# INFORMATION-THEORETIC LIMITATIONS OF FORMAL SYSTEMS

**Journal of the ACM 21 (1974), pp. 403–424**

**Gregory J. Chaitin**[1]
*Buenos Aires, Argentina*

## Abstract

*An attempt is made to apply information-theoretic computational complexity to metamathematics. The paper studies the number of bits of instructions that must be a given to a computer for it to perform finite and infinite tasks, and also the amount of time that it takes the computer to perform these tasks. This is applied to measuring the difficulty of proving a given set of theorems, in terms of the number of bits of axioms that are assumed, and the size of the proofs needed to deduce the theorems from the axioms.*

**Key Words and Phrases:**

complexity of sets, computational complexity, difficulty of theorem-proving, entropy of sets, formal systems, Gödel's incompleteness theorem, halting problem, information content of sets, information content of axioms, information theory, information time trade-offs, metamathematics, random strings, recursive functions, recursively enumerable sets, size of proofs, universal computers

**CR Categories:**

5.21, 5.25, 5.27, 5.6

# 1. Introduction

This paper attempts to study information-theoretic aspects of computation in a very general setting. It is concerned with the information that must be supplied to a computer for it to carry out finite or infinite computational tasks, and also with the time it takes the computer to do this. These questions, which have come to be grouped under the heading of abstract computational complexity, are considered to be of interest in themselves. However, the motivation for this investigation is primarily its metamathematical applications.

Computational complexity differs from recursive function theory in that, instead of just asking whether it is possible to compute something, one asks exactly how much effort is needed to do this. Similarly, instead of the usual metamathematical approach, we propose to measure the difficulty of proving something. How many bits of axioms are needed



An early version of this paper was presented at the Courant Institute Computational Complexity Symposium, New York, October 1971. [28] includes a nontechnical exposition of some results of this paper. [1] and [2] announce related results.

Author's address: Rivadavia 3580, Dpto. 10A, Buenos Aires, Argentina.

to be able to obtain a set of theorems? How long are the proofs needed to demonstrate them? What is the trade-off between how much is assumed and the size of the proofs?

We consider the axioms of a formal system to be a program for listing the set of theorems, and the time at which a theorem is written out to be the length of its proof.

We believe that this approach to metamathematics may yield valuable dividends. Mathematicians were at first greatly shocked at, and then ignored almost completely, Gödel's announcement that no set of axioms for number theory is complete. It wasn't clear what, in practice, was the significance of Gödel's theorem, how it should affect the everyday activities of mathematicians. Perhaps this was because the unprovable propositions appeared to be very pathological singular points.[2,3]

The approach of this paper, in contrast, is to measure the power of a set of axioms, to measure the information that it contains. We shall see that there are circumstances in which one only gets out of a set of axioms what one puts in, and in which it is possible to reason in the following manner. If a set of theorems constitutes $t$ bits of information, and a set of axioms contains less than $t$ bits of information, then it is impossible to deduce these theorems from these axioms.

We consider that this paper is only a first step in the direction of such an approach to metamathematics;[4] a great deal of work remains to be done to clarify these matters. Nevertheless, we would like to sketch here the conclusions which we have tentatively drawn.[5]

---

[2]In [3] and [4] von Neumann analyzes the effect of Gödel's theorem upon mathematicians. Weyl's reaction to Gödel's theorem is quoted by Bell [5]. The original source is [6]. See also Weyl's discussion [7] of Gödel's views regarding his incompleteness theorem.

[3]For nontechnical expositions of Gödel's incompleteness theorem, see [8, 9, 10, Sec. 1, pp. xv-xviii, 11, and 12]. [28] contains a nontechnical exposition of an incompleteness theorem analogous to Berry's paradox that is Theorem 4.1 of this paper.

[4][13–16] are related in approach to this paper. [13, 15, and 16] are concerned with measuring the size of proofs and the effect of varying the axioms upon their size. In [14] Cohen "measures the strength of a [formal] system by the ordinals which can be handled in the system."

[5]The analysis that follows of the possible significance of the results of this paper has been influenced by [17 and 18], in addition to the references cited in Footnote

After empirically exploring, in the tradition of Euler and Gauss, the properties of the natural numbers one may discover interesting regularities. One then has two options. The first is to accept the conjectures one has formulated on the basis of their empirical corroboration, as an experimental scientist might do. In this way one may have a great many laws to remember, but will not have to bother to deduce them from other principles. The other option is to try to find a theory for one's observations, or to see if they follow from existing theory. In this case it may be possible to reduce a great many observations into a few general principles from which they can be deduced. But there is a cost: one can now only arrive at the regularities one observed by means of long demonstrations.

Why use formal systems, instead of proceeding empirically? First of all, if the empirically derived conjectures aren't independent facts, reducing them to a few common principles allows one to have to remember less assumptions, and this is easier to do, and is much safer, as one is assuming less. The cost is, of course, the size of the proofs.

What attitude, then, does this suggest toward Gödel's theorem that any formalization of number theory is incomplete? It tends to provide theoretical justification for the attitude that number theorists have in fact adopted when they extensively utilize in their work hypotheses such as that of Riemann concerning the zeta function. Gödel's theorem does not mean that mathematicians must give up hope of understanding the properties of the natural numbers; it merely means that one may have to adopt new axioms as one seeks to order and interrelate, to organize and comprehend, ever more extensive mathematical observations. I.e. the mathematician shouldn't be more upset than the physicist when he needs to assume a new axiom; nor should he be too horrified when an axiom must be abandoned because it is found that it contradicts previously existing theory, or because it predicts properties of the natural numbers that are not corroborated empirically. In a word, we propose that there may be theoretical justification for regarding number theory somewhat more like a dynamic empirical science than as a closed static body of theory.

This paper grew out of work on the concept of an individual random,

2. Incidentally, it is interesting to examine [19, p. 112] in the light of this analysis.

patternless, chaotic, unpredictable string of bits. This concept has been rigorously defined in several ways, and the properties of these random strings have been studied by several authors (see, for example, [20–28]). Most strings are random; they have no special distinguishing features; they are typical and hard to tell apart. But can it be proved that a particular string is random? The answer is that about $n$ bits of axioms are needed to be able to prove that a particular $n$-bit string is random.

More precisely, the train of thought was as follows. The entropy, or information content, or complexity, of a string is defined to be the number of bits needed to specify it so effectively that it can be constructed. A random $n$-bit string is about $n$ bits of information, i.e. has complexity/entropy/information content $\approx n$; there is essentially nothing better to do if one wishes to specify such a string than just show it directly. But the string consisting of 1,000,000 repetitions of the 6-bit pattern 000101 has far less than 6,000,000 bits of complexity. We have just specified it using far fewer bits.

What if one wishes to be able to determine each string of complexity $\leq n$ and its complexity? It turns out that this requires $n + O(1)$ bits of axioms; at least $n - c$ bits are necessary (Theorem 4.1), and $n + c$ bits are sufficient (Theorem 4.3). But the proofs will be enormously long unless one essentially directly takes as axioms all the theorems that one wishes to prove, and in that case there will be an enormously great number of bits of axioms (Theorem 7.6(c)).

Another theme of this paper arises from the following metamathematical considerations, which are well known (see, for example, [29]). In a formal system without a decision method, it is impossible to bound the size of a proof of a theorem by a recursive function of the number of characters in the statement of the theorem. For if there were such a function $f$, one could decide whether or not an arbitrary proposition $p$ is a theorem, by merely checking if a proof for it appears among the finitely many possible proofs of size bounded by $f$ of the number of characters in $p$.

Thus, in a formal system having no decision method, there are very profound theorems, theorems that have short statements, but need immensely long proofs. In Section 10 we study the function $e(n)$, necessarily nonrecursive, defined to be the least $s$ such that all theorems of the formal system with $\leq n$ characters have proofs of size $\leq s$.

