asymptotic sense most instances chosen at random of problems like limited-size PCP or tiling will be difficult to solve. In a problem like satisfiability, however, difficult instances tend to occur only on the boundary between cases where the density of black or white squares implies that there is usually satisfaction or usually not satisfaction. If one looks at simple instances of problems (say PCP with short strings) then my experience is that many are easy to solve. But just as some fraction of cellular automata with very simple rules show immensely complex behavior, so similarly it seems that some fraction of even simple instances of many NP-complete problems also tend to be difficult to solve.

▪ **Page 770 · Rule 30 inversion.** The total numbers of sequences for *t* from 1 to 15 not yielding stripes of heights 1 and 2 are respectively

```
{1, 2, 2, 3, 3, 6, 6, 10, 16, 31, 52, 99, 165, 260}
```

```
{2, 5, 8, 14, 23, 40, 66, 111, 182,
  316, 540, 921, 1530, 2543, 4122}
```

The sideways evolution of rule 30 discussed on page 601 implies that if one fills cells from the left rather than the right then some sequence of length $t + 1$ will always yield any given stripe of height *t*.

If the evolution of rule 30 can be set up as on page 704 to emulate any Boolean function then the problem considered here is immediately equivalent to satisfiability.

▪ **Systems of limited size.** In the system *x → Mod[x + m, n]* from page 255 the repetition period *n/GCD[m, n]* can be computed using Euclid's algorithm in at most about *Log[GoldenRatio, n]* steps. In the system *x → Mod[2 x, n]* from page 257, the repetition period *MultiplicativeOrder[2, n]* probably cannot always be computed in any polynomial of *Log[n]* steps, since otherwise *FactorInteger[n]* could also be computed in about this number of steps. (But see note below.) In a cellular automaton with *n* cells, the problem of finding the repetition period is in general PSPACE-complete—as follows from the possibility of universality in the underlying cellular automaton. And even in a case like rule 30 I suspect that the period cannot be found much faster than by tracing nearly 2^n steps of evolution. (I know of no way for example to break the computation into parts that can be done in parallel.) With sufficiently simple behavior, a cellular automaton repetition period can readily be determined in some power of *Log[n]* steps. But even with an additive rule and nested behavior, the period depends on quantities like *MultiplicativeOrder[2, n]*, which probably take more like *n* steps to evaluate. (But see note below.)

▪ **Page 771 · Quantum computers.** In an ordinary classical setup one typically describes the state of something like a 2-color cellular automaton with *n* cells just by giving a list of *n* color values. But the standard formalism of quantum theory (see page 1058) implies that for an analogous quantum system—like a line of *n* quantum spins each either up or down—one instead has to give a whole vector of probability amplitudes for each of the 2^n possible complete underlying spin configurations. And these amplitudes a_i are assumed to be complex numbers with a continuous range of possible values, subject only to the conventional constraint of unit total probability $Sum[Abs[a_i]^2, \{i, 2^n\}] == 1$. The evolution of such a quantum system can then formally be represented by successive multiplication of the vector of amplitudes by appropriate $2^n \times 2^n$ unitary matrices.

In a classical system like a cellular automaton with *n* cells a probabilistic ensemble of states can similarly be described by a vector of 2^n probabilities p_i—now satisfying $Sum[p_i, \{i, 2^n\}] == 1$, and evolving by multiplication with $2^n \times 2^n$ matrices having a single *1* in each row. (If the system is reversible—as in the quantum case—then the matrices are invertible.) But even if one assumes that all 2^n states in the ensemble somehow manage to evolve in parallel, it is still fairly clear that to do reliable computations takes essentially as much effort as evolving single instances of the underlying system. For even though the vector of probabilities can formally give outcomes for 2^n different initial conditions, any specific individual outcome could have probability as small as 2^{-n}—and so would take 2^n trials on average to detect.

The idea of setting up quantum analogs of systems like Turing machines and cellular automata began to be pursued in the early 1980s by a number of people, including myself. At first it was not clear what idealizations to make, but by the late 1980s—especially through the work of David Deutsch—the concept had emerged that a quantum computer should be described in terms of a network of basic quantum gates. The idea was to have say *n* quantum spins (each representing a so-called qubit), then to do computations much like in the reversible logic systems of page 1097 or the sorting networks of page 1142 by applying some appropriate sequence of elementary operations. It was found to be sufficient to do operations on just one and two spins at a time, and in fact it was shown that any $2^n \times 2^n$ unitary matrix can be approximated arbitrarily closely by a suitable sequence of for example underlying 2-spin *{x, y} → {x, Mod[x + y, 2]}* operations (assuming values 0 and 1), together with 1-spin arbitrary phase change operations. Such phase changes can be produced by repeatedly applying a single irrational rotation, and using the fact that *Mod[h s, 2 π]* will eventually for some *s* come close to any given phase (see page 903). From the involvement of continuous numbers, one might at first imagine that it should be possible to do fundamentally more computations than can be done say in ordinary discrete cellular automata. But all the evidence is that—just as discussed on page 1128—this will not in fact be possible if one makes the assumption that at some level discrete must be used to set up the initial values of probability amplitudes.

From the fact that the basic evolution of an *n*-spin quantum system in effect involves superpositions of 2^n spin configurations one might however still imagine that in finite computations exponential speedups should be possible. And as a potential example, consider setting up a quantum computer that evaluates a given Boolean function—with its initial configurations of spins encoding possible inputs to the function, and the final configuration of a particular spin representing the output from the function. One might imagine that with such a computer it would be easy to solve the NP-complete problem of satisfiability from page 768: one would just start off with a superposition in which all 2^n possible inputs have equal amplitude, then look at whether the spin representing the output from the function has any amplitude to be in a particular configuration. But in an actual physical system one does not expect to be able to find values of amplitudes directly. For according to the standard formalism of quantum theory all amplitudes do is to determine probabilities for particular outcomes of measurements. And with the setup described, even if a particular function is ultimately satisfiable the probability for a single output spin to be measured say as up can be as little as 2^{-n}—requiring on average 2^n trials to distinguish from *0*, just as in the classical probabilistic case.

With a more elaborate setup, however, it appears sometimes to be possible to spread out quantum amplitudes so as to make different outcomes correspond to much larger probability differences. And indeed in 1994 Peter Shor found a way to do this so as to get quantum computers at least formally to factor integers of size *n* using resources only polynomial in *n*. As mentioned in the note above, it becomes straightforward to factor *m* if one can get the values of *MultiplicativeOrder[a, m]*. But these correspond to periodicities in the list *Mod[a ^ Range[m], m]*. Given *n* spins one can imagine using their 2^n possible configurations to represent each element of *Range[m]*. But now if one sets up a superposition of all these configurations, one can compute *Mod[a^#, m] &*, then essentially use *Fourier* to find periodicities—all with a polynomial number of quantum gates. And depending on *FactorInteger[m]* the resulting amplitudes show fairly large differences which can then be detected in the probabilities for different outcomes of measurements.

In the mid-1990s it was thought that quantum computers might perhaps give polynomial solutions to all NP problems. But in fact only a very few other examples were found—all ultimately based on very much the same ideas as factoring. And indeed it now seems decreasingly likely that quantum computers will give polynomial solutions to NP-complete problems. (Factoring is not known to be NP-complete.)

And even in the case of factoring there are questions about the idealizations used. It does appear that only modest precision is needed for the initial amplitudes. And it seems that perturbations from the environment can be overcome using versions of error-correcting codes. But it remains unclear just what might be needed actually to perform for example the final measurements required.

Simple physical versions of individual quantum gates have been built using particles localized for example in ion traps. But even modestly larger setups have been possible only in NMR and optical systems—which show formal similarities to quantum systems (and for example exhibit interference) but presumably do not have any unique quantum advantage. (There are other approaches to quantum computation that involve for example topology of 4D quantum fields. But it is difficult to see just what idealizations are realistic for these.)

▪ **Circuit complexity.** Any function with a fixed size of input can be computed by a circuit of the kind shown on page 619. How the minimal size or depth of circuit needed grows with input size then gives a measure of the difficulty of the computation, with circuit depth growing roughly like number of steps for a Turing machine. Note that much as on page 662 one can construct universal circuits that can be arranged by appropriate choice of parts of their input to compute any function of a given input size. (Compare page 703.)

▪ **Page 771 · Finding outcomes.** If one sets up a function to compute the outcome after *t* steps of evolution from some fixed initial condition—say a single black cell in a cellular automaton—then the input to this function need contain only *Log[2, t]* digits. But if the evolution is computationally irreducible then to find its outcome will involve explicitly following each of its *t* steps—thereby effectively finding results for each of the *2 ^ Log[2, t]* possible arrangements of digits corresponding to numbers less than *t*. Note that the computation that is involved is not necessarily in either NP or PSPACE.

▪ **P completeness.** If one allows arbitrary initial conditions in a cellular automaton with nearest-neighbor rules, then to compute the color of a particular cell after *t* steps in general requires specifying as input the colors of all *2 t + 1* initial cells up to distance *t* away (see page 960). And if one always does computations using systems that have only nearest-neighbor rules then just combining *2 t + 1* bits of information can take up to *t* steps—even if the bits are combined in a way that is not computationally irreducible. So to avoid this one can consider systems that are more like circuits in which any element can get data from any other. And given *t* elements operating in parallel one can consider the class NC studied by Nicholas Pippenger in 1978 of computations that can be done in a number of steps that is at most some power of *Log[t]*. Among such computations are *Plus*, *Times*, *Divide*, *Det* and *LinearSolve* for integers, as well as determining outcomes in additive cellular automata (see page 609). But I strongly suspect that computational irreducibility prevents outcomes in systems like rule 30 and rule 110 from being found by computations that are in NC—implying in effect that allowing arbitrary connections does not help much in computing the evolution of such systems. There is no way yet known to establish this for certain, but just as with NP and P one can consider showing that a computation is P-complete with respect to transformations in NC. It turns out that finding the outcome of evolution in any standard universal Turing machine or cellular automaton is P-complete in this sense, since the process of emulating any such system by any other one is in NC. Results from the mid-1970s established that finding the output from an arbitrary circuit with *And* or *Or* gates is P-complete, and this has made it possible to show that finding the outcome of evolution in various systems not yet known to be universal is P-complete. A notable example due to Cristopher Moore from 1996 is the 3D majority cellular automaton with rule *UnitStep[a + AxesTotal[a, 3] - 4]* (see page 927); another example is the Ising model cellular automaton from page 982.

Implications for Mathematics and Its Foundations

▪ **History.** Babylonian and Egyptian mathematics emphasized arithmetic and the idea of explicit calculation. But Greek mathematics tended to focus on geometry, and increasingly relied on getting results by formal deduction. For being unable to draw geometrical figures with infinite accuracy this seemed the only way to establish anything with certainty. And when Euclid around 330 BC did his work on geometry he started from 10 axioms (5 “common notions” and 5 “postulates”) and derived 465 theorems. Euclid’s work was widely studied for more than two millennia and viewed as a quintessential example of deductive thinking. But in arithmetic and algebra—which in effect dealt mostly with discrete entities—a largely calculational approach was still used. In the 1600s and 1700s, however, the development of calculus and notions of continuous functions made use of more deductive methods. Often the basic concepts were somewhat vague, and by the mid-1800s, as mathematics became more elaborate and abstract, it became clear that to get systematically correct results a more rigid formal structure would be needed.

The introduction of non-Euclidean geometry in the 1820s, followed by various forms of abstract algebra in the mid-1800s, and transfinite numbers in the 1880s, indicated that mathematics could be done with abstract structures that had no obvious connection to everyday intuition. Set theory and predicate logic were proposed as ultimate foundations for all of mathematics (see note below). But at the very end of the 1800s paradoxes were discovered in these approaches. And there followed an increasing effort—notably by David Hilbert—to show that everything in mathematics could consistently be derived just by starting from axioms and then using formal processes of proof.

Gödel’s Theorem showed in 1931 that at some level this approach was flawed. But by the 1930s pure mathematics had already firmly defined itself to be based on the notion of doing proofs—and indeed for the most part continues to do so even today (see page 859). In recent years, however, the increasing use of explicit computation has made proof less important, at least in most applications of mathematics.

▪ **Models of mathematics.** Gottfried Leibniz’s notion in the late 1600s of a “universal language” in which arguments in mathematics and elsewhere could be checked with logic can be viewed as an early idealization of mathematics. Starting in 1879 with his “formula language” (*Begriffsschrift*) Gottlob Frege followed a somewhat similar direction, suggesting that arithmetic and from there all of mathematics could be built up from predicate logic, and later an analog of set theory. In the 1890s Giuseppe Peano in his *Formulario* project organized a large body of mathematics into an axiomatic framework involving logic and set theory. Then starting in 1910 Alfred Whitehead and Bertrand Russell in their *Principia Mathematica* attempted to derive many areas of mathematics from foundations of logic and set theory. And although its methods were flawed and its notation obscure this work did much to establish the idea that mathematics could be built up in a uniform way.

Starting in the late 1800s, particularly with the work of Gottlob Frege and David Hilbert, there was increasing interest in so-called metamathematics, and in trying to treat mathematical proofs like other objects in mathematics. This led in the 1920s and 1930s to the introduction of various idealizations for mathematics—notably recursive functions, combinators, lambda calculus, string rewriting systems and Turing machines. All of these were ultimately shown to be universal (see page 784) and thus in a sense capable of reproducing any mathematical system. String rewriting systems—as studied particularly by Emil Post—are close to the multiway systems that I use in this section (see page 938).

Largely independent of mathematical logic the success of abstract algebra led by the end of the 1800s to the notion that any mathematical system could be represented in algebraic terms—much as in the operator systems of this section. Alfred Whitehead to some extent captured this in his 1898 *Universal Algebra*, but it was not until the 1930s that the theory of structures emphasized commonality in the axioms for different fields of mathematics—an idea taken further in the 1940s by category theory (and later by topos theory). And following the work of the Bourbaki group beginning at the end of the 1930s it has become almost universally accepted that structures together with set theory are the appropriate framework for all of pure mathematics.

But in fact the *Mathematica* language released in 1988 is now finally a serious alternative. For while it emphasizes calculation rather than proof its symbolic expressions and transformation rules provide an extremely general way to represent mathematical objects and operations—as for example the notes to this book illustrate.

(See also page 1176.)

■ **Page 773 · Axiom systems.** In the main text I argue that there are many consequences of axiom systems that are quite independent of their details. But in giving the specific axiom systems that have been used in traditional mathematics one needs to take account of all sorts of fairly complicated details.

As indicated by the tabs in the picture, there is a hierarchy to axiom systems in traditional mathematics, with those for basic and predicate logic, for example, being included in all others. (Contrary to usual belief my results strongly suggest however that the presence of logic is not in fact essential to many overall properties of axiom systems.)

As discussed in the main text (see also page 1155) one can think of axioms as giving rules for transforming symbolic expressions—much like rules in *Mathematica*. And at a fundamental level all that matters for such transformations is the structure of expressions. So notation like $a+b$ and $a \times b$, while convenient for interpretation, could equally well be replaced by more generic forms such as $f[a, b]$ or $g[a, b]$ without affecting any of the actual operation of the axioms.

My presentation of axiom systems generally follows the conventions of standard mathematical literature. But by making various details explicit I have been able to put all axiom systems in forms that can be used almost directly in *Mathematica*. Several steps are still necessary though to get the actual rules corresponding to each axiom system. First, the definitions at the top of page 774 must be used to expand out various pieces of notation. In basic logic I use the notation $u = v$ to stand for the pair of rules $u \to v$ and $v \to u$. (Note that $=$ has the precedence of $\to$ not $==$.) In predicate logic the tab at the top specifies how to construct rules (which in this case are often called rules of inference, as discussed on page 1155). $x\_ \wedge y\_ \to x\_$ is the *modus ponens* or detachment rule (see page 1155). $x\_ \to \forall_{y\_}\, x\_$ is the generalization rule. $x\_ \to x\_ \wedge \# \&$ is applied to the axioms given to get a list of rules. Note that while $=$ in basic logic is used in the underlying construction of rules, $==$ in predicate logic is just an abstract operator with properties defined by the last two axioms given.

As is typical in mathematical logic, there are some subtleties associated with variables. In the axioms of basic logic literal variables like a must be replaced with patterns like $a\_$ that can stand for any expression. A rule like $a\_ \wedge (b\_ \vee \neg b\_) \to a$ can then immediately be applied to part of an expression using *Replace*. But to apply a rule like $a\_ \to a \wedge (b \vee \neg b)$ requires in effect choosing some new expression for b (see page 1155). And one way to represent this process is just to have the pattern $a\_ \to a\_ \wedge (b\_ \vee \neg b\_)$ and then to say that any actual rule that can be used must match this pattern. The rules given in the tab for predicate logic work the same way. Note, however, that in predicate logic the expressions that appear on each side of any rule are required to be so-called well-formed formulas (WFFs) consisting of variables (such as a) and constants (such as 0 or $\emptyset$) inside any number of layers of functions (such as $+$, $\cdot$, or ⊿) inside a layer of predicates (such as $==$ or $\in$) inside any number of layers of logical connectives (such as $\wedge$ or $\Rightarrow$) or quantifiers (such as $\forall$ or $\exists$). (This setup is reflected in the grammar of the *Mathematica* language, where the operator precedences for functions are higher than for predicates, which are in turn higher than for quantifiers and logical connectives—thus yielding for example few parentheses in the presentation of axiom systems here.)

In basic logic any rule can be applied to any part of any expression. But in predicate logic rules can be applied only to whole expressions, always in effect using *Replace[expr, rules]*. The axioms below (devised by Matthew Szudzik as part of the development of this book) set up basic logic in this way.

$(a \lor b) \lor c = (b \lor a) \lor c$	$a \lor ((b \land c) \land d) = a \lor ((c \land b) \land d)$
$((a \lor b) \lor c) \lor d = (a \lor (b \lor c)) \lor d$	$a \lor (((b \land c) \land d) \land e) = a \lor ((b \land (c \land d)) \land e)$
$a \lor (b \land (c \land \neg c)) = a$	$a \lor (b \land (c \lor \neg c)) = a \lor b$
$a \lor (b \lor (c \land d)) = a \lor ((b \lor c) \land (b \lor d))$	$a \lor (b \land (c \lor d)) = a \lor ((b \land c) \lor (b \land d))$
$a \lor (b \land \neg (c \lor d)) = a \lor (b \land (\neg c \land \neg d))$	$a \lor (b \land \neg (c \land d)) = a \lor (b \land (\neg c \lor \neg d))$
$a \lor (b \land c) = a \lor (b \land \neg \neg c)$	

■ **Basic logic.** The formal study of logic began in antiquity (see page 1099), with verbal descriptions of many templates for valid arguments—corresponding to theorems of logic—being widely known by medieval times. Following ideas of abstract algebra from the early 1800s, the work of George Boole around 1847 introduced the notion of representing logic in a purely symbolic and algebraic way. (Related notions had been considered by Gottfried Leibniz in the 1680s.) Boole identified *1* with *True* and *0* with *False*, then noted that theorems in logic could be stated as equations in which *Or* is roughly *Plus* and *And* is *Times*—and that such equations can be manipulated by algebraic means. Boole's work was progressively clarified and simplified, notably by Ernst Schröder, and by around 1900, explicit axiom systems for Boolean algebra were being given. Often they included most of the 14 highlighted theorems of page 817, but slight simplifications led for example to the "standard version" of page 773. (Note that the duality between *And* and *Or* is no longer explicit here.) The "Huntington version" of page 773 was given by Edward Huntington in 1933, along with

$$\{(\neg \neg a) = a,\ (a \lor \neg (b \lor \neg b)) = a,$$
$$(\neg (\neg (a \lor \neg b) \lor \neg (a \lor \neg c))) = (a \lor \neg (b \lor c))\}$$

The "Robbins version" was suggested by Herbert Robbins shortly thereafter, but only finally proved correct in 1996 by William McCune using automated theorem proving (see page 1157). The "Sheffer version" based on *Nand* (see page 1173) was given by Henry Sheffer in 1913. The shorter version was devised by David Hillman as part of the development of this book. The shortest version is discussed on page 808. (See also page 1175.)

In the main text each axiom defines an equivalence between expressions. The tradition in philosophy and mathematical logic has more been to take axioms to be true statements from which others can be deduced by the *modus ponens* inference rule $\{x, x \Rightarrow y\} \to y$ (see page 1155). In 1879 Gottlob Frege used his diagrammatic notation to set up a symbolic representation for logic on the basis of the axioms

$$\{a \Rightarrow (b \Rightarrow a),\ (a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c)),$$
$$(a \Rightarrow (b \Rightarrow c)) \Rightarrow (b \Rightarrow (a \Rightarrow c)),$$
$$(a \Rightarrow b) \Rightarrow ((\neg b) \Rightarrow (\neg a)),\ (\neg \neg a) \Rightarrow a,\ a \Rightarrow (\neg \neg a)\}$$

Charles Peirce did something similar at almost the same time, and by 1900 this approach to so-called propositional or sentential calculus was well established. (Alfred Whitehead and Bertrand Russell used an axiom system based on *Or* and *Not* in their original 1910 edition of *Principia Mathematica*.) In 1948 Jan Łukasiewicz found the single axiom version

$$\{((a \Rightarrow (b \Rightarrow a)) \Rightarrow ((((\neg c) \Rightarrow (d \Rightarrow (\neg e))) \Rightarrow ((c \Rightarrow (d \Rightarrow f)) \Rightarrow$$
$$((e \Rightarrow d) \Rightarrow (e \Rightarrow f)))) \Rightarrow g)) \Rightarrow (h \Rightarrow g)\}$$

equivalent for example to

$$\{((\neg a) \Rightarrow (b \Rightarrow (\neg c))) \Rightarrow$$
$$((a \Rightarrow (b \Rightarrow d)) \Rightarrow ((c \Rightarrow b) \Rightarrow (c \Rightarrow d))),\ a \Rightarrow (b \Rightarrow a)\}$$

It turns out to be possible to convert any axiom system that works with *modus ponens* (and supports the properties of $\Rightarrow$) into a so-called equational one that works with equivalences between expressions by using

```
Module[{a}, Join[Thread[axioms == a ⇒ a],
    {((a ⇒ a) ⇒ b) == b, ((a ⇒ b) ⇒ b) == (b ⇒ a) ⇒ a}]]
```

An analog of *modus ponens* for *Nand* is $\{x, x \barwedge (y \barwedge z)\} \to z$, and with this Jean Nicod found in 1917 the single axiom

$$\{(a \barwedge (b \barwedge c)) \barwedge ((e \barwedge (e \barwedge e)) \barwedge ((d \barwedge b) \barwedge ((a \barwedge d) \barwedge (a \barwedge d))))\}$$

which was highlighted in the 1925 edition of *Principia Mathematica*. In 1931 Mordechaj Wajsberg found the slightly simpler

$$\{(a \barwedge (b \barwedge c)) \barwedge (((d \barwedge c) \barwedge ((a \barwedge d) \barwedge (a \barwedge d))) \barwedge (a \barwedge (a \barwedge b)))\}$$

Such an axiom system can be converted to an equational one using

```
Module[{a}, With[{t = a ⊼ (a ⊼ a), i = #1 ⊼ (#2 ⊼ #2) &},
    Join[Thread[axioms == t], {i[t ⊼ (b ⊼ c), c] == t,
        i[t, b] == b, i[i[a, b], b] == i[i[b, a], a]}]]]
```

but then involves 4 axioms.

The question of whether any particular statement in basic logic is true or false is always formally decidable, although in general it is NP-complete (see page 768).

■ **Predicate logic.** Basic logic in effect concerns itself with whole statements (or "propositions") that are each either *True* or *False*. Predicate logic on the other hand takes into account how such statements are built up from other constructs—like those in mathematics. A simple statement in predicate logic is $\forall_x (\forall_y x == y) \lor \forall_x (\exists_y (\neg x == y))$, where $\forall$ is "for all" and $\exists$ is "there exists" (defined in terms of $\forall$ on page 774)—and this particular statement can be proved *True* from the axioms. In general statements in predicate logic can contain arbitrary so-called predicates, say *p[x]* or *r[x, y]*, that are each either *True* or *False* for given x and y. When predicate logic is used as part of other axiom systems, there are typically axioms which define properties of the predicates. (In real algebra, for example, the predicate $>$ satisfies $a > b \Rightarrow a \neq b$.) But in pure predicate logic the predicates are not assumed to have any particular properties.

Notions of quantifiers like $\forall$ and $\exists$ were already discussed in antiquity, particularly in the context of syllogisms. The first explicit formulation of predicate logic was given by Gottlob Frege in 1879, and by the 1920s predicate logic had become widely accepted as a basis for mathematical axiom systems. (Predicate logic has sometimes also been used as a model for general reasoning—and particularly in the 1980s was the basis for several initiatives in artificial intelligence. But for the most part it has turned out to be too rigid to capture directly typical everyday reasoning processes.)

Monadic pure predicate logic—in which predicates always take only a single argument—reduces in effect to basic logic and is not universal. But as soon as there is even one arbitrary predicate with two arguments the system becomes universal (see page 784). And indeed this is the case even if one considers only statements with quantifiers $\forall\ \exists\ \forall$. (The system is also universal with one two-argument function or two one-argument functions.)

In basic logic any statement that is true for all possible assignments of truth values to variables can always be proved from the axioms of basic logic. In 1930 Kurt Gödel showed a similar result for pure predicate logic: that any statement that is true for all possible explicit values of variables and all possible forms of predicates can always be proved from the axioms of predicate logic. (This is often called Gödel's Completeness Theorem, but is not related to completeness of the kind I discuss on page 782 and elsewhere in this section.)

In discussions of predicate logic there is often much said about scoping of variables. A typical issue is that in, say, $\forall_x (\exists_y (\neg x == y))$, x and y are dummy variables whose specific names are not supposed to be significant; yet the names become significant if, say, x is replaced by y. In *Mathematica* most such issues are handled automatically. The axioms for predicate logic given here follow the work of Alfred Tarski in 1962 and use properties of $==$ to minimize issues of variable scoping.

(See also higher-order logics on page 1167.)

▪ **Arithmetic.** Most of the Peano axioms are straightforward statements of elementary facts about arithmetic. The last axiom is a schema (see page 1156) that states the principle of mathematical induction: that if a statement is valid for $a = 0$, and its validity for $a = b$ implies its validity for $a = b + 1$, then it follows that the statement must be valid for all a. Induction was to some extent already used in antiquity—for example in Euclid's proof that there are always larger primes. It began to be used in more generality in the 1600s. In effect it expresses the idea that the integers form a single ordered sequence, and it provides a basis for the notion of recursion.

In the early history of mathematics arithmetic with integers did not seem to need formal axioms, for facts like $x + y == y + x$ appeared to be self-evident. But in 1861 Hermann Grassmann showed that such facts could be deduced from more basic ones about successors and induction. And in 1891 Giuseppe Peano gave essentially the Peano axioms listed here (they were also given slightly less formally by Richard Dedekind in 1888)—which have been used unchanged ever since. (Note that in second-order logic—and effectively set theory— $+$ and $\times$ can be defined just in terms of Δ; see page 1160. In addition, as noted by Julia Robinson in 1948 it is possible to remove explicit mention of $+$ even in the ordinary Peano axioms, using the fact that if $c == a + b$ then $(\Delta a \times c) \times (\Delta b \times c) == \Delta(c \times c) \times (\Delta a \times b)$. Axioms 3, 4 and 6 can then be replaced by $a \times b == b \times a$, $a \times (b \times c) == (a \times b) \times c$ and $(\Delta a) \times (\Delta a \times b) == \Delta a \times (\Delta b \times (\Delta a))$. See also page 1163.)

The proof of Gödel's Theorem in 1931 (see page 1158) demonstrated the universality of the Peano axioms. It was shown by Raphael Robinson in 1950 that universality is also achieved by the Robinson axioms for reduced arithmetic (usually called Q) in which induction—which cannot be reduced to a finite set of ordinary axioms (see page 1156)—is replaced by a single weaker axiom. Statements like $x + y == y + x$ can no longer be proved in the resulting system (see pages 800 and 1169).

If any single one of the axioms given for reduced arithmetic is removed, universality is lost. It is not clear however exactly what minimal set of axioms is needed, for example, for the existence of solutions to integer equations to be undecidable (see page 787). (It is known, however, that essentially nothing is lost even from full Peano arithmetic if for example one drops axioms of logic such as $\neg\ \neg a = a$.)

A form of arithmetic in which one allows induction but removes multiplication was considered by Mojzesz Presburger in 1929. It is not universal, although it makes statements of size n potentially take as many as about 2^{2^n} steps to prove (though see page 1143).

The Peano axioms for arithmetic seem sufficient to support most of the whole field of number theory. But if as I believe there are fairly simple results that are unprovable from these axioms it may in fact be necessary to extend the Peano axioms to make certain kinds of progress even in practical number theory. (See also page 1166.)

▪ **Algebraic axioms.** Axioms like $a \circ (b \circ c) = (a \circ b) \circ c$ can be used in at least three ways. First, as equations which can be manipulated—like the axioms of basic logic—to establish whether expressions are equal. Second, as on page 773, as statements to be added to the axioms of predicate logic to yield results that hold for every possible system described by the axioms (say every possible semigroup). And third, as definitions of sets whose properties can be studied—and compared—using set theory. High-school algebra typically treats axioms as equations. More advanced algebra often uses predicate logic, but implicitly uses set theory whenever it addresses for example mappings between objects. Note that as discussed on page 1159 how one uses algebraic axioms can affect issues of universality and undecidability. (See also page 1169.)

