Thinking about Gödel and Turing

Essays on Complexity, 1970 – 2007

Gregory J Chaitin

With a Foreword by
Paul Davies

World Scientific

Thinking about Gödel and Turing

Essays on Complexity, 1970 - 2007

Gregory J Chaitin

IBM T J Watson Research Center, USA

With a Foreword by Paul Davis

World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

Author's photograph courtesy of Jacqueline Meyer.
For additional copyright information, see the Acknowledgements at the end of the book.

THINKING ABOUT GÖDEL AND TURING
Essays on Complexity, 1970–2007

ISBN-13 978-981-270-895-3
ISBN-10 981-270-895-2
ISBN-13 978-981-270-896-0 (pbk)
ISBN-10 981-270-896-0 (pbk)

Printed in Singapore.

Contents

This page intentionally left blank

Introductory note

How should this book be read? Well, the articles in it are independent, self-contained pieces, and I prefer to let readers wander through, having their own thoughts, exploring on their own, rather than offer a guided tour. In other words, I will let the individual essays stand on their own, unintroduced. And there is no need to read this book from cover to cover. Just read whatever strikes your fancy, enjoy whatever catches your eye.

However, if you do read this book from cover to cover in chronological order, you will see that the papers in it all deal with the same problem, they attempt to answer the same question: "What is the meaning of Gödel's incompleteness theorem?" Of course, my point of view changes and develops over time. Themes enter and disappear, but there is a central spine that never varies, a single thread that ties it all together. It's one train of thought, on different aspects of the same topic.

For those of you who would like a historical perspective, I have in fact put together a timeline explaining the evolution of my ideas. It's called "Algorithmic information theory: Some recollections." This, however, is a technical paper, not a popular account intended for the general reader. This timeline can be found in the *festschrift* volume assembled by Cristian Calude, *Randomness and Complexity, from Leibniz to Chaitin* (World Scientific, 2007).

The original sources of the papers in this collection are given in the table of contents, but more detailed information, including copyrights, appears in the **Acknowledgements** at the end of the book. And for those of you who would like to know where to go for more information on particular topics, I have included a **List of publications** with most of my technical and non-technical papers and books and some interviews.

This page intentionally left blank

On the difficulty of computations

Two practical considerations concerning the use of computing machinery are the amount of information that must be given to the machine for it to perform a given task and the time it takes the machine to perform it. The size of programs and their running time are studied for mathematical models of computing machines. The study of the amount of information (i.e., number of bits) in a computer program needed for it to put out a given finite binary sequence leads to a definition of a random sequence; the random sequences of a given length are those that require the longest programs. The study of the running time of programs for computing infinite sets of natural numbers leads to an arithmetic of computers, which is a distributive lattice. [This paper was presented at the Pan-American Symposium of Applied Mathematics, Buenos Aires, Argentina, August 1968.]

Section I

The modern computing machine sprang into existence at the end of World War II. But already in 1936 Turing and Post had proposed a mathematical model of computing machines (figure 1).[1] The mathematical model of the computing machine that Turing and Post proposed, commonly referred to as the Turing machine, is a black box with a finite number of internal states. The box can read and write on an infinite paper tape, which is divided into squares. A digit or letter may be written on each square of the tape, or the square may be blank. Each second the machine performs one of the following

[1]Their papers appear in Davis [1]. As general references on computability theory we may also cite Davis [2]–[4], Minsky [5], Rogers [6], and Arbib [7].

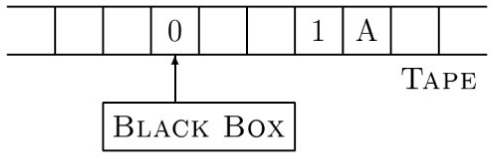

Figure 1. A Turing-Post machine

actions. It may stop, it may shift the tape one square to the right or one square to the left, it may erase the square on which the read-write head is positioned, or it may write a digit or letter on the square on which the read-write head is positioned. The action it performs is determined solely by the internal state of the black box at the moment, and the current state of the black box is determined solely by its previous internal state and the character read on the square of the tape on which its read-write head was positioned.

Incredible as it may seem at first, a machine of such primitive design can multiply numbers written on its tape, and can write on its tape the successive digits of π. Indeed, it is now generally accepted that any calculation that a modern electronic digital computer or a human computer can do, can also be done by such a machine.

Section II

How much information must be provided to a computer in order for it to perform a given task? The point of view we will present here is somewhat different from the usual one. In a typical scientific application, the computer may be used to analyze statistically huge amounts of data and produce a brief report in which a great many observations are reduced to a handful of statistical parameters. We would view this in the following manner. The same final result could have been achieved if we had provided the computer with a table of the results, together with instructions for printing them in a neat report. This observation is, of course, ridiculous for all practical purposes. For, had we known the results, it would not have been necessary to use a computer. This example, then, does not exemplify those aspects of

computation that we will emphasize.

Rather, we are thinking of such scientific applications as solving the Schrödinger wave equation for the helium atom. Here we have no data, only a program; and the program will produce after much calculation a great deal of printout. Or consider calculating the apparent positions of the planets as observed from the earth over a period of years. A small program incorporating the very simple Newtonian theory for this situation will predict a great many astronomical observations. In this problem there are no data—only a program that contains, of course, a table of the masses of the planets and their initial positions and velocities.

Section III

Let us now consider the problem of the amount of information that it is necessary to provide to a computer in order for it to calculate a given finite binary sequence. A computing machine is defined for these purposes to be a device that accepts as input a program, performs the calculations indicated to it in the program, and finally puts out the binary sequence it has calculated. In line with the mathematical theory of information, it is natural for the program to be viewed as a sequence of bits or 0's and 1's. Furthermore, in computer engineering all programs and data are represented in the machine's circuits in binary form. Thus, we may consider a computer to be a device that accepts one binary sequence (the program) and emits another (the result of the calculation).

011001001→COMPUTER→11111100100011001101000

As an example of a computer we would then have an electronic digital computer that accepts programs consisting of magnetized spots on magnetic tape and puts out its results in the same form. Another example is a Turing machine. The program is a series of 0's and 1's written on the machine's tape at the start of the calculation, and the result is a sequence of 0's and 1's written on its tape when it stops. As was mentioned, the second of these examples can do anything that the first can.

Section IV

We are interested in the amount of information that must be supplied to a computer M in order for it to calculate a given finite binary sequence S. We may now define this as the size or length of the smallest binary sequence that causes the machine M to calculate S. We denote the length of the shortest program for M to calculate S by $L(M, S)$. It has been shown that there is a computing machine M that has the following three properties.[2]

1) $L(M, S) \leq k + 1$ for all binary sequences S of length k.

In other words, any binary sequence of length k can be calculated by this computer M if it is given an appropriate program at most $k + 1$ bits in length. The proof is as follows. If no better way to calculate a binary sequence occurs to us, we can always include the binary sequence as a table in the program. This computer is so designed that we need add only a single bit to the sequence to obtain a program for computing it. The computer M emits the sequence S when it is given the program $S0$.

2) Those binary sequences S for which $L(M, S) < j$ are fewer than 2^j in number.

Thus, most binary sequences of length k require programs of about the same length k, and the number of sequences that can be computed by smaller programs decreases exponentially as the size of the program decreases. The proof is as follows. There are only $2^j - 2$ binary sequences less than j in length. Thus, there are fewer than 2^j programs less than j in length, for each program is a binary sequence. At best, a program will cause the computer to calculate a single binary sequence. At worst, an error in the program will trap the computer in an endless loop, and no binary sequence will be calculated. As each program causes the computer to calculate at most one binary sequence, the number of sequences calculated must be smaller than the number of programs. Thus, fewer than 2^j binary sequences can be calculated by means of programs less than j in length.

3) For any other computer M' there exists a constant $c(M')$ such that for all binary sequences S, $L(M, S) \leq L(M', S) + c(M')$.

[2]Solomonoff [8] was the first to employ computers of this kind.

In other words, this computer requires shorter programs than any other computer, or more exactly it does not require programs much longer than those required by any other computer. The proof is as follows. The computer M is designed to interpret the circuit diagrams of any other computer M'. Given a program for M' and the circuit diagrams of M', the computer M proceeds to calculate how M' would behave, i.e., it proceeds to simulate M'. Thus, we need only add a fixed number of bits to any program for M' in order to obtain a program that enables M to calculate the same result. This program for M is of the form $PC1$.