To close this introduction, we would like to mention without proof an example that shows particularly clearly the relationship between the number of bits of axioms that are assumed and what can be deduced. This example is based on the work of M. Davis, Ju. V. Matisjasevič, H. Putnam, and J. Robinson that settled Hilbert's tenth problem (cf. [30]). There is a polynomial $P$ in $k+2$ variables with integer coefficients that has the following property. Consider the infinite string whose $i$th bit is 1 or 0 depending on whether or not the set

$$S_i = \{n \in N | \exists x_1, \ldots, x_k \in N \, P(i, n, x_1, \ldots, x_k) = 0\}$$

is infinite. Here $N$ denotes the natural numbers. This infinite binary sequence is random, i.e. the complexity of an initial segment is asymptotic to its length. What is the number of bits of axioms that is needed to be able to prove for each natural number $i < n$ whether or not the set $S_i$ is infinite? By using the methods of Section 4, it is easy to see that the number of bits of axioms that is needed is asymptotic to $n$.

# 2. Definitions Related to Computers and Complexity

This paper is concerned with measuring the difficulty of computing finite and infinite sets of binary strings. The binary strings are considered to be ordered in the following fashion: $\Lambda$, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, ... In order to be able to also study the difficulty of computing finite or infinite sets of natural numbers, we consider each binary string to simultaneously be a natural number: the $n$th binary string corresponds to the natural number $n$. Ordinal numbers are considered to start with 0, not 1. For example, we speak of the 0th string of length $n$.

In order to be able to study the difficulty of computing finite and infinite sets of mathematical propositions, we also consider that each binary string is simultaneously a proposition. Propositions use a finite alphabet of characters which we suppose includes all the usual mathematical symbols. We consider the $n$th binary string to correspond to the $n$th proposition, where the propositions are in lexicographical order defined by an arbitrary ordering of the symbols of their alphabet.

Henceforth, we say "string" instead of "binary string," it being understood that this refers to a binary string. It should be clear from the context whether we are considering something to be a string, a natural number, or a proposition.

Operations with strings include exponentiation: $0^k$ and $1^k$ denote the string of $k$ 0's and $k$ 1's, respectively. $\lg(s)$ denotes the length of a string $s$. Note that the length $\lg(n)$ of a natural number $n$ is therefore $\lfloor \log_2(n+1) \rfloor$. The maximum element of a finite set of strings $S$ is denoted by $\max S$, and we stipulate that $\max \emptyset = 0$. $\#(S)$ denotes the number of elements in a finite set $S$.

We use these notational conventions in a somewhat tricky way to indicate how to compactly code several pieces of information into a single string. Two coding techniques are used.

(a) Consider two natural numbers $n$ and $k$ such that $0 \leq k < 2^n$. We code $n$ and $k$ into the string $s = 0^n + k$, i.e. the $k$th string of length $n$. Given the string $s$, one recovers $n$ and $k$ as follows: $n = \lg(s)$, $k = s - 0^{\lg(s)}$. This technique is used in the proofs of Theorems 4.3, 6.1, 7.4, and 10.1. In three of these proofs $k$ is $\#(S)$, where $S$ is a subset of the strings having length $< n$; $n$ and $\#(S)$ are coded into the string $s = 0^n + \#(S)$. In the case of Theorem 6.1, $k$ is the number that corresponds to a string $s$ of length $< n$ (thus $0 \leq k < 2^n - 1$); $n$ and $s$ are coded into the string $s' = 0^n + s$.

(b) Consider a string $p$ and a natural number $k$. We code $p$ and $k$ into the string $s = 0^{\lg(k)}1kp$, i.e. the string consisting of $\lg(k)$ 0's followed by a 1 followed by the $k$th string followed by the string $p$. The length of the initial run of 0's is the same as the length of the $k$th string and is used to separate $kp$ in two and recover $k$ and $p$ from $s$. Note that $\lg(s) = \lg(p) + 2\lg(k) + 1$. This technique is used in the proof of Theorem 10.4. The proof of Theorem 4.1 uses a simpler technique: $p$ and $k$ are coded into the string $s = 0^k1p$. But this coding is less economical, for $\lg(s) = \lg(p) + k + 1$.

We use an extremely general definition of computer; this has the advantage that if one can show that something is difficult to compute

using any such computer, this will be a very strong result. A computer is defined by indicating whether it has halted and what it has output, as a function of its program and the time. The formal definition of a computer $C$ is an ordered pair $\langle C, H_C \rangle$ consisting of two total recursive functions

$$C : X^* \times N \to \{S \in 2^{X^*} | S \text{ is finite}\}$$

and $H_C : X^* \times N \to X$. Here $X = \{0, 1\}$, $X^*$ is the set of all strings, and $N$ is the set of all natural numbers. It is assumed that the functions $C$ and $H_C$ have the following two properties:

(a) $C(p, t) \subset C(p, t + 1)$, and

(b) if $H_C(p, t) = 1$, then $H_C(p, t + 1) = 1$ and $C(p, t) = C(p, t + 1)$.

$C(p, t)$ is the finite set of strings output by the computer $C$ up to time $t$ when its program is $p$. If $H_C(p, t) = 1$ the computer $C$ is halted at time $t$ when its program is $p$. If $H_C(p, t) = 0$ the computer $C$ isn't halted at time $t$ when its program is $p$. Henceforth whether $H_C(p, t) = 1$ or 0 will be indicated by stating that "$C(p, t)$ is halted" or that "$C(p, t)$ isn't halted." Property (a) states that $C(p, t)$ is the cumulative output, and property (b) states that a computer that is halted remains halted and never outputs anything else.

$C(p)$, the output of the computation that $C$ performs when it is given the program $p$, is defined to be $\bigcup_t C(p, t)$. It is said that "$C(p)$ halts" iff there is a $t$ such that $C(p, t)$ is halted. Furthermore, if $C(p)$ halts, the time at which it halts is defined to be the least $t$ such that $C(p, t)$ is halted. We say that the program $p$ calculates the finite set $S$ when run on $C$ if $C(p) = S$ and halts. We say that the program $p$ enumerates the finite or infinite set $S$ when run on $C$ if $C(p) = S$.

We now define a class of computers that are especially suitable to use for measuring the information needed to specify a computation. A computer $U$ is said to be universal if it has the following property. For any computer $C$, there must be a natural number, denoted $\text{sim}(C)$ (the cost of simulating $C$), such that the following holds. For any program $p$, there exists a program $p'$ such that: $\lg(p') \leq \lg(p) + \text{sim}(C)$, $U(p')$ halts iff $C(p)$ halts, and $U(p') = C(p)$.

The idea of this definition is as follows. The universal computers are information-theoretically the most economical ones; their programs

are shortest. More precisely, a universal computer $U$ is able to simulate any other computer, and the program $p'$ for $U$ that simulates the program $p$ for $C$ need not be much longer than $p$. If there are instructions of length $n$ for computing something using the computer $C$, then there are instructions of length $\leq n + \mathrm{sim}(C)$ for carrying out the same computation using $U$; i.e. at most $\mathrm{sim}(C)$ bits must be added to the length of the instructions, to indicate the computer that is to be simulated. Note that we do not assume that there is an effective procedure for obtaining $p'$ given $C$ and $p$. We have no need for the concept of an effectively universal computer in this paper. Nevertheless, the most natural examples of universal computers are effectively universal. See the Appendix for examples of universal computers.

We shall suppose that a particular universal computer $U$ has somehow been chosen, and shall use it as our standard computer for measuring the information needed to specify a computation. The choice of $U$ corresponds to the choice of the standard of measurement.