▪ **Groups.** Groups have been used implicitly in the context of geometrical symmetries since antiquity. In the late 1700s specific groups began to be studied explicitly, mainly in the context of permutations of roots of polynomials, and notably by Evariste Galois in 1831. General groups were defined by Arthur Cayley around 1850 and their standard axioms became established by the end of the 1800s. The alternate axioms given in the main text are the shortest known. The first for ordinary groups was found by Graham Higman and Bernhard Neumann in 1952; the second by William McCune (using automated theorem proving) in 1992. For commutative (Abelian) groups the first alternate axioms were found by Alfred Tarski in 1938; the second by William McCune (using automated theorem proving) in 1992. In this case it is known that no shorter axioms are possible. (See page 806.) Note that in terms of the $\bar{\circ}$ operator $1 == a \mathbin{\bar{\circ}} a$, $\overline{b} == (a \mathbin{\bar{\circ}} a) \mathbin{\bar{\circ}} b$, and $a \cdot b == a \mathbin{\bar{\circ}} ((a \mathbin{\bar{\circ}} a) \mathbin{\bar{\circ}} b)$. Ordinary group theory is universal; commutative group theory is not (see page 1159).

▪ **Semigroups.** Despite their simpler definition, semigroups have been much less studied than groups, and there have for example been about 7 times fewer mathematical publications about them (and another 7 times fewer about monoids). Semigroups were defined by Jean-Armand de Séguier in 1904, and beginning in the late 1920s a variety of algebraic results about them were found. Since the 1940s they have showed up sporadically in various areas of mathematics—notably in connection with evolution processes, finite automata and category theory.

▪ **Fields.** With $\oplus$ being $+$ and $\otimes$ being $\times$ rational, real and complex numbers are all examples of fields. Ordinary integers lack inverses under $\times$, but reduction modulo a prime p gives a finite field. Since the 1700s many examples of fields have arisen, particularly in algebra and number theory. The general axioms for fields as given here emerged around the end of the 1800s. Shorter versions can undoubtedly be found. (See page 1168.)

▪ **Rings.** The axioms given are for commutative rings. With $\oplus$ being $+$ and $\otimes$ being $\times$ the integers are an example. Several examples of rings arose in the 1800s in number theory and algebraic geometry. The study of rings as general algebraic structures became popular in the 1920s. (Note that from the axioms of ring theory one can only expect to prove results that hold for any ring; to get most results in number theory, for example, one needs to use the axioms of arithmetic, which are intended to be specific to ordinary integers.) For non-commutative rings the last axiom given is replaced by $(a \oplus b) \otimes c == a \otimes c \oplus b \otimes c$. Non-commutative rings already studied in the 1800s include quaternions and square matrices.

▪ **Other algebraic systems.** Of algebraic systems studied in traditional mathematics the vast majority are special cases of either groups, rings or fields. Probably the most common other examples are those based on lattice theory. Standard axioms for lattice theory are ($\wedge$ is usually called meet, and $\vee$ join)

$$\{a \wedge b == b \wedge a,\ a \vee b == b \vee a,\ (a \wedge b) \wedge c == a \wedge (b \wedge c),$$
$$(a \vee b) \vee c == a \vee (b \vee c),\ a \wedge (a \vee b) == a,\ a \vee a \wedge b == a\}$$

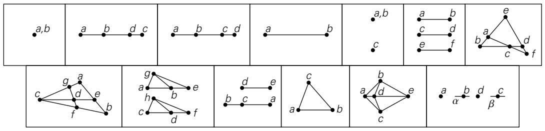
Boolean algebra (basic logic) is a special case of lattice theory, as is the theory of partially ordered sets (of which the causal networks in Chapter 9 are an example). The shortest single axiom currently known for lattice theory has *LeafCount* 79 and involves 7 variables. But I suspect that in fact a *LeafCount* less than about 20 is enough.

(See also page 1171.)

▪ **Real algebra.** A notion of real numbers as measures of space or quantity has existed since antiquity. The development of basic algebra gave a formal way to represent operations on such numbers. In the late 1800s there were efforts—notably by Richard Dedekind and Georg Cantor—to set up a general theory of real numbers relying only on basic concepts about integers—and these efforts led to set theory. For purely algebraic questions of the kind that might arise in high-school algebra, however, one can use just the axioms given here. These add to field theory several axioms for ordering, as well as the axiom at the bottom expressing a basic form of continuity (specifically that any polynomial which changes sign must have a zero). With these axioms one can prove results about real polynomials, but not about arbitrary mathematical functions, or integers. The axioms were shown to be complete by Alfred Tarski in the 1930s. The proof was based on setting up a procedure that could in principle resolve any set of real polynomial equations or inequalities. This is now in practice done by *Simplify* and other functions in *Mathematica* using methods of cylindrical algebraic decomposition invented in the 1970s—which work roughly by finding a succession of points of change using *Resultant*. (Note that with n variables the number of steps needed can increase like 2^{2^n}.) (See the note about real analysis below.)

▪ **Geometry.** Euclid gave axioms for basic geometry around 300 BC which were used with fairly little modification for more than 2000 years. In the 1830s, however, it was realized that the system would remain consistent even if the so-called parallel postulate was modified to allow space to be curved. Noting the vagueness of Euclid's original axioms there was then increasing interest in setting up more formal axiom systems for geometry. The best-known system was given by David Hilbert in 1899—and by describing geometrical figures using algebraic equations he showed that it was as consistent as the underlying axioms for numbers.

The axioms given here are illustrated below. They were developed by Alfred Tarski and others in the 1940s and 1950s. (Unlike Hilbert's axioms they require only first-order predicate logic.) The first six give basic properties of betweenness of points and congruence of line segments. The second- and third-to-last axioms specify that space has two dimensions; they can be modified for other dimensions. The last axiom is a schema that asserts the continuity of space. (The system is not finitely axiomatizable.)



The axioms given can prove most of the results in an elementary geometry textbook—indeed all results that are about geometrical figures such as triangles and circles specified by a fixed finite number of points, but which do not involve concepts like area. The axioms are complete and consistent—and thus not universal. They can however be made universal if axioms from set theory are added.

▪ **Category theory.** Developed in the 1940s as a way to organize constructs in algebraic topology, category theory works at the level of whole mathematical objects rather than their elements. In the basic axioms given here the variables represent morphisms that correspond to mappings between objects. (Often morphisms are shown as arrows in diagrams, and objects as nodes.) The axioms specify that when morphisms are composed their domains and codomains must have appropriately matching types. Some of the methodology of category theory has become widely used in mathematics, but until recently the basic theory itself was not extensively studied—and its axiomatic status remains unclear. Category theory can be viewed as a formalization of operations on abstract data types in computer languages—though unlike in *Mathematica* it normally requires that functions take a single bundle of data as an argument.

▪ **Set theory.** Basic notions of finite set theory have been used since antiquity—though became widespread only after their introduction into elementary mathematics education in the 1960s. Detailed ideas about infinite sets emerged in the 1880s through the work of Georg Cantor, who found it useful in studying trigonometric series to define sets of transfinite numbers of points. Several paradoxes associated with infinite sets were quickly noted—a 1901 example due to Bertrand Russell being to ask whether a set containing all sets that do not contain themselves in fact contains itself. To avoid such paradoxes Ernst Zermelo in 1908 suggested formalizing set theory using the first seven axioms given in the main text. (The axiom of infinity, for example, was included to establish that an infinite set such as the integers exists.) In 1922 Abraham Fraenkel noted that Zermelo's axioms did not support certain operations that seemed appropriate in a theory of sets, leading to the addition of Thoralf Skolem's axiom of replacement, and to what is usually called Zermelo-Fraenkel set theory (ZF). (The replacement axiom formally makes the subset axiom redundant.) The axiom of choice was first explicitly formulated by Zermelo in 1904 to capture the idea that in a set all elements can be ordered, so that the process of transfinite induction is possible (see page 1160). The non-constructive character of the axiom of choice has made it always remain somewhat controversial. It has arisen in many different guises and been useful in proving theorems in many areas of mathematics, but it has seemingly peculiar consequences such as the Banach-Tarski result that a solid sphere can be divided into six pieces (each a non-measurable set) that can be reassembled into a solid sphere twice the size. (The nine axioms with the axiom of choice are usually known as ZFC.) The axiom of regularity (or axiom of foundation) formulated by John von Neumann in 1929 explicitly forbids sets which for example can be elements of themselves. But while this axiom is convenient in simplifying work in set theory it has not been found generally useful in mathematics, and is normally considered optional at best.

A few additional axioms have also arisen as potentially useful. Most notable is the Continuum Hypothesis discussed on page 1127, which was proved independent of ZFC by Paul Cohen in 1963. (See also page 1166.)

Note that by using more complicated axioms the only construct beyond predicate logic needed to formulate set theory is ∈ . As discussed on page 1176, however, one cannot avoid axiom schemas in the formulation of set theory given here. (The von Neumann-Bernays-Gödel formulation does avoid these, but at the cost of introducing additional objects more general than sets.)

(See also page 1160.)

▪ **General topology.** The axioms given define properties of open sets of points in spaces—and in effect allow issues like connectivity and continuity to be discussed in terms of set theory without introducing any explicit distance function.

▪ **Real analysis.** The axiom given is Dedekind's axiom of continuity, which expresses the connectedness of the set of real numbers. Together with set theory it allows standard results about calculus to be derived. But as well as ordinary real numbers, these axioms allow non-standard analysis with constructs such as explicit infinitesimals (see page 1172).

▪ **Axiom systems for programs.** (See pages 794 and 1168.)

▪ **Page 775 · Implementation.** Given the axioms in the form

```
s[1] = (a_ ⊼ a_) ⊼ (a_ ⊼ b_) → a;
s[2, x_] := b_ → (b ⊼ b) ⊼ (b ⊼ x); s[3] =
  a_ ⊼ (a_ ⊼ b_) → a ⊼ (b ⊼ b); s[4] = a_ ⊼ (b_ ⊼ b_) → a ⊼ (a ⊼ b);
s[5] = a_ ⊼ (a_ ⊼ (b_ ⊼ c_)) → b ⊼ (b ⊼ (a ⊼ c));
```

the proof shown here can be represented by

```
{{s[2, b], {2}}, {s[4], {}}, {s[2, (b ⊼ b) ⊼ ((a ⊼ a) ⊼ (b ⊼ b))],
    {2, 2}}, {s[1], {2, 2, 1}}, {s[2, b ⊼ b], {2, 2, 2, 2, 2, 2}},
  {s[5], {2, 2, 2}}, {s[2, b ⊼ b], {2, 2, 2, 2, 2, 1}},
  {s[1], {2, 2, 2, 2, 2}}, {s[3], {2, 2, 2}},
  {s[1], {2, 2, 2, 2}}, {s[4], {2, 2, 2}}, {s[5], {}},
  {s[2, a], {2, 2, 1}}, {s[1], {2, 2}}, {s[3], {}}, {s[1], {2}}}
```

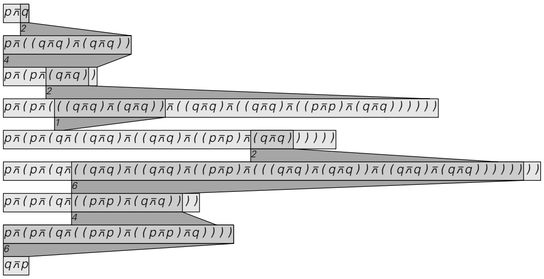
and applied using

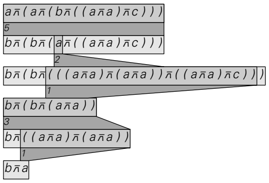
```
FoldList[Function[{u, v},
    MapAt[Replace[#, v[[1]]] &, u, {v[[2]]}]], a ⊼ b, proof]
```

▪ **Page 776 · Proof structures.** The proof shown is in a sense based on very low-level steps, each consisting of applying a single axiom from the original axiom system. But in practical mathematics it is usual for proofs to be built up in a more hierarchical fashion using intermediate results or lemmas. In the way I set things up lemmas can in effect be introduced as new axioms which can be applied repeatedly during a proof. And in the case shown here if one first proves the lemma

$$(a \barwedge (a \barwedge (b \barwedge ((a \barwedge a) \barwedge c)))) = (b \barwedge a)$$

and treats it as rule 6, then the main proof can be shortened:


p⊼q
2
p⊼((q⊼q)⊼(q⊼q))
4
p⊼(p⊼(q⊼q))
2
p⊼(p⊼(((q⊼q)⊼(q⊼q))⊼((q⊼q)⊼((q⊼q)⊼((p⊼p)⊼(q⊼q))))))
1
p⊼(p⊼(q⊼((q⊼q)⊼((q⊼q)⊼((p⊼p)⊼(q⊼q))))))
2
p⊼(p⊼(q⊼((q⊼q)⊼((q⊼q)⊼((p⊼p)⊼(((q⊼q)⊼(q⊼q))⊼((q⊼q)⊼(q⊼q))))))))
6
p⊼(p⊼(q⊼((p⊼p)⊼(q⊼q))))
4
p⊼(p⊼(q⊼((p⊼p)⊼((p⊼p)⊼q))))
6
q⊼p



When one just applies axioms from the original axiom system one is in effect following a single line of steps. But when one proves a lemma one is in effect on a separate branch, which only merges with the main proof when one uses the lemma. And if one has nested lemmas one can end up with a proof that is in effect like a tree. (Repeated use of a single lemma can also lead to cycles.) Allowing lemmas can in extreme cases probably make proofs as much as exponentially shorter. (Note that lemmas can also be used in multiway systems.)

In the way I have set things up one always gets from one step in a proof to the next by taking an expression and applying some transformation rule to it. But while this is familiar from algebraic mathematics and from the operation of *Mathematica* it is not the model of proofs that has traditionally been used in mainstream mathematical logic. For there one tends to think not so much about transforming expressions as about taking collections of true statements (such as equations $u == v$), and using so-called rules of inference to deduce other ones. Most often there are two basic rules of inference: *modus ponens* or detachment which uses the logic result $(x \wedge x \Rightarrow y) \Rightarrow y$ to deduce the statement y from statements x and $x \Rightarrow y$, and substitution, which takes statements x and y and deduces $x /. p \rightarrow y$, where p is a logical variable in x (see page 1151). And with this approach axioms enter merely as initial true statements, leaving rules of inference to generate successive steps in proofs. And instead of being mainly linear sequences of results, proofs instead become networks in which pairs of results are always combined when *modus ponens* is used. But it is still always in principle possible to convert any proof to a purely sequential one—though perhaps at the cost of having exponentially many more steps.

▪ **Substitution strategies.** With the setup I am using each step in a proof involves transforming an expression like $u = v$ using an expression like $s = t$. And for this to happen s or t must match some part w of u or v. The simplest way this can be achieved is for s or t to reproduce w when its variables are replaced by appropriate expressions. But in general one can make replacements not only for variables in s and t, but also for ones in w. And in practice this often makes many more matches possible. Thus for example the axiom $a \circ a = a$ cannot be applied directly to $(p \circ q) \circ (p \circ r) = q \circ r$. But after the replacement $r \to q$, $a \circ a$ matches $(p \circ q) \circ (p \circ r)$ with $a \to p \circ q$, yielding the new theorem $p \circ q = q \circ q$. These kinds of substitutions are used in the proof on page 810. One approach to finding them is so-called paramodulation, which was introduced around 1970 in the context of automated theorem-proving systems, and has been used in many such systems (see page 1157). (Such substitutions are not directly relevant to *Mathematica*, since it transforms expressions rather than theorems or equations. But when I built SMP in 1981, its semantic pattern matching mechanism did use essentially such substitutions.)

▪ **One-way transformations.** As formulated in the main text, axioms define two-way transformations. One can also set up axiom systems based on one-way transformations (as in multiway systems). For basic logic, examples of this were studied in the mid-1900s, and with the transformations thought of as rules of inference they were sometimes known as "axiomless formulations".

▪ **Axiom schemas.** An axiom like $a + 0 == a$ is a single well-formed formula in the sense of page 1150. But sometimes one needs infinite collections of such individual axioms, and in the main text these are represented by axiom schemas given as *Mathematica* patterns involving objects like *x_*. Such schemas are taken to stand for all individual axioms that match the patterns and are well-formed formulas. The induction axiom in arithmetic is an example of a schema. (See the note on finite axiomatizability on page 1176.) Note that as mentioned on page 1150 all the axioms given for basic logic should really be thought of as schemas.

▪ **Reducing axiom details.** Traditional axiom systems have many details not seen in the basic structure of multiway systems. But in most cases these details can be avoided—and in the end the universality of multiway systems implies that they can always be made to emulate any axiom system.

Traditional axiom systems tend to be based on operator systems (see page 801) involving general expressions, not just strings. But any expression can always be written as a string using something like *Mathematica FullForm*. (See also page 1169.) Traditional axiom systems also involve symbolic variables, not just literal string elements. But by using methods like those for combinators on page 1121 explicit mention of variables can always be eliminated.

▪ **Proofs in practice.** At some level the purpose of a proof is to establish that something is true. But in the practice of modern mathematics proofs have taken on a broader role; indeed they have become the primary framework for the vast majority of mathematical thinking and discourse. And from this perspective the kinds of proofs given on pages 810 and 811—or typically generated by automated theorem proving—are quite unsatisfactory. For while they make it easy at a formal level to check that certain statements are true, they do little at a more conceptual level to illuminate why this might be so. And indeed the kinds of proofs normally considered most mathematically valuable are ones that get built up in terms of concepts and constructs that are somehow expected to be as generally applicable as possible. But such proofs are inevitably difficult to study in a uniform and systematic way (though see page 1176). And as I argue in the main text, it is in fact only for the rather limited kinds of mathematics that have historically been pursued that such proofs can be expected to be sufficient. For in general proofs can be arbitrarily long, and can be quite devoid of what might be considered meaningful structure.

Among practical proofs that show signs of this (and whose mathematical value is thus often considered controversial) most have been done with aid of computers. Examples include the Four-Color Theorem (coloring of maps), the optimality of the Kepler packing (see page 986), the completeness of the Robbins axiom system (see page 1151) and the universality of rule 110 (see page 678).

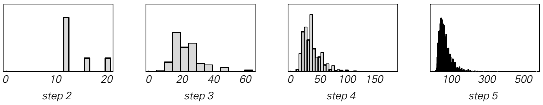
In the past it was sometimes claimed that using computers is somehow fundamentally incompatible with developing mathematical understanding. But particularly as the use of *Mathematica* has become more widespread there has been increasing recognition that computers can provide crucial raw material for mathematical intuition—a point made rather forcefully by the discoveries in this book. Less well recognized is the fact that formulating mathematical ideas in a *Mathematica* program is at least as effective a way to produce clarity of thinking and understanding as formulating a traditional proof.

▪ **Page 778 · Properties.** The second rule shown has the property that black elements always appear before white, so that strings can be specified just by the number of elements of each color that they contain—making the rule one of the sorted type discussed on page 937, based on the difference vector $\{\{2, -1\}, \{-1, 3\}, \{-4, -1\}\}$. The question of whether a given string can be generated is then analogous to finding whether there is a solution with certain positivity properties to a set of linear Diophantine equations.

▪ **Page 781 · NAND tautologies.** At each step every possible transformation rule in the axioms is applied wherever it can. New expressions are also created by replacing each possible variable with $x \barwedge y$, where x and y are new variables, and by setting every possible pair of variables equal in turn. The longest tautology at step t is

$$\text{Nest}[(\# \barwedge \#) \barwedge (\# \barwedge p_t)\,\&,\ p \barwedge (p \barwedge p),\ t-1]$$

whose *LeafCount* grows like 3^t. The distribution of sizes of statements generated at each step is shown below.



Even with the same underlying axioms the tautologies are generated in a somewhat different order if one uses a different strategy—say one based on paramodulation (see page 1156). Pages 818 and 1175 discuss the sequence of all NAND theorems listed in order of increasing complexity.

▪ **Proof searching.** To find a proof of some statement $p = q$ in a multiway system one can always in principle just start from p, evolve the system until it first generates q, then pick out the sequence of strings on the path from p to q. But doing this will usually involve building up a vast network of strings. And although at some level computational irreducibility and NP completeness (see page 766) imply that in general only a limited amount of this computational work can be saved, there are in practice quite often important optimizations that can be made. For finding a proof of $p = q$ is like searching for a path satisfying the constraint of going from p to q. And just like in the systems based on constraints in Chapter 5 one can usually do at least somewhat better than just to look at every possible path in turn.

For a start, in generating the network of paths one only ever need keep a single path that leads to any particular string; just like in many of my pictures of multiway systems one can in effect always drop any duplicate strings that occur. One might at first imagine that if p and q are both short strings then one could also drop any very long strings that are produced. But as we have seen, it is perfectly possible for long intermediate strings to be needed to get from p to q. Still, it is often reasonable to weight things so that at least at first one looks at paths that involve only shorter strings.

In the most direct approach, one takes a string and at each step just applies the underlying rules or axioms of the multiway system. But as soon as one knows that there is a path from a string u to a string v, one can also imagine applying the rule $u \to v$ to any string—in effect like a lemma. And one can choose which lemmas to try first by looking for example at which involve the shortest or commonest strings.

It is often important to minimize the number of lemmas one has to keep. Sometimes one can do this by reducing every lemma—and possibly every string—to some at least partially canonical form. One can also use the fact that in a multiway system if $u \to v$ and $r \to s$ then $u <> r \to v <> s$.

If one wants to get from p to q the most efficient thing is to use properties of q to avoid taking wrong turns. But except in systems with rather simple structure this is usually difficult to achieve. Nevertheless, one can for example always in effect work forwards from p, and backwards from q, seeing whether there is any overlap in the sets of strings one gets.

▪ **Automated theorem proving.** Since the 1950s a fair amount of work has been done on trying to set up computer systems that can prove theorems automatically. But unlike systems such as *Mathematica* that emphasize explicit computation none of these efforts have ever achieved widespread success in mathematics. And indeed given my ideas in this section this now seems not particularly surprising.

The first attempt at a general system for automated theorem proving was the 1956 Logic Theory Machine of Allen Newell and Herbert Simon—a program which tried to find proofs in basic logic by applying chains of possible axioms. But while the system was successful with a few simple theorems the searches it had to do rapidly became far too slow. And as the field of artificial intelligence developed over the next few years it became widely believed that what would be needed was a general system for imitating heuristics used in human thinking. Some work was nevertheless still done on applying results in mathematical logic to speed up the search process. And in 1963 Alan Robinson suggested the idea of resolution theorem proving, in which one constructs $\neg\, \mathit{theorem} \vee \mathit{axioms}$, then typically writes this in conjunctive normal form and repeatedly applies rules like $(\neg p \vee q) \wedge (p \vee q) \to q$ to try to reduce it to *False*, thereby proving given *axioms* that *theorem* is *True*. But after early enthusiasm it became clear that this approach could not be expected to make theorem proving easy—a point emphasized by the discovery of NP completeness in the early 1970s. Nevertheless, the approach was used with some success, particularly in proving that various mechanical and other engineering systems would behave as intended—although by the mid-1980s such verification was more often done by systematic Boolean function methods (see page 1097). In the 1970s simple versions of the resolution method were incorporated into logic programming languages such as Prolog, but little in the way of mathematical theorem proving was done with them. A notable system under development since the 1970s is the Boyer-Moore theorem prover Nqthm, which uses resolution together with methods related to induction to try to find proofs of statements in a version of LISP. Another family of systems under development at Argonne National Laboratory since the 1960s are intended to find proofs in pure operator (equational) systems (predicate logic with equations). Typical of this effort was the Otter system started in the mid-1980s, which uses the resolution method, together with a variety of ad hoc strategies that are mostly versions of the general ones for multiway systems in the previous note. The development of so-called unfailing completion algorithms (see page 1037) in the late 1980s made possible much more systematic automated theorem provers for pure operator systems—with a notable example being the Waldmeister system developed around 1996 by Arnim Buch and Thomas Hillenbrand.

Ever since the 1970s I at various times investigated using automated theorem-proving systems. But it always seemed that extensive human input—typically from the creators of the system—was needed to make such systems actually find non-trivial proofs. In the late 1990s, however, I decided to try the latest systems and was surprised to find that some of them could routinely produce proofs hundreds of steps long with little or no guidance. Almost any proof that was easy to do by hand almost always seemed to come out automatically in just a few steps. And the overall ability to do proofs—at least in pure operator systems—seemed vastly to exceed that of any human. But as page 810 illustrates, long proofs produced in this way tend to be difficult to read—in large part because they lack the higher-level constructs that are typical in proofs created by humans. As I discuss on page 821, such lack of structure is in some respects inevitable. But at least for specific kinds of theorems in specific areas of mathematics it seems likely that more accessible proofs can be created if each step is allowed to involve sophisticated computations, say as done by *Mathematica*.

■ **Proofs in *Mathematica*.** Most of the individual built-in functions of *Mathematica* I designed to be as predictable as possible—applying transformations in definite ways and using algorithms that are never of fundamentally unknown difficulty. But as their names suggest *Simplify* and *FullSimplify* were intended to be less predictable—and just to do what they can and then return a result. And in many cases these functions end up trying to prove theorems; so for example *FullSimplify[(a + b)/2 ≥ Sqrt[a b], a > 0 && b > 0]* must in effect prove a theorem to get the result *True*.

■ **Page 781 · Truth and falsity.** The notion that statements can always be classified as either true or false has been a common idealization in logic since antiquity. But in everyday language, computer languages and mathematics there are many ways in which this idealization can fail. An example is $x + y == z$, which cannot reasonably be considered either true or false unless one knows what x, y and z are. Predicate logic avoids this particular kind of case by implicitly assuming that what is meant is a general statement about all values of any variable—and avoids cases like the expression $x + y$ by requiring all statements to be well-formed formulas (see page 1150). In *Mathematica* functions like *TrueQ* and *IntegerQ* are set up always to yield *True* or *False*—but just by looking at the explicit structure of a symbolic expression.

Note that although the notion of negation seems fairly straightforward in everyday language it can be difficult to implement in computational or mathematical settings. And thus for example even though it may be possible to establish by a finite computation that a particular system halts, it will often be impossible to do the same for the negation of this statement. The same basic issue arises in the intuitionistic approach to mathematics, in which one assumes that any object one handles must be found by a finite construction. And in such cases one can set up an analog of logic in which one no longer takes $\neg \neg a = a$.

It is also possible to assume a specific number $k > 2$ of truth values, as on page 1175, or to use so-called modal logics.

(See also page 1167.)

■ **Page 782 · Gödel's Theorem.** What is normally known as "Gödel's Theorem" (or "Gödel's First Incompleteness Theorem") is the centerpiece of the paper "On Undecidable Propositions of *Principia Mathematica* and Related Systems" published by Kurt Gödel in 1931. What the theorem shows is that there are statements that can be formulated within the standard axiom system for arithmetic but which cannot be proved true or false within that system. Gödel's paper does this first for the statement "this statement is unprovable", and much of the paper is concerned with showing how such a statement can be encoded within arithmetic. Gödel in effect does this by first converting the statement to one about recursive functions and then—by using tricks of number theory such as the beta function of page 1120—to one purely about arithmetic. (Gödel's main achievement is sometimes characterized as the "arithmetization of metamathematics": the discovery that concepts such as provability related to the processes of mathematics can be represented purely as statements in arithmetic.) (See page 784.)

Gödel originally based his theorem on Peano arithmetic (as discussed in the context of *Principia Mathematica*), but expected that it would in fact apply to any reasonable formal system for mathematics—and in later years considered that this had been established by thinking about Turing machines. He suggested that his results could be avoided if some form of transfinite hierarchy of formalisms could be used, and appears to have thought that at some level humans and mathematics do this (compare page 1167).

Gödel's 1931 paper came as a great surprise, although the issues it addressed were already widely discussed in the field of mathematical logic. And while the paper is at a technical level rather clear, it has never been easy for typical mathematicians to read. Beginning in the late 1950s its results began to be widely known outside of mathematics, and by the late 1970s Gödel's Theorem and various misstatements of it were often assigned an almost mystical significance. Self-reference was commonly seen as its central feature, and connections with universality and computation were usually missed. And with the belief that humans must somehow have intrinsic access to all truths in mathematics, Gödel's Theorem has been used to argue for example that computers can fundamentally never emulate human thinking.

The picture on page 786 can be viewed as a modern proof of Gödel's Theorem based on Diophantine equations.

In addition to what is usually called Gödel's Theorem, Kurt Gödel established a second incompleteness theorem: that the statement that the axioms of arithmetic are consistent cannot be proved by using those axioms (see page 1168). He also established what is often called the Completeness Theorem for predicate logic (see page 1152)—though here "completeness" is used in a different sense.

■ **Page 783 · Properties.** The first multiway system here generates all strings that end in □■; the third all strings that end in ■. The second system generates all strings where the second-to-last element is white, or the string ends with a run of black elements delimited by white ones.

■ **Page 783 · Essential incompleteness.** If a consistent axiom system is complete this means that any statement in the system can be proved true or false using its axioms, and the question of whether a statement is true can always be decided by a finite procedure. If an axiom system is incomplete then this means that there are statements that cannot be proved true or false using its axioms—and which must therefore be considered independent of those axioms. But even given this it is still possible that a finite procedure can exist which decides whether a given statement is true, and indeed this happens in the theory of commutative groups (see note below). But often an axiom system will not only be incomplete, but will also be what is called essentially incomplete. And what this means is that there is no finite set of axioms that can consistently be added to make the system complete. A consequence of this is that there can be no finite procedure that always decides whether a given statement is true—making the system what is known as essentially undecidable. (When I use the term "undecidable" I normally mean "essentially undecidable". Early work on mathematical logic sometimes referred to statements that are independent as being undecidable.)