The 1 at the right end of the program indicates to the computer M that this is a simulation, C is a fixed binary sequence of length $c(M') - 1$ giving the circuit diagrams of the computer M', which is to be imitated, and P is the program for M'.[3]

Section V

Kolmogorov [9] and the author [11], [12] have independently suggested that computers such as those previously described be applied to the problem of defining what is meant by a random or patternless finite binary sequence of 0's and 1's. In the traditional foundations of the mathematical theory of probability, as expounded by Kolmogorov in his classic [10], there is no place for the concept of an individual random sequence of 0's and 1's. Yet it is not altogether meaningless to say that the sequence

$$11001011111001100101111000001 0$$

is more random or patternless than the sequences

$$11111111111111111111111111111 1$$
$$01010101010101010101010101010 1,$$

for we may describe these last two sequences as thirty 1's or fifteen 01's, but there is no shorter way to specify the first sequence than by just writing it all out.

We believe that the random or patternless sequences of a given length are those that require the longest programs. We have seen that most of the

[3]How can the computer M separate PC into P and C? C has each of its bits doubled, except the pair of bits at its left end. These are unequal and serve as punctuation separating C from P.

binary sequences of length k require programs of about length k. These, then, are the random or patternless sequences. Those sequences that can be obtained by putting into a computer a program much shorter than k are the nonrandom sequences, those that possess a pattern or follow a law. The more possible it is to compress a binary sequence into a short program calculation, the less random is the sequence.

As an example of this, let us consider those sequences of 0's and 1's in which 0's and 1's do not occur with equal frequency. Let p be the relative frequency of 1's, and let $q = 1 - p$ be the relative frequency of 0's. A long binary sequence that has the property that 1's are more frequent than 0's can be obtained from a computer program whose length is only that of the desired sequence reduced by a factor $H(p, q) = -p \log_2 p - q \log_2 q$. For example, if 1's occur approximately $\frac{3}{4}$ of the time and 0's occur $\frac{1}{4}$ of the time in a long binary sequence of length k, there is a program for computing that sequence with length only about $H(\frac{3}{4}, \frac{1}{4})k = 0.80k$. That is, the program need be only approximately 80 percent the length of the sequence it computes. In summary, if 0's and 1's occur with unequal frequencies, we can compress such sequences into programs only a certain percentage (depending on the frequencies) of the size of the sequence. Thus, random or incompressible sequences will have about as many 0's as 1's, which agrees with our intuitive expectations.

In a similar manner it can be shown that all groups of 0's and 1's will occur with approximately the expected frequency in a long binary sequence that we call random; 01100 will appear $2^{-5}k$ times in long sequences of length k, etc.[4]

Section VI

The definition of random or patternless finite binary sequences just presented is related to certain considerations in information theory and in the methodology of science.

The two problems considered in Shannon's classical exposition [15] are to transmit information as efficiently and as reliably as possible. Here we are interested in examining the viewpoint of information theory concerning the efficient transmission of information. An information source may be redundant, and information theory teaches us to code or compress messages

[4]Martin-Löf [14] also discusses the statistical properties of random sequences.

so that what is redundant is eliminated and communications equipment is optimally employed. For example, let us consider an information source that emits one symbol (either an A or a B) each second. Successive symbols are independent, and A's are three times more frequent than B's. Suppose it is desired to transmit the messages over a channel that is capable of transmitting either an A or a B each second. Then the channel has a capacity of 1 bit per second, while the information source has entropy 0.80 bits per symbol; and thus it is possible to code the messages in such a way that on the average $1/0.80 = 1.25$ symbols of message are transmitted over the channel each second. The receiver must decode the messages; that is, expand them into their original form.

In summary, information theory teaches us that messages from an information source that is not completely random (that is, which does not have maximum entropy) can be compressed. The definition of randomness is merely the converse of this fundamental theorem of information theory; if lack of randomness in a message allows it to be coded into a shorter sequence, then the random messages must be those that cannot be coded into shorter messages. A computing machine is clearly the most general possible decoder for compressed messages. We thus consider that this definition of randomness is in perfect agreement and indeed strongly suggested by the coding theorem for a noiseless channel of information theory.

Section VII

This definition is also closely related to classical problems of the methodology of science.[5]

Consider a scientist who has been observing a closed system that once every second either emits a ray of light or does not. He summarizes his observations in a sequence of 0's and 1's in which a 0 represents "ray not emitted" and a 1 represents "ray emitted." The sequence may start

$$0110101110\ldots$$

and continue for a few million more bits. The scientist then examines the sequence in the hope of observing some kind of pattern or law. What does he mean by this? It seems plausible that a sequence of 0's and 1's is patternless

[5]Solomonoff [8] also discusses the relation between program lengths and the problem of induction.

if there is no better way to calculate it than just by writing it all out at once from a table giving the whole sequence. The scientist might state:

My Scientific Theory: 0110101110...

This would not be considered an acceptable theory. On the other hand, if the scientist should hit upon a method by which the whole sequence could be calculated by a computer whose program is short compared with the sequence, he would certainly not consider the sequence to be entirely patternless or random. The shorter the program, the greater the pattern he may ascribe the sequence.

There are many parallels between the foregoing and the way scientists actually think. For example, a simple theory that accounts for a set of facts is generally considered better or more likely to be true than one that needs a large number of assumptions. By "simplicity" is not meant "ease of use in making predictions." For although general relativity is considered to be the simple theory par excellence, very extended calculations are necessary to make predictions from it. Instead, one refers to the number of arbitrary choices that have been made in specifying the theoretical structure. One is naturally suspicious of a theory whose number of arbitrary elements is of an order of magnitude comparable to the amount of information about reality that it accounts for.

Section VIII

Let us now turn to the problem of the amount of time necessary for computations.[6] We will develop the following thesis. Call an infinite set of natural numbers perfect if there is no essentially quicker way to compute infinitely many of its members than computing the whole set. Perfect sets exist. This thesis was suggested by the following vague and imprecise considerations.[7]

One of the most profound problems of the theory of numbers is that of calculating large primes. While the sieve of Eratosthenes appears to be as quick an algorithm for calculating all the primes as is possible, in recent times hope has centered on calculating large primes by calculating a subset

[6] As general references we may cite Blum [16] and Arbib and Blum [17]. Our exposition is a summary of that of [13].

[7] See Hardy and Wright [18], Sections 1.4 and 2.5 for the number-theoretic background of the following remarks.

of the primes, those that are Mersenne numbers. Lucas's test can decide the primality of a Mersenne number with rapidity far greater than is furnished by the sieve method. If there are an infinity of Mersenne primes, then it appears that Lucas has achieved a decisive advance in this classical problem of the theory of numbers.

An opposing point of view is that there is no essentially better way to calculate large primes than by calculating them all. If this is the case, it apparently follows that there must be only finitely many Mersenne primes.

These considerations, then, suggested that there are infinite sets of natural numbers that are arbitrarily difficult to compute, and that do not have any infinite subsets essentially easier to compute than the whole set. Here difficulty of computation refers to speed. Our development will be as follows. First, we define computers for calculating infinite sets of natural numbers. Then we introduce a way of comparing the rapidity of computers, a transitive binary relation, i.e., almost a partial ordering. Next we focus our attention on those computers that are greater than or equal to all others under this ordering, i.e., the fastest computers. Our results are conditioned on the computers having this property. The meaning of "arbitrarily difficult to compute" is then clarified. Last, we exhibit sets that are arbitrarily difficult to compute and do not have any subset essentially easier to compute than the whole set.

Section IX

We are interested in the speed of programs for generating the elements of an infinite set of natural numbers. For these purposes we may consider a computer to be a device that once a second emits a (possibly empty) finite set of natural numbers and that once started never stops. That is to say, a computer is now viewed as a function whose arguments are the program and the time and whose value is a finite set of natural numbers. If a program causes the computer to emit infinitely many natural numbers in size order and without any repetitions, we say that the computing machine calculates the infinite set of natural numbers that it emits.

A Turing machine can be used to compute infinite sets of natural numbers; it is only necessary to establish a convention as to when natural numbers are emitted. For example, we may divide the machine's tape into two halves, and stipulate that what is written on the right half cannot be erased.

The computational scratchwork is done on the left half of the tape, and the successive members of the infinite set of natural numbers are written on the nonerasable squares in decimal notation, separated by commas, with no blank spaces permitted between characters. The moment a comma has been written, it is considered that the digits between it and the previous comma form the numeral representing the next natural number emitted by the machine. We suppose that the Turing machine performs a single cycle of activity (read tape; shift, write, or erase tape; change internal state) each second. Last, we stipulate that the machine be started scanning the first nonerasable square of the tape, that initially the nonerasable squares be all blank, and that the program for the computer be written on the first erasable squares, with a blank serving as punctuation to indicate the end of the program and the beginning of an infinite blank region of tape.

Section X

We now order the computers according to their speeds. $C \geq C'$ is defined as meaning that C is not much slower than C'.