We now define $I(S)$, the information needed to calculate the finite set $S$, and $I_e(S)$, the information needed to enumerate the finite or infinite set $S$.

$$I(S) = \min \lg(p)\,(U(p) = S \text{ and halts});$$

$$I_e(S) = \begin{cases} \min \lg(p)\,(U(p) = S), \\ \infty \text{ if there are no such } p. \end{cases}$$

We say that $I(S)$ is the complexity of the finite set $S$, and that $I_e(S)$ is the e-complexity (enumeration complexity) of the finite or infinite set $S$. Note that $I(S)$ is the number of bits in the shortest program for $U$ that calculates $S$, and $I_e(S)$ is the number of bits in the shortest program for $U$ that enumerates $S$. Also, $I_e(S)$, the e-complexity of a set $S$, is $\infty$ if $S$ isn't r.e. (recursively enumerable).

We say that a program $p$ such that $U(p) = S$ and halts is a description of $S$, and a program $p$ such that $U(p) = S$ is an e-description (enumeration description) of $S$. Moreover, if $U(p) = S$ and halts and $\lg(p) = I(S)$, then we say that $p$ is a minimal description of $S$. Likewise, if $U(p) = S$ and $\lg(p) = I_e(S)$, then we say that $p$ is a minimal e-description of $S$.

Finally, we define $I_e(f)$, the e-complexity of a partial function $f$. This is defined to be the e-complexity of the graph of $f$, i.e. the set of all ordered pairs of the form $(n, f(n))$. Here the ordered pair $(i, j)$ is defined to be the natural number $(2i+1)2^j - 1$; this is an effective 1–1 correspondence between the ordered pairs of natural numbers and the natural numbers. Note that $I_e(f) = \infty$ if $f$ isn't partial recursive.

Before considering basic properties of these concepts, we introduce an abbreviated notation. Instead of $I(\{s\})$ and $I_e(\{s\})$ we shall write $I(s)$ and $I_e(s)$; i.e. the complexity or e-complexity of a string is defined to be complexity or e-complexity of its singleton set.

We now present basic properties of these concepts. First of all, note that there are precisely $2^n$ programs of length $n$, and $2^{n+1} - 1$ programs of length $\leq n$. It follows that the number of different sets of complexity $n$ and the number of different sets of e-complexity $n$ are both $\leq 2^n$. Also, the number of different sets of complexity $\leq n$ and the number of different sets of e-complexity $\leq n$ are both $\leq 2^{n+1} - 1$; i.e. the number of different objects of complexity or e-complexity $\leq n$ is bounded by the number of different descriptions or e-descriptions of length $\leq n$, which is $2^{n+1} - 1$. Thus it might be said that almost all sets are arbitrarily complex.

It is immediate from the definition of complexity and of a universal computer that $I_e(C(p)) \leq \lg(p) + \mathrm{sim}(C)$, and $I(C(p)) \leq \lg(p) + \mathrm{sim}(C)$ if $C(p)$ halts. This is used often, and without explicit mention. The following theorem lists for reference other basic properties of complexity and e-complexity that are used in this paper.

**Theorem 2.1.**

(a) There is a $c$ such that for all strings $s$, $I(s) \leq \lg(s) + c$.

(b) There is a $c$ such that for all finite sets $S$, $I(S) \leq \max S + c$.

(c) For any computer $C$, there is a $c$ such that for all programs $p$, $I_e(C(p)) \leq I(p) + c$, and $I(C(p)) \leq I(p) + c$ if $C(p)$ halts.

*Proof.* (a) There is a computer $C$ such that $C(s) = \{s\}$ and halts for all programs $s$. Thus $I(s) \leq \lg(s) + \mathrm{sim}(C)$.

(b) There is a computer $C$ such that $C(p)$ halts for all programs $p$, and $n \in C(p)$ iff $n < \lg(p)$ and the $n$th bit of $p$ is a 1. Thus $I(S) \leq \max S + 1 + \mathrm{sim}(C)$.

(c) There is a computer $C'$ that does the following when it is given the program $s$. First $C'$ simulates running $s$ on $U$, i.e. it simulates $U(s)$. If and when $U(s)$ halts, $C'$ has determined the set calculated by $U$ when it is given the program $s$. If this isn't a singleton set, $C'$ halts. If it is a singleton set $\{p\}$, $C'$ then simulates running $p$ on $C$. As $C'$ determines the strings output by $C(p)$, it also outputs them. And $C'$ halts if $C$ halts during the simulated run.

In summary, $C'(s) = C(p)$ and halts iff $C(p)$ does, if $s$ is a description of the program $p$. Thus, if $s$ is a minimal description of the string $p$, then

$$\begin{cases} I_e(C(p)) = I_e(C'(s)) \leq \lg(s) + \mathrm{sim}(C') = I(p) + \mathrm{sim}(C'), \text{ and} \\ I(C(p)) = I(C'(s)) \leq \lg(s) + \mathrm{sim}(C') = I(p) + \mathrm{sim}(C') \text{ if } C(p) \text{ halts.} \end{cases}$$

Q.E.D.

It follows from Theorem 2.1(a) that all strings of length $n$ have complexity $\leq n + c$. In conjunction with the fact that $< 2^{n-k}$ strings are of complexity $< n - k$, this shows that the great majority of the strings of length $n$ are of complexity $\approx n$. These are the random strings of length $n$. By taking $C = U$ in Theorem 2.1(c), it follows that there is a $c$ such that for any minimal description $p$, $I(p) + c \geq I(U(p)) = \lg(p)$. Thus minimal descriptions are highly random strings. Likewise, minimal e-descriptions are highly random. This corresponds in information theory to the fact that the most informative messages are the most unexpected ones, the ones with least regularities and redundancies, and appear to be noise, not meaningful messages.

# 3. Definitions Related to Formal Systems

This paper deals with the information and time needed to carry out computations. However, we wish to apply these results to formal systems. This section explains how this is done.

The abstract definition used by Post that a formal system is an r.e. set of propositions is close to the viewpoint of this paper (see [31]).[6]

[6]For standard definitions of formal systems, see, for example, [32–34] and [10, p. 117].

However, we are not quite this unconcerned with the internal details of formal systems.

The historical motivation for formal systems was of course to construct deductive theories with completely objective, formal criteria for the validity of a demonstration. Thus, a fundamental characteristic of a formal system is an algorithm for checking the validity of proofs. From the existence of this proof verification algorithm, it follows that the set of all theorems that can be deduced from the axioms $p$ by means of the rules of inference by proofs $\leq t$ characters in length is given by a total recursive function $C$ of $p$ and $t$. To calculate $C(p,t)$ one applies the proof verification algorithm to each of the finitely many possible demonstrations having $\leq t$ characters.

These considerations motivate the following definition. The rules of inference of a class of formal systems is a total recursive function $C : X^* \times N \to \{S \in 2^{X^*} | S \text{ is finite}\}$ with the property that $C(p,t) \subset C(p,t+1)$. The value of $C(p,t)$ is the finite (possibly empty) set of the theorems that can be proven from the axioms $p$ by means of proofs $\leq t$ in size. Here $p$ is a string and $t$ is a natural number. $C(p) = \bigcup_t C(p,t)$ is the set of theorems that are consequences of the axioms $p$. The ordered pair $\langle C, p \rangle$, which implies both the choice of rules of inference and axioms, is a particular formal system.

Note that this definition is the same as the definition of a computer with the notion of "halting" omitted. Thus given any rules of inference, there is a computer that never halts whose output up to time $t$ consists precisely of those propositions that can be deduced by proofs of size $\leq t$ from the axioms the computer is given as its program. And given any computer, there are rules of inference such that the set of theorems that can be deduced by proofs of size $\leq t$ from the program, is precisely the set of strings output by the computer up to time $t$. For this reason we consider the following notions to be synonymous: "computer" and "rules of inference," "program" and "axioms," and "output up to time $t$" and "theorems with proofs of size $\leq t$."