One might think that adding rules to a system could never reduce its computational sophistication. And this is correct if with suitable input one can always avoid the new rules. But often these rules will allow transformations that in effect short-circuit any sophisticated computation. And in the context of axiom systems, adding axioms can be thought of as putting more constraints on a system—thus potentially in effect forcing it to be simpler. The result of all this is that an axiom system that is universal can stop being universal when more axioms are added to it. And indeed this happens when one goes from ordinary group theory to commutative group theory, and from general field theory to real algebra.

■ **Page 784 · Predicate logic.** The universality of predicate logic with a single two-argument function follows immediately from the result on page 1156 that it can be used to emulate any two-way multiway system.

■ **Page 784 · Algebraic axioms.** How universality works with algebraic axioms depends on how those axioms are being used (compare page 1153). What is said in the main text here assumes that they are being used as on page 773—with each variable in effect standing for any object (compare page 1169), and with the axioms being added to predicate logic. The first of these points means that one is concerned with so-called pure group theory—and with finding results valid for all possible groups. The second means that the statements one considers need not just be of the form $\ldots == \ldots$, but can explicitly involve logic; an example is Cayley's theorem

$$a \cdot x == a \cdot y \Rightarrow (x == y \wedge \exists_z\, a \cdot z == x) \wedge$$
$$a \cdot x == b \cdot x \Rightarrow a == b \wedge (a \cdot b) \cdot x == a \cdot (b \cdot x)$$

With this setup, Alfred Tarski showed in 1946 that any statement in Peano arithmetic can be encoded as a statement in group theory—thus demonstrating that group theory is universal, and that questions about it can be undecidable. This then also immediately follows for semigroup theory and monoid theory. It was shown for ring theory and field theory by Julia Robinson in 1949. But for commutative group theory it is not the case, as shown by Wanda Szmielew around 1950. And indeed there is a procedure based on quantifier elimination for determining in a finite number of steps whether any statement in commutative group theory can be proved. (Commutative group theory is thus a decidable theory. But as mentioned in the note above, it is not complete—since for example it cannot establish the theorem $a == b$ which states that a group has just one element. It is nevertheless not essentially incomplete—and for example adding the axiom $a == b$ makes it complete.) Real algebra is also not universal (see page 1153), and the same is for example true for finite fields—but not for arbitrary fields.

As discussed on page 1141, word problems for systems such as groups are undecidable. But to set up a word problem in general formally requires going beyond predicate logic, and including axioms from set theory. For a word problem relates not, say, to groups in general, but to a particular group, specified by relations between generators. Within predicate logic one can give the relations as statements, but in effect one cannot specify that no other relations hold. It turns out, however, that undecidability for word problems occurs in essentially the same places as universality for axioms with predicate logic. Thus, for example, the word problem is undecidable for groups and semigroups, but is decidable for commutative groups.

One can also consider using algebraic axioms without predicate logic—as in basic logic or in the operator systems of page 801. And one can now ask whether there is then universality. In the case of semigroup theory there is not. But certainly systems of this type can be universal—since for example they can be set up to emulate any multiway system. And it seems likely that the axioms of ordinary group theory are sufficient to achieve universality.

■ **Page 784 · Set theory.** Any integer n can be encoded as a set using for example $\mathit{Nest}[\mathit{Union}[\#, \{\#\}]\,\&, \{\}, n]$. And from this a statement s in Peano arithmetic (with each variable explicitly quantified) can be translated to a statement in set theory by using

$$\begin{aligned}&\mathit{Replace}[s, \{\forall_{a\_}\ b\_ \to \forall_a\ (a \in \mathbb{N} \Rightarrow b),\\ &\quad \exists_{a\_}\ b\_ \to \exists_a\ (a \in \mathbb{N} \wedge b)\}, \{0, \infty\}]\end{aligned}$$

and then adding the statements below to provide definitions ($\mathbb{N}$ is the set of non-negative integers, $\langle x, y, z\rangle$ is an ordered triple, and $\updownarrow a$ determines whether each triple in a set a is of the form $\langle x, y, f[x, y]\rangle$ specifying a single-valued function).

$a = \mathbb{N} \Leftrightarrow \forall_b\ ((\emptyset \in b \wedge \forall_c\ (c \in b \Rightarrow \bigcup\{c, \{c\}\} \in b)) \Rightarrow a \subseteq b)$
$a = \Delta\, b \Leftrightarrow a = \bigcup\{b, \{b\}\}$
$a = \langle b, c, d\rangle \Leftrightarrow a = \{\{\{\{b, c\}, \{c\}\}, d\}, \{d\}\}$
$\updownarrow a \Leftrightarrow (\forall_b\ \forall_c\ \forall_d\ (\langle b, c, d\rangle \in a \Rightarrow \forall_e\ (\langle b, c, e\rangle \in a \Rightarrow d = e)) \wedge$ $\forall_b\ \forall_c\ ((b \in \mathbb{N} \wedge c \in \mathbb{N}) \Rightarrow \exists_d\ (d \in \mathbb{N} \wedge \langle b, c, d\rangle \in a)))$
$a = b + c \Leftrightarrow \forall_d\ ((\updownarrow d \wedge \forall_e\ \forall_f\ \forall_g\ (\langle \Delta\, e, f, g\rangle \in d \Rightarrow \langle e, f, \Delta\, g\rangle \in d) \wedge$ $\forall_f\ \forall_g\ (\langle \emptyset, f, g\rangle \in d \Rightarrow g = f)) \Rightarrow \langle b, c, a\rangle \in d)$
$a = b \times c \Leftrightarrow \forall_d\ ((\updownarrow d \wedge \forall_e\ \forall_f\ \forall_g\ (\langle \Delta\, e, f, g\rangle \in d \Rightarrow \langle e, f, f + g\rangle \in d) \wedge$ $\forall_f\ \forall_g\ (\langle \emptyset, f, g\rangle \in d \Rightarrow g = \emptyset)) \Rightarrow \langle b, c, a\rangle \in d)$

This means that set theory can be used to prove any statement that can be proved in Peano arithmetic. But it can also prove other statements—such as Goodstein's result (see note below), and the consistency of arithmetic (see page 1168). An important reason for this is that set theory allows not just ordinary induction over sequences of integers but also transfinite induction over arbitrary ordered sets (see below).

■ **Page 786 · Universal Diophantine equation.** The equation is built up from ones whose solutions are set up to be integers that satisfy particular relations. So for example the equation $a^2 + b^2 == 0$ has solutions that are exactly those integers that satisfy the relation $a == 0 \wedge b == 0$. Similarly, assuming as in the rest of this note that all variables are non-negative, $b == a + c + 1$ has solutions that are exactly those integers that satisfy $a < b$, with c having some allowed value. From various number-theoretical results many relations can readily be encoded as integer equations:

$$(a == 0 \vee b == 0) \longleftrightarrow a\, b == 0$$

$$(a == 0 \wedge b == 0) \longleftrightarrow a + b == 0$$

$$a < b \longleftrightarrow b == a + c + 1$$

$$a == \mathit{Mod}[b, c] \longleftrightarrow (b == a + c\, d \wedge a < c)$$

$$a == \mathit{Quotient}[b, c] \longleftrightarrow (b == a\, c + d \wedge d < c)$$

$$\begin{aligned}&a == \mathit{Binomial}[b, c] \longleftrightarrow \mathit{With}[\{n = 2^b + 1\},\\ &\qquad (n + 1)^b == n^c\,(a + d\, n) + e \wedge e < n^c \wedge a < n]\end{aligned}$$

$$a == b\,! \longleftrightarrow a == \mathit{Quotient}[c^b, \mathit{Binomial}[c, b]]$$

$$a == \mathit{GCD}[b, c] \longleftrightarrow (b\, c > 0 \wedge a\, d == b \wedge a\, e == c \wedge a + c\, f == b\, g)$$

$$a == \mathit{Floor}[b/c] \longleftrightarrow (a\, c + d == b \wedge d < c)$$

$$\mathit{PrimeQ}[a] \longleftrightarrow (\mathit{GCD}[(a - 1)\,!, a] == 1 \wedge a > 1)$$

$$\begin{aligned}&a == \mathit{BitAnd}[c, d] \wedge b == \mathit{BitOr}[c, d] \longleftrightarrow\\ &\qquad (\sigma[c, a] \wedge \sigma[d, a] \wedge \sigma[b, c] \wedge \sigma[b, d] \wedge a + b == c + d)\ /.\\ &\quad \sigma[x\_, y\_] \to \mathit{Mod}[\mathit{Binomial}[x, y], 2] == 1\end{aligned}$$

where the last encoding uses the result on page 608. (Note that any variable a can be forced to be non-negative by including an equation $a == w^2 + x^2 + y^2 + z^2$, as on page 910.)

Given an integer a for which $\mathit{IntegerDigits}[a, 2]$ gives the cell values for a cellular automaton, a single step of evolution according say to rule 30 is given by

$$\mathit{BitXor}[a, 2\, \mathit{BitOr}[a, 2\, a]]$$

where (see page 871)

$$\mathit{BitXor}[x, y] == \mathit{BitOr}[x, y] - \mathit{BitAnd}[x, y]$$

and a is assumed to be padded with 0's at each end. The corresponding form for rule 110 is

$$\mathit{BitXor}[\mathit{BitAnd}[a, 2 a, 4 a], \mathit{BitOr}[2 a, 4 a]]$$

The final equation is then obtained from

$$\begin{aligned}
&\{1 + x_4 + x_{12} == 2^{(1+x_3)(x_1+2 x_3)}, x_2 + x_{13} == 2^{x_1},\\
&\quad 1 + x_5 + x_{14} == 2^{x_1}, 2^{x_3} x_5 + 2^{x_1+2 x_3} x_6 + 2^{x_1+x_3} x_{15} + x_{16} == x_4,\\
&\quad 1 + x_{15} + x_{17} == 2^{x_3}, 1 + x_{16} + x_{18} == 2^{x_3},\\
&\quad 2^{1+x_3 (1+x_1+2 x_3)} (-1 + x_2) - x_{10} + x_{11} == 2 x_4,\\
&\quad x_7 == \mathit{BitAnd}[x_6, 2 x_6] \wedge x_8 == \mathit{BitOr}[x_6, 2 x_6],\\
&\quad x_9 == \mathit{BitAnd}[x_6, 2 x_7] \wedge x_{19} == \mathit{BitOr}[x_6, 2 x_7],\\
&\quad x_{10} == \mathit{BitAnd}[x_9, 2 x_8] \wedge x_{11} == \mathit{BitOr}[x_9, 2 x_8]\}
\end{aligned}$$

where x_1 through x_4 have the meanings indicated in the main text, and satisfy $x_i \geq 0$. Non-overlapping subsidiary variables are introduced for *BitOr* and *BitAnd*, yielding a total of 79 variables.

Note that it is potentially somewhat easier to construct Diophantine equations to emulate register machines—or arithmetic systems from page 673—than to emulate cellular automata, but exactly the same basic methods can be used.

In the universal equation in the main text variables appear in exponents. One can reduce such an exponential equation to a pure polynomial equation by encoding powers using integer equations. The simplest known way of doing this (see note below) involves a degree 8 equation with 60 variables:

$$\begin{aligned}
&a == b^c \longleftrightarrow \alpha[d, 4 + b e, 1 + z] \wedge \alpha[f, e, 1 + z] \wedge\\
&\quad a == \mathit{Quotient}[d, f] \wedge \alpha[g, 4 + b, 1 + z] \wedge e == 16 g (1 + z)
\end{aligned}$$

$$\begin{aligned}
&\lambda[a\_, b\_, c\_] := \mathit{Module}[\{x\},\\
&\qquad 2 a + x_1 == c \wedge (\mathit{Mod}[b - a, c] == 0 \vee \mathit{Mod}[b + a, c] == 0)]
\end{aligned}$$

$$\begin{aligned}
&\alpha[a\_, b\_, c\_] := \mathit{Module}[\{x\}, {x_1}^2 - b x_1 x_2 + {x_2}^2 == 1 \wedge\\
&\quad {x_3}^2 - b x_3 x_4 + {x_4}^2 == 1 \wedge 1 + x_4 + x_5 == x_3 \wedge \mathit{Mod}[x_3, {x_1}^2] ==\\
&\quad 0 \wedge 2 x_4 + x_7 == b x_3 \wedge \mathit{Mod}[-b + x_8, x_7] == 0 \wedge\\
&\quad \mathit{Mod}[-2 + x_8, x_1] == 0 \wedge x_8 - x_{11} == 3 \wedge {x_{12}}^2 - x_8 x_{12} x_{13} +\\
&\quad {x_{13}}^2 == 1 \wedge 1 + 2 a + x_{14} == x_1 \wedge \lambda[a, x_{12}, x_7] \wedge \lambda[c, x_{12}, x_1]]
\end{aligned}$$

(This roughly uses the idea that solutions to Pell equations grow exponentially, so that for example $x^2 == 2 y^2 + 1$ has solutions $\mathit{With}[\{u = 3 + 2 \sqrt{2}\}, (u^n + u^{-n})/2]$.) From this representation of *Power* the universal equation can be converted to a purely polynomial equation with 2154 variables—which when expanded has 1683150 terms, total degree 16 (average per term 6.8), maximum coefficient 17827424 and *LeafCount* 16540206.

Note that the existence of universal Diophantine equations implies that any problem of mathematics—even, say, the Riemann Hypothesis—can in principle be formulated as a question about the existence of solutions to a Diophantine equation. It also means that given any specific enumeration of polynomials, there must be some universal polynomial u which if fed the enumeration number of a polynomial p, together with an encoding of the values of its variables, will yield the corresponding value of p as a solution to $u == 0$.

■ **Hilbert's Tenth Problem.** Beginning in antiquity various procedures were developed for solving particular kinds of Diophantine equations (see page 1164). In 1900, as one of his list of 23 important mathematical problems, David Hilbert posed the problem of finding a single finite procedure that could systematically determine whether a solution exists to any specified Diophantine equation. The original proof of Gödel's Theorem from 1931 in effect involves showing that certain logical and other operations can be represented by Diophantine equations—and in the end Gödel's Theorem can be viewed as saying that certain statements about Diophantine equations are unprovable. The notion that there might be universal Diophantine equations for which Hilbert's Tenth Problem would be fundamentally unsolvable emerged in work by Martin Davis in 1953. And by 1961 Davis, Hilary Putnam and Julia Robinson had established that there are exponential Diophantine equations that are universal. Extending this to show that Hilbert's original problem about ordinary polynomial Diophantine equations is unsolvable required proving that exponentiation can be represented by a Diophantine equation, and this was finally done by Yuri Matiyasevich in 1969 (see note above).

By the mid-1970s, Matiyasevich had given a construction for a universal Diophantine equation with 9 variables—though with a degree of about 10^{45}. It had been known since the 1930s that any Diophantine equation can be reduced to one with degree 4—and in 1980 James Jones showed that a universal Diophantine equation with degree 4 could be constructed with 58 variables. In 1979 Matiyasevich also showed that universality could be achieved with an exponential Diophantine equation with many terms, but with only 3 variables. As discussed in the main text I believe that vastly simpler Diophantine equations can also be universal. It is even conceivable that a Diophantine equation with 2 variables could be universal: with one variable essentially being used to represent the program and input, and the other the execution history of the program—with no finite solution existing if the program does not halt.

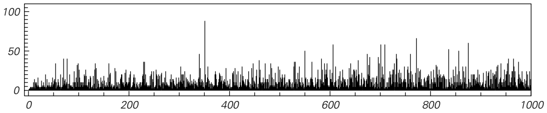
■ **Polynomial value sets.** Closely related to issues of solving Diophantine equations is the question of what set of positive values a polynomial can achieve when fed all possible positive integer values for its variables. A polynomial with a single variable must always yield either be a finite set, or a simple polynomial progression of values. But already the sequence of values for $x^2 y - x y^3$ or even $x (y^2 + 1)$ seem quite complicated. And for example from the fact that $x^2 == y^2 + (x y \pm 1)$ has solutions $\mathit{Fibonacci}[n]$ it follows that the positive values of $(2-(x^2-y^2-x\,y)^2)\,x$ are just Fibonacci[n] (achieved when {x, y} is Fibonacci[{n, n - 1}]). This is the simplest polynomial giving Fibonacci[n], and there are for example no polynomials with 2 variables, up to 4 terms, total degree less than 4, and integer coefficients between -2 and +2, that give any of 2^n, 3^n or Prime[n]. Nevertheless, from the representation for PrimeQ in the note above it has been shown that the positive values of a particular polynomial with 26 variables, 891 terms and total degree 97 are exactly the primes. (Polynomials with 42 variables and degree 5, and 10 variables and degree 10^{45}, are also known to work, while it is known that one with 2 variables cannot.) And in general the existence of a universal Diophantine equation implies that any set obtained by any finite computation must correspond to the positive values of some polynomial. The analog of doing a long computation to find a result is having to go to large values of variables to find a positive polynomial value. Note that one can imagine, say, emulating the evolution of a cellular automaton by having the t^{th} positive value of a polynomial represent the t^{th} step of evolution. That universality can be achieved just in the positive values of a polynomial is already remarkable. But I suspect that in the end it will take only a surprisingly simple polynomial, perhaps with just three variables and fairly low degree.

(See also page 1165.)

■ **Statements in Peano arithmetic.** Examples include:

- $\sqrt{2}$ is irrational:
  $\neg\, \exists_a\, (\exists_b\, (b \neq 0 \wedge a \times a == (\Delta\Delta 0) \times (b \times b)))$
- There are infinitely many primes of the form n^2+1:
  $\neg\, \exists_n\, (\forall_c\, (\exists_a\, (\exists_b\, (n+c) \times (n+c) + \Delta 0 == (\Delta\Delta a) \times (\Delta\Delta b))))$
- Every even number (greater than 2) is the sum of two primes (Goldbach's Conjecture; see page 135):
  $\forall_a\, (\exists_b\, (\exists_c\, ((\Delta\Delta 0) \times (\Delta\Delta a) == b + c \wedge \forall_d\, (\forall_e\, (\forall_f\, ((f == (\Delta\Delta d) \times (\Delta\Delta e) \vee f == \Delta 0) \Rightarrow (f \neq b \wedge f \neq c)))))))$

The last two statements have never been proved true or false, and remain unsolved problems of number theory. The picture shows spacings between n for which n^2+1 is prime.



■ **Transfinite numbers.** For most mathematical purposes it is quite adequate just to have a single notion of infinity, usually denoted ∞. But as Georg Cantor began to emphasize in the 1870s, it is possible to distinguish different levels of infinity. Most of the details of this have not been widely used in typical mathematics, but they can be helpful in studying foundational issues. Cantor's theory of ordinal numbers is based on the idea that every integer must have a successor. The next integer after all of the ordinary ones—the first infinite integer—is given the name ω. In Cantor's theory $\omega+1$ is still larger (though $1+\omega$ is not), as are $2\,\omega$, ω^2 and ω^ω. Any arithmetic expression involving ω specifies an ordinal number—and can be thought of as corresponding to a set containing all integers up to that number. The ordinary axioms of arithmetic do not apply, but there are still fairly straightforward rules for manipulating such expressions. In general there are many different expressions that correspond to a given number, though there is always a unique Cantor normal form—essentially a finite sequence of digits giving coefficients of descending powers of ω. However, not all infinite integers can be represented in this way. The first one that cannot is ϵ_0, given by the limit $\omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}$, or effectively Nest[$\omega$^# &, ω, ω]. ϵ_0 is the smallest solution to $\omega^\epsilon == \epsilon$. Subsequent solutions ($\epsilon_1$, ..., ϵ_ω, ..., ϵ_{ϵ_0}, ...) define larger ordinals, and one can go on until one reaches the limit $\epsilon_{\epsilon_{\epsilon_{\cdot_{\cdot_{\cdot}}}}}$, which is the first solution to $\epsilon_\alpha == \alpha$. Giving this ordinal a name, one can then go on again, until eventually one reaches another limit. And it turns out that in general one in effect has to introduce an infinite sequence of names in order to be able to specify all transfinite integers. (Naming a single largest or "absolutely infinite" integer is never consistent, since one can always then talk about its successor.) As Cantor noted, however, even this only allows one to reach the lowest class of transfinite numbers—in effect those corresponding to sets whose size corresponds to the cardinal number $\aleph_0$. Yet as discussed on page 1127, one can also consider larger cardinal numbers, such as $\aleph_1$, considered in connection with the number of real numbers, and so on. And at least for a while the ordinary axioms of set theory can be used to study the sets that arise.

■ **Growth rates.** One can characterize most functions by their ultimate rates of growth. In basic mathematics these might be n, $2\,n$, $3\,n$, ... or n^2, n^3, ..., or 2^n, 3^n, ..., or 2^n, 2^{2^n}, 2^{2^n}, ... To go further one begins by defining an analog to the Ackermann function of page 906:

```
f[1][n_] = 2 n; f[s_][n_] := Nest[f[s - 1], 1, n]
```

f[2][n] is then 2^n, f[3] is iterated power, and so on. Given this one can now form the "diagonal" function

```
f[ω][n_] := f[n][n]
```

and this has a higher growth rate than any of the f[s][n] with finite s. This higher growth rate is indicated by the transfinite index ω. And in direct analogy to the transfinite numbers discussed above one can then in principle form a hierarchy of functions using operations like

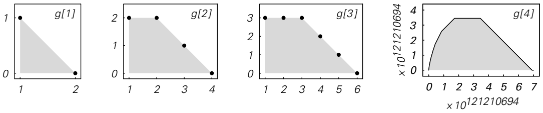
```
f[ω + s][n_] := Nest[f[ω + s - 1], 1, n]
```

together with diagonalization at limit ordinals. In practice, however, it gets more and more difficult to determine that the functions defined in this way actually in a sense halt and yield definite values—and indeed for $f[\epsilon_0]$ this can no longer be proved using the ordinary axioms of arithmetic (see below). Yet it is still possible to define functions with even more rapid rates of growth. An example is the so-called busy beaver function (see page 1144) that gives the maximum number of steps that it takes for any Turing machine of size n to halt when started from a blank tape. In general this function must grow faster than any computable function, and is not itself computable.

■ **Page 787 · Unprovable statements.** After the appearance of Gödel's Theorem a variety of statements more or less directly related to provability were shown to be unprovable in Peano arithmetic and certain other axiom systems. Starting in the 1960s the so-called method of forcing allowed certain kinds of statements in strong axiom systems—like the Continuum Hypothesis in set theory (see page 1155)—to be shown to be unprovable. Then in 1977 Jeffrey Paris and Leo Harrington showed that a variant of Ramsey's Theorem (see page 1068)—a statement that is much more directly mathematical—is also unprovable in Peano arithmetic. The approach they used was in essence based on thinking about growth rates—and since the 1970s almost all new examples of unprovability have been based on similar ideas. Probably the simplest is a statement shown to be unprovable in Peano arithmetic by Laurence Kirby and Jeff Paris in 1982: that certain sequences `g[n]` defined by Reuben Goodstein in 1944 are of limited length for all n, where

```
g[n_] := Map[First, NestWhileList[
        {f[#] - 1, Last[#] + 1} &, {n, 3}, First[#] > 0 &]]
f[{0, _}] = 0; f[{n_, k_}] := Apply[Plus, MapIndexed[#1
     k ^ f[{#2[[1]] - 1, k}] &, Reverse[IntegerDigits[n, k - 1]]]]
```

As in the pictures below, `g[1]` is `{1, 0}`, `g[2]` is `{2, 2, 1, 0}` and `g[3]` is `{3, 3, 3, 2, 1, 0}`. `g[4]` increases quadratically for a long time, with only element $3 \times 2^{402653211} - 2$ finally being 0. And the point is that in a sense `Length[g[n]]` grows too quickly for its finiteness to be provable in general in Peano arithmetic.



The argument for this as usually presented involves rather technical results from several fields. But the basic idea is roughly just to set up a correspondence between elements of `g[n]` and possible proofs in Peano arithmetic—then to use the fact that if one knew that `g[n]` always terminated this would establish the validity of all these proofs, which would in turn prove the consistency of arithmetic—a result which is known to be unprovable from within arithmetic.

Every possible proof in Peano arithmetic can in principle be encoded as an ordinary integer. But in the late 1930s Gerhard Gentzen showed that if proofs are instead encoded as ordinal numbers (see note above) then any proof can validly be reduced to a preceding one just by operations in logic. To cover all possible proofs, however, requires going up to the ordinal ϵ_0. And from the unprovability of consistency one can conclude that this must be impossible using the ordinary operation of induction in Peano arithmetic. (Set theory, however, allows transfinite induction—essentially induction on arbitrary sets—letting one reach such ordinals and thus prove the consistency of arithmetic.) In constructing `g[n]` the integer n is in effect treated like an ordinal number in Cantor normal form, and a sequence of numbers that should precede it are found. That this sequence terminates for all n is then provable in set theory, but not Peano arithmetic—and in effect `Length[g[n]]` must grow like $f[\epsilon_0][n]$.)

In general one can imagine characterizing the power of any axiom system by giving a transfinite number κ which specifies the first function $f[\kappa]$ (see note above) whose termination cannot be proved in that axiom system (or similarly how rapidly the first example of y must grow with x to prevent $\exists_y\, p[x, y]$ from being provable). But while it is known that in Peano arithmetic $\kappa = \epsilon_0$, quite how to describe the value of κ for, say, set theory remains unknown. And in general I suspect that there are a vast number of functions with simple definitions whose termination cannot be proved not just because they grow too quickly but instead for the more fundamental reason that their behavior is in a sense too complicated.

Whenever a general statement about a system like a Turing machine or a cellular automaton is undecidable, at least some instances of that statement encoded in an axiom system must be unprovable. But normally these tend to be complicated and not at all typical of what arise in ordinary mathematics. (See page 1167.)

■ **Encodings of arithmetic.** Statements in arithmetic are normally written in terms of $+$, $\times$ and Δ (and logical operations). But it turns out also to be possible to encode such statements in terms of other basic operations. This was for example done by Julia Robinson in 1949 with Δ (or $a + 1$) and `Mod[a, b] == 0`. And in the 1990s Ivan Korec and others showed that it could be done just with *Mod[Binomial[a + b, a], k]* with $k = 6$ or any product of primes—and that it could not be done with k a prime or prime power. These operations can be thought of as finding elements in nested Pascal's triangle patterns produced by k-color additive cellular automata. Korec showed that finding elements in the nested pattern produced by the $k = 3$ cellular automaton with rule *{{1, 1, 3}, {2, 2, 1}, {3, 3, 2}}⟦#1, #2⟧ &* (compare page 886) was also enough.

■ **Page 788 · Infinity.** See page 1162.

■ **Page 789 · Diophantine equations.** If variables appear only linearly, then it is possible to use *ExtendedGCD* (see page 944) to find all solutions to any system of Diophantine equations—or to show that none exist. Particularly from the work of Carl Friedrich Gauss around 1800 there emerged a procedure to find solutions to any quadratic Diophantine equation in two variables—in effect by reduction to the Pell equation $x^2 == a y^2 + 1$ (see page 944), and then computing *ContinuedFraction[*$\sqrt{a}$*]*. The minimal solutions can be large; the largest ones for successive coefficient sizes are given below. (With size s coefficients it is for example known that the solutions must always be less than $(14 s)^{5 s}$.).

1	$1 + x + x^2 + y - x y == 0$	$\{x == 2, y == 7\}$
2	$1 + 2 x + 2 x^2 + 2 y + x y - 2 y^2 == 0$	$\{x == 687, y == 881\}$
3	$2 + 2 x + 3 x^2 + 3 y + x y - y^2 == 0$	$\{x == 545759, y == 1256763\}$
4	$-4 - x + 4 x^2 - y - 3 x y - 4 y^2 == 0$	$\{x == 251996202018, y == 174633485974\}$

There is a fairly complete theory of homogeneous quadratic Diophantine equations with three variables, and on the basis of results from the early and mid-1900s a finite procedure should in principle be able to handle quadratic Diophantine equations with any number of variables. (The same is not true of simultaneous quadratic Diophantine equations, and indeed with a vector x of just a few variables, a system $m . x^2 == a$ of such equations could quite possibly show undecidability.)

Ever since antiquity there have been an increasing number of scattered results about Diophantine equations involving higher powers. In 1909 Axel Thue showed that any equation of the form $p[x, y] == a$, where $p[x, y]$ is a homogeneous irreducible polynomial of degree at least 3 (such as $x^3 + x y^2 + y^3$) can have only a finite number of integer solutions. (He did this by formally factoring $p[x, y]$ into terms $x - \alpha_i y$, then looking at rational approximations to the algebraic numbers α_i.) In 1966 Alan Baker then proved an explicit upper bound on such solutions, thereby establishing that in principle they can be found by a finite search procedure. (The proof is based on having bounds for how close to zero $Sum[\alpha_i Log[\alpha_j], i, j]$ can be for independent algebraic numbers α_k.) His bound was roughly $Exp[(c s)^{10^5}]$—but later work in essence reduced this, and by the 1990s practical algorithms were being developed. (Even with a bound of 10^{100}, rational approximations to real number results can quickly give the candidates that need to be tested.)