What do we mean by saying that computer C is not much slower than computer C' for the purpose of computing infinite sets of natural numbers? There is a computable change of C's time scale that makes C as fast as C' or faster. More exactly, there is a computable function $f(n)$ (for example $n!$ or $n^{n^{n^{\cdots}}}$ with n exponents) with the following property. Let P' be any program that makes C' calculate an infinite set of natural numbers. Then there exists a program P that makes C calculate the same set of natural numbers and has the additional property that every natural number emitted by C' during the first t seconds of calculation is emitted by C during the first $f(t)$ second of calculation, for all but a finite number of values of t. We may symbolize this relation between the computers C and C' as $C \geq C'$, for it has the property that $C \geq C'$ and $C' \geq C''$ only if $C \geq C''$.

In this way, we have introduced an ordering of the computers for computing infinite sets of natural numbers, and it can be shown that a distributive lattice results. The most important property of this ordering for our present purposes is that there is a set of computers $\geq$ all other computers. In what follows we assume that the computer that is used is a member of this set of fastest computers.

Section XI

We now clarify what we mean by "arbitrarily difficult to compute."

Let $f(n)$ be any computable function that carries natural numbers into natural numbers. Such functions can get big very quickly indeed. For example consider the function $n^{n^{n^{\cdots}}}$ in which there are n^n exponents. There are infinite sets of natural numbers such that, no matter how the computer is programmed, at least $f(n)$ seconds will pass before the computer emits all those elements of the set that are less than or equal to n. Of course, a finite number of exceptions are possible, for any finite part of an infinite set can be computed very quickly by including in the computer's program a table of the first few elements of the set. Note that the difficulty in computing such sets of natural numbers does not lie in the fact that their elements get very big very quickly, for even small elements of such sets require more than astronomical amounts of time to be computed. What is more, there are infinite sets of natural numbers that are arbitrarily difficult to compute and include 90 percent of the natural numbers.

We finally exhibit infinite sets of natural numbers that are arbitrarily difficult to compute, and do not have any infinite subsets essentially easier to compute than the whole set. Consider the following tree of natural numbers (figure 2).[8] The infinite sets of natural numbers that we promised to exhibit are obtained by starting at the root of the tree (that is, at 0) and walking forward, including in the set every natural number that is stepped on.

It is easy to see that no infinite subset of such a set can be computed much more quickly than the whole set. For suppose we are told that n is in such a set. Then we know at once that the greatest integer less than $n/2$ is the previous element of the set. Thus, knowing that 1 000 000 is in the set, we immediately produce all smaller elements in it, by walking backwards through the tree. They are 499 999, 249 999, 124 999, etc. It follows that there is no appreciable difference between generating an infinite subset of such a set, and generating the whole set, for gaps in an incomplete generation can be filled in very quickly.

It is also easy to see that there are sets that can be obtained by walking through this tree and are arbitrarily difficult to compute. These, then, are the sets that we wished to exhibit.

[8]This tree is used in Rogers [6], p. 158, in connection with retraceable sets. Retraceable sets are in some ways analogous to those sets that concern us here.

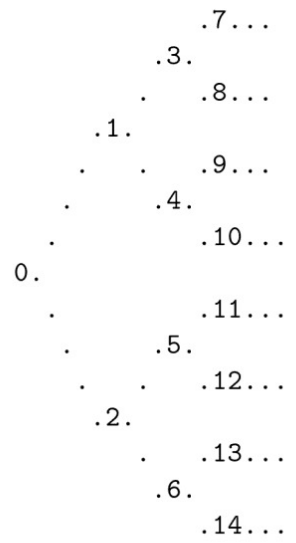

Figure 2. A tree of natural numbers

Acknowledgment

The author wishes to express his gratitude to Prof. G. Pollitzer of the University of Buenos Aires, whose constructive criticism much improved the clarity of this presentation.

References

[1] M. Davis, Ed., *The Undecidable.* Hewlett, N.Y.: Raven Press, 1965.

[2] —, *Computability and Unsolvability.* New York: McGraw-Hill, 1958.

[3] —, "Unsolvable problems: A review," *Proc. Symp. on Mathematical Theory of Automata.* Brooklyn, N.Y.: Polytech. Inst. Brooklyn Press, 1963, pp. 15–22.

[4] —, "Applications of recursive function theory to number theory," *Proc. Symp. in Pure Mathematics,* vol. 5. Providence, R.I.: AMS, 1962, pp. 135–138.

[5] M. Minsky, *Computation: Finite and Infinite Machines.* Englewood Cliffs, N.J.: Prentice-Hall, 1967.

[6] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability.* New York: McGraw-Hill, 1967.

[7] M. A. Arbib, *Theories of Abstract Automata.* Englewood Cliffs, N.J.: Prentice-Hall (to be published).

[8] R. J. Solomonoff, "A formal theory of inductive inference," *Inform. and Control,* vol. 7, pp. 1–22, March 1964; pp. 224–254, June 1964.

[9] A. N. Kolmogorov, "Three approaches to the definition of the concept 'quantity of information'," *Probl. Peredachi Inform.,* vol. 1, pp. 3–11, 1965.

[10] —, *Foundations of the Theory of Probability.* New York: Chelsea, 1950.

[11] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *J. ACM,* vol. 13, pp. 547–569, October 1966.

[12] —, "On the length of programs for computing finite binary sequences: statistical considerations," *J. ACM,* vol. 16, pp. 145–159, January 1969.

[13] —, "On the simplicity and speed of programs for computing infinite sets of natural numbers," *J. ACM,* vol. 16, pp. 407–422, July 1969.

[14] P. Martin-Löf, "The definition of random sequences," *Inform. and Control,* vol. 9, pp. 602–619, December 1966.

[15] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication.* Urbana, Ill.: University of Illinois Press, 1949.

[16] M. Blum, "A machine-independent theory of the complexity of recursive functions," *J. ACM,* vol. 14, pp. 322–336, April 1967.

[17] M. A. Arbib and M. Blum, "Machine dependence of degrees of difficulty," *Proc. AMS,* vol. 16, pp. 442–447, June 1965.

[18] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers.* Oxford: Oxford University Press, 1962.

The following references have come to the author's attention since this lecture was given.

[19] D. G. Willis, "Computational complexity and probability constructions," Stanford University, Stanford, Calif., March 1969.

[20] A. N. Kolmogorov, "Logical basis for information theory and probability theory," *IEEE Trans. Information Theory,* vol. IT-14, pp. 662–664, September 1968.

[21] D. W. Loveland, "A variant of the Kolmogorov concept of complexity," Dept. of Math., Carnegie-Mellon University, Pittsburgh, Pa., Rept. 69-4.

[22] P. R. Young, "Toward a theory of enumerations," *J. ACM,* vol. 16, pp. 328–348, April 1969.

[23] D. E. Knuth, *The Art of Computer Programming;* vol. 2, *Seminumerical Algorithms.* Reading, Mass.: Addison-Wesley, 1969.

[24] *1969 Conf. Rec. of the ACM Symp. on Theory of Computing* (Marina del Rey, Calif.).

This page intentionally left blank

Information-theoretic computational complexity

This paper attempts to describe, in nontechnical language, some of the concepts and methods of one school of thought regarding computational complexity. It applies the viewpoint of information theory to computers. This will first lead us to a definition of the degree of randomness of individual binary strings, and then to an information-theoretic version of Gödel's theorem on the limitations of the axiomatic method. Finally, we will examine in the light of these ideas the scientific method and von Neumann's views on the basic conceptual problems of biology. [This paper was presented at the IEEE International Congress of Information Theory, Ashkelon, Israel, June 1973.]

This field's fundamental concept is the complexity of a binary string, that is, a string of bits, of zeros and ones. The complexity of a binary string is the minimum quantity of information needed to define the string. For example, the string of length n consisting entirely of ones is of complexity approximately $\log_2 n$, because only $\log_2 n$ bits of information are required to specify n in binary notation.

However, this is rather vague. Exactly what is meant by the definition of a string? To make this idea precise a computer is used. One says that a string defines another when the first string gives instructions for constructing the second string. In other words, one string defines another when it is a program for a computer to calculate the second string. The fact that a string of n ones is of complexity approximately $\log_2 n$ can now be translated more correctly into the following. There is a program $\log_2 n + c$ bits long that calculates the string of n ones. The program performs a loop for printing

ones n times. A fixed number c of bits are needed to program the loop, and $\log_2 n$ bits more for specifying n in binary notation.

Exactly how are the computer and the concept of information combined to define the complexity of a binary string? A computer is considered to take one binary string and perhaps eventually produce another. The first string is the program that has been given to the machine. The second string is the output of this program; it is what this program calculates. Now consider a given string that is to be calculated. How much information must be given to the machine to do this? That is to say, what is the length in bits of the shortest program for calculating the string? This is its complexity.

It can be objected that this is not a precise definition of the complexity of a string, inasmuch as it depends on the computer that one is using. Moreover, a definition should not be based on a machine, but rather on a model that does not have the physical limitations of real computers.