The rules of inference that correspond to the universal computer $U$ are especially interesting, because they permit axioms to be very economical. When using the rules of inference $U$, the number of bits of axioms needed to deduce a given set of propositions is precisely the e-complexity of the set of propositions. If $n$ bit of axioms are needed to

obtain a set $T$ of theorems using the rules of inference $U$, then at least $n - \text{sim}(C)$ bits of axioms are needed to obtain them using the rules of inference $C$; i.e. if $C(a) = T$, then $\text{lg}(a) \geq I_e(T) - \text{sim}(C)$. Thus it could be said that $U$ is among the rules of inference that permit axioms to be most economical. In Section 4 we are interested exclusively in the number of bits needed to deduce certain sets of propositions, not in the size of the proofs. We shall therefore only consider the rules of inference $U$ in Section 4, i.e. formal systems of the form $\langle U, p\rangle$.

As a final comment regarding the rules of inference $U$, we would like to point out the interesting fact that if these rules of inference are used, then a minimal set of axioms for obtaining a given set of theorems must necessarily be random. This is just another way of saying that a minimal e-description is a highly random string, which was mentioned at the end of Section 2.

The following theorem also plays a role in the interpretation of our results in terms of formal systems.

**Theorem 3.1.**

Let $f$ be a recursive function, and $g$ be a recursive predicate.

(a) Let $C$ be a computer. There is a computer $C'$ that never halts such that $C'(p,t) = \{f(s) | s \in C(p,t)\ \&\ g(s)\}$ for all $p$ and $t$.

(b) There is a $c$ such that $I_e(\{f(s) | s \in S\ \&\ g(s)\}) \leq I_e(S) + c$ for all r.e. sets $S$.

*Proof.* (a) is immediate; (b) follows by taking $C = U$ in part (a). Q.E.D.

The following is an example of the use of Theorem 3.1. Suppose we wish to study the size of the proofs that "$n \in H$" in a formal system $\langle C, p\rangle$, where $n$ is a numeral for a natural number. If we have a result concerning the speed with which any computer can enumerate the set $H$, we apply this result to the computer $C'$ that has the property that $n \in C'(p,t)$ iff "$n \in H$"$\in C(p,t)$ for all $n$, $p$, and $t$. In this case the predicate $g$ selects those strings that are propositions of the form "$n \in H$," and the function $f$ transforms "$n \in H$" to $n$.

Here is another kind of example. Suppose there is a computer $C$ that enumerates a set $H$ very quickly. Then there is a computer $C'$

that enumerates propositions of the form "$n \in H$" just as quickly. In this case the predicate $g$ is taken to be always true, and the function $f$ transforms $n$ to "$n \in H$."

# 4. The Number of Bits of Axioms Needed to Determine the Complexity of Specific Strings

The set of all programs that halt when run on $U$ is r.e. Similarly, the set of all true propositions of the form "$I(s) \leq n$" where $s$ is a string and $n$ is a natural number, is an r.e. set. In other words, if a program halts, or if a string is of complexity less than or equal to $n$, one will eventually find this out. To do this one need only try on $U$ longer and longer test runs of more and more programs, and do this in a systematic way.

The problem is proving that a program doesn't halt, or that a string is of complexity greater than $n$. In this section we study how many bits of axioms are needed, and, as was pointed out in Section 3, it is sufficient to consider only the rules of inference $U$. We shall see that with $n$ bits of axioms it is impossible to prove that a particular string is of complexity greater than $n + c$, where $c$ doesn't depend on the particular axioms chosen (Theorem 4.1). It follows from this that if a formal system has $n$ bits of axioms, then there is a program of length $\leq n + c$ that doesn't halt, but the fact that this program doesn't halt can't be proven in this formal system (Theorem 4.2).

Afterward, we show that $n + c$ bits of axioms suffice to be able to determine each program of length not greater than $n$ that halts (Theorem 4.4), and thus to determine each string of complexity less than or equal to $n$, and its complexity (Theorem 4.3). Furthermore, the remaining strings must be of complexity greater than $n$.

Next, we construct an r.e. set of strings $P$ that has the property that infinitely many strings aren't in it, but a formal system with $n$ bits of axioms can't prove that a particular string isn't an element of $P$ if the length of this string is greater than $n + c$ (Theorem 4.5). It follows that $P$ is what Post called a simple set; that is, $P$ is r.e., and

its complement is infinite, but contains no infinite r.e. subset (see [31], Sec. 5, pp. 319–320). Moreover, $n + c$ bits suffice to determine each string of length not greater than $n$ that isn't an element of $P$.

Finally, we show that not only are $n$ bits of axioms insufficient to exhibit a string of complexity $> n+c$, they are also insufficient to exhibit (by means of an e-description) an r.e. set of e-complexity greater than $n + c$ (Theorem 4.6). This is because no set can be of e-complexity much greater than the complexity of one of its e-descriptions, and thus $I_e(U(s)) > k$ implies $I(s) > k - c$, where $c$ is a constant that doesn't depend on $s$.

Although these results clarify how many bits of axioms are needed to determine the complexity of individual strings, they raise several questions regarding the size of proofs.

$n + c$ bits of axioms suffice to determine each string of complexity $\leq n$ and its complexity, but the method used here to do this appears to be extremely slow, that is, the proofs appear to be extremely long. Is this necessarily the case? The answer is "yes," as is shown in Section 7.

We have pointed out that there is a formal system having as theorems all true propositions of the form "$U(p)$ halts." The size of the proof that $U(p)$ halts must grow faster than any recursive function $f$ of $\lg(p)$. For suppose that such a recursive bound on the length of proofs existed. Then all true propositions of the form "$U(p)$ doesn't halt" could be enumerated by checking to see if there is no proof that $U(p)$ halts of size $< f(\lg(p))$. This is impossible, by Theorem 4.2. The size of these proofs is studied in Section 10.

**Theorem 4.1.** (a) There is a $c$ such that for all programs $p$, if a proposition of the form "$I(s) > n$" ($s$ a string, $n$ a natural number) is in $U(p)$ only if $\mathrm{I}(s) > n$, then "$I(s) > n$" is in $U(p)$ only if $n < \lg(p)+c$.

In other words: (b) There is a $c$ such that for all formal systems $\langle U, p\rangle$, if "$I(s) > n$" is a theorem of $\langle U, p\rangle$ only if it is true, then "$I(s) > n$" is a theorem of $\langle U, p\rangle$ only if $n < \lg(p) + c$.

For any r.e. set of propositions $T$, one obtains the following from (a) by taking $p$ to be a minimal e-description of $T$: (c) If $T$ has the property that "$I(s) > n$" is in $T$ only if $I(s) > n$, then $T$ has the property that "$I(s) > n$" is in $T$ only if $n < I_e(T) + c$.

*Idea of Proof.* The following is essentially the classical Berry

paradox.[7] For each natural number $n$ greater than 1, consider "the least natural number that can't be defined in less than $N$ characters." Here $N$ denotes the numeral for the number $n$. This is a $\lfloor \log_{10} n \rfloor + c$ character phrase defining a number that supposedly needs at least $n$ characters to be defined. This is a paradox if $\lfloor \log_{10} n \rfloor + c < n$, which holds for all sufficiently great values of $n$.

The following is a sharper version of Berry's paradox. Consider: "the least natural number whose definition requires more characters than there are in this phrase." This $c$-character phrase defines a number that supposedly needs more than $c$ characters to be defined.

The following version is analogous to our proof. Consider this program: "Calculate the first string that can be proven in $\langle U, p \rangle$ to be of complexity greater than the number of bits in this program, where $p$ is the following string: ..." Here "first" refers to first in the recursive enumeration $U(p, t)$ $(t = 0, 1, 2, \ldots)$ of the theorems of the formal system $\langle U, p \rangle$. This program is only a constant number $c$ of bits longer than the number of bits in $p$. It is no longer a paradox; it shows that in $\langle U, p \rangle$ no string can be proven to be of complexity greater than $c + \lg(p) = c +$ the number of bits of axioms of $\langle U, p \rangle$.