Starting in the late 1800s and continuing ever since a series of progressively more sophisticated geometric and algebraic views of Diophantine equations have developed. These have led for example to the 1993 proof of Fermat's Last Theorem and to the 1983 Faltings theorem (Mordell conjecture) that the topology of the algebraic surface formed by allowing variables to take on complex values determines whether a Diophantine equation has only a finite number of rational solutions—and shows for example that this is the case for any equation of the form $x^n == a y^n + 1$ with $n > 3$. Extensive work has been done since the early 1900s on so-called elliptic curve equations such as $x^2 == a y^3 + b$ whose corresponding algebraic surface has a single hole (genus 1). (A crucial feature is that given any two rational solutions to such equations, a third can always be found by a simple geometrical construction.) By the 1990s explicit algorithms for such equations were being developed—with bounds on solutions being found by Baker's method (see above). In the late 1990s similar methods were applied to superelliptic (e.g. $x^n == p[y]$) and hyperelliptic (e.g. $x^2 == p[y]$) equations involving higher powers, and it now at least definitely seems possible to handle any two-variable cubic Diophantine equation with a finite procedure. Knowing whether Baker's method can be made to work for any particular class of equations involves, however, seeing whether certain rather elaborate algebraic constructions can be done—and this may perhaps in general be undecidable. Most likely there are already equations of degree 4 where Baker's method cannot be used—perhaps ones like $x^3 == y^4 + x y + a$. But in recent years there have begun to be results by other methods about two-variable Diophantine equations, giving, for example, general upper bounds on the number of possible solutions. And although this has now led to the assumption that all two-variable Diophantine equations will eventually be resolved, based on the results of this book I would not be surprised if in fact undecidability and universality appeared in such equations—even perhaps at degree 4 with fairly small coefficients.

The vast majority of work on Diophantine equations has been for the case of two variables (or three for some homogeneous equations). No clear analog of Baker's method is known beyond two variables, and my suspicion is that with three variables undecidability and universality may already be present even in cubic equations.

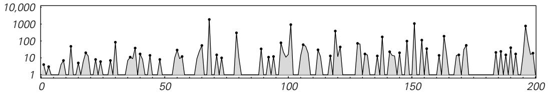
As mentioned in the main text, proving that even simple specific Diophantine equations have no solutions can be very difficult. Obvious methods involve for example showing that no solutions exist for real variables, or for variables reduced modulo some n. (For quadratic equations Hasse's Principle implies that if no solutions exist for any n then there are no solutions for ordinary integers—but a cubic like $3x^3 + 4y^3 + 5z^3 == 0$ is a counterexample.) If one can find a bound on solutions—say by Baker's method—then one can also try to show that no values below this bound are actually solutions. Over the history of number theory the sophistication of equations for which proofs of no solutions can be given has gradually increased—though even now it is state of the art to show say that $x == y == 1$ is the only solution to $x^2 == 3y^4 - 2$.

Just as for all sorts of other systems with complex behavior, some idea of overall properties of Diophantine equations can be found on the basis of an approximation of perfect randomness. Writing equations in the form $p[x_1, x_2, \ldots, x_n] == 0$ the distribution of values of p will in general be complicated (see page 1161), but as a first approximation one can try taking it to be purely random. (Versions of this for large numbers of variables are validated by the so-called circle method from the early 1900s.) If p has total degree d then with $x_i < x$ the values of $\text{Abs}[p]$ will range up to about x^d. But with n variables the number of different cases sampled for $x_i < x$ will be x^n. The assumption of perfect randomness then suggests that for $d < n$, more and more cases with $p == 0$ will be seen as x increases, so that the equation will have an infinite number of solutions. For $d > n$, on the other hand, it suggests that there will often be no solutions, and that any solutions that exist will usually be small. In the boundary case $d == n$ it suggests that even for arbitrarily large x an average of about one solution should exist—suggesting that the smallest solution may be very large, and presumably explaining the presence of so many large solutions in the $n = d = 2$ and $n = d = 3$ examples in the main text. Note that even though large solutions may be rare when $d > n$ they must always exist in at least some cases whenever there is undecidability and universality in a class of equations. (See also page 1161.)

If one wants to enumerate all possible Diophantine equations there are many ways to do this, assigning different weights to numbers of variables, and sizes of coefficients and of exponents. But with several ways I have tried, it seems that of the first few million equations, the vast majority have no solutions—and this can in most cases be established by fairly elementary methods that are presumably within Peano arithmetic. When solutions do exist, most are fairly small. But as one continues the enumeration there are increasingly a few equations that seem more and more difficult to handle.

■ **Page 790 · Properties.** (All variables are assumed positive.)

- $2x + 3y == a$. There are $\text{Ceiling}[a/2] + \text{Ceiling}[2a/3] - (a + 1)$ solutions, the one with smallest x being $\{\text{Mod}[2a + 2, 3] + 1, 2\,\text{Floor}[(2a + 2)/3] - (a + 2)\}$. Linear equations like this were already studied in antiquity. (Compare page 915.)
- $x^2 == y^2 + a$. Writing a in terms of distinct factors as $r\,s$, $\{r + s, r - s\}/2$ gives a solution if it yields integers—which happens when $\text{Abs}[a] > 4$ and $\text{Mod}[a, 4] \neq 2$.
- $x^2 == a\,y^2 + 1$ (Pell equation). As discussed on page 944, whenever a is not a perfect square, there are always an infinite number of solutions given in terms of $\text{ContinuedFraction}[\sqrt{a}\,]$. Note that even when the smallest solution is not very large, subsequent solutions can rapidly get large. Thus for example when $a = 13$, the second solution is already $\{842401, 233640\}$.
- $x^2 == y^3 + a$ (Mordell equation). First studied in the 1600s, a complete theory of this so-called elliptic curve equation was only developed in the late 1900s—using fairly sophisticated algebraic number theory. The picture below shows as a function of a the minimum x that solves the equation. For $a = 68$, the only solution is $x = 1874$; for $a = 1090$, it is $x = 149651610621$. The density of cases with solutions gradually thins out as a increases (for $0 < a \leq 10000$ there are 2468 such cases). There are always only a finite number of solutions (for $0 < a \leq 10000$ the maximum is 12, achieved for $a = 8900$).



- $x^2 == a\,y^3 + 1$. Also an elliptic curve equation.
- $x^3 == y^4 + x\,y + a$. For most values of a (including specifically $a = 1$) the continuous version of this equation defines a surface of genus 3, so there are at most a finite number of integer solutions. (An equation of degree d generically defines a surface of genus $1/2\,(d - 1)(d - 2)$.) Note that $x^3 == y^4 + a$ is equivalent to $x^3 == z^2 + a$ by a simple substitution.
- $x^2 == y^5 + a\,y + 3$. The second smallest solution to $x^2 == y^5 + 5y + 3$ is $\{45531, 73\}$. As for the equations above, there are always at most a finite number of integer solutions.
- $x^3 + y^3 == z^2 + a$. For the homogenous case $a = 0$ the complete solution was found by Leonhard Euler in 1756.

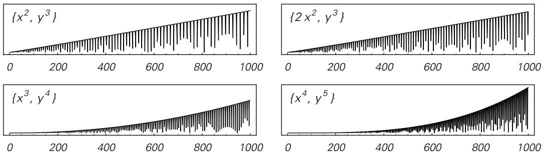
- $x^3 + y^3 == z^3 + a$. No solutions exist when $a = 9n \pm 4$; for $a = n^3$ or $2n^3$ infinite families of solutions are known. Particularly in its less strict form $x^3 + y^3 + z^3 == a$ with x, y, z positive or negative the equation was mentioned in the 1800s and again in the mid-1900s; computer searches for solutions were begun in the 1960s, and by the mid-1990s solutions such as $\{283059965, 2218888517, 2220422932\}$ for the case $a = -30$ had been found. Any solution to the difficult case $x^3 + y^3 == z^3 - 3$ must have $Mod[x, 9] == Mod[y, 9] == Mod[z, 9]$. (Note that $x^2 + y^2 + z^2 == a$ always has solutions except when $a = 4^s (8n + 7)$, as mentioned on page 135.)

■ **Large solutions.** A few other 2-variable equations with fairly large smallest solutions are:

- $x^3 == 3y^3 - xy + 63$: $\{7149, 4957\}$
- $x^4 == y^3 + 2xy - 2y + 81$: $\{19674, 531117\}$
- $x^4 == 5y^3 + xy + y - 8x$: $\{69126, 1659072\}$

The equation $x^x y^y == z^z$ is known to have smallest non-trivial solution $\{2985984, 1679616, 4478976\}$.

■ **Nearby powers.** One can potentially find integer equations with large solutions but small coefficients by looking say for pairs of integer powers close in value. The pictures below show what happens if one computes x^m and y^n for many x and y, sorts these values, then plots successive differences. The differences are trivially zero when $x = s^n$, $y = s^m$. Often they are large, but surprisingly small ones can sometimes occur (despite various suggestions from the so-called ABC conjecture). Thus, for example, $5853886516781223^3 - 1641843$ is a perfect square, as found by Noam Elkies in 1998. (Another example is $55^5 - 22434^2 == 19$.)



■ **Page 791 · Unsolved problems.** Problems in number theory that are simple to state (say in the notation of Peano arithmetic) but that so far remain unsolved include:

- Is there any odd number equal to the sum of its divisors? (Odd perfect number; 4th century BC) (See page 911.)
- Are there infinitely many primes that differ by 2? (Twin Prime Conjecture; 1700s?) (See page 909.)
- Is there a cuboid in which all edges and all diagonals are of integer length? (Perfect cuboid; 1719)
- Is there any even number which is not the sum of two primes? (Goldbach's Conjecture; 1742) (See page 135.)
- Are there infinitely many primes of the form $n^2 + 1$? (Quadratic primes; 1840s?) (See page 1162.)
- Are there infinitely many primes of the form $2^{2^n} + 1$? (Fermat primes; 1844)
- Are there no solutions to $x^m - y^n == 1$ other than $3^2 - 2^3 == 1$? (Catalan's Conjecture; 1844)
- Can every integer not of the form $9n \pm 4$ be written as $a^3 \pm b^3 \pm c^3$? (See note above.)
- How few n^{th} powers need be added to get any given integer? (Waring's Problem; 1770)

(See also Riemann Hypothesis on page 918.)

■ **Page 791 · Fermat's Last Theorem.** That $x^n + y^n == z^n$ has no integer solutions for $n > 2$ was suggested by Pierre Fermat around 1665. Fermat proved this for $n = 4$ around 1660; Leonhard Euler for $n = 3$ around 1750. It was proved for $n = 5$ and $n = 7$ in the early 1800s. Then in 1847 Ernst Kummer used ideas of factoring with algebraic integers to prove it for all $n < 37$. Extensions of this method gradually allowed more cases to be covered, and by the 1990s computers had effectively given proofs for all n up to several million. Meanwhile, many connections had been found between the general case and other areas of mathematics—notably the theory of elliptic curves. And finally around 1995, building on extensive work in number theory, Andrew Wiles managed to give a complete proof of the result. His proof is long and complicated, and relies on sophisticated ideas from many areas of mathematics. But while the statement of the proof makes extensive use of concepts from areas like set theory, it seems quite likely that in the end a version of it could be given purely in terms of Peano arithmetic. (By the 1970s it had for example been shown that many classic proofs with a similar character in analytic number theory could at least in principle be carried out purely in Peano arithmetic.)

■ **Page 791 · More powerful axioms.** If one looks for example at progressively more complicated Diophantine equations then one can expect that one will find examples where more and more powerful axiom systems are needed to prove statements about them. But my guess is that almost as soon as one reaches cases that cannot be handled by Peano arithmetic one will also reach cases that cannot be handled by set theory or even by still more powerful axiom systems.

Any statement that one can show is independent of the Peano axioms and at least not inconsistent with them one can potentially consider adding as a new axiom. Presumably it is best to add axioms that allow the widest range of new statements to be proved. But I strongly suspect that the set of statements that cannot be proved is somehow sufficiently fragmented that adding a few new axioms will actually make very little difference.

In set theory (see page 1155) a whole sequence of new axioms have historically been added to allow particular kinds of statements to be proved. And for several decades additional so-called large cardinal axioms have been discussed, that in effect state that sets exist larger than any that can be reached with the current axioms of set theory. (As discussed on page 816 any axiom system that is universal must in principle be able to prove any statement that can be proved in any axiom system—but not with the kinds of encodings normally considered in mathematical logic.)

It is notable, however, that if one looks at classic theorems in mathematics many can actually be derived from remarkably weak axioms. And indeed the minimal axioms needed to obtain most of mathematics as it is now practiced are probably much weaker than those on pages 773 and 774.

(If one considers for example theorems about computational issues such as whether Turing machines halt, then it becomes inevitable that to cover more Turing machines one needs more axioms—and to cover all possible machines one needs an infinite set of axioms, that cannot even be generated by any finite set of rules.)

▪ **Higher-order logics.** In ordinary predicate—or so-called first-order—logic the objects x that $\forall_x$ and $\exists_x$ range over are variables of the kind used as arguments to functions (or predicates) such as f[x]. To set up second-order logic, however, one imagines also being able to use $\forall_f$ and $\exists_f$ where f is a function (say the head of f[x]). And then in third-order logic one imagines using $\forall_g$ and $\exists_g$ where g appears in g[f][x].

Early formulations of axiom systems for mathematics made little distinction between first- and second-order logic. The theory of types used in *Principia Mathematica* introduced some distinction, and following the proof of Gödel's Completeness Theorem for first-order logic in 1930 (see page 1152) standard axiom systems for mathematics (as given on pages 773 and 774) began to be reformulated in first-order form, with set theory taking over many of the roles of second-order logic.

In current mathematics, second-order logic is sometimes used at the level of notation, but almost never in its full form beyond. And in fact with any standard computational system it can never be implemented in any explicit way. For even to enumerate theorems in second-order logic is in general impossible for a system like a Turing machine unless one assumes that an oracle can be added. (Note however that this is possible in Henkin versions of higher-order logic that allow only limited function domains.)

▪ **Truth and incompleteness.** In discussions of the foundations of mathematics in the early 1900s it was normally assumed that truth and provability were in a sense equivalent—so that all true statements could in principle be reached by formal processes of proof from fixed axioms (see page 782). Gödel's Theorem showed that there are statements that can never be proved from given axioms. Yet often it seemed inevitable just from the syntactic structure of statements (say as well-formed formulas) that each of them must at some level be either true or false. And this led to the widespread claim that Gödel's Theorem implies the existence of mathematical statements that are true but unprovable—with their negations being false but unprovable. Over the years this often came to be assigned a kind of mystical significance, mainly because it was implicitly assumed that somehow it must still ultimately be possible to know whether any given statement is true or false. But the Principle of Computational Equivalence implies that in fact there are all sorts of statements that simply cannot be decided by any computational process in our universe. So for example, it must in some sense be either true or false that a given Turing machine halts with given input—but according to the Principle of Computational Equivalence there is no finite procedure in our universe through which we can guarantee to know which of these alternatives is correct.

In some cases statements can in effect have default truth values—so that showing that they are unprovable immediately implies, say, that they must be true. An example in arithmetic is whether some integer equation has no solution. For if there were a solution, then given the solution it would be straightforward to give a proof that it is correct. So if it is unprovable that there is no solution, then it follows that there must in fact be no solution. And similarly, if it could be shown for example that Goldbach's Conjecture is unprovable then it would follow that it must be true, for if it were false then there would have to be a specific number which violates it, and this could be proved. Not all statements in mathematics have this kind of default truth value. And thus for example the Continuum Hypothesis in set theory is unprovable but could be either of true or false: it is just independent of the axioms of set theory. In computational systems, showing that it is unprovable that a given Turing machine halts with given input immediately implies that in fact it must not halt. But showing that it is unprovable whether a Turing machine halts with every input (a Π_2 statement in the notation of page 1139) does not immediately imply anything about whether this is in fact true or false.

▪ **Page 793 · Generalization in mathematics.** Systems that have evolved from the basic notion of numbers provide a characteristic example of the process of progressive generalization in mathematics. The main such systems and their dates of earliest known reasonably formalized use have been (see also page 901): positive integers (before 10,000 BC), rationals (3000 BC), square roots (2000 BC), other roots (1800 BC), all integers (600 AD, 1600s), decimals (950 AD), complex numbers (1500s, 1800s), polynomials (1591), infinitesimals (1635), algebraic numbers (1744), quaternions (1843), Grassmann algebra (1844), ideals (1844, 1871), octonions (~1845), Boolean algebra (1847), fields (1850s, 1871), matrices (1858), associative algebras (1870), axiomatic real numbers (1872), vectors (1881), transfinite ordinals (1883), transfinite cardinals (1883), operator calculus (1880s), Boolean algebras (1890), algebraic number fields (1893), rings (1897), *p*-adic numbers (1897), non-Archimedean fields (1899), *q*-numbers (1926), non-standard integers (1930s), non-standard reals (hyperreals) (1960), interval arithmetic (1968), fuzzy arithmetic (1970s), surreal numbers (1970s). New systems have usually been introduced in connection with extending the domains of particular existing operations. But in almost all cases the systems are set up so as to preserve as many theorems as possible—a notion that was for example made explicit in the Principle of Permanence discussed by George Peacock in 1830 and extended by Hermann Hankel in 1869.

▪ **Page 794 · Cellular automaton axioms.** The first 4 axioms are general to one-dimensional cellular automata. The next 8 are specific to rule 110. The final 3 work whenever patterns are embedded in a background of white cells. The universality of rule 110 presumably implies that the axiom system given is universal. (A complete proof would require handling various issues about boundary conditions.)

If the last 2 axioms are dropped any statement can readily be proved true or false essentially just by running rule 110 for a finite number of steps equal to the number of nested ↓ plus ⟨…⟩ in the statement. In practice, a large number of steps can however be required. As an example the statement

```
O■ ⇒ (∃_a (∃_b O a ◇ (■ ◇ (■ ◇ (■ ◇ (□ ◇
 (■ ◇ (■ ◇ (■ ◇ (□ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (□ ◇
  (□ ◇ (□ ◇ (■ ◇ (□ ◇ (□ ◇ (■ ◇ (■ ◇ (□ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇
   (□ ◇ (□ ◇ (□ ◇ (■ ◇ (□ ◇ b)))))))))))))))))))))))))))))))))))))
```

asserts that a particular localized structure occurs in the evolution of rule 110 from a single black cell. But page 38 shows that this happens for the first time after 2867 steps. (A proof of this without lemmas would probably have to be of length at least 32,910,300.)

The axioms as they are stated apply to any rule 110 evolution, regardless of initial conditions. One can establish that the statement at the bottom on the right cannot be proved either true or false from the axioms by showing that it is true for some initial conditions and false for others. Note from page 279 that the sequence □■□■□ cannot occur in rule 110 evolution except as an initial condition. So this means that the statement is false if the initial condition is ■ and true if the initial condition is □■□■□.

▪ **Practical programs.** Any equivalence between programs in a programming language can be thought of as a theorem. Simple examples in *Mathematica* include:

```
First[Prepend[p, q]] === q
Join[Join[p, q], r] === Join[p, Join[q, r]]
Partition[Union[p], 1] === Split[Union[p]]
```

One can set up axiom systems say by combining definitions of programming languages with predicate logic (as done by John McCarthy for LISP in 1963). And for programs whose structure is simple enough it has sometimes been possible to prove theorems useful for optimization or verification. But in the vast majority of cases this has been essentially impossible in practice. And I suspect that this is a reflection of widespread fundamental unprovability. In setting up programs with specific purposes there is inevitably some computational reducibility (see page 828). But I suspect that enough computational irreducibility usually remains to make unprovability common when one asks about all possible forms of behavior of the program.

▪ **Page 796 · Rules.** The examples shown here (roughly in order of increasing complexity) correspond respectively to cases (a), (k), (b), (q), (p), (r), (o), (d) on page 798.

▪ **Page 797 · Consistency.** Any axiom system that is universal can represent the statement that the system is consistent. But normally such a statement cannot be proved true or false within the system itself. And thus for example Kurt Gödel showed this in 1931 for Peano arithmetic (in his so-called second incompleteness theorem). In 1936, however, Gerhard Gentzen showed that the axioms of set theory imply the consistency of Peano arithmetic (see page 1160). In practical mathematics set theory is always taken to be consistent, but to set up a proof of this would require axioms beyond set theory.

▪ **Page 798 · Properties.** For most of the rules shown, there ultimately turn out to be quite easy characterizations of what strings can be produced.

- (a) At step t, the only new string produced is the one containing t black elements.

- (b) All strings of length n containing exactly one black cell are produced—after at most $2n-1$ steps.
- (c) All strings containing even-length runs of white cells are produced.
- (d) The set of strings produced is complicated. The last length 4 string produced is □□□■, after 16 steps; the last length 6 one is □□□□□■, after 26 steps.
- (e) All strings that begin with a black element are produced.
- (f) All strings that end with a white element but contain at least one black element, or consist of all white elements ending with black, are produced. Strings of length n take n steps to produce.
- (g) The same strings as in (f) are produced, but now a string of length n with m black elements takes $n+m-1$ steps.
- (h) All strings appear in which the first run of black elements is of length 1; a string of length n with m black elements appears after $n+m-1$ steps.
- (i) All strings containing an odd number of black elements are produced; a string of length n with m black cells occurs at step $n+m-1$.
- (j) All strings that end with a black element are produced.
- (k) Above length 1, the strings produced are exactly those starting with a white element. Those of length n appear after at most $3n-3$ steps.
- (l) The same strings as in (k) are produced, taking now at most $2n+1$ steps.
- (m) All strings beginning with a black element are produced, after at most $3n+1$ steps.
- (n) The set of strings produced is complicated, and seems to include many but not all that do not end with ■□.
- (o) All strings that do not end in ■□ are produced.
- (p) All strings are produced, except ones in which every element after the first is white. ■□□■ takes 14 steps.
- (q) All strings are produced, with a string of length n with m white elements taking $n+2m$ steps.
- (r) All strings are ultimately produced—which is inevitable after the lemmas ■ → ■■ and ■ → □ appear at steps 12 and 13. (See the first rule on page 778.)

Page 800 · Non-standard arithmetic. Goodstein's result from page 1163 is true for all ordinary integers. But since it is independent of the axioms of arithmetic there must be objects that still satisfy the axioms but for which it is false. It turns out however that any such objects must in effect be infinite. For any set of objects that satisfy the axioms of arithmetic must include all finite ordinary integers, since each of these can be reached just by using Δ repeatedly. And the axioms then turn out to imply that any additional objects must be larger than all these integers—and must therefore be infinite. But for any such truly infinite objects operations like $+$ and $\times$ cannot be computed by finite procedures, making it difficult to describe such objects in an explicit way. Ever since the work of Thoralf Skolem in 1933 non-standard models of arithmetic have been discussed, particularly in the context of ultrafilters and constructs like infinite trees. (See also page 1172.)

Page 800 · Reduced arithmetic. (See page 1152.) Statements that can be proved with induction but are not provable only with Robinson's axioms are: $x \neq \Delta x$; $x+y == y+x$; $x+(y+z) == (x+y)+z$; $0+x == x$; $\exists_x (\Delta x + y == z \Rightarrow y \neq z)$; $x \times y == y \times x$; $x \times (y \times z) == (x \times y) \times z$; $x \times (y+z) == x \times y + x \times z$.

Page 800 · Generators and relations. In the axiom systems of page 773, a single variable can stand for any element—much like a *Mathematica* pattern object such as `x_`. In studying specific instances of objects like groups one often represents elements as products of constants or generators, and then for example specifies the group by giving relations between these products. In traditional mathematical notation such relations normally look just like ordinary axioms, but in fact the variables that appear in them are now assumed to be literal objects—like x in *Mathematica*—that are generically taken to be unequal. (Compare page 1159.)

Page 801 · Comparison to multiway systems. Operator systems are normally based on equations, while multiway systems are based on one-way transformations. But for multiway systems where each rule $p \to q$ is accompanied by its reverse $q \to p$, and such pairs are represented say by "AAB" ↔ "BBAA", an equivalent operator system can immediately be obtained either from

```
Apply[Equal,
  Map[Fold[#2[#1] &, x, Characters[#]] &, rules, {2}], {1}]
```

or from (compare page 1172)

```
Append[Apply[Equal,
    Map[(Fold[f, First[#], Rest[#]] &)[Characters[#]] &,
      rules, {2}], {1}], f[f[a, b], c] == f[a, f[b, c]]]
```

where now objects like "A" and "B" are treated as constants—essentially functions with zero arguments. With slightly more effort multiway systems with ordinary one-way rules can also be converted to operator systems. Converting from operator systems to multiway systems is more difficult, though ultimately always possible (see page 1156).

As discussed on page 898, one can set up operator evolution systems similar to symbolic systems (see page 103) that have essentially the same relationship to operator systems as sequential substitution systems do to multiway systems. (See also page 1172.)

■ **Page 802 · Operator systems.** One can represent the possible values of expressions like *f[f[p, q], p]* by rule numbers analogous to those used for cellular automata. Specifying an operator *f* (taken in general to have *n* arguments with *k* possible values) by giving the rule number *u* for *f[p, q, …]*, the rule number for an expression with variables *vars* can be obtained from

```
With[{m = Length[vars]}, FromDigits[
 Block[{f = Reverse[IntegerDigits[u, k, k^n]][[FromDigits[
    {##}, k] + 1]] &}, Apply[Function[Evaluate[vars], expr],
  Reverse[Array[IntegerDigits[# - 1, k, m] &, k^m]], {1}]], k]]
```

■ **Truth tables.** The method of finding results in logic by enumerating all possible combinations of truth values seems to have been rediscovered many times since antiquity. It began to appear regularly in the late 1800s, and became widely known after its use by Emil Post and Ludwig Wittgenstein in the early 1920s.

■ **Page 803 · Proofs of axiom systems.** One way to prove that an axiom system can reproduce all equivalences for a given operator is to show that its axioms can be used to transform any expression to and from a unique standard form. For then one can start with an expression, convert it to standard form, then convert back to any expression that is equivalent. We saw on page 616 that in ordinary logic there is a unique DNF representation in terms of *And*, *Or* and *Not* for any expression, and in 1921 Emil Post used essentially this to give the first proof that an axiom system like the first one on page 773 can completely reproduce all theorems of logic. A standard form in terms of *Nand* can be constructed essentially by direct translation of DNF; other methods can be used for the various other operators shown. (See also page 1175.)

Given a particular axiom system that one knows reproduces all equivalences for a given operator one can tell whether a new axiom system will also work by seeing whether it can be used to derive the axioms in the original system. But often the derivations needed may be very long—as on page 810. And in fact in 1948 Samuel Linial and Emil Post showed that in general the problem is undecidable. They did this in effect by arguing (much as on page 1169) that any multiway system can be emulated by an axiom system of the form on page 803, then knowing that in general it is undecidable whether a multiway system will ever reach some given result. (Note that if an axiom system does manage to reproduce logic in full then as indicated on page 814 its consequences can always be derived by proofs of limited length, if nothing else by using truth tables.)

Since before 1920 it has been known that one way to disprove the validity of a particular axiom system is to show that with $k > 2$ truth values it allows additional operators (see page 805). (Note that even if it works for all finite k this does not establish its validity.) Another way to do this is to look for invariants that should not be present—seeing if there are features that differ between equivalent expressions, yet are always left the same by transformations in the axiom system. (Examples for logic are axiom systems which never change the size of an expression, or which are of the form *{expr == a}* where *Flatten[expr]* begins or ends with *a*.)

■ **Junctional calculus.** Expressions are equivalent when *Union[Level[expr, {-1}]]* is the same, and this canonical form can be obtained from the axiom system of page 803 by flattening using $(a \circ b) \circ c == a \circ (b \circ c)$, sorting using $a \circ b == b \circ a$, and removing repeats using $a \circ a == a$. The operator can be either *And* or *Or* (8 or 14). With $k = 3$ there are 9 operators that yield the same results:

```
{13203, 15633, 15663, 16401,
  17139, 18063, 19539, 19569, 19599}
```

With $k = 4$ there are 3944 such operators (see below). No single axiom can reproduce all equivalences, since such an axiom must of the form *expr == a*, yet *expr* cannot contain variables other than *a*, and so cannot for example reproduce $a \circ b = b \circ a$.

■ **Equivalential calculus.** Expressions with variables *vars* are equivalent if they give the same results for

```
Mod[Map[Count[expr, #, {-1}] &, vars], 2]
```

With n variables, there are thus 2^n equivalence classes of expressions (compared to 2^{2^n} for ordinary logic). The operator can be either *Xor* or *Equal* (6 or 9). With $k = 3$ there are no operators that yield the same results; with $k = 4$ *{458142180, 1310450865, 2984516430, 3836825115}* work (see below). The shortest axiom system that works up to $k = 2$ is $\{(a \circ b) \circ a = b\}$. With *modus ponens* as the rule of inference, the shortest single-axiom system that works is known to be $\{(a \circ b) \circ ((c \circ b) \circ (a \circ c))\}$. Note that equivalential calculus describes the smallest non-trivial group, and can be viewed as an extremely minimal model of algebra.

■ **Implicational calculus.** With $k = 2$ the operator can be either 2 or 11 (*Implies*), with $k = 3$ *{2694, 9337, 15980}*, and with $k = 4$ any of 16 possibilities. (Operators exist for any k.) No single axiom, at least with up to 7 operators and 4 variables, reproduces all equivalences. With *modus ponens* as the rule of inference, the shortest single-axiom system that works is known to be $\{((a \circ b) \circ c) \circ ((c \circ a) \circ (d \circ a))\}$. Using the method of page 1151 this can be converted to the equational form

$$\{((a \circ b) \circ c) \circ ((c \circ a) \circ (d \circ a)) = d \circ d,$$
$$(a \circ a) \circ b = b,\ (a \circ b) \circ b = (b \circ a) \circ a\}$$

from which the validity of the axiom system in the main text can be established.