Here we will not define the computer used in the definition of complexity. However, this can indeed be done with all the precision of which mathematics is capable. Since 1936 it has been known how to define an idealized computer with unlimited memory. This was done in a very intuitive way by Turing and also by Post, and there are elegant definitions based on other principles [2]. The theory of recursive functions (or computability theory) has grown up around the questions of what is computable and what is not.

Thus it is not difficult to define a computer mathematically. What remains to be analyzed is which definition should be adopted, inasmuch as some computers are easier to program than others. A decade ago Solomonoff solved this problem [7]. He constructed a definition of a computer whose programs are not much longer than those of any other computer. More exactly, Solomonoff's machine simulates running a program on another computer, when it is given a description of that computer together with its program.

Thus it is clear that the complexity of a string is a mathematical concept, even though here we have not given a precise definition. Furthermore, it is a very natural concept, easy to understand for those who have worked with computers. Recapitulating, the complexity of a binary string is the information needed to define it, that is to say, the number of bits of information that must be given to a computer in order to calculate it, or in other words, the size in bits of the shortest program for calculating it. It is understood that a certain mathematical definition of an idealized computer is being used, but it is not given here, because as a first approximation it is sufficient to think of the length in bits of a program for a typical computer in use today.

Now we would like to consider the most important properties of the complexity of a string. First of all, the complexity of a string of length n is less than $n + c$, because any string of length n can be calculated by putting it directly into a program as a table. This requires n bits, to which must be added c bits of instructions for printing the table. In other words, if nothing betters occurs to us, the string itself can be used as its definition, and this requires only a few more bits than its length.

Thus the complexity of each string of length n is less than $n+c$. Moreover, the complexity of the great majority of strings of length n is approximately n, and very few strings of length n are of complexity much less than n. The reason is simply that there are much fewer programs of length appreciably less than n than strings of length n. More exactly, there are 2^n strings of length n, and less than 2^{n-k} programs of length less than $n - k$. Thus the number of strings of length n and complexity less than $n - k$ decreases exponentially as k increases.

These considerations have revealed the basic fact that the great majority of strings of length n are of complexity very close to n. Therefore, if one generates a binary string of length n by tossing a fair coin n times and noting whether each toss gives head or tail, it is highly probable that the complexity of this string will be very close to n. In 1965 Kolmogorov proposed calling random those strings of length n whose complexity is approximately n [8]. We made the same proposal independently [9]. It can be shown that a string that is random in this sense has the statistical properties that one would expect. For example, zeros and ones appear in such strings with relative frequencies that tend to one-half as the length of the strings increases.

Consequently, the great majority of strings of length n are random, that is, need programs of approximately length n, that is to say, are of complexity approximately n. What happens if one wishes to show that a particular string is random? What if one wishes to prove that the complexity of a certain string is almost equal to its length? What if one wishes to exhibit a specific example of a string of length n and complexity close to n, and assure oneself by means of a proof that there is no shorter program for calculating this string?

It should be pointed out that this question can occur quite naturally to a programmer with a competitive spirit and a mathematical way of thinking. At the beginning of the sixties we attended a course at Columbia University in New York. Each time the professor gave an exercise to be programmed, the students tried to see who could write the shortest program. Even though

several times it seemed very difficult to improve upon the best program that had been discovered, we did not fool ourselves. We realized that in order to be sure, for example, that the shortest program for the IBM 650 that prints the prime numbers has, say, 28 instructions, it would be necessary to prove it, not merely to continue for a long time unsuccessfully trying to discover a program with less than 28 instructions. We could never even sketch a first approach to a proof.

It turns out that it was not our fault that we did not find a proof, because we faced a fundamental limitation. One confronts a very basic difficulty when one tries to prove that a string is random, when one attempts to establish a lower bound on its complexity. We will try to suggest why this problem arises by means of a famous paradox, that of Berry [1, p. 153].

Consider the smallest positive integer that cannot be defined by an English phrase with less than 1 000 000 000 characters. Supposedly the shortest definition of this number has 1 000 000 000 or more characters. However, we defined this number by a phrase much less than 1 000 000 000 characters in length when we described it as "the smallest positive integer that cannot be defined by an English phrase with less than 1 000 000 000 characters!"

What relationship is there between this and proving that a string is complex, that its shortest program needs more than n bits? Consider the first string that can be proven to be of complexity greater than 1 000 000 000. Here once more we face a paradox similar to that of Berry, because this description leads to a program with much less than 1 000 000 000 bits that calculates a string supposedly of complexity greater than 1 000 000 000. Why is there a short program for calculating "the first string that can be proven to be of complexity greater than 1 000 000 000?"

The answer depends on the concept of a formal axiom system, whose importance was emphasized by Hilbert [1]. Hilbert proposed that mathematics be made as exact and precise as possible. In order to avoid arguments between mathematicians about the validity of proofs, he set down explicitly the methods of reasoning used in mathematics. In fact, he invented an artificial language with rules of grammar and spelling that have no exceptions. He proposed that this language be used to eliminate the ambiguities and uncertainties inherent in any natural language. The specifications are so precise and exact that checking if a proof written in this artificial language is correct is completely mechanical. We would say today that it is so clear whether a proof is valid or not that this can be checked by a computer.

Hilbert hoped that this way mathematics would attain the greatest pos-

sible objectivity and exactness. Hilbert said that there can no longer be any doubt about proofs. The deductive method should be completely clear.

Suppose that proofs are written in the language that Hilbert constructed, and in accordance with his rules concerning the accepted methods of reasoning. We claim that a computer can be programmed to print all the theorems that can be proven. It is an endless program that every now and then writes on the printer a theorem. Furthermore, no theorem is omitted. Each will eventually be printed, if one is very patient and waits long enough.

How is this possible? The program works in the following manner. The language invented by Hilbert has an alphabet with finitely many signs or characters. First the program generates the strings of characters in this alphabet that are one character in length. It checks if one of these strings satisfies the completely mechanical rules for a correct proof and prints all the theorems whose proofs it has found. Then the program generates all the possible proofs that are two characters in length, and examines each of them to determine if it is valid. The program then examines all possible proofs of length three, of length four, and so on. If a theorem can be proven, the program will eventually find a proof for it in this way, and then print it.

Consider again "the first string that can be proven to be of complexity greater than 1 000 000 000." To find this string one generates all theorems until one finds the first theorem that states that a particular string is of complexity greater than 1 000 000 000. Moreover, the program for finding this string is short, because it need only have the number 1 000 000 000 written in binary notation, $\log_2$ 1 000 000 000 bits, and a routine of fixed length c that examines all possible proofs until it finds one that a specific string is of complexity greater than 1 000 000 000.

In fact, we see that there is a program $\log_2 n + c$ bits long that calculates the first string that can be proven to be of complexity greater than n. Here we have Berry's paradox again, because this program of length $\log_2 n + c$ calculates something that supposedly cannot be calculated by a program of length less than or equal to n. Also, $\log_2 n + c$ is much less than n for all sufficiently great values of n, because the logarithm increases very slowly.

What can the meaning of this paradox be? In the case of Berry's original paradox, one cannot arrive at a meaningful conclusion, inasmuch as one is dealing with vague concepts such as an English phrase's defining a positive integer. However our version of the paradox deals with exact concepts that have been defined mathematically. Therefore, it cannot really be a contradiction. It would be absurd for a string not to have a program of length

less than or equal to n for calculating it, and at the same time to have such a program. Thus we arrive at the interesting conclusion that such a string cannot exist. For all sufficiently great values of n, one cannot talk about "the first string that can be proven to be of complexity greater than n," because this string cannot exist. In other words, for all sufficiently great values of n, it cannot be proven that a particular string is of complexity greater than n. If one uses the methods of reasoning accepted by Hilbert, there is an upper bound to the complexity that it is possible to prove that a particular string has.

This is the surprising result that we wished to obtain. Most strings of length n are of complexity approximately n, and a string generated by tossing a coin will almost certainly have this property. Nevertheless, one cannot exhibit individual examples of arbitrarily complex strings using methods of reasoning accepted by Hilbert. The lower bounds on the complexity of specific strings that can be established are limited, and we will never be mathematically certain that a particular string is very complex, even though most strings are random.[1]

In 1931 Gödel questioned Hilbert's ideas in a similar way [1], [2]. Hilbert had proposed specifying once and for all exactly what is accepted as a proof, but Gödel explained that no matter what Hilbert specified so precisely, there would always be true statements about the integers that the methods of reasoning accepted by Hilbert would be incapable of proving. This mathematical result has been considered to be of great philosophical importance. Von Neumann commented that the intellectual shock provoked by the crisis in the foundations of mathematics was equaled only by two other scientific events in this century: the theory of relativity and quantum theory [4].