*Proof.* Consider the computer $C$ that does the following when it is given the program $p'$. First, it solves the equation $p' = 0^k 1 p$. If this isn't possible (i.e. $p' = 0^k$), then $C$ halts without outputting anything. If this is possible, $C$ continues by simulating running the program $p$ on $U$. It generates $U(p)$ searching for a proposition of the form "$I(s) > n$" in which $s$ is a string, $n$ is a natural number, and $n \geq \lg(p') + k$. If and when it finds such a proposition "$I(s) > n$" in $U(p)$, it outputs $s$ and halts.

Suppose that $p$ satisfies the hypothesis of the theorem, i.e. "$I(s) > n$" is in $U(p)$ only if $I(s) > n$. Consider $C(0^{\mathrm{sim}(C)} 1 p)$. If $C(0^{\mathrm{sim}(C)} 1 p) = \{s\}$, then $I(s) \leq \lg(0^{\mathrm{sim}(C)} 1 p) + \mathrm{sim}(C) = \lg(p) + 2\,\mathrm{sim}(C) + 1$. But $C$ outputs $s$ and halts because it found the proposition "$I(s) > n$" in $U(p)$ and

$$n \geq \lg(p') + k = \lg(0^{\mathrm{sim}(C)} 1 p) + \mathrm{sim}(C) = \lg(p) + 2\,\mathrm{sim}(C) + 1.$$

[7] Although due to Berry, its importance was recognized by Russell and it was published by him [35, p. 153].

Thus, by the hypothesis of the theorem, $I(s) > n \geq \lg(p)+2\,\mathrm{sim}(C)+1$, which contradicts the upper bound on $I(s)$. Consequently, $C(0^{\mathrm{sim}(C)}1p)$ doesn't output anything (i.e. equals $\emptyset$), for there is no proposition "$I(s) > n$" in $U(p)$ with $n \geq \lg(p) + 2\,\mathrm{sim}(C) + 1$. The theorem is proved with $c = 2\,\mathrm{sim}(C) + 1$. Q.E.D.

**Definition 4.1.** $H = \{p|U(p) \text{ halts}\}$. ($H$ is r.e., as was pointed out in the first paragraph of this section.)

**Theorem 4.2.** There is a $c$ such that for all formal systems $\langle U, p\rangle$, if a proposition of the form "$s \in H$" or "$s \notin H$" ($s$ a string) is a theorem of $\langle U, p\rangle$ only if it is true, then there is a string $s$ of length $\leq \lg(p) + c$ such that neither "$s \in H$" nor "$s \notin H$" is a theorem of $\langle U, p\rangle$.

*Proof.* Consider the computer $C$ that does the following when it is given the program $p$. It simulates running $p$ on $U$, and as it generates $U(p)$, it checks each string in it to see if it is a proposition of the form "$s \in H$" or "$s \notin H$," where $s$ is a string. As soon as $C$ has determined in this way for each string $s$ of length less than or equal to some natural number $n$ whether or not "$s \in H$" or "$s \notin H$" is in $U(p)$, it does the following.

$C$ supposes that these propositions are true, and thus that it has determined the set $\{s \in H | \lg(s) \leq n\}$. Then it simulates running each of the programs in this set on $U$ until $U$ halts, and thus determines the set $S = \bigcup U(s)$ ($s \in H$ & $\lg(s) \leq n$). $C$ then outputs the proposition "$I(f) > n$," where $f$ is the first string not in $S$, and then continues generating $U(p)$ as was indicated in the first paragraph of this proof. Inasmuch as $f$ isn't output by any program of length $\leq n$ that halts, it must in fact be of complexity $> n$.

Thus, $C(p)$ enumerates true propositions of the form "$I(f) > n$" if $p$ satisfies the hypothesis of the theorem. Hence, by Theorem 4.1, "$I(f) > n$" is in $C(p)$ only if $n < I_e(C(p)) + c' \leq \lg(p) + \mathrm{sim}(C) + c'$. It is easy to see that the theorem is proved with $c = \mathrm{sim}(C) + c'$. Q.E.D.

**Theorem 4.3.** Consider the set $T_n$ consisting of all true propositions of the form "$I(s) = k$" ($s$ a string, $k$ a natural number $\leq n$) and all true propositions of the form "$I(s) > n$." $I_e(T_n) = n + O(1)$.

In other words, a formal system $\langle U, p'\rangle$ whose theorems consist precisely of all true propositions of the form "$I(s) = k$" with $k \leq n$, and all true propositions of the form "$I(s) > n$," requires $n + O(1)$ bits of axioms; i.e. $n - c$ bits are necessary and $n + c$ bits are sufficient to

obtain this set of theorems.

*Idea of Proof.* If one knows $n$ and how many programs of length $\leq n$ halt when run of $U$, then one can find them all, and see what they calculate. $n$ and this number $h$ can be coded into an $(n+1)$-bit string. In other words, the axiom of this formal system with theorem set $T_n$ is essentially "the number of programs of length $\leq n$ that halt when run on $U$ is $h$," where $n$ and $h$ are particular natural numbers. This axiom is $n + O(1)$ bits of information.

*Proof.* By Theorem 4.1, $I_e(T_n) \geq n - c$. It remains to show that $I_e(T_n) \leq n + c$.

Consider the computer $C$ that does the following when it is given the program $p$ of length $\geq 1$. It generates the r.e. set $H$ until it has found $p - 0^{\lg(p)}$ programs of length $\leq \lg(p) - 1$ that halt when run on $U$. If and when it has found this set $S$ of programs, it simulates running each program in $S$ on $U$ until it halts. $C$ then examines each string that is calculated by a program in $S$, and determines the length of the shortest program in $S$ that calculates it. If $p - 0^{\lg(p)} =$ the number $h$ of programs of length $\leq \lg(p) - 1$ that halt when run of $U$, then $C$ has determined each string of complexity $\leq \lg(p) - 1$ and its complexity. If $p - 0^{\lg(p)} < h$, then $C$'s estimates of the complexity of strings are too high. And if $p - 0^{\lg(p)} > h$, then $C$ never finishes generating $H$. Finally, $C$ outputs its estimates as propositions of the form "$I(s) = k$" with $k \leq \lg(p) - 1$, and as propositions of the form "$I(s) > k$" with $k = \lg(p) - 1$ indicating that all other strings are of complexity $> \lg(p) - 1$.

We now show how $C$ can be used to enumerate $T_n$ economically. Consider $h = \#(\{s \in H | \lg(s) \leq n\})$. As there are precisely $2^{n+1} - 1$ strings of length $\leq n$, $0 \leq h \leq 2^{n+1} - 1$. Let $p$ be $0^{n+1} + h$, that is, the $h$th string of length $n + 1$. Then $C(p) = T_n$, and thus $I_e(T_n) \leq \lg(p) + \mathrm{sim}(C) = n + 1 + \mathrm{sim}(C)$. Q.E.D.

**Theorem 4.4.** Let $T_n$ be the set of all true propositions of the form "$s \in H$" or "$s \notin H$" with $s$ a string of length $\leq n$. $I_e(T_n) = n + O(1)$.

In other words, a formal system $\langle U, p\rangle$ whose theorems consist precisely of all true propositions of the form "$s \in H$" or "$s \notin H$" with $\lg(s) \leq n$, requires $n + O(1)$ bits of axioms; i.e. $n - c$ bits are necessary and $n + c$ bits are sufficient to obtain this set of theorems.

*Proof.* Theorem 4.2 shows that $I_e(T_n) \geq n - c$. The proof that

$I_e(T_n) \leq n+c$ is obtained from the proof of Theorem 4.3 by simplifying the definition of the computer $C$ so that it outputs $T_n$, instead of, in effect, using $T_n$ to determine each string of complexity $\leq n$ and its complexity. Q.E.D.

**Definition 4.2.** $P = \{s|I(s) < \lg(s)\}$; i.e. $P$ contains each string $s$ whose complexity $I(s)$ is less than its length $\lg(s)$.

