Information-Theoretic Computational Complexity
Invited Paper

IEEE Transactions on Information Theory IT-20 (1974), pp. 10-15

Gregory J. Chaitin

Abstract

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 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 possible 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.

(*Footnote*: 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.)

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 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. (See the Appendix).

1. In a formal system with n bits of axioms it is impossible to prove that 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. (In an important paper [14], Eigen studies these questions from the point of view of thermodynamics and biochemistry.) 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. (Chandrasekaran and Reeker [15] discuss the relevance of complexity to artificial intelligence.)

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. (See [11], [12] for different approaches.)

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

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, ...). The length of a binary string s is denoted lg(s). Thus if s is considered to be a natural number, then lg(s) = $\lfloor \log_2 (s+1) \rfloor$. Here $\lfloor x \rfloor$ is the greatest integer $\leqslant x$.

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, $I_U(s) \leqslant I_C(s) + c$, 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 $U(0^i1p) = C_i(p)$, 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 $F(a, h) \subseteq F(a, h+1)$. 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 $\leqslant h$ 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 $I(s) \leqslant \lg(s) + c$ 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, $I(s) \leqslant I_C(s) + c = \lg(s) + c$.

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 ``$I(s) \geqslant n$'' is in $F(a)$ only if it is true, i.e., only if $I(s) \geqslant n$. Then a proposition of the form ``$I(s) \geqslant n$'' is in $F(a)$ only if $n \leqslant \lg(a) + c$, 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 $I(s_k) \leqslant \lg(a) + k + c'$, 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 $\leqslant \lg(a) + 2c'$, 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 $I(s_k) \leqslant \lg(a) + k + c'$. 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 ``$I(s) \geqslant n$'' 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

$$I(s_k) = I(C(0^k 1a))$$
$$\leqslant I_C(C(0^k 1a)) + c''$$
$$\leqslant \lg(0^k 1a) + c'' = \lg(a) + k + (c'' + 1) = \lg(a) + k + c'.$$

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 ``$I(s) \geqslant n$."

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 $\geqslant n$. This completes our sketch of how all true propositions of the form ``$I(s) = k$" with $k < n$ and of the form ``$I(s) \geqslant n$" 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 $\leqslant n$, i.e.,

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

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

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

$$I(f(n)) \leqslant I_C(f(n)) + c \leqslant \lg(n) + c = \lfloor \log_2 (n+1) \rfloor + c < n$$

for all sufficiently great values of n. Thus $B(n) \geqslant f(n)$ 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 $\leqslant B(n)$, 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 $I(s_n) \leqslant n + c$, 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 $\leqslant n$ such that $U(p_0) = B(n)$, it follows that

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

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 $\lg(a) \leqslant U(p)$, 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 $k \leqslant n + c$, 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 F^1, A_n 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 $\leqslant \lg(A_n) + c = n + c$, and $B(n+c)$ is $\geqslant$ 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.

Manuscript received January 29, 1973; revised July 18, 1973. This paper was presented at the IEEE International Congress of Information Theory, Ashkelon, Israel, June 1973.

The author is at Mario Bravo 249, Buenos Aires, Argentina.