We have combined ideas from information theory and computability theory in order to define the complexity of a binary string, and have then used this concept to give a definition of a random string and to show that a formal axiom system enables one to prove that a random string is indeed random in only finitely many cases.

Now we would like to examine some other possible applications of this

[1]This is a particularly perverse example of Kac's comment [13, p. 16] that "as is often the case, it is much easier to prove that an overwhelming majority of objects possess a certain property than to *exhibit* even one such object." The most familiar example of this is Shannon's proof of the coding theorem for a noisy channel; while it is shown that most coding schemes achieve close to the channel capacity, in practice it is difficult to implement a good coding scheme.

viewpoint. In particular, we would like to suggest that the concept of the complexity of a string and the fundamental methodological problems of science are intimately related. We will also suggest that this concept may be of theoretical value in biology.

Solomonoff [7] and the author [9] proposed that the concept of complexity might make it possible to precisely formulate the situation that a scientist faces when he has made observations and wishes to understand them and make predictions. In order to do this the scientist searches for a theory that is in agreement with all his observations. We consider his observations to be represented by a binary string, and a theory to be a program that calculates this string. Scientists consider the simplest theory to be the best one, and that if a theory is too "ad hoc," it is useless. How can we formulate these intuitions about the scientific method in a precise fashion? The simplicity of a theory is inversely proportional to the length of the program that constitutes it. That is to say, the best program for understanding or predicting observations is the shortest one that reproduces what the scientist has observed up to that moment. Also, if the program has the same number of bits as the observations, then it is useless, because it is too "ad hoc." If a string of observations only has theories that are programs with the same length as the string of observations, then the observations are random, and can neither be comprehended nor predicted. They are what they are, and that is all; the scientist cannot have a theory in the proper sense of the concept; he can only show someone else what he observed and say "it was this."

In summary, the value of a scientific theory is that it enables one to compress many observations into a few theoretical hypotheses. There is a theory only when the string of observations is not random, that is to say, when its complexity is appreciably less than its length in bits. In this case the scientist can communicate his observations to a colleague much more economically than by just transmitting the string of observations. He does this by sending his colleague the program that is his theory, and this program must have much fewer bits than the original string of observations.

It is also possible to make a similar analysis of the deductive method, that is to say, of formal axiom systems. This is accomplished by analyzing more carefully the new version of Berry's paradox that was presented. Here we only sketch the three basic results that are obtained in this manner.[2]

1. In a formal system with n bits of axioms it is impossible to prove that

[2]See the Appendix.

a particular binary string is of complexity greater than $n + c$.

2. Contrariwise, there are formal systems with $n + c$ bits of axioms in which it is possible to determine each string of complexity less than n and the complexity of each of these strings, and it is also possible to exhibit each string of complexity greater than or equal to n, but without being able to know by how much the complexity of each of these strings exceeds n.

3. Unfortunately, any formal system in which it is possible to determine each string of complexity less than n has either one grave problem or another. Either it has few bits of axioms and needs incredibly long proofs, or it has short proofs but an incredibly great number of bits of axioms. We say "incredibly" because these quantities increase more quickly than any computable function of n.

It is necessary to clarify the relationship between this and the preceding analysis of the scientific method. There are less than 2^n strings of complexity less than n, but some of them are incredibly long. If one wishes to communicate all of them to someone else, there are two alternatives. The first is to directly show all of them to him. In this case one will have to send him an incredibly long message because some of these strings are incredibly long. The other alternative is to send him a very short message consisting of n bits of axioms from which he can deduce which strings are of complexity less than n. Although the message is very short in this case, he will have to spend an incredibly long time to deduce from these axioms the strings of complexity less than n. This is analogous to the dilemma of a scientist who must choose between directly publishing his observations, or publishing a theory that explains them, but requires very extended calculations in order to do this.

Finally, we would like to suggest that the concept of complexity may possibly be of theoretical value in biology.

At the end of his life von Neumann tried to lay the foundation for a mathematics of biological phenomena. His first effort in this direction was his work *Theory of Games and Economic Behavior,* in which he analyzes what is a rational way to behave in situations in which there are conflicting interests [3]. *The Computer and the Brain,* his notes for a lecture series, was published shortly after his death [5]. This book discusses the differences and similarities between the computer and the brain, as a first step to a theory of

how the brain functions. A decade later his work *Theory of Self-Reproducing Automata* appeared, in which von Neumann constructs an artificial universe and within it a computer that is capable of reproducing itself [6]. But von Neumann points out that the problem of formulating a mathematical theory of the evolution of life in this abstract setting remains to be solved; and to express mathematically the evolution of the complexity of organisms, one must first define complexity precisely.[3] We submit that "organism" must also be defined, and have tried elsewhere to suggest how this might perhaps be done [10].

We believe that the concept of complexity that has been presented here may be the tool that von Neumann felt is needed. It is by no means accidental that biological phenomena are considered to be extremely complex. Consider how a human being analyzes what he sees, or uses natural languages to communicate. We cannot carry out these tasks by computer because they are as yet too complex for us—the programs would be too long.[4]

Appendix

In this Appendix we try to give a more detailed idea of how the results concerning formal axiom systems that were stated are established.[5]

Two basic mathematical concepts that are employed are the concepts of a recursive function and a partial recursive function. A function is recursive if there is an algorithm for calculating its value when one is given the value of its arguments, in other words, if there is a Turing machine for doing this. If it is possible that this algorithm never terminates and the function is thus undefined for some values of its arguments, then the function is called partial recursive.[6]

In what follows we are concerned with computations involving binary strings. The binary strings are considered to be ordered in the following manner: Λ, 0, 1, 00, 01, 10, 11, 000, 001, 010, ... The natural number n is represented by the nth binary string ($n = 0, 1, 2, \ldots$). The length of a binary

[3]In an important paper [14], Eigen studies these questions from the point of view of thermodynamics and biochemistry.

[4]Chandrasekaran and Reeker [15] discuss the relevance of complexity to artificial intelligence.

[5]See [11], [12] for different approaches.

[6]Full treatments of these concepts can be found in standard texts, e.g., Rogers [16].

string s is denoted $\lg(s)$. Thus if s is considered to be a natural number, then $\lg(s) = [\log_2(s+1)]$. Here $[x]$ is the greatest integer $\leq x$.

Definition 1. A *computer* is a partial recursive function $C(p)$. Its argument p is a binary string. The value of $C(p)$ is the binary string output by the computer C when it is given the program p. If $C(p)$ is undefined, this means that running the program p on C produces an unending computation.

Definition 2. The *complexity* $I_C(s)$ of a binary string s is defined to be the length of the shortest program p that makes the computer C output s, i.e.,

$$I_C(s) = \min_{C(p)=s} \lg(p).$$

If no program makes C output s, then $I_C(s)$ is defined to be infinite.

Definition 3. A computer U is *universal* if for any computer C and any binary string s, $I_U(s) \leq I_C(s) + c$, where the constant c depends only on C.

It is easy to see that there are universal computers. For example, consider the computer U such that $U(0^i 1p) = C_i(p)$, where C_i is the ith computer, i.e., a program for U consists of two parts: the left-hand part indicates which computer is to be simulated, and the right-hand part gives the program to be simulated. We now suppose that some particular universal computer U has been chosen as the standard one for measuring complexities, and shall henceforth write $I(s)$ instead of $I_U(s)$.

Definition 4. The *rules of inference* of a class of formal axiom systems is a recursive function $F(a,h)$ (a a binary string, h a natural number) with the property that $F(a,h) \subset F(a,h+1)$. The value of $F(a,h)$ is the finite (possibly empty) set of theorems that can be proven from the axioms a by means of proofs $\leq h$ characters in length. $F(a) = \bigcup_h F(a,h)$ is the set of theorems that are consequences of the axioms a. The ordered pair $\langle F, a\rangle$, which implies both the choice of rules of inference and axioms, is a particular formal axiom system.

This is a fairly abstract definition, but it retains all those features of formal axiom systems that we need. Note that although one may not be interested in some axioms (e.g., if they are false or incomprehensible), it is stipulated that $F(a,h)$ is always defined.

Theorem 1. a) There is a constant c such that $I(s) \leq \lg(s) + c$ for all binary strings s. b) There are less than 2^n binary strings of complexity less than n.

Proof of a). There is a computer C such that $C(p) = p$ for all programs p. Thus for all binary strings s, $I(s) \leq I_C(s) + c = \lg(s) + c$.

Proof of b). As there are less than 2^n programs of length less than n, there must be less than this number of binary strings of complexity less than n. Q.E.D.

Thesis. A random binary string s is one having the property that $I(s) \approx \lg(s)$.

Theorem 2. Consider the rules of inference F. Suppose that a proposition of the form "$I(s) \geq n$" is in $F(a)$ only if it is true, i.e., only if $I(s) \geq n$. Then a proposition of the form "$I(s) \geq n$" is in $F(a)$ only if $n \leq \lg(a) + c$, where c is a constant that depends only on F.