**Theorem 4.5.** (a) $P$ is r.e., i.e. there is a formal system with the property that "$s \in P$" is a theorem iff $s \in P$.

(b) $\overline{P}$ is infinite, because for each $n$ there is a string of length $n$ that isn't an element of $P$.

(c) There is a $c$ such that for all formal systems $\langle U, p\rangle$, if "$s \notin P$" is a theorem only if it is true, then "$s \notin P$" is a theorem only if $\lg(s) < \lg(p) + c$. Thus, by the definition of e-complexity, if an r.e. set $T$ of propositions has the property that "$s \notin P$" is in $T$ only if it is true, then "$s \notin P$" is in $T$ only if $\lg(s) < I_e(T) + c$.

(d) There is a $c$ such that for all r.e. sets of strings $S$, if $S$ contains no string in $P$ (i.e. $S \subset \overline{P}$), then $\lg(\max S) < I_e(S)+c$. Thus $\max S < 0^{I_e(S)+c} = 2^{I_e(S)+c} - 1$, and $\#(S) < 2^{I_e(S)+c}$.

(e) Let $T_n$ be the set of all true propositions of the form "$s \notin P$" with $\lg(s) \leq n$. $I_e(T_n) = n + O(1)$. In other words, a formal system $\langle U, p\rangle$ whose theorems consist precisely of all true propositions of the form "$s \notin P$" with $\lg(s) \leq n$, requires $n + O(1)$ bits of axioms; i.e. $n - c$ bits are necessary and $n + c$ bits are sufficient to obtain this set of theorems.

*Proof.* (a) This is an immediate consequence of the fact that the set of all true propositions of the form "$I(s) \leq n$" is r.e.

(b) We must show that for each $n$ there is a string of length $n$ whose complexity is greater than or equal to its length. There are $2^n$ strings of length $n$. As there are exactly $2^n - 1$ program of length $< n$, there are $< 2^n$ strings of complexity $< n$. Thus at least one string of length $n$ must be of complexity $\geq n$.

(c) Consider the computer $C$ that does the following when it is given the program $p$. It simulates running $p$ on $U$. As $C$ generates $U(p)$, it examines each string in it to see if it is a proposition of the form "$s \notin P$," where $s$ is a string of length $\geq 1$. If it is, $C$ outputs the proposition "$I(s) > n$" where $n = \lg(s) - 1$.

If $p$ satisfies the hypothesis, i.e. "$s \notin P$" is in $U(p)$ only if it is

true, then $C(p)$ enumerates true propositions of the form "$I(s) > n$" with $n = \lg(s) - 1$. It follows by Theorem 4.1 that $n$ must be $< I_e(C(p)) + c' \leq \lg(p) + \mathrm{sim}(C) + c'$. Thus $\lg(s) - 1 < \lg(p) + \mathrm{sim}(C) + c'$, and part (c) of the theorem is proved with $c = \mathrm{sim}(C) + c' + 1$.

(d) Consider the computer $C$ that does the following when it is given the program $p$. It simulates running $p$ on $U$. As $C$ generates $U(p)$, it takes each string $s$ in $U(p)$, and outputs the proposition "$s \notin P$."

Suppose $S$ contains no string in $P$. Let $p$ be a minimal e-description of $S$, i.e. $U(p) = S$ and $\lg(p) = I_e(S)$. Then $C(p)$ enumerates true propositions of the form "$s \notin P$" with $s \in S$. By part (c) of this theorem,

$$\lg(s) < I_e(C(p)) + c' \leq \lg(p) + \mathrm{sim}(C) + c' = I_e(S) + \mathrm{sim}(C) + c'.$$

Part (d) of the theorem is proved with $c = \mathrm{sim}(C) + c'$.

(e) That $I_e(T_n) \geq n - c$ follows from part (c) of this theorem. The proof that $I_e(T_n) \leq n + c$ is obtained by changing the definition of the computer $C$ in the proof of Theorem 4.3 in the following manner. After $C$ has determined each string of complexity $\leq n$ and its complexity, $C$ determines each string $s$ of complexity $\leq n$ whose complexity is greater than or equal to its length, and then $C$ outputs each such $s$ in a proposition of the form "$s \notin P$." Q.E.D.

**Theorem 4.6.** (a) There is a $c$ such that for all programs $p$, if a proposition of the form "$I_e(U(s)) > n$" ($s$ a string, $n$ a natural number) is in $U(p)$ only if $I_e(U(s)) > n$, then "$I_e(U(s)) > n$" is in $U(p)$ only if $n < \lg(p) + c$.

In other words: (b) There is a c such that for all formal systems $\langle U, p\rangle$, if "$I_e(U(s)) > n$" is a theorem of $\langle U, p\rangle$ only if it is true, then "$I_e(U(s)) > n$" is a theorem of $\langle U, p\rangle$ only if $n < \lg(p) + c$.

For any r.e. set of propositions $T$, one obtains the following from (a) by taking p to be a minimal e-description of $T$: (c) If $T$ has the property that "$I_e(U(s)) > n$" is in $T$ only if $I_e(U(s)) > n$, then $T$ has the property that "$I_e(U(s)) > n$" is in $T$ only if $n < I_e(T) + c$.

*Proof.* By Theorem 2.1(c), there is a $c'$ such that $I_e(U(s)) > n$ implies $I(s) > n - c'$.

Consider the computer $C$ that does the following when it is given the program $p$. It simulates running $p$ on $U$. As $C$ generates $U(p)$, it checks

each string in it to see if it is a proposition of the form "$I_e(U(s)) > n$" with $s$ a string and $n$ a natural number. Each time it finds such a proposition in which $n \geq c'$, $C$ outputs the proposition "$I(s) > m$" where $m = n - c' \geq 0$.

If $p$ satisfies the hypothesis of the theorem, then $C(p)$ enumerates true propositions of the form "$I(s) > m$." "$I(s) > m$" $(m = n - c' \geq 0)$ is in $C(p)$ iff "$I_e(U(s)) > n$" $(n \geq c')$ is in $U(p)$. By Theorem 4.1, "$I(s) > m$" is in $C(p)$ only if

$$m < I_e(C(p)) + c'' \leq \lg(p) + \mathrm{sim}(C) + c''.$$

Thus "$I_e(U(s)) > n$" $(n \geq c')$ is in $U(p)$ only if $n - c' < \lg(p) + \mathrm{sim}(C) + c''$. The theorem is proved with $c = \mathrm{sim}(C) + c'' + c'$. Q.E.D.

# 5. The Greatest Natural Number of Complexity $\leq$ N

The growth of $a(n)$, the greatest natural number of complexity $\leq n$ as a partial function of $n$, serves as a benchmark for measuring a number of computational phenomena. The general approach in Sections 6 to 10 will be to use a partial function of $n$ to measure some quantity of computational interest, and to compare the growth of this partial function as $n$ increases with that of $a(n)$.

We compare rates of growth in the following fashion.

**Definition 5.1.** We say that a partial function $f$ grows at least as quickly as another partial function $g$, written $f \succeq g$ or $g \preceq f$, when a shift of $f$ overbounds $g$. That is, when there is a $c$ such that for all $n$, if $g(n)$ is defined, then $f(n + c)$ is defined and $f(n + c) \geq g(n)$. Note that $f \preceq g$ and $g \preceq h$ implies $f \preceq h$.

**Definition 5.2.** We say that the partial functions $f$ and $g$ grow equally quickly, written $f \asymp g$, iff $f \preceq g$ and $g \preceq f$.

We now formally define $a(n)$, and list its basic properties for future reference.

**Definition 5.3.** $a(n) = \max k\ (I(k) \leq n)$. The maximum is taken over all natural numbers $k$ of complexity $\leq n$. If there are no such $k$, then $a(n)$ is undefined.