■ **Page 803 · Operators on sets.** There is always more than one operator that yields a given collection of equivalences. So for ordinary logic both *Nand* and *Nor* work. And with $k = 4$ any of the 12 operators

```
{1116699, 169585339, 290790239, 459258879,
  1090522958, 1309671358, 1430343258, 1515110058,
  2184380593, 2575151445, 2863760025, 2986292093}
```

also turn out to work. One can see why this happens by considering the analogy between operations in logic and operations on sets. As reflected in their traditional notations—and emphasized by Venn diagrams—*And* ($\wedge$), *Or* ($\vee$) and *Not* correspond directly to *Intersection* ($\cap$), *Union* ($\cup$) and *Complement*. If one starts from the single-element set {1} then applying *Union*, *Intersection* and *Complement* one always gets either {} or {1}. And applying *Complement[s, Intersection[a, b]]* to these two elements gives the same results and same equivalences as $a \bar{\wedge} b$ applied to *True* and *False*. But if one uses instead s = {1, 2} then starts with {1} and {2} one gets any of {{}, {1}, {2}, {1, 2}} and in general with s = Range[n] one gets any of the 2^n elements in the powerset

```
Distribute[Map[{{}, {#}} &, s], List, List, List, Join]
```

But applying *Complement[s, Intersection[a, b]]* to these elements still always produces the same equivalences as with $a \bar{\wedge} b$. Yet now $k = 2^n$. And so one therefore has a representation of Boolean algebra of size 2^n. For ordinary logic based on *Nand* it turns out that there are no other finite representations (though there are other infinite ones). But if one works, say, with *Implies* then there are for example representations of size 3 (see above). And the reason for this is that with s = {1, 2} the function *Union[Complement[s, a], b]* corresponding to $a \Rightarrow b$ only ever gets to the 3 elements {{1}, {2}, {1, 2}}. Indeed, in general with operators *Implies*, *And* and *Or* one gets to $2^n - 1$ elements, while with operators *Xor* and *Equal* one gets to 2^(2 Floor[n/2]) elements.

(One might think that one could force there only ever to be two elements by adding an axiom like a == b ∨ b == c ∨ c == a. But all this actually does is to force there to be only two objects analogous to *True* and *False*.)

■ **Page 805 · Implementation.** Given an axiom system in the form {f[a, f[a, a]] = a, f[a, b] = f[b, a]} one can find rule numbers for the operators f[x, y] with k values for each variable that are consistent with the axiom system by using

```
Module[{c, v}, c = Apply[Function,
  {v = Union[Level[axioms, {-1}]], Apply[And, axioms]}];
 Select[Range[0, k^k^2 - 1], With[{u = IntegerDigits[#, k, k^2]},
   Block[{f}, f[x_, y_] := u[[-1 - k x - y]];
    Array[c, Table[k, {Length[v]}], 0, And]]] &]]
```

For $k = 4$ this involves checking nearly 16^4 or 4 billion cases, though many of these can often be avoided, for example by using analogs of the so-called Davis-Putnam rules. (In searching for an axiom system for a given operator it is in practice often convenient first to test whether each candidate axiom holds for the operator one wants.)

■ **Page 805 · Properties.** There are k^{k^2} possible forms for binary operators with k possible values for each argument. There is always at least some operator that satisfies the constraints of any given axiom system—though in a case like a = b it has $k = 1$. Of the 274,499 axiom systems of the form {… = a} where … involves ∘ up to 6 times, 32,004 allow only operators {6, 9}, while 964 allow only {1, 7}. The only cases of 2 or less operators that appear with $k = 2$ are {{}, {10}, {12}, {1, 7}, {3, 12}, {5, 10}, {6, 9}, {10, 12}}. (See page 1174.)

■ **Page 806 · Algebraic systems.** Operator systems can be viewed as algebraic systems of the kind studied in universal algebra (see page 1150). With a single two-argument operator (such as ∘) what one has is in general known as a groupoid (though this term means something different in topology and category theory); with two such operators a ringoid. Given a particular algebraic system, it is sometimes possible—as we saw on page 773—to reduce the number of operators it involves. But the number of systems that have traditionally been studied in mathematics and that are known to require only one 2-argument operator are fairly limited. In addition to basic logic, semigroups and groups, there are essentially only the rather obscure examples of semilattices, with axioms {a∘(b∘c) = (a∘b)∘c, a∘b = b∘a, a∘a = a}, central groupoids, with axioms {(b∘a)∘(a∘c) = b}, and squags (quasigroup representations of Steiner triple systems), with axioms {a∘b = b∘a, a∘a = a, a∘(a∘b) = b} or equivalently {a∘((b∘(b∘(((c∘c)∘d)∘c)))∘a) = d}. (Ordinary quasigroups are defined by {a∘c = b, d∘a = b} with c, d unique for given a, b—so that their table is a Latin square; their axioms can be set up with 3 operators as {a\a∘b = b, a∘b / b = a, a∘(a\b) = b, (a / b)∘b = a}.)

Pages 773 and 774 indicate that most axiom systems in mathematics involve operators with at most 2 arguments (there are exceptions in geometry). (Constants such as 1 or ∅ can be viewed as 0-argument operators.) One can nevertheless generalize say to polyadic groups, with 3-argument composition and analogs of associativity such as

```
f[f[a, b, c], d, e] = f[a, f[b, c, d], e] = f[a, b, f[c, d, e]]
```

Another example is the cellular automaton axiom system of page 794; see also page 886. (A perhaps important generalization is to have expressions that are arbitrary networks rather than just trees.)

■ **Symbolic systems.** By introducing constants (0-argument operators) and interpreting $\circ$ as function application one can turn any symbolic system such as $e[x][y] \to x[x[y]]$ from page 103 into an algebraic system such as $(e \circ a) \circ b = a \circ (a \circ b)$. Doing this for the combinator system from page 711 yields the so-called combinatory algebra $\{((s \circ a) \circ b) \circ c = (a \circ c) \circ (b \circ c), (k \circ a) \circ b = a\}$.

■ **Page 806 · Groups and semigroups.** With k possible values for each variable, the forms of operators allowed by axiom systems for group theory and semigroup theory correspond to multiplication tables for groups and semigroups with k elements. Note that the first group that is not commutative (Abelian) is the group S3 with $k = 6$ elements. The total number of commutative groups with k elements is just

```
Apply[Times,
  Map[PartitionsP[Last[#]] &, FactorInteger[k]]]
```

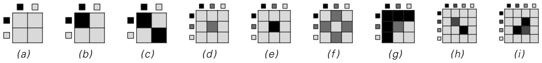
(Relabelling of elements makes the number of possible operator forms up to $k!$ times larger.) (See also pages 945, 1153 and 1173.)

■ **Forcing of operators.** Given a particular set of forms for operators one can ask whether an axiom system can be found that will allow these but no others. As discussed in the note on operators on sets on page 1171 some straightforwardly equivalent forms will always be allowed. And unless one limits the number of elements k it is in general undecidable whether a given axiom system will allow no more than a given set of forms. But even with fixed k it is also often not possible to force a particular set of forms. And as an example of this one can consider commutative group theory. The basic axioms for this allow forms of operators corresponding to multiplication tables for all possible commutative groups (see note above). So to force particular forms of operators would require setting up axioms satisfied only by specific commutative groups. But it turns out that given the basic axioms for commutative group theory any non-trivial set of additional axioms can always be reduced to a single axiom of the form $a^n == 1$ (where exponentiation is repeated application of $\circ$). Yet even given a particular number of elements k, there can be several distinct groups satisfying $a^n == 1$ for a given exponent n. (The groups can be written as products of cyclic ones whose orders correspond to the possible factors of n.) (Something similar is also known in principle to be true for general groups, though the hierarchy of axioms in this case is much more complicated.)

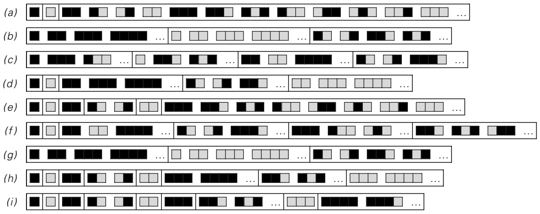
■ **Model theory.** In model theory each form of operator that satisfies the constraints of a given axiom system is called a model of that axiom system. If there is only one inequivalent model the axiom system is said to be categorical—a notion discussed for example by Richard Dedekind in 1887. The Löwenheim-Skolem theorem from 1915 implies that any axiom system must always have a countable model. (For an operator system such a model can have elements which are simply equivalence classes of expressions equal according to the axioms.) So this means that even if one tries to set up an axiom system to describe an uncountable set—such as real numbers—there will inevitably always be extra countable models. Any axiom system that is incomplete must always allow more than one model. The model intended when the axiom system was originally set up is usually called the standard model; others are called non-standard. In arithmetic non-standard models are obscure, as discussed on page 1169. In analysis, however, Abraham Robinson pointed out in 1960 that one can potentially have useful non-standard models, in which there are explicit infinitesimals—much along the lines suggested in the original work on calculus in the late 1600s.

■ **Pure equational logic.** Proofs in operator systems always rely on certain underlying rules about equality, such as the equivalence of $u == v$ and $v == u$, and of $u == v$ and $u == v\ /.\ a \to b$. And as Garrett Birkhoff showed in 1935, any equivalence between expressions that holds for all possible forms of operator must have a finite proof using just these rules. (This is the analog of Gödel's Completeness Theorem from page 1152 for pure predicate logic.) But as soon as one introduces actual axioms that constrain the operators this is no longer true—and in general it can be undecidable whether or not a particular equivalence holds.

■ **Multiway systems.** One can use ideas from operator systems to work out equivalences in multiway systems (compare page 1169). One can think of concatenation of strings as being an operator, in terms of which a string like "ABB" can be written $f[f[a, b], b]$. (The arguments to $\circ$ should strictly be distinct constants, but no equivalences are lost by allowing them to be general variables.) Assuming that the rules for a multiway system come in pairs $p \to q$, $q \to p$, like "AB" → "AAA", "AAA" → "AB", these can be written as statements about operators, like $f[a, b] == f[f[a, a], a]$. The basic properties of concatenation then also imply that $f[f[a, b], c] == f[a, f[b, c]]$. And this means that the possible forms for the operator $\circ$ correspond to possible semigroups. Given a particular such semigroup satisfying axioms derived from a multiway system, one can see whether the operator representations of particular strings are equal—and if they are not, then it follows that the strings can never be reached from each other through evolution of the multiway system. (Such operator representations are a rough analog for multiway systems of truth tables.) As an example, with the multiway system "AB" ↔ "BA" some possible forms of operators are shown below. (In this case these are the commutative semigroups. With $k = 2$, elements 6 out of the total of 8 possible semigroups appear; with $k = 3$, 63 out of 113, and with $k = 4$, 1140 out of 3492—all as shown on page 805.) (See also page 952.)



Taking ∘ to be each of these operators, one can work out a representation for any given string like "ABAA" by for example constructing the expression *f[f[f[a, b], a], a]* and finding its value for each of the k^2 possible pairs of values of *a* and *b*. Then for each successive operator, the sets of strings where the arrays of values are the same are as shown below.



Ultimately the sets of strings equivalent under the multiway system are exactly those containing particular numbers of black and white elements. But as the pictures above suggest, only some of the distinctions between sets of strings are ever captured when any specific form for the operator is used.

Just as for operator systems, any bidirectional multiway system will allow a certain set of operators. (When there are multiple rules in the multiway system, tighter constraints are obtained by combining them with *And*.) And the pattern of results for simple multiway systems is roughly similar to those on page 805 for operator systems—although, for example, the associativity of concatenation makes it impossible for example to get the operators for *Nand* and basic logic.

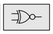
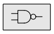
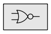
■ **Page 806 · Logic in languages.** Human languages always seem to have single words for AND, OR and NOT. A few have distinct words for OR and XOR: examples are Latin with *vel* and *aut* and Finnish with *vai* and *tai*. NOR is somewhat rare, though Dutch has *noch* and Old English *ne*. (Modern English has only the compound form *neither ... nor*.) But remarkably enough it appears that no ordinary language has a single word for NAND. The reason is not clear. Most people seem to find it difficult to think in terms of NAND (NAND is for example not associative, but then neither is NOR). And NAND on the face of it rarely seems useful in everyday situations. But perhaps these are just reflections of the historical fact that NAND has never been familiar from ordinary languages.

Essentially all computer languages support AND, OR and NOT as ways to combine logical statements; many support AND, OR and XOR as bitwise operations. Circuit design languages like Verilog and VHDL also support NAND, NOR and XNOR. (NAND is the operation easiest to implement with CMOS FETs—the transistors used in most current chips; it was also implemented by pentode vacuum tubes.) Circuit designers sometimes use the linguistic construct "*p* nand *q*".

The Laws of Form presented by George Spencer Brown in 1969 introduce a compact symbolic notation for NAND with any number of arguments and in effect try to develop a way of discussing NAND and reasoning directly in terms of it. (The axioms normally used are essentially the Sheffer ones from page 773.)

■ **Page 806 · Properties.** Page 813 lists theorems satisfied by each function. {0, 1, 6, 7, 8, 9, 14, 15} are commutative (orderless) so that $a∘b = b∘a$, while {0, 6, 8, 9, 10, 12, 14, 15} are associative (flat), so that $a∘(b∘c) = (a∘b)∘c$. (Compare page 886.)

■ **Notations.** Among those in current use are (highlighted ones are supported directly in *Mathematica*):

True	T	1	⊤	⋎						
False	F	0	⊥	⋏						
Not[p]	**¬ p**	p̄	~p	p′	-p	**! p**	⌐ p	N p		(negation)
And[p, q]	**p ∧ q**	p & q	p · q	p q	**p && q**	K p q		(conjunction)		
Or[p, q]	**p ∨ q**	p + q	**p ‖ q**	A p q		(disjunction)				
Xor[p, q]	**p ⊻ q**	p ⊕ q	p ≢ q	p ∨̇ q	J p q		(inequivalence)			
Implies[p, q]	**p ⇒ q**	p ⊃ q	p → q	If[p, q]	C p q	(material implication)				
Equal[p, q]	**p = q**	p ⇔ q	p ≡ q	p ↔ q	p ~ q	E p q		(material equivalence; xnor)		
Nand[p, q]	**p ⊼ q**	p | q	p ⌐ q	(p q)	D p q		(Sheffer stroke; alternative denial)			
Nor[p, q]	**p ⊽ q**	p↓ q	X p q		(joint denial)					

The grouping of terms is normally inferred from precedence of operators (typically ordered ==, ¬, ⊼, ∧, ⊻, ⊽, ∨, ⇒), or explicitly indicated by parentheses, function brackets, or sometimes nested underbars or dots. So-called Polish notation given second-to-last above avoids all explicit delimiters (see page 896).

■ **Page 807 · Universal logical functions.** The fact that combinations of *Nand* or *Nor* are adequate to reproduce any logical function was noted by Charles Peirce around 1880, and became widely known after the work of Henry Sheffer in 1913. (See also page 1096.) *Nand* and *Nor* are the only 2-input functions universal in this sense. ({*Equal*} can for example reproduce only functions *{9, 10, 12, 15}*, *{Implies}* only functions *{10, 11, 12, 13, 14, 15}*, and *{Equal, Implies}* only functions *{8, 9, 10, 11, 12, 13, 14, 15}*.) For 3-input functions, corresponding to elementary cellular automaton rules, 56 of the 256 possibilities turn out to be universal. Of these, 6 are straightforward generalizations of *Nand* and *Nor*. Other universal functions include rules 1, 45 and 202 (*If[a == 1, b, c]*), but not 30, 60 or 110. For large n roughly 1/4 of all n-input functions are universal. (See also page 1175.)

■ **Page 808 · Searching for logic.** For axiom systems of the form $\{\ldots = a\}$ one finds:

	2 variables					3 variables				
number of $\circ$	2	3	4	5	6	2	3	4	5	6
total systems	4	16	80	448	2688	54	405	3402	30618	288684
allow $\barwedge$	0	5	44	168	1532	0	9	124	744	8764
allow only $\barwedge$ etc. for $k=2$	0	0	2	12	76	0	0	12	84	868
allow only $\barwedge$ etc. for $k\leq3$	0	0	0	0	0	0	0	8	16	296
allow only $\barwedge$ etc. for $k\leq4$	0	0	0	0	0	0	0	0	0	100

$\{((b\circ b)\circ a)\circ(a\circ b) = a\}$ allows the $k = 3$ operator 15552 for which the NAND theorem $(p\circ p)\circ q = (p\circ q)\circ q$ is not true. $\{(((b\circ a)\circ c)\circ a)\circ(a\circ c) = a\}$ allows the $k = 4$ operator 95356335 for which even $p\circ q = q\circ p$ is not true. Of the 100 cases that remain when $k = 4$, the 25 inequivalent under renaming of variables and reversing arguments of $\circ$ are

$\{(b\circ(b\circ(a\circ a)))\circ(a\circ(b\circ c)),$
$(b\circ(b\circ(a\circ a)))\circ(a\circ(c\circ b)),\ (b\circ(b\circ(a\circ b)))\circ(a\circ(b\circ c)),$
$(b\circ(b\circ(a\circ b)))\circ(a\circ(c\circ b)),\ (b\circ(b\circ(a\circ c)))\circ(a\circ(c\circ b)),$
$(b\circ(b\circ(b\circ a)))\circ(a\circ(b\circ c)),\ (b\circ(b\circ(b\circ a)))\circ(a\circ(c\circ b)),$
$(b\circ(b\circ(c\circ a)))\circ(a\circ(b\circ c)),\ (b\circ((a\circ b)\circ b))\circ(a\circ(b\circ c)),$
$(b\circ((a\circ b)\circ b))\circ(a\circ(c\circ b)),\ (b\circ((a\circ c)\circ b))\circ(a\circ(c\circ b)),$
$((b\circ c)\circ a)\circ(b\circ(b\circ(a\circ b))),\ ((b\circ c)\circ a)\circ(b\circ(b\circ(a\circ c))),$
$((b\circ c)\circ a)\circ(b\circ((a\circ a)\circ b)),\ ((b\circ c)\circ a)\circ(b\circ((a\circ b)\circ b)),$
$((b\circ c)\circ a)\circ(b\circ((a\circ c)\circ b)),\ ((b\circ c)\circ a)\circ(b\circ((b\circ a)\circ b)),$
$((b\circ c)\circ a)\circ(b\circ((c\circ a)\circ b)),\ ((b\circ c)\circ a)\circ(c\circ(c\circ(a\circ b))),$
$((b\circ c)\circ a)\circ(c\circ(c\circ(a\circ c))),\ ((b\circ c)\circ a)\circ(c\circ((a\circ a)\circ c)),$
$((b\circ c)\circ a)\circ(c\circ((a\circ b)\circ c)),\ ((b\circ c)\circ a)\circ(c\circ((a\circ c)\circ c)),$
$((b\circ c)\circ a)\circ(c\circ((b\circ a)\circ c)),\ ((b\circ c)\circ a)\circ(c\circ((c\circ a)\circ c))\}$

Of these I was able in 2000—using automated theorem proving—to show that the ones given as (g) and (h) in the main text are indeed axiom systems for logic. (My proof essentially as found by Waldmeister is given on page 810.)

If one adds $a\circ b = b\circ a$ to any of the other 23 axioms above then in all cases the resulting axiom system can be shown to reproduce logic. But from any of the 23 axioms on their own I have never managed to derive $p\circ q = q\circ p$. Indeed, it seems difficult to derive much at all from them—though for example I have found a fairly short proof of $(p\circ p)\circ(p\circ q) = p$ from $\{(b\circ(b\circ(b\circ a)))\circ(a\circ(b\circ c)) = a\}$.

It turns out that the first of the 25 axioms allows the $k = 6$ operator 1885760537655023865453442036 and so cannot be logic. Axioms 3, 19 and 23 allow similar operators, leaving 19 systems as candidate axioms for logic.

It has been known since the 1940s that any axiom system for logic must have at least one axiom that involves more than 2 variables. The results above now show that 3 variables suffice. And adding more variables does not seem to help. The smallest axiom systems with more than 3 variables that work up to $k = 2$ are of the form $\{(((b\circ c)\circ d)\circ a)\circ(a\circ d) = a\}$. All turn out also to work at $k = 3$, but fail at $k = 4$. And with 6 NANDs (as in (g) and (h)) no system of the form $\{\ldots = a\}$ works even up to $k = 4$.

For axiom systems of the form $\{\ldots = a,\ a\circ b = b\circ a\}$:

	2 variables					3 variables				
number of $\circ$	4	5	6	7	8	4	5	6	7	8
total systems	4	16	80	448	2688	54	405	3402	30618	288684
allow $\barwedge$	0	5	44	168	1532	0	9	124	744	8764
allow only $\barwedge$ etc. for $k=2$	0	4	20	160	748	0	8	80	736	6248
allow only $\barwedge$ etc. for $k\leq3$	0	0	0	64	16	0	0	32	416	2752
allow only $\barwedge$ etc. for $k\leq4$	0	0	0	48	16	0	0	32	384	2368

With 2 variables the inequivalent cases that remain are

$\{(a\circ b)\circ(a\circ(b\circ(a\circ b))),$
$(a\circ b)\circ(a\circ(b\circ(b\circ b))),\ (a\circ(b\circ b))\circ(a\circ(b\circ(b\circ b)))\}$

but all of these allow the $k = 6$ operator

1885760537655125429738480884

and so cannot correspond to basic logic. With 3 variables, all 32 cases with 6 NANDs are equivalent to $(a\circ b)\circ(a\circ(b\circ c))$, which is axiom system (f) in the main text. With 7 NANDs there are 8 inequivalent cases:

$\{(a\circ a)\circ(b\circ(b\circ(a\circ c))),\ (a\circ b)\circ(a\circ(b\circ(a\circ b))),\ (a\circ b)\circ(a\circ(b\circ(a\circ c))),$
$(a\circ b)\circ(a\circ(b\circ(b\circ b))),\ (a\circ b)\circ(a\circ(b\circ(b\circ c))),$
$(a\circ b)\circ(a\circ(b\circ(c\circ c))),\ (a\circ b)\circ(a\circ(c\circ(a\circ c))),\ (a\circ b)\circ(a\circ(c\circ(c\circ c)))\}$

and of these at least 5 and 6 can readily be proved to be axioms for logic.

Any axiom system must consist of equivalences valid for the operator it describes. But the fact that there are fairly few short such equivalences for *Nand* (see page 818) implies that there can be no axiom system for *Nand* with 6 or less NANDs except the ones discussed above.

■ **Two-operator logic.** If one allows two operators then one can get standard logic if one of these operators is forced to be *Not* and the other is forced to be *And*, *Or* or *Implies*—or in fact any of operators 1, 2, 4, 7, 8, 11, 13, 14 from page 806.

A simple example that allows *Not* and either *And* or *Or* is the Robbins axiom system from page 773. Given the first two axioms (commutativity and associativity) it turns out that no shorter third axiom will work in this case (though ones such as $f[g[f[a, g[f[a, b]]]], g[g[b]]] = b$ of the same size do work).

Much as in the single-operator case, to reproduce logic two pairs of operators must be allowed for $k = 2$, none for $k = 3$, 12 for $k = 4$, and so on. Among single axioms, the shortest that works up to $k = 2$ is $(\neg(\neg(\neg b \vee a) \vee \neg(a \vee b))) = a$. The shortest that works up to $k = 3$ is $(\neg(\neg(a \vee b) \vee \neg b) \vee \neg(\neg a \vee a)) = b$. It is known, however, that at least 3 variables must appear in order to reproduce logic, and an example of a single axiom with 4 variables that has been found recently to work is $\{(\neg(\neg(c \vee b) \vee \neg a) \vee \neg(\neg(\neg d \vee d) \vee \neg a \vee c)) = a\}$.

■ **Page 808 · History.** (See page 1151.) (c) was found by Henry Sheffer in 1913; (e) by Carew Meredith in 1967. Until this book, very little work appears to have been done on finding short axioms for logic in terms of *Nand*. Around 1949 Meredith found the axiom system

$$\{(a \circ (b \circ c)) \circ (a \circ (b \circ c)) = \\ ((c \circ a) \circ a) \circ ((b \circ a) \circ a),\ (a \circ a) \circ (b \circ a) = a\}$$

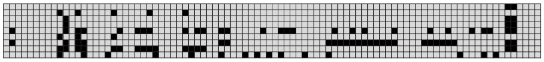
In 1967 George Spencer Brown found (see page 1173)

$$\{(a \circ a) \circ ((b \circ b) \circ b) = a, \\ a \circ (b \circ c) = (((c \circ c) \circ a) \circ ((b \circ b) \circ a)) \circ (((c \circ c) \circ a) \circ ((b \circ b) \circ a))\}$$

and in 1969 Meredith also gave the system

$$\{a \circ (b \circ (a \circ c)) = a \circ (b \circ (b \circ c)),\ (a \circ a) \circ (b \circ a) = a,\ a \circ b = b \circ a\}$$

■ **Page 812 · Theorem distributions.** The picture below shows which of the possible theorems from page 812 hold for each of the numbered standard mathematical theories from page 805. The theorem close to the right-hand end valid in many cases is $(p \circ p) \circ p = p \circ (p \circ p)$. The lack of regularity in this picture can be viewed as a sign that it is difficult to tell which theorems hold, and thus in effect to do mathematics.

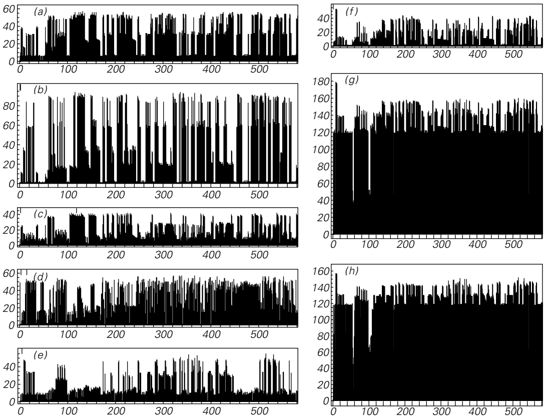


■ **Page 814 · Multivalued logic.** As noted by Jan Łukasiewicz and Emil Post in the early 1920s, it is possible to generalize ordinary logic to allow k values *Range[0, 1, 1/(k - 1)]*, say with 0 being *False*, and 1 being *True*. Standard operations in logic can be generalized as *Not[a_] = 1 - a*, *And[a_, b_] = Min[a, b]*, *Or[a_, b_] = Max[a, b]*, *Xor[a_, b_] = Abs[a - b]*, *Equal[a_, b_] = 1 - Abs[a - b]*, *Implies[a_, b_] = 1 - UnitStep[a - b] (a - b)*. An alternative generalization for *Not* is *Not[a_] := Mod[(k - 1) a + 1, k]/(k - 1)*. The function *Nand[a_, b_] := Not[And[a, b]]* used in the main text turns out to be universal for any k. Axiom systems can be set up for multivalued logic, but they are presumably more complicated than for ordinary $k = 2$ logic. (Compare page 1171.)

The idea of intermediate truth values has been discussed intermittently ever since antiquity. Often—as in the work of George Boole in 1847—a continuum of values between 0 and 1 are taken to represent probabilities of events, and this is the basis for the field of fuzzy logic popular since the 1980s.

■ **Page 814 · Proof lengths in logic.** As discussed on page 1170 equivalence between expressions can always be proved by transforming to and from canonical form. But with n variables a DNF-type canonical form can be of size 2^n—and can take up to at least 2^n proof steps to reach. And indeed if logic proofs could in general be done in a number of steps that increases only like a polynomial in n this would mean that the NP-complete problem of satisfiability could also be solved in this number of steps, which seems very unlikely (see page 768).

In practice it is usually extremely difficult to find the absolute shortest proof of a given logic theorem—and the exact length will depend on what axiom system is used, and what kinds of steps are allowed. In fact, as mentioned on page 1155, if one does not allow lemmas some proofs perhaps have to become exponentially longer. The picture below shows in each of the axiom systems from page 808 the lengths of the shortest proofs found by a version of Waldmeister (see page 1158) for all 582 equivalences (see page 818) that involve two variables and up to 3 NANDs on either side.



The longest of these are respectively $\{57, 94, 42, 57, 55, 53, 179, 157\}$ and occur for theorems

$$\{(((a \barwedge a) \barwedge b) \barwedge b) = (((a \barwedge b) \barwedge a) \barwedge a), \\ (a \barwedge (a \barwedge (a \barwedge a))) = (a \barwedge ((a \barwedge b) \barwedge b)),\ (((a \barwedge a) \barwedge a) \barwedge a) = \\ (((a \barwedge a) \barwedge b) \barwedge a),\ (((a \barwedge a) \barwedge b) \barwedge b) = (((a \barwedge b) \barwedge a) \barwedge a), \\ (a \barwedge ((b \barwedge b) \barwedge a)) = (b \barwedge ((a \barwedge a) \barwedge b)),\ ((a \barwedge a) \barwedge a) = ((b \barwedge b) \barwedge b), \\ ((a \barwedge a) \barwedge a) = ((b \barwedge b) \barwedge b),\ ((a \barwedge a) \barwedge a) = ((b \barwedge b) \barwedge b)\}$$

Note that for systems that do not already have it as an axiom, most theorems use the lemma $(a \barwedge b) = (b \barwedge a)$ which takes respectively $\{6, 1, 8, 49, 8, 1, 119, 118\}$ steps to prove.