Proof. Consider that binary string s_k having the shortest proof from the axioms a that it is of complexity $> \lg(a) + 2k$. We claim that $I(s_k) \leq \lg(a) + k + c'$, where c' depends only on F. Taking $k = c'$, we conclude that the binary string $s_{c'}$ with the shortest proof from the axioms a that it is of complexity $> \lg(a) + 2c'$ is, in fact, of complexity $\leq \lg(a) + 2c'$, which is impossible. It follows that s_k doesn't exist for $k = c'$, that is, no binary string can be proven from the axioms a to be of complexity $> \lg(a) + 2c'$. Thus the theorem is proved with $c = 2c'$.

It remains to verify the claim that $I(s_k) \leq \lg(a) + k + c'$. Consider the computer C that does the following when it is given the program 0^k1a. It calculates $F(a, h)$ for $h = 0, 1, 2, \ldots$ until it finds the first theorem in $F(a, h)$ of the form "$I(s) \geq n$" with $n > \lg(a) + 2k$. Finally C outputs the binary string s in the theorem it has found. Thus $C(0^k1a)$ is equal to s_k, if s_k exists. It follows that

$$\begin{aligned} I(s_k) &= I(C(0^k1a)) \\ &\leq I_C(C(0^k1a)) + c'' \\ &\leq \lg(0^k1a) + c'' = \lg(a) + k + (c'' + 1) = \lg(a) + k + c'. \end{aligned}$$

Q.E.D.

Definition 5. A_n is defined to be the kth binary string of length n, where k is the number of programs p of length $< n$ for which $U(p)$ is defined, i.e., A_n has n and this number k coded into it.

Theorem 3. There are rules of inference F^1 such that for all n, $F^1(A_n)$ is the union of the set of all true propositions of the form "$I(s) = k$" with $k < n$ and the set of all true propositions of the form "$I(s) \geq n$."

Proof. From A_n one knows n and for how many programs p of length $< n$ $U(p)$ is defined. One then simulates in parallel, running each program p of length $< n$ on U until one has determined the value of $U(p)$ for each p of

length $< n$ for which $U(p)$ is defined. Knowing the value of $U(p)$ for each p of length $< n$ for which $U(p)$ is defined, one easily determines each string of complexity $< n$ and its complexity. What's more, all other strings must be of complexity $\geq n$. This completes our sketch of how all true propositions of the form "$I(s) = k$" with $k < n$ and of the form "$I(s) \geq n$" can be derived from the axiom A_n. Q.E.D.

Recall that we consider the nth binary string to be the natural number n.

Definition 6. The partial function $B(n)$ is defined to be the biggest natural number of complexity $\leq n$, i.e.,

$$B(n) = \max_{I(k) \leq n} k = \max_{\lg(p) \leq n} U(p).$$

Theorem 4. Let f be a partial recursive function that carries natural numbers into natural numbers. Then $B(n) \geq f(n)$ for all sufficiently great values of n.

Proof. Consider the computer C such that $C(p) = f(p)$ for all p.

$$I(f(n)) \leq I_C(f(n)) + c \leq \lg(n) + c = [\log_2(n+1)] + c < n$$

for all sufficiently great values of n. Thus $B(n) \geq f(n)$ for all sufficiently great values of n. Q.E.D.

Theorem 5. Consider the rules of inference F. Let

$$F_n = \bigcup_a F(a, B(n)),$$

where the union is taken over all binary strings a of length $\leq B(n)$, i.e., F_n is the (finite) set of all theorems that can be deduced by means of proofs with not more than $B(n)$ characters from axioms with not more than $B(n)$ bits. Let s_n be the first binary string s not in any proposition of the form "$I(s) = k$" in F_n. Then $I(s_n) \leq n + c$, where the constant c depends only on F.

Proof. We claim that there is a computer C such that if $U(p) = B(n)$, then $C(p) = s_n$. As, by the definition of B, there is a p_0 of length $\leq n$ such that $U(p_0) = B(n)$, it follows that

$$I(s_n) \leq I_C(s_n) + c = I_C(C(p_0)) + c \leq \lg(p_0) + c \leq n + c,$$

which was to be proved.

It remains to verify the claim that there is a C such that if $U(p) = B(n)$, then $C(p) = s_n$. C works as follows. Given the program p, C first simulates running the program p on U. Once C has determined $U(p)$, it calculates $F(a, U(p))$ for all binary strings a such that $\lg(a) \leq U(p)$, and forms the union of these $2^{U(p)+1} - 1$ different sets of propositions, which is F_n if $U(p) = B(n)$. Finally C outputs the first binary string s not in any proposition of the form "$I(s) = k$" in this set of propositions; s is s_n if $U(p) = B(n)$. Q.E.D.

Theorem 6. Consider the rules of inference F. If $F(a, h)$ includes all true propositions of the form "$I(s) = k$" with $k \leq n + c$, then either $\lg(a) > B(n)$ or $h > B(n)$. Here c is a constant that depends only on F.

Proof. This is an immediate consequence of Theorem 5. Q.E.D.

The following theorem gives an upper bound on the size of the proofs in the formal systems $\langle F^1, A_n \rangle$ that were studied in Theorem 3, and also shows that the lower bound on the size of these proofs that is given by Theorem 6 cannot be essentially improved.

Theorem 7. There is a constant c such that for all n $F^1(A_n, B(n + c))$ includes all true propositions of the form "$I(s) = k$" with $k < n$.

Proof. We claim that there is a computer C such that for all n, $C(A_n) =$ the least natural number h such that $F^1(A_n, h)$ includes all true propositions of the form "$I(s) = k$" with $k < n$. Thus the complexity of this value of h is $\leq \lg(A_n) + c = n + c$, and $B(n + c)$ is $\geq$ this value of h, which was to be proved.

It remains to verify the claim. C works as follows when it is given the program A_n. First, it determines each binary string of complexity $< n$ and its complexity, in the manner described in the proof of Theorem 3. Then it calculates $F^1(A_n, h)$ for $h = 0, 1, 2, \ldots$ until all true propositions of the form "$I(s) = k$" with $k < n$ are included in $F^1(A_n, h)$. The final value of h is then output by C. Q.E.D.

References

[1] J. van Heijenoort, Ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931.* Cambridge, Mass.: Harvard Univ. Press, 1967.

[2] M. Davis, Ed., *The Undecidable—Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions.* Hewlett, N.Y.: Raven Press, 1965.

[3] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior.* Princeton, N.J.: Princeton Univ. Press, 1944.

[4] —, "Method in the physical sciences," in *John von Neumann—Collected Works.* New York: Macmillan, 1963, vol. 6, no. 35.

[5] —, *The Computer and the Brain.* New Haven, Conn.: Yale Univ. Press, 1958.

[6] —, *Theory of Self-Reproducing Automata.* Urbana, Ill.: Univ. Illinois Press, 1966. (Edited and completed by A. W. Burks.)

[7] R. J. Solomonoff, "A formal theory of inductive inference," *Inform. Contr.,* vol. 7, pp. 1–22, Mar. 1964; also, pp. 224–254, June 1964.

[8] A. N. Kolmogorov, "Logical basis for information theory and probability theory," *IEEE Trans. Inform. Theory,* vol. IT-14, pp. 662–664, Sept. 1968.

[9] G. J. Chaitin, "On the difficulty of computations," *IEEE Trans. Inform. Theory,* vol. IT-16, pp. 5–9, Jan. 1970.

[10] —, "To a mathematical definition of 'life'," *ACM SICACT News,* no. 4, pp. 12–18, Jan. 1970.

[11] —, "Computational complexity and Gödel's incompleteness theorem," (Abstract) *AMS Notices,* vol. 17, p. 672, June 1970; (Paper) *ACM SIGACT News,* no. 9, pp. 11–12, Apr. 1971.

[12] —, "Information-theoretic limitations of formal systems," presented at the Courant Institute Computational Complexity Symp., N.Y., Oct. 1971. A revised version will appear in *J. Ass. Comput. Mach.*

[13] M. Kac, *Statistical Independence in Probability, Analysis, and Number Theory,* Carus Math. Mono., Mathematical Association of America, no. 12, 1959.

[14] M. Eigen, "Selforganization of matter and the evolution of biological macromolecules," *Die Naturwissenschaften,* vol. 58, pp. 465–523, Oct. 1971.

[15] B. Chandrasekaran and L. H. Reeker, "Artificial intelligence—a case for agnosticism," Ohio State University, Columbus, Ohio, Rep. OSU-CISRC-TR-72-9, Aug. 1972; also, *IEEE Trans. Syst., Man, Cybern.,* vol. SMC-4, pp. 88–94, Jan. 1974.