**Theorem 5.1.**

(a) If $a(n)$ is defined, then $I(a(n)) \leq n$.

(b) If $a(n)$ is defined, then $a(n+1)$ is defined and $a(n) \leq a(n+1)$.

(c) If $I(n) \leq m$, then $n \leq a(m)$.

(d) $n \leq a(I(n))$.

(e) If $a(m)$ is defined, then $n > a(m)$ implies $I(n) > m$.

(f) If $I(n) > i$ for all $n \geq m$, then $m > a(i)$ if $a(i)$ is defined.

(g) $I(a(n)) = n + O(1)$.

(h) There is a $c$ such that for all finite sets $S$ of strings, $\max S \leq a(I(S) + c)$.

(i) There is a $c$ such that for all finite sets $S$ of strings and all $n$, if $a(n)$ is defined and $a(n) \in S$, then $I(S) > n - c$.

*Proof.* (a) to (f) follow immediately from the definition of $a(n)$.

(g) Consider the two computers $C$ and $C'$ that always halt and such that $C(n) = \{n+1\}$ and $C'(n+1) = \{n\}$. It follows by Theorem 2.1(c) that $I(n) = I(n+1) + O(1)$. By part (e) of this theorem, if $a(n)$ is defined then $I(a(n)+1) > n$. By part (a) of this theorem, $I(a(n)) \leq n$. Hence if $a(n)$ is defined we have $I(a(n)) = I(a(n)+1)+O(1)$, $I(a(n)) \leq n$, and $I(a(n)+1) > n$. It follows that $I(a(n)) = n + O(1)$.

(h) Consider the computer $C$ such that $C(p) = \{\max S\}$ and halts if $p$ is a description of $S$, i.e. if $U(p) = S$ and halts. It follows that $I(\max S) \leq I(S) + \mathrm{sim}(C)$. Thus by part (c) of this theorem, $\max S \leq a(I(S) + c)$, where $c = \mathrm{sim}(C)$.

(i) Consider the computer $C$ such that $C(p) = \{1 + \max S\}$ and halts if $p$ is a description of $S$, i.e. if $U(p) = S$ and halts. It follows that $I(1 + \max S) \leq I(S) + \mathrm{sim}(C)$. If $a(n) \in S$, then $1 + \max S > a(n)$, and thus by part (e) of this theorem $I(1 + \max S) > n$. Hence $n < I(1 + \max S) \leq I(S) + \mathrm{sim}(C)$, and thus $I(S) > n - c$, where $c = \mathrm{sim}(C)$. Q.E.D.

# 6. How Fast Does the Greatest Natural Number of Complexity ≤ N Grow with Increasing N?

In Theorem 6.2 we show that an equivalent definition of $a(n)$ is the greatest value at $n$ of any partial recursive function of complexity $\leq n$. In Theorem 6.3 we use this to show that any partial function $\succeq a$ eventually overtakes any partial recursive function. This will apply directly to all the functions that will be shown in succeeding sections to be $\asymp$ to $a$.

In Theorem 6.4 it is shown that for any partial recursive function $f$, $f(a(.)) \preceq a$. Thus there is a $c$ such that for all $n$, if $a(n)$ is defined, then $a(n) < a(n+c)$ (Theorem 6.5).

**Theorem 6.1.** There is a $c$ such that if $f : N \to N$ is a partial recursive function defined at $n$ and $n \geq I_e(f)$, then $I(f(n)) \leq n + c$.

*Proof.* Given a minimal e-description $s$ of the graph of $f$, we add it to $0^{n+1}$. As $n \geq I_e(f) = \lg(s)$, the resulting string $p = 0^{n+1} + s$ has both $n$ $(= \lg(p) - 1)$ and the graph of $f$ $(= U(s) = U(p - 0^{\lg(p)}))$ coded into it. Given this string $p$ as its program, a computer $C$ generates the graph of $f$ searching for the pair $(n, f(n))$. If and when it is found, the computer outputs $f(n)$ and halts. Thus, $f(n)$, if defined, is of complexity $\leq \lg(p) + \mathrm{sim}(C) = n + 1 + \mathrm{sim}(C)$. Q.E.D.

**Definition 6.1.** $b(n) = \max f(n)$ $(I_e(f) \leq n)$. The maximum is taken over all partial recursive functions $f : N \to N$ that are defined at $n$ and are of e-complexity $\leq n$. If there are no such functions, then $b(n)$ is undefined.

**Theorem 6.2.** $a \asymp b$.

*Proof.* First we show that $b \preceq a$. If $b(n)$ is defined, then there is a partial recursive function $f : N \to N$ defined at $n$ with $I_e(f) \leq n$, such that $f(n) = b(n)$. By Theorem 6.1, $I(f(n)) \leq n + c$, and thus $f(n) \leq a(n + c)$ by Theorem 5.1(c). Hence if $b(n)$ is defined, $b(n) = f(n) \leq a(n + c)$, and thus $b \preceq a$.

Now we show that $a \preceq b$. Suppose that $a(n)$ is defined, and consider the constant function $f_n : N \to N$ whose value is always $a(n)$, and the computer $C$ such that $C(n) = \{(0, n), (1, n), (2, n), \ldots\}$. It follows by Theorem 2.1(c) that $I_e(f_n) \leq I(a(n)) + c$, which by Theorem 5.1(a)

is $\leq n+c$. Thus if $a(n)$ is defined, $a(n) = f_n(n+c) \leq \max f(n+c)$ $(I_e(f) \leq n+c) = b(n+c)$. Hence $a \preceq b$. Q.E.D.

**Theorem 6.3.** Let the partial function $x : N \to N$ have the property that $x \succeq a$. There is a constant $c'$ such that the following holds for all partial recursive functions $f : N \to N$. If $f(n)$ is defined and $n \geq I_e(f) + c'$, then $x(n)$ is defined and $x(n) \geq f(n)$.

*Proof.* By Theorem 6.2 and the transitivity of $\succeq$, $x \succeq a \succeq b$. Thus there is a $c$ such that $x(n+c)$ is defined and $x(n+c) \geq b(n)$ if $b(n)$ is defined. Consider the shifted function $f'(n) = f(n+c)$. The existence of a computer $C$ such that $(i,j) \in C(p)$ iff $(i+c,j) \in U(p)$ shows that $I_e(f') \leq I_e(f) + \mathrm{sim}(C)$. By the definition of $b$, $x(n+c) \geq b(n) \geq f'(n)$ if $f'$ is defined at $n$ and $I_e(f') \leq n$. Thus $x(n+c) \geq f(n+c)$ if $f$ is defined at $n+c$ and $I_e(f') \leq I_e(f) + \mathrm{sim}(C) \leq n$. In other words, $x(n) \geq f(n)$ if $f$ is defined at $n$ and $I_e(f) + \mathrm{sim}(C) + c \leq n$. The theorem is proved with $c' = \mathrm{sim}(C) + c$. Q.E.D.

**Theorem 6.4.** Let $f : N \to N$ be a partial recursive function. $f(a(.)) \preceq a$.

*Proof.* There is a computer $C$ such that $C(n) = \{f(n)\}$ and halts if $f(n)$ is defined. Thus by Theorem 2.1(c), if $f(n)$ is defined, $I(f(n)) \leq I(n) + c$. Substituting $a(n)$ for $n$, we obtain $I(f(a(n))) \leq I(a(n)) + c \leq n + c$, for by Theorem 5.1(a), $I(a(n)) \leq n$. Thus if $f(a(n))$ is defined, $f(a(n)) \leq a(n+c)$, by Theorem 5.1(c). Q.E.D.

**Theorem 6.5.** There is a $c$ such that for all $n$, if $a(n)$ is defined, then $a(n) < a(n+c)$.

*Proof.* Taking $f(n) = n+1$ in Theorem 6.4, we obtain $a(.)+1 \preceq a$. Q.E.D.