■ **Page 818 · NAND theorems.** The total number of expressions with n NANDs and s variables is: *Binomial[2 n, n]* $s^{n+1}/(n+1)$ (see page 897). With $s = 2$ and n from 0 to 7 the number of these *True* for all values of variables is $\{0, 0, 4, 0, 80, 108, 2592, 7296\}$, with the first few distinct ones being (see page 781)

$$\{(p \barwedge p) \barwedge p,\ (((p \barwedge p) \barwedge p) \barwedge p) \barwedge p,\ (((p \barwedge p) \barwedge p) \barwedge q) \barwedge q\}$$

The number of unequal expressions obtained is *{2, 3, 3, 7, 10, 15, 12, 16}* (compare page 1096), with the first few distinct ones being

$$\{p, p \barwedge p, p \barwedge q, (p \barwedge p) \barwedge p, (p \barwedge q) \barwedge p, (p \barwedge p) \barwedge q\}$$

Most of the axioms from page 808 are too long to appear early in the list of theorems. But those of system (d) appear at positions *{3, 15, 568}* and those of (e) at *{855, 4}*.

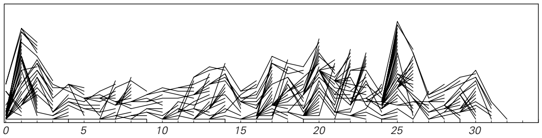
(See also page 1096.)

▪ **Page 819 · Finite axiomatizability.** It is known that the axiom systems (such as Peano arithmetic and set theory) given with axiom schemas on pages 773 and 774 can be set up only with an infinite number of individual axioms. But because such axioms can be described by schemas they must all have similar forms, so that even though the definition in the main text suggests that each corresponds to an interesting theorem these theorems are not in a sense independently interesting. (Note that for example the theory of specifically finite groups cannot be set up with a finite number even of schemas—or with any finite procedure for checking whether a given candidate axiom should be included.)

▪ **Page 820 · Empirical metamathematics.** One can imagine a network representing some field of mathematics, with nodes corresponding to theorems and connections corresponding to proofs, gradually getting filled in as the field develops. Typical rough characterizations of mathematical results—independent of their detailed history—might then end up being something like the following:

- lemma: short theorem appearing at an intermediate stage in a proof
- corollary: theorem connected by a short proof to an existing theorem
- easy theorem: theorem having a short proof
- difficult theorem: theorem having a long proof
- elegant theorem: theorem whose statement is short and somewhat unique
- interesting theorem (see page 817): theorem that cannot readily be deduced from earlier theorems, but is well connected
- boring theorem: theorem for which there are many others very much like it
- useful theorem: theorem that leads to many new ones
- powerful theorem: theorem that substantially reduces the lengths of proofs needed for many others
- surprising theorem: theorem that appears in an otherwise sparse part of the network of theorems
- deep theorem: theorem that connects components of the network of theorems that are otherwise far away
- important theorem: theorem that allows a broad new area of the network to be reached.

The picture below shows the network of theorems associated with Euclid's *Elements*. Each stated theorem is represented by a node connected to the theorems used in its stated proof. (Only the shortest connection from each theorem is shown explicitly.) The axioms (postulates and common notions) are given in the first column on the left, and successive columns then show theorems with progressively longer proofs. (Explicit annotations giving theorems used in proofs were apparently added to editions of Euclid only in the past few centuries; the picture below extends the usual annotations in a few cases.) The theorem with the longest proof is the one that states that there are only five Platonic solids.



▪ **Speedups in other systems.** Multiway systems are almost unique in being able to be sped up just by adding "results" already derived in the multiway system. In other systems, there is no such direct way to insert such results into the rules for the system.

▪ **Character of mathematics.** Since at least the early 1900s several major schools of thought have existed:

- *Formalism* (e.g. David Hilbert): Mathematics studies formal rules that have no intrinsic meaning, but are relevant because of their applications or history.
- *Platonism* (e.g. Kurt Gödel): Mathematics involves trying to discover the properties of a world of ideal mathematical forms, of which we in effect perceive only shadows.
- *Logicism* (e.g. Gottlob Frege, Bertrand Russell): Mathematics is an elaborate application of logic, which is itself fundamental.
- *Intuitionism* (e.g. Luitzen Brouwer): Mathematics is a precise way of handling ideas that are intuitive to the human mind.

The results in this book establish a new point of view somewhere between all of these.

▪ **Invention versus discovery in mathematics.** One generally considers things invented if they are created in a somewhat arbitrary way, and discovered if they are identified by what seems like a more inexorable process. The results of this section thus strongly suggest that the basic directions taken by mathematics as currently practiced should mostly be considered invented, not discovered. The new kind of science that I describe in this book, however, tends to suggest forms of mathematics that involve discovery rather than invention.

▪ **Ordering of constructs.** One can deduce some kind of ordering among standard mathematical constructs by seeing how difficult they are to implement in various systems—such as cellular automata, Turing machines and Diophantine equations. My experience has usually been that addition is easiest, followed by multiplication, powers, Fibonacci numbers, perfect numbers and then primes. And perhaps this is similar to the order in which these constructs appeared in the early history of mathematics. (Compare page 640.)

▪ **Mathematics and the brain.** A possible reason for some constructs to be more common in mathematics is that they are somehow easier for human brains to manipulate. Typical human experience makes small positive integers and simple shapes familiar—so that all human brains are at least well adapted to such constructs. Yet of the limited set of people exposed to higher mathematics, different ones often seem to think in bizarrely different ways. Some think symbolically, presumably applying linguistic capabilities to algebraic or other representations. Some think more visually, using mechanical experience or visual memory. Others seem to think in terms of abstract patterns, perhaps sometimes with implicit analogies to musical harmony. And still others—including some of the purest mathematicians—seem to think directly in terms of constraints, perhaps using some kind of abstraction of everyday geometrical reasoning.

In the history of mathematics there are many concepts that seem to have taken surprisingly long to emerge. And sometimes these are ones that people still find hard to grasp. But they often later seem quite simple and obvious—as with many examples in this book.

It is sometimes thought that people understand concepts in mathematics most easily if they are presented in the order in which they arose historically. But for example the basic notion of programmability seems at some level quite easy even for young children to grasp—even though historically it appeared only recently.

In designing *Mathematica* one of my challenges was to use constructs that are at least ultimately easy for people to understand. Important criteria for this in my experience include specifying processes directly rather than through constraints, the explicitness in the representation of input and output, and the existence of small, memorable, examples. Typically it seems more difficult for people to understand processes in which larger numbers of different things happen in parallel. (Notably, *FoldList* normally seems more difficult to understand than *NestList*.) Tree structures such as *Mathematica* expressions are fairly easy to understand. But I have never found a way to make general networks similarly easy, and I am beginning to suspect that they may be fundamentally difficult for brains to handle.

▪ **Page 821 · Frameworks.** Symbolic integration was in the past done by a collection of ad hoc methods like substitution, partial fractions, integration by parts, and parametric differentiation. But in *Mathematica* *Integrate* is now almost completely systematic, being based on structure theorems for finding general forms of integrals, and on general representations in terms of *MeijerG* and other functions. (In recognizing, for example, whether an expression involving a parameter can have a pole undecidable questions can in principle come up, but they seem rare in practice.) Proofs are essentially always still done in an ad hoc way—with a few minor frameworks like enumeration of cases, induction, and proof by contradiction (reductio ad absurdum) occasionally being used. (More detailed frameworks are used in specific areas; an example are ϵ-δ arguments in calculus.) But although still almost unknown in mainstream mathematics, methods from automated theorem proving (see page 1157) are beginning to allow proofs of many statements that can be formulated in terms of operator systems to be found in a largely systematic way (e.g. page 810). (In the case of Euclidean geometry—which is a complete axiom system—algebraic methods have allowed complete systematization.) In general, the more systematic the proofs in a particular area become, the less relevant they will typically seem compared to the theorems that they establish as true.

Intelligence in the Universe

▪ **Page 822 · Animism.** Attributing abstract human qualities such as intelligence to systems in nature is a central part of the idea of animism, discussed on page 1195.

▪ **Page 822 · The weather.** Almost all the intricate variations of atmospheric temperature, pressure, velocity and humidity that define the weather we see are in the end determined by fairly simple underlying rules for fluid behavior. (Details of phase changes in water are also important, as are features of topography, ocean currents, solar radiation levels and presumably land use.) Our everyday personal attempts to predict the weather tend to be based just on looking at local conditions and then recalling what happened when we saw these conditions before. But ever since the mid-1800s synoptic weather maps of large areas have been available that summarize conditions in terms of features like fronts and cyclones. And predictions made by looking at simple trends in these features tend at least in some situations to work fairly well. Starting in the 1940s more systematic efforts to predict weather were made by using computers to run approximations to fluid equations. The approximations have improved as larger computers have become available. But even though millions of discrete samples are now used, each one typically still represents something much larger then for example a single cloud. Yet ever since the 1970s, the approach has had at least some success in making predictions up to several days in advance. But although there has been gradual improvement it is usually assumed that—like in the Lorenz equations—the phenomenon of chaos must make forecasts that are based on even slightly different initial measurements eventually diverge exponentially (see page 972). Almost certainly this does indeed happen in at least some critical situations. But it seems that over most of a typical weather map there is no such sensitivity—so that in the end the difficulties of weather prediction are probably much more a result of computational irreducibility and of the sophisticated kinds of computations that the Principle of Computational Equivalence implies should often occur even in simple fluids.

▪ **Page 822 · Defining intelligence.** The problem of defining intelligence independent of specific education and culture has been considered important for human intelligence testing since the beginning of the 1900s. Charles Spearman suggested in 1904 that there might be a general intelligence factor (usually called *g*) associated with all intellectual tasks. Its nature was never very clear, but it was thought that its value could be inferred from performance on puzzles involving numbers, words and pictures. By the 1980s, however, there was increasing emphasis on the idea that different types of human tasks require different types of intelligence. But throughout the 1900s psychologists occasionally tried to give general definitions of intelligence—initially usually in terms of learning or problem-solving capabilities; later more often in terms of adaptation to complex environments.

Particularly starting at the end of the 1800s there was great interest in whether animals other than humans could be considered intelligent. The most common general criterion used was the ability to show behavior based on conceptual or abstract thinking rather than just simple instincts. More specific criteria also included ability to use tools, plan actions, use language, solve logical problems and do arithmetic. But by the mid-1900s it became increasingly clear that it was very difficult to interpret actual observations—and that unrecognized cues could for example often account for the behavior seen.

When the field of artificial intelligence began in the mid-1900s there was some discussion of appropriate definitions of intelligence (see page 1099). Most focused on mathematical or other problem solving, though some—such as the Turing test—emphasized everyday conversation with humans.

▪ **Page 823 · Mimesis.** The notion of inanimate analogs of memory—such as impressions in wax—was discussed for example by Plato in antiquity.

▪ **Page 823 · Defining life.** Greek philosophers such as Aristotle defined life by the presence of some form of soul, and the idea that there must be a single unique feature associated with life has always remained popular. In the 1800s the notion of a "life force" was discussed—and thought to be associated perhaps with chemical properties of protoplasm, and perhaps with electricity. The discovery from the mid-1800s to the mid-1900s of all sorts of elaborate chemical processes in living systems led biologists often to view life as defined by its ability to maintain fixed overall structure while achieving chemical functions such as metabolism. When the Second Law of Thermodynamics was formulated in the mid-1800s living systems were usually explicitly excluded (see page 1021), and by the 1930s physicists often considered local entropy decrease a defining feature of life. Among geneticists and soon mathematicians self-reproduction was usually viewed as the defining feature of life, and following the discovery of the structure of DNA in 1953 it came to be widely believed that the presence of self-replicating elements must be fundamental to life. But the recognition that just copying information is fairly easy led in the 1960s to definitions of life based on the large amounts of information encoded in its genetic material, and later to ones based on the apparent difficulty of deriving this information (see page 1069). And perhaps in part reacting to my discoveries about cellular automata it became popular in the 1980s to mention adaptation and essential interdependence of large numbers of different kinds of parts as further necessary characteristics of life. Yet in the end every single general definition that has been given both includes systems that are not normally considered alive, and excludes ones that are. (Self-reproduction, for example, suggests that flames are alive, but mules are not.)

One of the features that defines life on Earth is the presence of DNA, or at least RNA. But as one looks at smaller molecules they become less specific to living systems. It is sometimes thought significant that living systems perpetuate the use of only one chirality of molecules, but actually this can quite easily be achieved by various forms of non-chemical input without life.

The Viking spacecraft that landed on Mars in the 1970s tried specific tests for life on soil samples—essentially whether gases were generated when nutrients were added, whether this behavior changed if the samples were first heated, and whether molecules common in terrestrial life were present. The tests gave confusing results, presumably having to do not with life, but rather with details of martian soil chemistry

▪ **Origin of life.** Fossil traces of living cells have been found going back more than 3.8 billion years—to perhaps as little as 700 million years after the formation of the Earth. There were presumably simpler forms of life that preceded the advent of recognizable cells, and even if life arose more than once it is unlikely that evidence of this would remain. (One sees many branches in the fossil record—such as organisms with dominant symmetries other than fivefold—but all seem to have the same ancestry.)

From antiquity until the 1700s it was widely believed that smaller living organisms arise spontaneously in substances like mud, and this was not finally disproved until the 1860s. Controversy surrounding the theory of biological evolution in the late 1800s dissuaded investigation of non-biological origins for life, and at the end of the 1800s it was for example suggested that life on Earth might have arisen from spores of extraterrestrial origin. In the 1920s the idea developed that electrical storms in the atmosphere of the early Earth could lead to production of molecules seen in living systems—and this was confirmed by the experiment of Stanley Miller and Harold Urey in 1953. The molecules obtained were nevertheless still fairly simple—and as it turns out most of them have now also been found in interstellar space. Starting in the 1960s suggestions were made for the chemical and other roles of constituents of the crust as well as atmosphere. Schemes for early forms of self-replication were invented based on molecules such as RNA and on patterns in clay-like materials. (The smallest known system that independently replicates itself is a mycoplasma bacterium with about 580,000 base pairs and perhaps 470 genes. Viroids can be as small as 10,000 atoms but require a host for replication.) In the 1970s it then became popular to investigate complicated cycles of chemical reactions that seemed analogous to ones found in living systems. But with the advent of widespread computer simulations in the 1980s it became clear that all sorts of features normally associated with life were actually rather easy to obtain. (See note above.)

▪ **Page 824 · Self-reproduction.** That one can for example make a mold that will produce copies of a shape has been known since antiquity (see note above). The cybernetics movement highlighted the question of what it takes for self-reproduction to occur autonomously, and in 1952 John von Neumann designed an elaborate 2D cellular automaton that would automatically make a copy of its initial configuration of cells (see page 876). In the course of the 1950s suggestions of several increasingly simple mechanical systems capable of self-reproduction were made—notably by Lionel Penrose. The phenomenon in the main text was noticed around 1961 by Edward Fredkin (see page 877). But while it shows some of the essence of self-reproduction, it lacks many of the more elaborate features common in biological self-reproduction. In the 1980s, however, such features were nevertheless surprisingly often present in computer viruses and worms. (See also page 1092.)

▪ **Page 825 · Extraterrestrial life.** Conditions thermally and chemically similar to those on Earth have presumably existed on other bodies in the solar system. Venus, Mars, Europa (a moon of Jupiter) and Titan (a moon of Saturn) have for example all probably had liquid water at some time. But there is so far no evidence for life now or in the past on any of these. Yet if life had arisen one might expect it to have become widespread, since at least on Earth it has managed to spread to many extremes of temperature, pressure and chemical composition. On several occasions structures have been found in extraterrestrial rocks that look somewhat like small versions of microorganism fossils (most notably in 1996 in a meteorite from Mars discovered in Antarctica). But almost certainly these structures have nothing to do with life, and are instead formed by ordinary precipitation of minerals. And although even up to the 1970s it was thought that life might well be found on Mars, it now seems likely that there is nothing quite like terrestrial life anywhere else in our solar system. (Even if life is found elsewhere it might still have originally come from Earth, say via meteorites, since dormant forms such as spores can apparently survive for long periods in space.)

Away from our solar system there is increasing evidence that most stars have planets with a distribution of sizes—so presumably conditions similar to Earth are fairly common. But thus far it has not been possible to see—say in planetary atmospheres—whether there are for example molecules similar to ones characteristically found in life on Earth.

▪ **Forms of living systems.** This book has shown that even with underlying rules of some fixed type a vast range of different forms can often be produced. And this makes it reasonable to expect that with appropriate genetic programs the chemical building blocks of life on Earth should in principle allow a vast range of forms. But the comparative weakness of natural selection (see page 391) has meant that only a limited set of such forms have actually been explored. And from the experience of this book I suspect that what others might even be nearby is effectively impossible to foresee. The appearance in engineering of forms somewhat like those in living systems should not be taken to imply that other forms are fundamentally difficult to produce; instead I suspect that it is more a reflection of the copying of living systems for engineering purposes. The overall morphology of living systems on Earth seems to be greatly affected by their basically gelatinous character. So even systems based on solids or gases would likely not be recognized by us as life.

▪ **Page 825 · History.** Although Greek philosophers such as Democritus believed that there must be an infinite number of worlds all with inhabitants like us, the prevailing view in antiquity—later supported by theological arguments—was that the Earth is special, and the only abode of life. However, with the development of Copernican ideas in the 1600s it came to be widely though not universally believed, even in theological circles, that other planets—as well as the Moon—must have inhabitants like us. Many astronomers attributed features they saw on the Moon to life if not intelligence, but in the late 1800s, after it was found that the Moon has no atmosphere, belief in life there began to wane. Starting in the 1870s, however, there began to be great interest in life on Mars, and it was thought—perhaps following the emphasis on terrestrial canal-building at the time—that a vast network of canals on Mars had been observed. And although in 1911 the apparent building of new canals on Mars was still being soberly reported by newspapers, there was by the 1920s increasing skepticism. The idea that lichens might exist on Mars and be responsible for seasonal changes in color nevertheless became popular, especially after the discovery of atmospheric carbon dioxide in 1947. Particularly in the 1920s there had been occasional claims of extraterrestrial radio signals (see page 1188), but by the 1950s interest in extraterrestrial intelligence had largely transferred to science fiction (see page 1190). Starting in the late 1940s many sightings were reported of UFOs believed to be alien spacecraft, but by the 1960s these were increasingly discredited. It had been known since the mid-1800s that many other stars are much like the Sun, but it was not until the 1950s that evidence of planets around other stars began to accumulate. Following a certain amount of discussion in the physics community in the 1950s, the first explicit search for extraterrestrial intelligence with a radio telescope was done in 1960 (see page 1189). In the 1960s landings of spacecraft on the Moon confirmed the absence of life there—though returning Apollo astronauts were still quarantined to guard against possible lunar microbes. And despite substantial expectations to the contrary, when spacecraft landed on Mars in 1976 they found no evidence of life there. Some searches for extraterrestrial signals have continued in the radio astronomy community, but perhaps because of its association with science fiction, the topic of extraterrestrial intelligence has generally not been popular with professional scientists. With the rise of amateur science on the web and the availability of low-cost radio telescope components the late 1990s may however have seen a renewal of serious interest.

▪ **Page 826 · Bird songs.** Essentially all birds produce calls of some kind, but complex songs are mainly produced by male songbirds, usually in breeding season. Their general form is inherited, but specifics are often learned through imitation during a fixed period of infancy, leading birds in local areas to have distinctive songs. The songs sometimes seem to be associated with attracting mates, and sometimes with defining territory—but often their function is unclear, even when one bird seems to sing in response to another. (There are claims, however, that parrots can learn to have meaningful conversations with humans.) The syrinxes of songbirds have two membranes, which can vibrate independently, in a potentially complex way. A specific region in bird brains appears to coordinate singing; the region contains a few tens of thousands of nerve cells, and is larger in species with more complex songs.

Famous motifs from human music are heard in bird songs probably more often than would be expected by chance. It may be that some common neural mechanism makes the motifs seem pleasing to both birds and humans. Or it could be that humans find them pleasing because they are familiar from bird songs.

▪ **Page 826 · Whale songs.** Male whales can produce complex songs lasting tens of minutes during breeding season. The songs often include rhyme-like repeating elements. At a given time all whales in a group typically sing almost the same song, which gradually changes. The function of the song is quite unclear. It has been claimed that its frequencies are optimized for long-range transmission in the ocean, but this appears not to be the case. In dolphins, it is known that one dolphin can produce patterns of sound that are repeated by a specific other dolphin.

▪ **Page 826 · Animal communication.** Most animals that live in groups have the capability to produce at least a few specific auditory, visual (e.g. gestures and displays), chemical (e.g. pheromones) or other signals in response to particular situations such as danger. Some animals have also been found to produce much more complex and varied signals.

For example it was discovered in the 1980s that elephants can generate elaborate patterns of sounds—but at frequencies below human hearing. Animals such as octopuses and particularly cuttlefish can show complex and changing patterns of pigmentation. But despite a fair amount of investigation it remains unclear whether these represent more than just simple responses to the environment.

■ **Page 826 · Theories of communication.** Over the course of time the question of what the essential features of communication are has been discussed from many different angles. It appears to have always been a common view that communication somehow involves transferring thoughts from one mind to another. Even in antiquity it was nevertheless recognized that all sorts of aspects of language are purely matters of convention, so that shared conventions are necessary for verbal communication to be possible. In the 1600s the philosophical idea that the only way to get information with certainty is from the senses led to emphasis on observable aspects of communication, and to the conclusion that there is no way to tell whether an accurate transfer of abstract thoughts has occurred between one mind and another. In the late 1600s Gottfried Leibniz nevertheless suggested that perhaps a universal language—modelled on mathematics—could be created that would represent all truths in an objective way accessible to any mind (compare page 1149). But by the late 1800s philosophers like Charles Peirce had developed the idea that communication must be understood purely in terms of its observable features and effects. Three levels of so-called semiotics were then discussed. The first was syntax: the grammatical or other structure of a sequence of verbal or other elements. The second was semantics: the standardized meaning or meanings of the sequence of elements. And the third was pragmatics: the observable effect on those involved in the communication. In the early 1900s, the logical positivism movement suggested that perhaps a universal language or formalism based on logic could be developed that would allow at least scientific truths to be communicated in an unambiguous way not affected by issues of pragmatics—and that anything that could not be communicated like this was somehow meaningless. But by the 1940s it came to be believed—notably by Ludwig Wittgenstein—that ordinary language, with its pragmatic context, could in the end communicate fundamentally more than any formalized logical system, albeit more ambiguously.

Ever since antiquity work has been done to formalize grammatical and other rules of individual human languages. In the early 1900s—notably with the work of Ferdinand de Saussure—there began to be more emphasis on the general question of how languages really operate, and the point was made that the verbal elements or signs in a language should be viewed as somehow intermediate between tangible entities like sounds and abstract thoughts and concepts. The properties of any given sign were recognized as arbitrary, but what was then thought to be essential about a language is the structure of the network of relations between signs—with the ultimate meaning of any given sign inevitably depending on the meanings of signs related to it (as later emphasized in deconstructionism). By the 1950s anthropological studies of various languages—notably by Benjamin Whorf—had encouraged the idea that concepts that did not appear to fit in certain languages simply could not enter the thinking of users of those languages. Evidence to the contrary (notably about past and future among Hopi speakers) eroded this strong form of the so-called Sapir-Whorf hypothesis, so that by the 1970s it was generally believed just that language can have an influence on thinking—a phenomenon definitely seen with mathematical notation and computer languages. Starting in the 1950s, especially with the work of Noam Chomsky, there were claims of universal features in human languages—independent of historical or cultural context (see page 1103). But at least among linguists these are generally assumed just to reflect common aspects of verbal processing in the human brain, not features that must necessarily appear in any conceivable language. (And it remains unclear, for example, to what extent non-verbal forms of communication such as music, gestures and visual ornament show the same grammatical features as ordinary languages.)

The rise of communications technology in the early 1900s led to work on quantitative theories of communication, and for example in 1928 Ralph Hartley suggested that an objective measure of the information content of a message with n possible forms is `Log[n]`. (Similar ideas arose around the same time in statistics, and in fact there had already been work on probabilistic models of written language by Andrei Markov in the 1910s.) In 1948 Claude Shannon suggested using a measure of information based on p `Log[p]`, and there quickly developed the notion that this could be used to find the fundamental redundancy of any sequence of data, independent of its possible meaning (compare page 1071). Human languages were found on this basis to have substantial redundancy (see page 1086), and it has sometimes been suggested that this is important to their operation—allowing errors to be corrected and details of different users to be ignored. (There are also obvious features which reduce redundancy—for example that in most languages common words tend to be short. One can also imagine models of the historical development of languages which will tend to lead to redundancy at the level of Shannon information.)

▪ **Mathematical notation.** While it is usually recognized that ordinary human languages depend greatly on history and context, it is sometimes believed that mathematical notation is somehow more universal. But although it so happens that essentially the same mathematical notation is in practice used all around the world by speakers of every ordinary language, I do not believe that it is in any way unique or inevitable, and in fact I think it shows most of the same issues of dependence on history and context as any ordinary language.

As a first example, consider the case of numbers. One can always just use *n* copies of the same symbol to represent an integer *n*—and indeed this idea seems historically to have arisen independently quite a few times. But as soon as one tries to set up a more compact notation there inevitably seem to be many possibilities. And so for example the Greek and Roman number systems were quite different from current Hindu-Arabic base-10 positional notation. Particularly from working with computers it is often now assumed that base-2 positional notation is somehow the most natural and fundamental. But as pages 560 and 916 show, there are many other quite different ways to represent numbers, each with different levels of convenience for different purposes. And it is fairly easy to see how a different historical progression might have ended up making another one of these seem the most natural.

The idea of labelling entities in geometrical diagrams by letters existed in Babylonian and Greek times. But perhaps because until after the 1200s numbers were usually also represented by letters, algebraic notation with letters for variables did not arise until the late 1500s. The idea of having notation for operators emerged in the early 1600s, and by the end of the 1600s, notably with the work of Gottfried Leibniz, essentially all the basic notation now used in algebra and calculus had been established. Most of it was ultimately based on shortenings and idealizations of ordinary language, an important early motivation just being to avoid dependence on particular ordinary languages. Notation for mathematical logic began to emerge in the 1880s, notably with the work of Giuseppe Peano, and by the 1930s it was widely used as the basis for notation in pure mathematics.

In its basic structure of operators, operands, and so on, mathematical notation has always been fairly systematic—and is close to being a context-free language. (In many ways it is like a simple idealization of ordinary language, with operators being like verbs, operands nouns, and so on.) And while traditional mathematical notation suffers from some inconsistencies and ambiguities, it was possible in developing *Mathematica* StandardForm to set up something very close that can be interpreted uniquely in all cases. Mathematical notation works well for things like ordinary formulas that involve a comparatively small number of basic operations. But there has been no direct generalization for more general constructs and computations. And indeed my goal in designing *Mathematica* was precisely to provide a uniform notation for these (see page 852). Yet to make this work I had to use names derived from ordinary language to specify the primitives I defined.

▪ **Computer communication.** Most protocols for exchanging data between computers have in the end traditionally had rather simple structures—with different pieces of information usually being placed at fixed positions, or at least being arranged in predefined sequences—or sometimes being given in name-value pairs. A more general approach, however, is to use tree-structured symbolic expressions of the kind that form the basis for *Mathematica*—and now in essence appear in XML. In the most general case one can imagine directly exchanging a representation of a program, that is run on the computer that receives it, and induces whatever effect one wants. A simple example from 1984 is *PostScript*, which can specify a picture by giving a program for constructing it; a more sophisticated example from the late 1990s is client-side Java. (Advanced forms of data compression can also be thought of as working by sending simple programs.) But a practical problem in exchanging arbitrary programs is the difficulty of guarding against malicious elements like viruses. And although at some level most communications between present-day computers are very regular and structured, this is often obscured by compression or encryption.

When a program is sent between computers it is usually encoded in a syntactically very straightforward way. But computer languages intended for direct use by humans almost always have more elaborate syntax that is a simple idealization of ordinary human language (see page 1103). There are in practice many similarities between different computer languages. Yet except in very low-level languages few of these are necessary consequences of features or limitations of actual computers. And instead most of them must be viewed just as the results of shared history—and the need to fit in with human cognitive abilities.

▪ **Meaning in programs.** Many issues about meaning arise for computer languages in more defined versions of the ways they arise for ordinary languages. Input to a computer language will immediately fail to be meaningful if it does not conform to a certain definite syntax. Before the input can have a chance of specifying meaningful action there are often all sorts of issues about whether variables in it refer to entities that can be considered to exist. And even if this is resolved, one can still get something that is in effect nonsense and does not usefully run. In most traditional computer languages it is usually the case that most programs chosen at random will just crash if run, often as a result of trying to write to memory outside the arrays they have allocated. In *Mathematica*, there is almost no similar issue, and programs chosen at random tend instead just to return unchanged. (Compare page 101.)

For the kinds of systems like cellular automata that I have discussed in this book programs chosen at random do very often produce some sort of non-trivial behavior. But as discussed in the main text there is still an issue of when this behavior can reasonably be considered meaningful.

For some purposes a more direct analog of messages is not programs or rules for systems like cellular automata but instead initial conditions. And one might imagine that the very process of running such initial conditions in a system with appropriate underlying rules would somehow be what corresponds to their meaning. But if one was just given a collection of initial conditions without any underlying rules one would then need to find out what underlying rules one was supposed to use in order to determine their meaning. Yet the system will always do something, whatever rules one uses. So then one is back to defining criteria for what counts as meaningful behavior in order to determine—by a kind of generalization of cryptanalysis—what rules one is supposed to use.