[16] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability.* New York: McGraw-Hill, 1967.

Randomness and mathematical proof

Although randomness can be precisely defined and can even be measured, a given number cannot be proved to be random. This enigma establishes a limit to what is possible in mathematics.

Almost everyone has an intuitive notion of what a random number is. For example, consider these two series of binary digits:

01010101010101010101
01101100110111100010

The first is obviously constructed according to a simple rule; it consists of the number 01 repeated ten times. If one were asked to speculate on how the series might continue, one could predict with considerable confidence that the next two digits would be 0 and 1. Inspection of the second series of digits yields no such comprehensive pattern. There is no obvious rule governing the formation of the number, and there is no rational way to guess the succeeding digits. The arrangement seems haphazard; in other words, the sequence appears to be a random assortment of 0's and 1's.

The second series of binary digits was generated by flipping a coin 20 times and writing a 1 if the outcome was heads and a 0 if it was tails. Tossing a coin is a classical procedure for producing a random number, and one might think at first that the provenance of the series alone would certify that it is random. This is not so. Tossing a coin 20 times can produce any one of 2^{20} (or a little more than a million) binary series, and each of them has

exactly the same probability. Thus it should be no more surprising to obtain the series with an obvious pattern than to obtain the one that seems to be random; each represents an event with a probability of 2^{-20}. If origin in a probabilistic event were made the sole criterion of randomness, then both series would have to be considered random, and indeed so would all others, since the same mechanism can generate all the possible series. The conclusion is singularly unhelpful in distinguishing the random from the orderly.

Clearly a more sensible definition of randomness is required, one that does not contradict the intuitive concept of a "patternless" number. Such a definition has been devised only in the past 10 years. It does not consider the origin of a number but depends entirely on the characteristics of the sequence of digits. The new definition enables us to describe the properties of a random number more precisely than was formerly possible, and it establishes a hierarchy of degrees of randomness. Of perhaps even greater interest than the capabilities of the definition, however, are its limitations. In particular the definition cannot help to determine, except in very special cases, whether or not a given series of digits, such as the second one above, is in fact random or only seems to be random. This limitation is not a flaw in the definition; it is a consequence of a subtle but fundamental anomaly in the foundation of mathematics. It is closely related to a famous theorem devised and proved in 1931 by Kurt Gödel, which has come to be known as Gödel's incompleteness theorem. Both the theorem and the recent discoveries concerning the nature of randomness help to define the boundaries that constrain certain mathematical methods.

Algorithmic Definition

The new definition of randomness has its heritage in information theory, the science, developed mainly since World War II, that studies the transmission of messages. Suppose you have a friend who is visiting a planet in another galaxy, and that sending him telegrams is very expensive. He forgot to take along his tables of trigonometric functions, and he has asked you to supply them. You could simply translate the numbers into an appropriate code (such as the binary numbers) and transmit them directly, but even the most modest tables of the six functions have a few thousand digits, so that the cost would be high. A much cheaper way to convey the same information would be to transmit instructions for calculating the tables from the underlying

trigonometric formulas, such as Euler's equation $e^{ix} = \cos x + i \sin x$. Such a message could be relatively brief, yet inherent in it is all the information contained in even the largest tables.

Suppose, on the other hand, your friend is interested not in trigonometry but in baseball. He would like to know the scores of all the major-league games played since he left the earth some thousands of years before. In this case it is most unlikely that a formula could be found for compressing the information into a short message; in such a series of numbers each digit is essentially an independent item of information, and it cannot be predicted from its neighbors or from some underlying rule. There is no alternative to transmitting the entire list of scores.

In this pair of whimsical messages is the germ of a new definition of randomness. It is based on the observation that the information embodied in a random series of numbers cannot be "compressed," or reduced to a more compact form. In formulating the actual definition it is preferable to consider communication not with a distant friend but with a digital computer. The friend might have the wit to make inferences about numbers or to construct a series from partial information or from vague instructions. The computer does not have that capacity, and for our purposes that deficiency is an advantage. Instructions given the computer must be complete and explicit, and they must enable it to proceed step by step without requiring that it comprehend the result of any part of the operations it performs. Such a program of instructions is an algorithm. It can demand any finite number of mechanical manipulations of numbers, but it cannot ask for judgments about their meaning.

The definition also requires that we be able to measure the information content of a message in some more precise way than by the cost of sending it as a telegram. The fundamental unit of information is the "bit," defined as the smallest item of information capable of indicating a choice between two equally likely things. In binary notation one bit is equivalent to one digit, either a 0 or a 1.

We are now able to describe more precisely the differences between the two series of digits presented at the beginning of this article:

01010101010101010101

01101100110111100010

The first could be specified to a computer by a very simple algorithm, such as "Print 01 ten times." If the series were extended according to the same

rule, the algorithm would have to be only slightly larger; it might be made to read, for example, "Print 01 a million times." The number of bits in such an algorithm is a small fraction of the number of bits in the series it specifies, and as the series grows larger the size of the program increases at a much slower rate.

For the second series of digits there is no corresponding shortcut. The most economical way to express the series is to write it out in full, and the shortest algorithm for introducing the series into a computer would be "Print 01101100110111100010." If the series were much larger (but still apparently patternless), the algorithm would have to be expanded to the corresponding size. This "incompressibility" is a property of all random numbers; indeed, we can proceed directly to define randomness in terms of incompressibility: A series of numbers is random if the smallest algorithm capable of specifying it to a computer has about the same number of bits of information as the series itself.

This definition was independently proposed about 1965 by A. N. Kolmogorov of the Academy of Science of the U.S.S.R. and by me, when I was an undergraduate at the City College of the City University of New York. Both Kolmogorov and I were then unaware of related proposals made in 1960 by Ray J. Solomonoff of the Zator Company in an endeavor to measure the simplicity of scientific theories. During the past decade we and others have continued to explore the meaning of randomness. The original formulations have been improved and the feasibility of the approach has been amply confirmed.

Model of Inductive Method

The algorithmic definition of randomness provides a new foundation for the theory of probability. By no means does it supersede classical probability theory, which is based on an ensemble of possibilities, each of which is assigned a probability. Rather, the algorithmic approach complements the ensemble method by giving precise meaning to concepts that had been intuitively appealing but that could not be formally adopted.

The ensemble theory of probability, which originated in the 17th century, remains today of great practical importance. It is the foundation of statistics, and it is applied to a wide range of problems in science and engineering. The algorithmic theory also has important implications, but they are primar-

ily theoretical. The area of broadest interest is its amplification of Gödel's incompleteness theorem. Another application (which actually preceded the formulation of the theory itself) is in Solomonoff's model of scientific induction.

Solomonoff represented a scientist's observations as a series of binary digits. The scientist seeks to explain these observations through a theory, which can be regarded as an algorithm capable of generating the series and extending it, that is, predicting future observations. For any given series of observations there are always several competing theories, and the scientist must choose among them. The model demands that the smallest algorithm, the one consisting of the fewest bits, be selected. Stated another way, this rule is the familiar formulation of Occam's razor: Given differing theories of apparently equal merit, the simplest is to be preferred.

Thus in the Solomonoff model a theory that enables one to understand a series of observations is seen as a small computer program that reproduces the observations and makes predictions about possible future observations. The smaller the program, the more comprehensive the theory and the greater the degree of understanding. Observations that are random cannot be reproduced by a small program and therefore cannot be explained by a theory. In addition the future behavior of a random system cannot be predicted. For random data the most compact way for the scientist to communicate his observations is for him to publish them in their entirety.

Defining randomness or the simplicity of theories through the capabilities of the digital computer would seem to introduce a spurious element into these essentially abstract notions: the peculiarities of the particular computing machine employed. Different machines communicate through different computer languages, and a set of instructions expressed in one of those languages might require more or fewer bits when the instructions are translated into another language. Actually, however, the choice of computer matters very little. The problem can be avoided entirely simply by insisting that the randomness of all numbers be tested on the same machine. Even when different machines are employed, the idiosyncrasies of various languages can readily be compensated for. Suppose, for example, someone has a program written in English and wishes to utilize it with a computer that reads only French. Instead of translating the algorithm itself he could preface the program with a complete English course written in French. Another mathematician with a French program and an English machine would follow the opposite procedure. In this way only a fixed number of bits need be added to the program,

and that number grows less significant as the size of the series specified by the program increases. In practice a device called a compiler often makes it possible to ignore the differences between languages when one is addressing a computer.

Since the choice of a particular machine is largely irrelevant, we can choose for our calculations an ideal computer. It is assumed to have unlimited storage capacity and unlimited time to complete its calculations. Input to and output from the machine are both in the form of binary digits. The machine begins to operate as soon as the program is given it, and it continues until it has finished printing the binary series that is the result. The machine then halts. Unless an error is made in the program, the computer will produce exactly one output for any given program.

Minimal Programs and Complexity

Any specified series of numbers can be generated by an infinite number of algorithms. Consider, for example, the three-digit decimal series 123. It could be produced by an algorithm such as "Subtract 1 from 124 and print the result," or "Subtract 2 from 125 and print the result," or an infinity of other programs formed on the same model. The programs of greatest interest, however, are the smallest ones that will yield a given numerical series. The smallest programs are called minimal programs; for a given series there may be only one minimal program or there may be many.

Any minimal program is necessarily random, whether or not the series it generates is random. This conclusion is a direct result of the way we have defined randomness. Consider the program P, which is a minimal program for the series of digits S. If we assume that P is not random, then by definition there must be another program, P', substantially smaller than P that will generate it. We can then produce S by the following algorithm: "From P' calculate P, then from P calculate S." This program is only a few bits longer than P', and thus it must be substantially shorter than P. P is therefore not a minimal program.

The minimal program is closely related to another fundamental concept in the algorithmic theory of randomness: the concept of complexity. The complexity of a series of digits is the number of bits that must be put into a computing machine in order to obtain the original series as output. The complexity is therefore equal to the size in bits of the minimal programs of

the series. Having introduced this concept, we can now restate our definition of randomness in more rigorous terms: A random series of digits is one whose complexity is approximately equal to its size in bits.

The notion of complexity serves not only to define randomness but also to measure it. Given several series of numbers each having n digits, it is theoretically possible to identify all those of complexity $n-1$, $n-10$, $n-100$ and so forth and thereby to rank the series in decreasing order of randomness. The exact value of complexity below which a series is no longer considered random remains somewhat arbitrary. The value ought to be set low enough for numbers with obviously random properties not to be excluded and high enough for numbers with a conspicuous pattern to be disqualified, but to set a particular numerical value is to judge what degree of randomness constitutes actual randomness. It is this uncertainty that is reflected in the qualified statement that the complexity of a random series is *approximately* equal to the size of the series.

Properties of Random Numbers

The methods of the algorithmic theory of probability can illuminate many of the properties of both random and nonrandom numbers. The frequency distribution of digits in a series, for example, can be shown to have an important influence on the randomness of the series. Simple inspection suggests that a series consisting entirely of either 0's or 1's is far from random, and the algorithmic approach confirms that conclusion. If such a series is n digits long, its complexity is approximately equal to the logarithm to the base 2 of n. (The exact value depends on the machine language employed.) The series can be produced by a simple algorithm such as "Print 0 n times," in which virtually all the information needed is contained in the binary numeral for n. The size of this number is about $\log_2 n$ bits. Since for even a moderately long series the logarithm of n is much smaller than n itself, such numbers are of low complexity; their intuitively perceived pattern is mathematically confirmed.

Another binary series that can be profitably analyzed in this way is one where 0's and 1's are present with relative frequencies of three-fourths and one-fourth. If the series is of size n, it can be demonstrated that its complexity is no greater than four-fifths n, that is, a program that will produce the series can be written in $4n/5$ bits. This maximum applies regardless of the

sequence of the digits, so that no series with such a frequency distribution can be considered very random. In fact, it can be proved that in any long binary series that is random the relative frequencies of 0's and 1's must be very close to one-half. (In a random decimal series the relative frequency of each digit is, of course, one-tenth.)

Numbers having a nonrandom frequency distribution are exceptional. Of all the possible n-digit binary numbers there is only one, for example, that consists entirely of 0's and only one that is all 1's. All the rest are less orderly, and the great majority must, by any reasonable standard, be called random. To choose an arbitrary limit, we can calculate the fraction of all n-digit binary numbers that have a complexity of less than $n - 10$. There are 2^1 programs one digit long that might generate an n-digit series; there are 2^2 programs two digits long that could yield such a series, 2^3 programs three digits long and so forth, up to the longest programs permitted within the allowed complexity; of these there are 2^{n-11}. The sum of this series $(2^1 + 2^2 + \cdots + 2^{n-11})$ is equal to $2^{n-10} - 2$. Hence there are fewer than 2^{n-10} programs of size less than $n-10$, and since each of these programs can specify no more than one series of digits, fewer than 2^{n-10} of the 2^n numbers have a complexity less than $n - 10$. Since $2^{n-10}/2^n = 1/1{,}024$, it follows that of all the n-digit binary numbers only about one in 1,000 have a complexity less than $n-10$. In other words, only about one series in 1,000 can be compressed into a computer program more than 10 digits smaller than itself.

A necessary corollary of this calculation is that more than 999 of every 1,000 n-digit binary numbers have a complexity equal to or greater than $n - 10$. If that degree of complexity can be taken as an appropriate test of randomness, then almost all n-digit numbers are in fact random. If a fair coin is tossed n times, the probability is greater than .999 that the result will be random to this extent. It would therefore seem easy to exhibit a specimen of a long series of random digits; actually it is impossible to do so.

Formal Systems

It can readily be shown that a specific series of digits is not random; it is sufficient to find a program that will generate the series and that is substantially smaller than the series itself. The program need not be a minimal program for the series; it need only be a small one. To demonstrate that a particular series of digits is random, on the other hand, one must prove that no small

program for calculating it exists.

It is in the realm of mathematical proof that Gödel's incompleteness theorem is such a conspicuous landmark; my version of the theorem predicts that the required proof of randomness cannot be found. The consequences of this fact are just as interesting for what they reveal about Gödel's theorem as they are for what they indicate about the nature of random numbers.

Gödel's theorem represents the resolution of a controversy that preoccupied mathematicians during the early years of the 20th century. The question at issue was: "What constitutes a valid proof in mathematics and how is such a proof to be recognized?" David Hilbert had attempted to resolve the controversy by devising an artificial language in which valid proofs could be found mechanically, without any need for human insight or judgement. Gödel showed that there is no such perfect language.

Hilbert established a finite alphabet of symbols, an unambiguous grammar specifying how a meaningful statement could be formed, a finite list of axioms, or initial assumptions, and a finite list of rules of inference for deducing theorems from the axioms or from other theorems. Such a language, with its rules, is called a formal system.

A formal system is defined so precisely that a proof can be evaluated by a recursive procedure involving only simple logical and arithmetical manipulations. In other words, in the formal system there is an algorithm for testing the validity of proofs. Today, although not in Hilbert's time, the algorithm could be executed on a digital computer and the machine could be asked to "judge" the merits of the proof.

Because of Hilbert's requirement that a formal system have a proof-checking algorithm, it is possible in theory to list one by one all the theorems that can be proved in a particular system. One first lists in alphabetical order all sequences of symbols one character long and applies the proof-testing algorithm to each of them, thereby finding all theorems (if any) whose proofs consist of a single character. One then tests all the two-character sequences of symbols, and so on. In this way all potential proofs can be checked, and eventually all theorems can be discovered in order of the size of their proofs. (The method is, of course, only a theoretical one; the procedure is too lengthy to be practical.)

to unusual mathematical propositions that were not likely to be of interest in practice, algorithmic information theory has shown that incompleteness and randomness are natural and pervasive. This suggests to me that the possibility of searching for new axioms applying to the whole numbers should perhaps be taken more seriously.

Indeed, the fact that many mathematical problems have remained unsolved for hundreds and even thousands of years tends to support my contention. Mathematicians steadfastly assume that the failure to solve these problems lies strictly within themselves, but could the fault not lie in the incompleteness of their axioms? For example, the question of whether there are any perfect odd numbers has defied an answer since the time of the ancient Greeks. (A perfect number is a number that is exactly the sum of its divisors, excluding itself. Hence 6 is a perfect number, since 6 equals 1 plus 2 plus 3.) Could it be that the statement "There are no odd perfect numbers" is unprovable? If it is, perhaps mathematicians had better accept it as an axiom.

This may seem like a ridiculous suggestion to most mathematicians, but to a physicist or a biologist it may not seem so absurd. To those who work in the empirical sciences the usefulness of a hypothesis, and not necessarily its "self-evident truth," is the key criterion by which to judge whether it should be regarded as the basis for a theory. If there are many conjectures that can be settled by invoking a hypothesis, empirical scientists take the hypothesis seriously. (The nonexistence of odd perfect numbers does not appear to have significant implications and would therefore not be a useful axiom by this criterion.)

Actually in a few cases mathematicians have already taken unproved but useful conjectures as a basis for their work. The so-called Riemann hypothesis, for instance, is often accepted as being true, even though it has never been proved, because many other important theorems are based on it. Moreover, the hypothesis has been tested empirically by means of the most powerful computers, and none has come up with a single counterexample. Indeed, computer programs (which, as I have indicated, are equivalent to mathematical statements) are also tested in this way—by verifying a number of test cases rather than by rigorous mathematical proof.