■ **Meaning and regularity.** If one considers something to show regularity one may or may not consider it meaningful. But if one considers something random then usually one will also consider it meaningless. For to say that something is meaningful normally implies that one somehow comes to a conclusion from it. And this typically implies that one can find some summary of some aspect of it—and thus some regularity. Yet there are still cases where things that are presumably quite random are considered meaningful—prices in financial markets being one example.

■ **Page 828 · Forms of artifacts.** Much as in biological evolution, once a particular engineering construct has been found to work it normally continues to be used. Examples with characteristic forms include (in rough order of their earliest known use): arrowheads, boomerangs, saws, boxes, stairs, fishhooks, wheels, arches, forks, balls, kites, lenses, springs, catenaries, cogs, screws, chains, trusses, cams, linkages, propellers, clocksprings, parabolic reflectors, airfoils, corrugation, zippers, and geodesic domes. It is notable that not even nested shapes are common, though they appear in cross-sections of rope (see page 874), as well as in address decoder trees on chips—and have recently been used in broadband antennas. (Some self-similarity is also present in standard log-periodic antennas.) When several distinct components are involved, more complicated structures are not uncommon—as in escapements, and many bearings and joints. More complex shapes for single elements sometimes arise when an analog of area maximization is desired—as with tire treads or fins in devices such as heat exchangers. Quadratic residue sequences *Mod[Range[n]^2, n]* (see page 1081) are used to give profiles for acoustic diffusers that operate uniformly over a range of frequencies. Musical instruments can have fairly complicated shapes maintained for historical reasons to considerable precision. Some knots can also be thought of as objects with complex forms. It is notable that elaborate types of mechanical motion (and sometimes surprising phenomena in general) are often first implemented in toys. Examples are early mechanical automata and model airplanes, and modern executive toys claiming to illustrate chaos theory through linkages, magnets or fluid systems. Complex trajectories (compare page 972) have sometimes been proposed or used for spacecraft. (See also notes on ornamental art on page 872.)

■ **Page 828 · Recognizing artifacts.** Various situations require picking out artifacts automatically. One example is finding buildings or machines from aerial reconnaissance images; another is finding boat or airplane wreckage on an ocean floor from sonar data. In both these cases the most common approach is to look for straight edges. Outdoor security systems also often need ways to distinguish animals and wind-induced motion from intentional human activity—and tend to have fairly simple procedures for doing this.

To recognize a regular crystal as not being a carefully cut artifact can take specific knowledge. The same can be true of patterns produced by wind on sand or rocks. Lenticular clouds are sometimes mistaken for UFOs on account of their regular shape. The exact cuboid form of the monolith in the movie *2001* was intended to suggest that it was an artifact.

Recognizing artifacts can be a central—and controversial—issue in prehistoric archeology. Sometimes human bones are found nearby. And sometimes chemical analysis suggests controlled fire—as with charcoal or baked clay. But to tell whether for example a piece of rock was formed naturally or was carefully made to be a stone tool can in general be very difficult. And a large part of the way this has been done in practice is just through comparison with known examples that fit into an overall pattern of gradual historical change. In recent decades there has been increasing emphasis on trying to understand and reproduce the whole process of making and using artifacts. And in the field of lithic analysis there are beginning to be fairly systematic ways to recognize for example the effects of the hundreds of orderly impacts needed to make a typical flint arrowhead by knapping. (Sometimes it is also possible to recognize microscopic features characteristic of particular kinds of use or wear—and it is conceivable that in the future analysis of trillions of atomic-scale features could reveal all sorts of details of the history of an object.)

To tell whether or not some arrangement of soil or rocks is an artifact can be extremely difficult—and there are many notorious cases of continuing controversy. Beyond looking for similarities to known examples, a typical approach is just to look for correlations with topographic or other features that might reveal some possible purpose.

▪ **Artifacts in data.** In fields like accounting and experimental science it is usually a sign of fraud if primary data is being created for a purpose, rather than merely being reported. If a large amount of numerical data has been made up by a person this can be detectable through statistical deviations from expected randomness—particularly in structural details such as frequency of digits. (So-called artifacts can also be the unintentional result of details of methods used to obtain or process data.)

In numerical computations effects are often called artifacts if they are believed not to be genuine features of an underlying mathematical system, but merely to reflect the computational scheme used. Such effects are usually first noticed through unexpected regularities in some detail of output. But in cases like chaos theory it remains unclear to what extent complex behavior seen in computations is an artifact (see page 920).

▪ **Animal artifacts.** Structures like mollusc shells, radiolarian skeletons and to some extent coral are formed through processes of growth like those discussed in Chapter 8. Structures like spider webs, wasp nests, termite mounds, bird nests and beaver dams rely on behavior determined by animal brains. (Even spider webs end up looking quite different if psychoactive drugs are administered to the spider.) And much like human artifacts, many of these structures tend to be distinguished by their comparative geometrical simplicity. In a few cases—particularly with insects—somewhat complicated forms are seen, but it seems likely that these are actually produced by rather simple local rules like those in aggregation systems (see page 1011).

▪ **Molecular biology.** DNA sequences of organisms can be thought of as artifacts created by biological evolution, and current data suggests that they contain some long-range correlations not present in typical random sequences. Most likely, however, these have fairly simple origins, perhaps being associated with iterative splicing of subsequences. And in the few thousand proteins currently known, standard statistical tests reveal no significant overall regularities in their sequences of up to a few thousand amino acids. (Some of the 20 standard amino acids do however occur more frequently than others.) Nevertheless, if one looks at overall shapes into which these proteins fold, there is some evidence that the same patterns of behavior are often seen. But probably such patterns would also occur in purely random proteins—at least if their folding happened in the same cellular apparatus. (See page 1003.) Note that the antibodies of the immune system are much like short random proteins—whose range of shapes must be sufficient to match any antigen. (See also page 1194.)

▪ **Messages in DNA.** Science fiction has sometimes suggested that an extraterrestrial source of life might have left some form of message in the DNA sequences of all terrestrial organisms, but to get evidence of this would seem to require extensive other knowledge of the source. (See also page 1190.)

▪ **Decompilers.** Trying to reverse engineer source code in a programming language like C from machine code output by compilers involves in effect trying to deduce higher-level purposes from lower-level computational steps. And normally this can be done with any reliability only when the machine code represents a fairly direct translation that has not been extensively rearranged or optimized.

▪ **Page 828 · Complexity and theology.** See page 861.

▪ **Page 829 · Purpose in archeology.** Ideas about the purpose of archeological objects most often ultimately tend to come from comparisons to similar-looking objects in use today. But great differences in typical beliefs and ways of life can make comparisons difficult. And certainly it is now very hard for us to imagine just what range of purposes the first known stone tools from 2.6 million years ago might have been put to—or what purpose the arrays of dots or handprints in cave paintings from 30,000 BC might have had. And even when it comes to early buildings from perhaps 10,000 BC it is still difficult to know just how they were used. Stone circles like Stonehenge from perhaps 3000 BC presumably served some community purpose, but beyond that little can convincingly be said. Definite geographical or astronomical alignments can be identified for many large prehistoric structures, but whether these were actually intentional is almost never clear. After the development of writing starting around 4000 BC, purposes can often be deduced from inscriptions and other written material. But still to work out for example the purpose of the Antikythera device from around 100 BC is very difficult, and depends on being able to trace a long historical tradition of astronomical clocks and orreries.

▪ **Dead languages.** Particularly over the past century or so, most of the known written human languages from every point in history have successfully been decoded. But to do this has essentially always required finding a case where there is explicit overlap with a known language—say a

Rosetta stone with the same text in multiple languages, or at least words or proper names that are transliterated. As in cryptanalysis, it is sometimes remarkable how small an amount of text is needed to find a decoding scheme. But usually what is done relies critically on the slowness with which human languages change, and the comparatively limited number of different basic ways in which they work.

■ **Teleology.** There is a common tendency to project human purposes onto natural objects and events—and this is for example almost universally done by young children. Ancient beliefs often held that things in nature are set up by a variety of gods for a variety of purposes. By 400 BC, following ideas of Anaxagoras, Socrates and Plato discussed the notion that things in nature might in effect be optimally designed for coherent purposes by a single mind. Around 350 BC Aristotle claimed that a full explanation of anything should include its purpose (or so-called final cause, or telos)—but said that for systems in nature this is often just to make the final forms of these systems (their so-called formal cause). The rise of monotheistic religions led to the widespread belief that the universe and everything in it was created for definite purposes by a single god. But the development of mathematical science in the 1600s—and its focus on mechanisms ("efficient causes")—led away from ascribing explicit purposes to physical systems. In the mid-1700s David Hume then claimed on philosophical grounds that we fundamentally have no basis for ascribing purposes to any kind of natural system—though in the 1790s Immanuel Kant argued that even though we cannot know whether there really are such purposes, it is still often necessary for us to think in terms of them. And in fact the notion that systems in biology are so complex that they must have been intelligently designed for a purpose remained common. In the late 1800s Darwinian evolution nevertheless suggested that no such purposeful design was necessary—though in a sense it again introduced a notion of purpose associated with optimization of fitness. Ever since the 1700s economics had been discussed in terms of purposeful activities of rational agents. In the early 1900s there were however general attempts to develop mechanistic explanations in the social sciences, but by the mid-1900s purpose was again widely discussed, especially in economics. And in fact, even in physics, a notion of purpose had actually always been quite common. For whenever a physical system satisfies any kind of implicit equation, this defines a constraint that can be viewed as corresponding to some kind of purpose. (See page 940.) That something like a notion of purpose is being used has been more widely recognized for variational principles like the Principle of Least Action in mechanics from the mid-1700s. Results in the late 1900s in astrophysics and cosmology seemed to suggest that for us to exist our universe must satisfy all sorts of constraints—and to avoid explaining this in terms of purpose the Anthropic Principle was introduced (see page 1026). What I do in this book goes significantly further than traditional science in getting rid of notions of purpose from investigations of nature. For I essentially always consider systems that are based on explicit evolution rules rather than implicit constraints. And in fact I argue that simple programs constructed without known purposes are what one needs to study to find the kinds of complex behavior we see.

■ **Possible purposes.** As part of asking whether the rules for a system are somehow minimal for a given purpose, one can ask what properties the system has that could reasonably be considered a purpose at all. In general one tends to talk of purpose only when doing so allows one to give a simpler description of some aspect of behavior than just describing the behavior directly. But whether one can give a simple description can depend greatly on the framework in which one is operating. And so, for example, while the digits of π have a simple description in terms of traditional mathematics, the results in Chapter 4 suggest that outside of this framework they normally do not. And what this means is that if one saw a system that had the property of generating the digits of π one would be unlikely to think that this could represent a meaningful purpose—unless one happened to be operating in traditional mathematics. And so similarly, one would be unlikely to think that generating the center column from rule 30 could represent any sort of meaningful purpose—unless one was operating within the framework that I have developed in this book.

■ **Page 830 · Purposeful computation.** See page 638.

■ **Page 832 · Doubling rules.** Rule (a) is

```
{{0, 2, _}→ 5, {5, 3, _}→ 5, {5, _, _}→ 1,
  {_, 5, _}→ 1, {_, 2, _}→ 3, {_, 3, 2}→ 2, {_, 1, 2}→ 4,
  {_, 4, _}→ 3, {4, 3, _}→ 4, {4, 0, _}→ 2, {_, x_, _}→ x}
```

and takes $2 n^2 + n$ steps to yield `Table[1, {2 n}]` given input `Append[Table[1, {n - 1}], 2]`. Rule (b) is

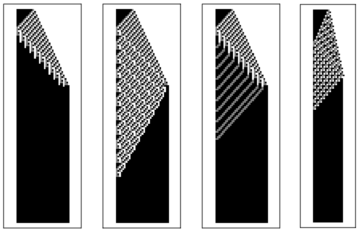
```
{{_, 2, _}→ 3, {_, 1, 2}→ 2, {3, 0, _}→ 1,
  {3, _, _}→ 3, {_, 3, _}→ 1, {_, x_, _}→ x}
```

and takes $3 n$ steps. Rule (c) is $k = 3$, $r = 1$ rule *5407067979* and takes $3 n - 1$ steps.

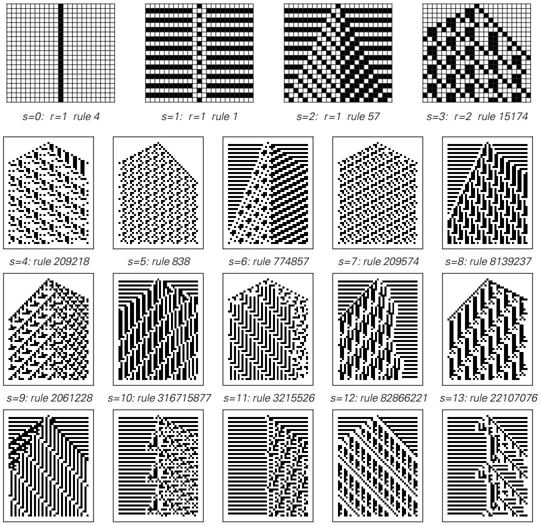
■ **Page 833 · Searching.** No symmetric $k = 3$, $r = 1$ rule yields doubling. General rules can show subtle bugs; rule *1340716537107* for example first fails at $n = 24$. The total number of $k = 3$, $r = 1$ rules that need to be searched can easily be reduced from 3^{27} to 3^{21}. Several different rules that work can behave identically, since up to 6 of the 27 cases in each rule are not sampled with the initial conditions used. In rules that work, between 8 and 19 cases lead to a change in the color of a cell, with 14 cases being the most common.

■ **Page 833 · Properties.** The number of steps increases irregularly but roughly quadratically with n in rule (a), and roughly linearly in (d) and (e). Rule (b) in the end repeats every 128 steps. The center of the complex pattern in both (d) and (e) emulates $k = 2$ rule 90.

■ **Other functions.** The first three pictures below show rules that yield $3n$ (no $k = 3$ rules yield $4n$, $5n$ or n^2), and the last picture $2n-2$ (corresponding to doubling with initial conditions analogous to page 639).

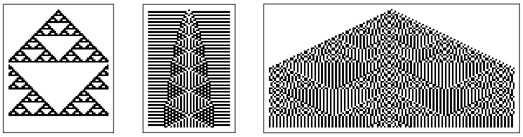


■ **Page 834 · Minimal cellular automata for sequences.** Given any particular sequence of black and white cells one can look for the simplest cellular automaton which generates that sequence as its center column when evolving from a single black cell (compare page 956). The pictures below show the lowest-numbered cellular automaton rules that manage to generate repetitive sequences containing black cells with successively greater separations s.

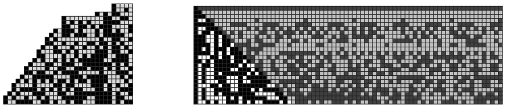


Elementary ($k = 2$, $r = 1$) cellular automata can be found only up to separations $s = 2$. But $k = 2$, $r = 2$ cellular automata can be found for all separations up to 15, as well as 17, 19 and 23. (Note that for example in the $s = 15$ case the lowest-numbered rule exhibits a complex 350-step transient away from the center column.)

The pictures below show the lowest-numbered cellular automata that generate respectively powers of two, squares and the nested Thue-Morse sequence of page 83 (compare rule 150). Of the 4 billion $k = 2$, $r = 2$ cellular automata none turn out to be able to produce for example sequences corresponding to the cubes, powers of 3, Fibonacci numbers, primes, digits of $\sqrt{2}$, or concatenation sequences.



If one looks not just at specific sequences, but instead at all 2^n possible sequences of length n, one can ask how many cellular automaton rules (say with $k = 2$, $r = 2$) one has to go through in order to generate every one of these. The pictures below show on the left the last rules needed to generate any sequence of each successive length—and on the right the form of the sequence (as well as its continuation after length n). Since some different rules generate the same sequences (see page 956) one needs to go through somewhat more than 2^n rules to get every sequence of length n. The sequences shown below can be thought of as being in a sense the ones of each length that are the most difficult to generate—or have the highest algorithmic information content. (Note that the sequence ■■■■□ is the first one that cannot be generated by any of the 256 elementary cellular automata; the first sequence that cannot be generated by any $k = 2$, $r = 2$ cellular automata is probably of length 26.)



■ **Other examples.** Minimal systems achieving particular purposes are shown on page 619 for Boolean functions evaluated with NANDs, pages 759 and 889 for Turing machines, page 1142 for sorting networks, and page 1035 for firing squad synchronization.

■ **Page 834 · Minimal theories.** Particularly in fundamental physics it has been found that the correct theory is often the minimal one consistent with basic observations. Yet barring supernatural intervention, the laws of physics embodied in such a theory presumably cannot be considered to have been created for any particular purpose. (See page 1025.)

▪ **Page 835 · Earth from space.** Human activity has led to a few large simple geometrical structures that are visually noticeable from space. One is the almost-straight 30-mile railroad causeway built in 1959 that divides halves of the Great Salt Lake in Utah where the water is colored blue and orange. Another is the almost-circular 12-mile-diameter national park created in 1900 that encloses ungrazed vegetation on the Egmont Volcano in New Zealand. On the scale of a few miles, there is also rectilinear arrangement of fields in the U.S. Midwest, as well as straight-line political boundaries with different agriculture on each side. Large geometrical patterns of logging were for example briefly visible after snow in 1961 near Cochrane, Canada—as captured by an early weather satellite. Perfectly straight sections of roads (such as the 90-mile Balladonia-Caiguna road in Australia), as well as the 4-mile-diameter perfectly circular Fermilab accelerator ring are not so easy to see. The Great Wall of China from 200 BC follows local topography and so is not straight.

Some of the most dramatic geometrical structures—such as the dendritic fossil drainage pattern in south Yemen or the bilaterally symmetric coral reefs around islands like Bora Bora—are not artifacts. The same is true of fields of parallel sand dunes, as well as of almost-circular structures such as the 40-mile-diameter impact crater in Manicouagan, Canada (highlighted by an annular lake) and the 30-mile-diameter Richat structure in the Sahara desert of Mauritania. On the Moon, the 50-mile-diameter crater Tycho is also almost circular—and has 1000-mile almost-straight rays coming out from it.

At night, lights of cities are obvious—notably hugging the coast of the Mediterranean—as are fire plumes from oil rigs. In addition, in some areas, sodium streetlamps make the light almost monochromatic. But it would seem difficult to be sure that these were artifacts without more information. In western Kansas there is however a 200-mile square region with light produced by a strikingly regular grid of towns—many at the centers of square counties laid out around 1870 in connection with land grants for railroad development. In addition, there is an isolated 1000-mile straight railroad built in the late 1800s across Kazakhstan between Aktyubinsk and Tashkent, with many towns visible at night along it. There are also 500-mile straight railroads built around the same time between Makat and Nukus, and Yaroslavl and Archangel. All these railroads go through flat empty terrain that previously had only a few nomadic inhabitants—and no settlements to define a route. But in many ways such geometrical forms seem vastly simpler to imagine producing than for example the elaborate pattern of successive lightning strikes visible especially in the tropics from space.

▪ **Page 835 · Astronomical objects.** Stars and planets tend to be close to perfect spheres. Lagrange points and resonances often lead to simple geometrical patterns of orbiting bodies. (The orbits of most planets in our solar system are also close to perfect circles; see page 973.) Regular spirograph-like patterns can occur for example in planetary nebulas formed by solar mass exploding stars. Unexplained phenomena that could conceivably be at least in part artifacts include gamma ray bursts and ultra high-energy cosmic rays. The local positions of stars are generally assumed to be random. 88 constellations are usually named—quite a few presumably already identified by the Babylonians and Sumerians around 2000 BC.

▪ **Page 835 · Natural radio emissions.** Each of the few million lightning flashes that occur on the Earth each day produce bursts of radio energy. At kilohertz frequencies reflection from the ionosphere allows these signals to propagate up to thousands of miles around the Earth, leading to continual intermittent crackling and popping. Particularly at night such signals can also travel within the ionosphere, but different frequencies travel at different rates, leading to so-called tweeks involving ringing or pinging. Signals can sometimes travel through the magnetosphere along magnetic field lines from one hemisphere to the other, yielding so-called whistlers with frequencies that fall off in a highly regular way with time. (Occasionally the signals can also travel back and forth between hemispheres, giving more complex results.) Radio emission can also occur when charged particles from the Sun excite plasma waves in the magnetosphere. And particularly at dawn or when an aurora is present an elaborate chorus of different elements can be produced—and heard directly on a VLF radio receiver.

Sunspots and solar flares make the Sun the most intense radio source in the solar system. Artificial radio signals from the Earth come next. The interaction of the solar wind with the magnetosphere of Jupiter produces radio emissions that exhibit variations reminiscent of gusting.

Outside the solar system, gas clouds show radio emission at discrete gigahertz frequencies from rotational transitions in molecules and spin-flip transitions in hydrogen atoms. (The narrowest lines come from natural masers and have widths around 1 kHz.) The cosmic microwave background, and processes such as thermal emission from dust, radiation from electrons in ionized gases, and synchrotron radiation from relativistic electrons in magnetic fields yield radio emissions with characteristic continuous frequency spectra. A total of over a million radio sources inside and outside our galaxy have now been catalogued, most with frequency spectra apparently consistent with known natural phenomena. Variations of source properties on timescales of months or years are not uncommon; variations of signals on timescales of tens of minutes can be introduced by propagation through turbulence in the interstellar medium.

Most radio emission from outside the solar system shows little apparent regularity. The almost perfectly repetitive signals from pulsars are an exception. Pulsars appear to be rapidly rotating neutron stars—perhaps 10 miles across—whose magnetic fields trap charged particles that produce radio emissions. When they first form after a supernova pulsars have millisecond repetition rates, but over the course of a few million years they slow to repetition rates of seconds through a series of glitches, associated perhaps with cracking in their solid crusts or perhaps with motion of quantized vortices in their superfluid interiors. Individual pulses from pulsars show some variability, presumably largely reflecting details of plasma dynamics in their magnetospheres.

■ **Page 835 · Artificial radio signals.** In current technology radio signals are essentially always based on carriers of the form $Sin[\omega\, t]$ with frequencies $\omega/(2\,\pi)$. When radio was first developed around 1900 information was normally encoded using amplitude modulation (AM) $s[t]\, Sin[\omega\, t]$. In the 1940s it also became popular to use frequency modulation (FM) $Sin[(1 + s[t])\,\omega\, t]$, and in the 1970s pulse code modulation (PCM) (pulse trains for $IntegerDigits[s[t], 2]$). All such methods yield signals that remain roughly in the range of frequencies $\{\omega - \delta,\ \omega + \delta\}$ where δ is the data rate in $s[t]$. But in the late 1990s—particularly for the new generation of cellular telephones—it began to be common to use spread spectrum CDMA methods, in which many signals with the same carrier frequency are combined. Each is roughly of the form $BitXor[u[t], s[t]]\, Sin[\omega\, t]$, where $u[t]$ is a pseudonoise (PN) sequence generated by a linear feedback shift register (LFSR) (see page 1084); the idea is that by using a different PN sequence for each signal the corresponding $s[t]$ can be recovered even if thousands are superimposed.

The radio spectrum from about 9 kHz to 300 GHz is divided by national and international legislation into about 460 bands designated for different purposes. And except when spread spectrum methods are used, most bands are then divided into between a few and a few thousand channels in which signals with identical structures but different frequencies are sent.

If one steps through frequencies with an AM radio scanner, one sometimes hears intelligible speech—from radio or TV broadcasts, or two-way radio communication. But in many frequency bands one hears instead either very regular or seemingly quite random signals. (A few bands allocated for example to distress signals or radio astronomy are normally quiet.) The regular signals come from such sources as navigation beacons, time standards, identification transponders and radars. Most have characteristic almost perfectly repetitive forms (radar pulses, for example, typically have the chirped form $Sin[(1 + \alpha\, t)\,\omega\, t]$)—and some sound uncannily like pulsars. When there are seemingly random signals some arise say from transmission of analog video (though this typically has very rigid overall structure associated with successive lines and frames), but most are now associated with digital data. And when CDMA methods are used there can be spreading over a significant range of frequencies—with regularities being recognizable only if one knows or can cryptanalyze LFSR sequences.

In general to send many signals together one just needs to associate each with a function $f[i, t]$ orthogonal to all other functions $f[j, t]$ (see page 1072). Current electronics (with analog elements such as phase-locked loops) make it easy to handle functions $Sin[\omega\, t]$, but other functions can yield better data density and perhaps better signal propagation. And as faster digital electronics makes it easier to implement these it seems likely that it will become less and less common to have simple carriers with definite frequencies.

In addition, there is a continuing trend towards greater spatial localization of signals—whether by using phased arrays or by explicitly using technologies like fiber optics.

At present, the most intense overall artificial radio emission from the Earth is probably the 50 or 60 Hz hum from power lines. The most intense directed signals are probably from radars (such as those used for ballistic missile detection) that operate at a few hundred megahertz and put megawatts of power into narrow beams. (Some such systems are however being replaced by lower-power phased array systems.)

■ **Page 835 · SETI.** First claims of extraterrestrial radio signals were made by Nikola Tesla in 1899. More widely believed claims were made by Guglielmo Marconi in 1922, and for several years searches were done—notably by the U.S. military—for signals presumed to be coming from Mars. But it became increasingly accepted that in fact nothing beyond natural radio emissions such as whistlers (see note above) were actually being detected.

When galactic radio emission was first noticed by Karl Jansky in 1931 it seemed too random to be of intelligent origin. And when radio astronomy began to develop it essentially ignored extraterrestrial intelligence. But in 1959

Giuseppe Cocconi and Philip Morrison analyzed the possibility of interstellar radio communication, and in 1960 Frank Drake used a radio telescope to look for explicit signals from two nearby stars.

In 1965 a claim was made that there might be intensity variations of intelligent origin in radio emission from the quasar CTA-102—but this was quickly retracted. Then in 1967 when the first pulsar was discovered it was briefly thought that perhaps its precise 1.33730113-second repetition rate might be of intelligent origin.

Since the 1960s around a hundred different SETI (search for extraterrestrial intelligence) experiments have been done. Most use the same basic scheme: to look for signals that show a narrow band of frequencies—say only 1 Hz wide—perhaps changing in time. (The corresponding waveform is thus required to be an almost perfect sinusoid.) Some concentrate on specific nearby stars, while others look at the whole sky, or test the stream of data from all observations at a particular radio telescope, sometimes scanning for repetitive trains of pulses rather than single frequencies. The best current experiments could successfully detect radio emission at the level now produced on Earth only from about 10 light years away—or from about the nearest 10 stars. The detection distance increases like the square root of the signal strength, covering all 10^{11} stars in our galaxy when the signal uses the total power output of a star.

Most SETI has been done with specially built systems or with existing radio telescopes. But starting in the mid-1990s it became possible to use standard satellite receivers, and there are now plans to set up a large array of these specifically for SETI. In addition, it is now possible to use software instead of hardware to implement SETI signal-processing algorithms—both traditional ones and presumably much more general ones that can for example pick out much weaker signals.

Many SETI experiments look for signals in the so-called "water hole" between the 1420 MHz frequency associated with the 21 cm line of hydrogen and the 1720 MHz frequency associated with hydroxyl (OH). But although there are now practical constraints associated with the fact that on Earth only a few frequency regions have been left clear for radio astronomy I consider this to be a remarkable example of reliance on details of human intellectual development.

Already in the early 1960s it was suggested that lasers instead of radio could be used for interstellar communication, and there have been various attempts to detect interstellar optical pulses. Other suggested methods of communication have included optical solitons, neutrinos and as-yet-unknown faster-than-light quantum effects.

It is sometimes suggested that there must be fundamental limits to detection of radio signals based on such issues as collection areas, noise temperatures and signal degradation. But even existing technology has provided a steady stream of examples where limits like these have been overcome—most often by the use of more sophisticated signal processing.

■ **Detection methods.** Ways to identify computational origins include looking for repeatability in apparently random signals and comparing with output from large collections of possible simple programs. At a practical level, the one-dimensional character of data from radio signals makes it difficult for us to apply our visual systems—which remain our most powerful general-purpose analysis tools.

■ **Higher perception and analysis.** See page 632.

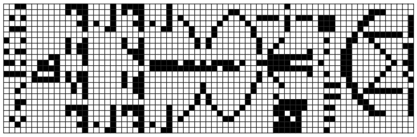
■ **Page 837 · Messages to send.** The idea of trying to send messages to extraterrestrials has existed since at least the early 1800s. The proposed content and medium of the messages has however steadily changed, usually reflecting what seemed to be the most significant human achievements of the time—yet often seeming quaint within just a few decades.

Starting in the 1820s various scientists (notably Carl Friedrich Gauss) suggested signalling the Moon by using such schemes as cutting clearings in a forest to illustrate the Pythagorean theorem or reflecting sunlight from mirrors in different countries placed so as to mimic an observed constellation of stars. In the 1860s, with the rise of telegraphy, schemes for sending flashes of light to Mars were discussed, and the idea developed that mathematics should somehow be the basic language used. In the 1890s radio signals were considered, and were tried by Nikola Tesla in 1899. Discussion in the 1920s led to the idea of sending radio pulses that could be assembled into a bitmap image, and some messages intended for extraterrestrials were probably sent by radio enthusiasts.

There is a long history of attempts to formulate universal languages (see page 1181). The Lincos language of Hans Freudenthal from 1960 was specifically designed for extraterrestrial communication. It was based on predicate logic, and attempted to use this to build up first mathematics, then science, then a general presentation of human affairs.

When the Pioneer 10 spacecraft was launched in 1972 it carried a physical plaque designed by Carl Sagan and others. The plaque is surprisingly full of implicit assumptions based on details of human intellectual development. For example, it has line drawings of humans—whose interpretation inevitably seems very specific to our visual system and artistic culture. It also has a polar plot of the positions of 14 pulsars relative to the Sun, with the pulsars specified by giving their periods as base 2 integers—but with trailing zeros inserted to cover inadequate precision. Perhaps the most peculiar element, however, is a diagram indicating the 21 cm transition in hydrogen—by showing two abstract quantum mechanical spin configurations represented in a way that seems completely specific to the particular details of human mathematics and physics. In 1977 the Voyager spacecraft carried phonograph records that included bitmap images and samples of spoken languages and music.

In 1974 the bitmap image below was sent as a radio signal from the Arecibo radio telescope. At the left-hand end is a version of the pattern of digits from page 117—but distorted so it has no obvious nested structure. There follow atomic numbers for various elements, and bitvectors for components of DNA. Next are idealized pictures of a DNA molecule, a human, and the telescope. All these parts seem to depend almost completely on detailed common conventions—and I suspect that without all sorts of human context their meaning would be essentially impossible to recognize.



In all, remarkably few messages have been sent—perhaps in part because of concerns that they might reveal us to extraterrestrial predators (see page 1191). There has also been a strong tendency to make messages hard even for humans to understand—perhaps on the belief that they must then be more scientific and more universal.

The main text argues that it will be essentially impossible to give definitive evidence of intelligence. Schemes that might however get at least some distance include sending:

- waveforms made of simple underlying elements;
- long complicated sequences that repeat precisely;
- a diversity of kinds of sequences;
- something complicated that satisfies simple constraints.

Examples of the latter include pattern-avoiding sequences (see page 944), magic squares and other combinatorial designs, specifications of large finite groups, and maximal length linear feedback shift register sequences (see page 1084). Notably, the last of these are already being transmitted by GPS satellites and CDMA communications systems. (If cases could be found where the sequences as a whole were forced not to have any obvious regularities, then pattern-avoiding sequences might perhaps be good since they have constraints that are locally fairly easy to recognize.)

Extrapolation of trends in human technology suggest that it will become ever easier to detect weak signals that might be assumed distorted beyond recognition or swamped by noise.

▪ **Page 838 · P versus NP.** Given a constraint, it may be an NP-complete problem to find out what object satisfies it. So it may be difficult to generate the object from the constraint. But if one allows oneself to generate the object in any way at all, this may still be easy, even if $P \neq NP$.

▪ **Science fiction.** Inhabitants of the Moon were described in stories by Lucian around 150 AD and Johannes Kepler in 1634—and in both cases were closely modelled on terrestrial organisms. Interest in fiction about extraterrestrials increased greatly at the end of the 1800s—perhaps because by then few parts of the Earth remained unexplored. And as science fiction developed, accounts of the future sometimes treated extraterrestrials as commonplace—and sometimes did not mention them at all. Most often extraterrestrials have been easy to recognize, being little more than simple combinations of terrestrial animals (and occasionally plants)—though fairly often with extra features like telepathy. Some stories have nevertheless explored extraterrestrial intelligence based for example on solids, gases or energy fields. An example is Fred Hoyle's 1957 *The Black Cloud* in which a large cloud of hydrogen gas achieves intelligence by exchanging electromagnetic signals between rocks whose surface molecular configurations store memories.

The most common fictional scenario for first contact with extraterrestrials is the arrival of spacecraft—often induced by us having passed a technology threshold such as radio, nuclear explosions or faster-than-light travel. Other scenarios sometimes considered include archeological discovery of extraterrestrial artifacts and receipt of radio signals.

In the movie *2001* a black cuboid with side ratios 1:4:9 detected on the Moon through its anomalous magnetic properties sends a radio pulse in response to sunlight. Later there are also a few frames of flashing octahedra, presumably intended to be extraterrestrial artifacts, or perhaps extraterrestrials themselves.

In *The Black Cloud* intelligence is suggested by responsiveness to radio stimuli. Communication is established—as often in science fiction—by the intelligence interpreting material that we supply, and then replying in the same format.

The movie *Contact* centers on a radio signal with several traditional SETI ideas: it is transmitted at $1420\,\pi$ MHz, and involves a sequence of primes to draw attention, an amplified TV signal from Earth and a description of a machine to build.

The various *Star Trek* television series depict many encounters with "new life and new civilizations". Sometimes intelligence is seen not associated with something that is considered a lifeform.

Particularly in short stories various scenarios have been explored where it is difficult ever to recognize intelligence. These include one-of-a-kind beings that have nothing to communicate with, as well as beings with inner intellectual activity but no effect on the outside world. When there are extraterrestrials substantially more advanced than humans few efforts have been made to describe their motives and purposes directly—and usually what is emphasized is just their effects on humans.

(See also page 1184.)

▪ **Page 839 · Practical arguments.** If extraterrestrials exist at all an obvious question—notably asked by Enrico Fermi in the 1940s—is why we have not encountered them. For there seems no fundamental reason that even spacecraft could not colonize our entire galaxy within just a few million years.

Explanations suggested for apparent absence include:

- Extraterrestrials are visiting, but we do not detect them;
- Extraterrestrials have visited, but not in recorded history;
- Extraterrestrials choose to exist in other dimensions;
- Interstellar travel is somehow infeasible;
- Colonization is somehow ecologically limited;
- Physical travel is not worth it; only signals are ever sent.

Explanations for apparent lack of radio signals include:

- Broadcasting is avoided for fear of conquest;
- There are active efforts to prevent us being contaminated;
- Extraterrestrials have no interest in communicating;
- Radio is the wrong medium;
- There are signals, but we do not understand them.

The so-called Drake equation gives a straightforward upper bound on the number of cases of extraterrestrial intelligence that could have arisen in our galaxy through the same basic chain of circumstances as humans. The result is a product of: rate of formation of suitable stars; fraction with planetary systems; number of Earth-like planets per system; fraction where life develops; fraction where intelligence develops; fraction where technology develops; time communicating civilizations survive. It now seems fairly certain that there are at least hundreds of millions of Earth-like planets in our galaxy. Biologists sometimes argue that intelligence is a rare development—though in the Darwinian approach it certainly has clear benefit. In addition, particularly in the Cold War period, it was often said that technological civilizations would quickly tend to destroy themselves, but now it seems more likely that intelligence—once developed—will tend to survive indefinitely, at least in machine form.

It is obviously difficult to guess the possible motivations of extraterrestrials, but one might expect that—just as with humans—different extraterrestrials would tend to do different things, so that at least some would choose to send out signals if not spacecraft. Out of about 6 billion humans, however, it is notable that only extremely few choose, say, to explore life in the depths of the oceans—though perhaps this is just because technology has not yet made it easy to do. In human history a key motivator for exploration has been trade. But trade requires that there be things of value to exchange; yet it is not clear that with sufficiently advanced technology there would be. For if the fundamental theory of physics is known, then everything about what is possible in our universe can in principle be worked out purely by a computation. Often irreducible work will be required, which one might imagine it would be worthwhile to trade. But as a practical matter, it seems likely that there will be vastly more room to do more extensive computations by using smaller components than by trading and collaborating with even millions of other civilizations. (It is notable that just a couple of decades ago, it was usually assumed that extraterrestrials would inevitably want to use large amounts of energy, and so would eventually for example tap all the output of a star. But seeing the increasing emphasis on information rather than mechanical work in human affairs this now seems much less clear.)

Extrapolating from our development, one might expect that most extraterrestrials would be something like immortal disembodied minds. And what such entities might do has to some extent been considered in the context of the notion of heaven in theology and art. And it is perhaps notable that while such activities as music and thought are often discussed, exploration essentially never is.

▪ **Physics as intelligence.** From the point of view of traditional thinking about intelligence in the universe it might seem like an extremely bizarre possibility that perhaps intelligence could exist at a very small scale, and in effect have spread throughout the universe, building as an artifact everything we see. But at least with a broad interpretation of intelligence this is at some level exactly what the Principle of Computational Equivalence suggests has actually happened. For it implies that even at the smallest scales the laws of physics will show the same computational sophistication that we normally associate with intelligence. So in some sense this supports the theological notion that there might be a kind of intelligence that permeates our universe. (See page 1195.)

Implications for Technology

■ **Covering technology.** In writing this book I have tried to achieve some level of completeness in covering the obvious scientific implications of my ideas. But to cover technological implications at anything like the same level would require at least as long a book again. And in my experience many of the intellectually most interesting aspects of technology emerge only when one actually tries to build technology for real—and they are often in a sense best captured by the technology itself rather than by a book about it.

■ **Page 840 · Applications of randomness.** Random drawing of lots has been used throughout recorded history as an unbiased way to distribute risks or rewards. Also common have been games of chance (see page 968). Randomness is in general a convenient way to allow local decisions to be made while maintaining overall averages. In biological organisms it is used in determining sex of offspring, as well as in achieving uniform sampling, say in foraging for food. (Especially in antiquity, all sorts of seemingly random phenomena have been used as a basis for fortune telling.)

The notion of taking random samples as a basis for making unbiased deductions has been common since the early 1900s, notably in polling and market research. And in the past few decades explicit randomization has become common as a way of avoiding bias in cases such as clinical trials of drugs.

In the late 1800s it was noted in recreational mathematics that one could find the value of π by looking at randomly dropped needles. In the early 1900s devices based on randomness were built to illustrate statistics and probability (see page 312), and were used for example to find the form of the Student *t*-distribution. With the advent of digital computers in the mid-1940s Monte Carlo methods (see page 968) were introduced, initially as a way to approximate processes like neutron diffusion. (Similar ideas had been used in 1901 by Kelvin to study the Boltzmann equation.) Such methods became increasingly popular, especially for simulating systems like telephone networks and particle detectors that have many heterogeneous elements—as well as in statistical physics. In the 1970s they also became widely used for high-dimensional numerical integration, notably for Feynman diagram evaluation in quantum electrodynamics. But eventually it was realized that quasi-Monte Carlo methods based on simple sequences could normally do better than ones based on pure randomness (see page 1085).

A convenient way to tell whether expressions are equal is to evaluate them with random numerical values for variables. (Care must be taken with branch cuts and bounding intervals for inexact numbers.) In the late 1970s it was noted that by evaluating *PowerMod[a, n - 1, n] == 1* for several random integers *a* one can with high probability quickly deduce *PrimeQ[n]*. (In the 1960s it had been noted that one can factor polynomials by filling in random integers for variables and factoring the resulting numbers.) And in the 1980s many such randomized algorithms were invented, but by the mid-1990s it was realized that most did not require any kind of true randomness, and could readily be derandomized and made more predictable. (See page 1085.)

There are all sorts of situations where in the absence of anything better it is good to use randomness. Thus, for example, many exploratory searches in this book were done randomly. And in testing large hardware and software systems random inputs are often used.

Randomness is a common way of avoiding pathological cases and deadlocks. (It requires no communication between components so is convenient in parallel systems.) Examples include message routing in networks, retransmission times after ethernet collisions, partitionings for sorting algorithms, and avoiding getting stuck in minimization procedures like simulated annealing. (See page 347.) As on page 333, it is common for randomness to add robustness—as for example in cellular automaton fluids, or in saccadic eye movements in biology.

In cryptography randomness is used to make messages look typical of all possibilities (see page 598). It is also used in roughly the same way in hashing (see page 622). Such randomness must be repeatable. But for cryptographic keys it should not be. And the same is true when one picks unique IDs, say to keep track of repeat web transactions with a low probability of collisions. Randomness is in effect also used in a similar way in the shotgun method for DNA sequencing, as well as in creating radar pulses that are difficult to forge. (In biological organisms random diversity in faces and voices may perhaps have developed for similar reasons.)

The unpredictability of randomness is often useful, say for animals or military vehicles avoiding predators (see page 1105). Such unpredictability can also be used in simulating human or natural processes, say for computer graphics, videogames, or mock handwriting. Random patterns are often used as a way to hide regularities—as in camouflage, security envelopes, and many forms of texturing and distressing. (See page 1077.)

In the past, randomness was usually viewed as a thing to be avoided. But with the spread of computers and consumer electronics that normally operate predictably, it has become increasingly popular as an option.

Microscopic randomness is implicitly used whenever there is dissipation or friction in a system, and generally it adds robustness to the behavior that occurs in systems.

▪ **Page 841 · Self-assembly.** Given elements (such as pieces of molecules) that fit together only when certain specified constraints are satisfied it is fairly straightforward to force, say, cellular automaton patterns to be generated, as on page 221. (Notable examples of such self-assembly occur for instance in spherical viruses.)

▪ **Page 841 · Nanotechnology.** Popular since the late 1980s, especially through the work of Eric Drexler, nanotechnology has mostly involved investigation of several approaches to making essentially mechanical devices out of small numbers of atoms. One approach extrapolates chip technology, and studies placing atoms individually on solid surfaces using for example scanning probe microscopy. Another extrapolates chemical synthesis—particularly of fullerenes—and considers large molecules made for example out of carbon atoms. And another involves for example setting up fragments of DNA to try to force particular patterns of self-assembly. Most likely it will eventually be possible to have a single universal system that can manufacture almost any rigid atomic-scale structure on the basis of some kind of program. (Ribosomes in biological cells already construct arbitrary proteins from DNA sequences, but ordinary protein shapes are usually difficult to predict.) Existing work has tended to concentrate on trying to make rather elaborate components suitable for building miniature versions of familiar machines. The discoveries in this book imply however that there are much simpler components that can also be used to set up systems that have behavior with essentially any degree of sophistication. Such systems can either have the kind of chemical and mechanical character most often considered in nanotechnology, or can be primarily electronic, for example along the lines of so-called quantum-dot cellular automata. Over the next several decades applications of nanotechnology will no doubt include much higher-capacity computers, active materials of various kinds, and cellular-scale biomedical devices.

▪ **Page 842 · Searching for technology.** Many inventions are made by pure ingenuity (sometimes aided by mathematical calculation) or by mimicking processes that go on in nature. But there are also cases where systematic searches are done. Notable examples were the testing of thousands of materials as candidate electric light bulb filaments by Thomas Edison in 1879, and the testing of 606 substances for chemotherapy by Paul Ehrlich in 1910. For at least fifty years it has now been quite routine to test many hundreds or thousands of substances in looking, say, for catalysts or drugs with particular functions. (Other kinds of systematic searches done include ones for metal alloys, cooking recipes and plant hybrids.) Starting in the late 1980s the methods of combinatorial chemistry (see note below) began to make it possible to do biochemical tests on arrays of millions of related substances. And by the late 1990s, similar ideas were being used for example in materials science: in a typical case an array of different combinations of substances is made by successively spraying through an appropriate sequence of masks, with some physical or chemical test then applied to all the samples.

In the late 1950s maximal length shift register sequences (page 1084) and some error-correcting codes (page 1101) were found by systematic searches of possible polynomials. Most subsequent codes, however, have been found by explicit mathematical constructions. Optimal circuit blocks for operations such as addition and sorting (see page 1142) have occasionally been found by searches, but are more often found by explicit construction, progressive improvement or systematic logic minimization (see page 1097). In some compilers searches are occasionally done for optimal sequences of instructions to implement particular simple functions. And in recent years—notably in the building of *Mathematica*—optimal algorithms for operations such as function evaluation and numerical integration have sometimes been found through searches. In addition, my 1984 identification of rule 30 as a randomness generator was the result of a small-scale systematic search.

Particularly since the 1970s, many systematic methods have been tried for optimizing engineering designs by computer. Usually they are based on iterative improvement rather than systematic search. Some rely on linear programming or gradient descent. Others use methods such as simulated annealing, neural networks and genetic algorithms. But as discussed on page 342, except in very simple cases, the results are usually far from any absolute optimum. (Plant and animal breeding can be viewed as a simple form of randomized search done since the dawn of civilization.)

▪ **Page 843 · Methodology in this book.** Much of what is presented in this book comes from systematic enumeration of all possible systems of particular types. However, sometimes I have done large searches for systems (see e.g. page 112). And especially in Chapter 11 I have occasionally explicitly constructed systems that show particular features.

▪ **Chemistry.** Chemical compounds are a little like cellular automata and other kinds of programs. For even though the basic physical laws relevant to chemical compounds have been known since the early 1900s, it remains extremely difficult to predict the actual properties of a given compound. And I suspect that the ultimate reason for this—just as in the case of simple programs—is computational irreducibility.

For a single molecule, the minimum energy configuration can presumably always be found by a limited amount of computational work—though potentially increasing rapidly with the number of atoms. But if one allows progressively more molecules computational irreducibility can make it take progressively more computational work to see what will happen. And much as in determining whether constraints like those on page 213 can be satisfied for an infinite region, it can take an infinite amount of computational work to determine bulk properties of an infinite collection of molecules. Thus in practice it has typically been difficult to predict for example boiling and particularly melting points (see note below). So this means in the end that most of chemistry must be based on facts determined experimentally about specific compounds that happen to have been studied.

There are currently about 10 million compounds listed in standard chemical databases. Of these, most were first identified as extracts from biological or other natural systems. In trying to discover compounds that might be useful say as drugs the traditional approach was to search large libraries of compounds, then to study variations on those that seemed promising. But in the 1980s it began to be popular to try so-called rational design in which molecules were created that could at least to some extent specifically be computed to have relevant shapes and chemical functions. Then in the 1990s so-called combinatorial chemistry became popular, in which—somewhat in imitation of the immune system—large numbers of possible compounds were created by successively adding at random several different possible amino acids or other units. But although it will presumably change in the future it remained true in 2001 that half of all drugs in use are derived from just 32 families of compounds.

Doing a synthesis of a chemical is much like constructing a network by applying a specified sequence of transformations. And just like for multiway systems it is presumably in principle undecidable whether a given set of possible transformations can ever be combined to yield a particular chemical. Yet ever since the 1960s there have been computer systems like LHASA that try to find synthesis pathways automatically. But perhaps because they lack even the analog of modern automated theorem-proving methods, such systems have never in practice been extremely successful.

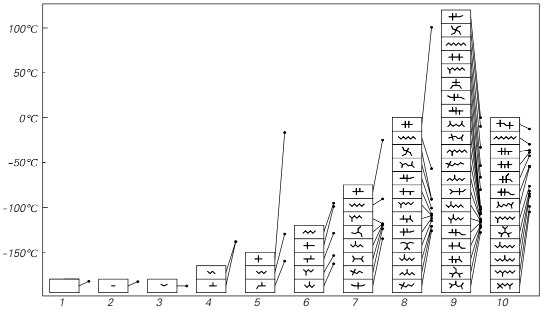
■ **Interesting chemicals.** The standard IUPAC system for chemical nomenclature assigns a name to essentially any possible compound. But even among hydrocarbons with fairly few atoms not all have ever been considered interesting enough to list in standard chemical databases. Thus for example the following compares the total number of conceivable alkanes (paraffins) to the number actually listed in the 2001 standard Beilstein database:

n	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
total	*1*	*1*	*1*	*2*	*3*	*5*	*9*	*18*	*35*	*75*	*159*	*355*	*802*	*1858*	*4347*	*10359*
listed	*1*	*1*	*1*	*2*	*3*	*5*	*9*	*18*	*35*	*75*	*68*	*108*	*60*	*60*	*41*	*62*

Any tree with up to 4 connections at each node can in principle correspond to an alkane with chemical formula C_nH_{2n+2}. The total number of such trees—studied since 1875—increases roughly like $2.79^n n^{-5/2}$. If every node has say 4 connections, then eventually one gets dendrimers that cannot realistically be constructed in 3D. But long before this happens one runs into many alkanes that presumably exist, but apparently have never explicitly been studied. The small unbranched ones (methane, ethane, propane, butane, pentane, etc.) are all well known, but ones with more complicated branching are decreasingly known. In coal and petroleum a continuous range of alkanes occur. Branched octanes are used to reduce knocking in car engines. Biological systems contain many specific alkanes—often quite large—that happen to be produced through chemical pathways in biological cells. (The $n = 11$ and $n = 13$ unbranched alkanes are for example known to serve as ant pheromones.)

In general the main way large molecules have traditionally ended up being considered chemically interesting is if they occur in biological systems—or mimic ones that do. Since the 1980s, however, molecules such as the fullerenes that instead have specific regular geometrical shapes have also begun to be considered interesting.

■ **Alkane properties.** The picture on the facing page shows melting points measured for alkanes. (Note that even when alkanes are listed in chemical databases—as discussed above—their melting points may not be given.) Unbranched alkanes yield melting points that increase smoothly for n even and for n odd. Highly symmetrical branched alkanes tend to have high melting points, presumably because they pack well in space. No reliable general method for predicting melting points is however known (see note above), and in fact for large n alkanes tend to form jellies with no clear notion of melting.



Things appear somewhat simpler with boiling points, and as noticed by Harry Wiener in 1947 (and increasingly discussed since the 1970s) these tend to be well fit as being linearly proportional to the so-called topological index given by the sum of the smallest numbers of connections visited in getting between all pairs of carbon atoms in an alkane molecule.

■ **Page 843 · Components for technology.** The Principle of Computational Equivalence suggests that a vast range of systems in nature can all ultimately be used to make computers. But it is remarkable to what extent even the components of present-day computer systems involve elements of nature originally studied for quite different reasons. Examples include electricity, semiconductors (used for chips), ferrites (used for magnetic storage), liquid crystals (used for displays), piezoelectricity (used for microphones), total internal reflection (used for optical fibers), stimulated emission (used for lasers) and photoconductivity (used for xerographic printing).

■ **Future technology.** The purposes technology should serve inevitably change as human civilization develops. But at least in the immediate future many of these purposes will tend to relate to the current character of our bodies and minds. For certainly technology must interface with these. But presumably as time progresses it will tend to become more integrated, with systems that we have created eventually being able to fit quite interchangeably into our usual biological or mental setup. At first most such systems will probably tend either to be based on standard engineering, or to be quite direct emulations of human components that we see. But particularly by using the ideas and methods of this book I suspect that significant progressive enhancements will be possible. And probably there will be many features that are actually quite easy to take far beyond the originals. One example is memory and the recall of history. Human memory is in many ways quite impressive. Yet for ordinary physical objects we are used to the idea that they remember little of their history, for at a macroscopic level we tend to see only the coarsest traces. But at a microscopic scale something like the surface of a solid has in at least some form remarkably detailed information about its history. And as technological systems get smaller it should become possible to read and manipulate this. And much as in the discussion at the end of Chapter 10 the ability to interact at such a level will yield quite different experiences, which in turn will tend to suggest different purposes to pursue with technology.

Historical Perspectives

■ **Page 844 · Human uniqueness.** The idea that there is something unique and special about humans has deep roots in Judeo-Christian tradition—and despite some dilution from science remains a standard tenet of Western thought today. Eastern religions have however normally tended to take a different view, and to consider humans as just one of many elements that make up the universe as a whole. (See note below.)

■ **Page 845 · Animism.** Belief in animism remains strong in perhaps several hundred million indigenous people around the world. In its typical form, it involves not only explaining natural phenomena by analogy to human behavior but also assuming that they can be influenced as humans might be, say by offerings or worship. (See also page 1177.)

Particularly since Edward Tylor in 1871 animism has often been thought of as the earliest identifiable form of religion. Polytheism is then assumed to arise when the idea of localized spirits associated with individual natural objects is generalized to the idea of gods associated with types of objects or concepts (as for example in many Roman beliefs). Following their rejection in favor of monotheism by Judaism—and later Christianity and Islam—such ideas have however tended to be considered primitive and pagan. In Europe through the Middle Ages there nevertheless remained widespread belief in animistic kinds of explanations. And even today some Western superstitions center on animism, as do rituals in countries like Japan. Animism is also a key element of the New Age movement of the 1960s, as well as of such ideas as the Gaia Hypothesis.

Particularly since the work of Jean Piaget in the 1940s, young children are often said to go through a phase of animism, in which they interact with complex objects much as if they were alive and human.

■ **Page 845 · Universe as intelligent.** Whether or not something like thinking can be attributed to the universe has long been discussed in philosophy and theology. Theism and the standard Western religions generally attribute thinking to a person-like God who governs the universe but is separate from it. Deism emphasizes that God can govern the universe only according to natural laws—but whether or not this involves thinking is unclear. Pantheism generally identifies the universe and God. In its typical religious form in Eastern metaphysics—as well as in philosophical idealism—the contents of the universe are identified quite directly with the thoughts of God. In scientific pantheism the abstract order of the universe is identified with God (often termed "Nature's God" or "Spinoza's God"), but whether this means that thinking is involved in the operation of the universe is not clear. (See also pages 822 and 1191.)

▪ **Non-Western thinking.** Some of my conclusions in this book may seem to resonate with ideas of Eastern thinking. For example, what I say about the fundamental similarity of human thinking to other processes in nature may seem to fit with Buddhism. And what I say about the irreducibility of processes in nature to short formal rules may seem to fit with Taoism. Like essentially all forms of science, however, what I do in this book is done in a rational tradition—with limited relation to the more mystical traditions of Eastern thinking.

▪ **Aphorisms.** Particularly from ancient and more fragmentary texts aphorisms have survived that may sometimes seem at least vaguely related to this book. (An example from the pre-Socratics is "everything is full of gods".) But typically it is impossible to see with any definiteness what such aphorisms might really have been intended to mean.

▪ **Postmodernism.** Since the mid-1960s postmodernism has argued that science must have fundamental limitations, based on its general belief that any single abstract system must somehow be as limited—and as arbitrary in its conclusions—as the context in which it is set up. My work supports the notion that—despite implicit assumptions made especially in the physical sciences—context can in fact be crucial to the choice of subject matter and interpretation of results in science (see e.g. page 1105). But the Principle of Computational Equivalence suggests at some level a remarkable uniformity among systems, that allows all sorts of general scientific statements to be made without dependence on context. It so happens that some of these statements then imply intrinsic general limitations on science—but even the very fact that such statements can be made is in a sense an example of successful generality in science that goes against the conclusions of postmodernism. (See also page 1131.)

▪ **Microcosm.** The notion that a human mind might somehow be analogous to the whole universe was discussed by Plato and others in antiquity, and known in the Middle Ages. But it was normally assumed that this was something fairly unique to the human mind—and nothing with the generality of the Principle of Computational Equivalence was ever imagined.

▪ **Human future.** The Principle of Computational Equivalence and the results of this book at first suggest a rather bleak view of the end point of the development of technology. As I argued in Chapter 10 computers will presumably be able to emulate human thinking. And particularly using the methods of this book one will be able to use progressively smaller physical components as elements of computers. So before too long it will no doubt be possible to implement all the processes of thinking that go on in a single human—or even in billions of humans—in a fairly small piece of material. Each piece of human thinking will then correspond to some microscopic pattern of changes in the atoms of the material. In the past one might have assumed that these changes would somehow show fundamental evidence of representing sophisticated human thinking. But the Principle of Computational Equivalence implies that many ordinary physical processes are computationally just as sophisticated as human thinking. And this means that the pattern of microscopic changes produced by such processes can at some level be just as sophisticated as those corresponding to human thinking. So given, say, an ordinary piece of rock in which there is all sorts of complicated electron motion this may in a fundamental sense be doing no less than some system of the future constructed with nanotechnology to implement operations of human thinking. And while at first this might seem to suggest that the rich history of biology, civilization and technology needed to reach this point would somehow be wasted, what I believe instead is that this just highlights the extent to which such history is what is ultimately the defining feature of the human condition.

▪ **Philosophical implications.** The Principle of Computational Equivalence has implications for many issues long discussed in the field of philosophy. Most important are probably those in epistemology (theory of knowledge). In the past, it has usually been assumed that if we could only build up in our minds an adequate model of the world, then we would immediately know whatever we want about the world. But the Principle of Computational Equivalence now implies that even given a model it may be irreducibly difficult to work out its consequences. In effect, computational irreducibility introduces a new kind of limit to knowledge. And it implies that one needs a criterion more sophisticated than immediate predictability to assess a scientific theory—since when computational irreducibility is present this will inevitably be limited. In the past, it has sometimes been assumed that truths that can be deduced purely by operations like those in logic must somehow always be trivial. But computational irreducibility implies that in general they are not. Indeed it implies that even once the basic laws are known there are still an endless series of questions that are worth investigating in science. It is often assumed that one cannot learn much about the world just by studying purely formal systems—and that one has to rely on empirical input. But the Principle of Computational Equivalence implies that at some level there are inevitably common features across both abstract and natural systems. In ontology (theory of being) the Principle of Computational Equivalence implies that special components are vastly less necessary than might have been thought. For it shows that all sorts of sophisticated characteristics can emerge from the very same kinds of simple components. (My discussion of fundamental physics in Chapter 9 also suggests that no separate entities beyond simple rules are needed to capture space, time or matter.) Arguments in several areas of philosophy involve in effect considering fundamentally different intelligences. But the Principle of Computational Equivalence implies that in fact above a certain threshold there is an ultimate equivalence between possible intelligences. In addition, the Principle of Computational Equivalence implies that all sorts of systems in nature and elsewhere will inevitably exhibit features that in the past have been considered unique to intelligence—and this has consequences for the mind-body problem, the question of free will, and recognition of other minds. It has often been thought that traditional logic—and to some extent mathematics—are somehow fundamentally special and provide in a sense unique foundations. But the Principle of Computational Equivalence implies that in fact there are a huge range of other formal systems, equivalent in their ultimate richness, but different in their details, and in the questions to which they naturally lead. In philosophy of science the Principle of Computational Equivalence forces a new methodology based on formal experiments—that is ultimately the foundation for the whole new kind of science that I describe in this book.