# 7. The Resources Needed to Calculate/ Enumerate the Set of All Strings of Complexity $\leq$ N

## 7.1

We first discuss the metamathematical implications of the material in this section.

The basic fact used in this section (see the proof of Theorem 7.3) is that for any computer $C$ there is a $c$ such that for all $n$, if $a(n)$ is defined then $\max \bigcup C(p, a(n))$ $(\lg(p) \leq a(n))$ is less than $a(n+c)$. Thus $a(n+c)$ cannot be output by programs of length $\leq a(n)$ in time $\leq a(n)$. If we use Theorem 3.1(a) to take $C$ to be such that $s \in C(p,t)$ iff "$I(s) = k$"$\in C^*(p,t)$, and we recall that $a(n+c)$ is a string of complexity $\leq n + c$, we obtain the following result. Any formal system $\langle C^*, p\rangle$ whose theorems include all true propositions of the form "$I(s) = k$" with $k \leq n + c$, must either have more than $a(n)$ bits of axioms, or need proofs of size greater than $a(n)$ to be able to demonstrate these propositions. Here $c$ depends only on the rules of inference $C^*$. This is a strong result, in view of the fact that $a(n)$ is greater than or equal to any partial recursive function $f(n)$ for $n \geq I_e(f) + c'$ (Theorem 6.3).

The idea of Section 9 is to show that both extremes are possible and there is a drastic trade-off. We can deduce these results from a few bits of axioms ($\leq n + c$ bits by Theorem 4.3) by means of enormous proofs, or we can directly take as axioms all that we wish to prove. This gives short proofs, but we are assuming an enormous number of bits of axioms.

From the fact that $a(n + c) > \max \bigcup C(p, a(n))$ $(\lg(p) \leq a(n))$, it also follows that if one wishes to prove a numerical upper bound on $a(n + c)$, one faces the same drastic alternatives. Lin and Rado, in trying to determine particular values of $\Sigma(n)$ and $SH(n)$, have, in fact, essentially been trying to do this (see [36]). In their paper they explain the difficulties they encountered and overcame for $n = 3$, and expect to be insurmountable for greater values of $n$.

## 7.2

Now we begin the formal exposition, which is couched exclusively in terms of computers.

In this section we study the set $K(n)$ consisting of all strings of complexity $\leq n$. This set turns out to be extremely difficult to calculate, or even to enumerate a superset of—either the program or the time needed must be extremely large. In order to measure this difficulty, we will first measure the resources needed to output $a(n)$.

**Definition 7.1.** $K(n) = \{s | I(s) \leq n\}$. Note that this set may be

empty, and $\#(K(n))$ isn't greater than $2^{n+1}-1$, inasmuch as there are exactly $2^{n+1}-1$ programs of length $\leq n$.

We shall show that $a(n)$ and the resources required to calculate/enumerate $K(n)$ grow equally quickly. What do we mean by the resources required to calculate a finite set, or to enumerate a superset of it? It is assumed that the computer $C$ is being used to do this.

**Definition 7.2.** Let $S$ be a finite set of strings. $r(S)$, the resources required to calculate $S$, is the least $r$ such that there is a program $p$ of length $\leq r$ having the property that $C(p,r) = S$ and is halted. If there is no such $r$, $r(S)$ is undefined. $r_e(S)$, the resources required to enumerate a superset of $S$, is the least $r$ such that there is a program $p$ of length $\leq r$ with the property that $S \subset C(p,r)$. If there is no such $r$, $r_e(S)$ is undefined. We abbreviate $r(\{s\})$ and $r_e(\{s\})$ as $r(s)$ and $r_e(s)$.

We shall find very useful the notion of the set of all output produced by the computer $C$ with information and time resources limited to $r$. We denote this by $C_r$.

**Definition 7.3.** $C_r = \bigcup C(p,r)$ $(\lg(p) \leq r)$.

We now list for future reference basic properties of these concepts.

**Theorem 7.0.**

(a) $a(n) = \begin{cases} \max K(n) \text{ if } K(n) \neq \emptyset, \\ \text{undefined if } K(n) = \emptyset. \end{cases}$

(b) $K(n) \neq \emptyset$, and $a(n)$ is defined, iff $n \geq n^*$. Here $n^* = \min I(s)$, where the minimum is taken over all strings $s$.

(c) For all $r$, $C_r \subset C_{r+1}$.

In (d) to (k), $S$ and $S'$ are arbitrary finite sets of strings.

(d) $S \subset C_{r_e(S)}$ if $r_e(S)$ is defined.

(e) $r_e(S) \leq r(S)$ if $r(S)$ is defined.

(f) If $r_e(S')$ is defined, then $S \subset S'$ implies $r_e(S) \leq r_e(S')$.

(g) If $S \subset C(p,t)$, then either $\lg(p) \geq r_e(S)$ or $t \geq r_e(S)$.

(h) If $C(p) = S$ and halts, then either $\lg(p) \geq r(S)$, or the time at which $C(p)$ halts is $\geq r(S)$.

(i) If $C(p) = S$ and halts, and $\lg(p) < r(S)$, then $C(p)$ halts at time $\geq r(S)$.

In (j) and (k) it is assumed that $C$ is $U$. Thus $r(S)$ and $r_e(S)$ are always defined.

(j) If $r(S) > I(S)$, then there is a program $p$ of length $I(S)$ such that $U(p) = S$ and halts at time $\geq r(S)$.

(k) $r(S) \geq I(S)$.

*Proof.* These results follow immediately from the definitions. Q.E.D.

**Theorem 7.1.**

There is a $c$ such that for all finite sets $S$ of strings,

(a) $I(r(S)) \leq I(S) + c$ if $r(S)$ is defined, and

(b) $I(r_e(S)) \leq I(S) + c$ if $r_e(S)$ is defined.

*Proof.* (a) Consider the computer $C'$ that does the following when it is given the program $p$. First, it simulates running $p$ on $U$. If and when $U$ halts during the simulated run, $C'$ has determined the finite set $S = U(p)$ of strings. Then $C'$ repeats the following operations for $r = 0, 1, 2, \ldots$

$C'$ determines $C(p', r)$ for each program $p'$ of length $\leq r$. It checks those $C(p', r)$ ($\lg(p') \leq r$) that are halted to see if one of them is equal to $S$. If none of them are, $C'$ adds 1 to $r$ and repeats this operation. If one of them is equal to $S$, $C'$ outputs $r$ and halts.

Let $p$ be a minimal description of a finite set $S$ of strings, i.e. $U(p) = S$ and halts, and $\lg(p) = I(S)$. Then if $r(S)$ is defined, $C'(p) = \{r(S)\}$ and halts, and thus $I(r(S)) \leq \lg(p) + \mathrm{sim}(C') = I(S) + \mathrm{sim}(C')$. This proves part (a) of the theorem.

(b) The proof of part (b) of the theorem is obtained from the proof of part (a) by changing the definition of the computer $C'$ so that it checks all $C(p', r)$ ($\lg(p') \leq r$) to see if one of them includes $S$, instead of checking all those $C(p', r)$ ($\lg(p') \leq r$) that are halted to see if one of them is equal to $S$. Q.E.D.

**Theorem 7.2.** $\max C_{a(.)} \preceq a$.

*Proof.* Theorem 2.1(c) and the existence of a computer $C'$ such that $C'(r) = C_r$ and halts, shows that there is a $c$ such that for all $r$, $I(C_r) \leq I(r) + c$. Thus by Theorem 5.1(h) and (b) there is a $c'$ such that for all $r$, $\max C_r \leq a(I(r) + c')$. Hence if $a(n)$ is defined, $\max C_{a(n)} \leq a(I(a(n)) + c') \leq a(n + c')$ by Theorem 5.1(a) and (b). Q.E.D.

**Theorem 7.3.**

(a) If $r_e(a(n))$ is defined when $a(n)$ is, then $r_e(a(.)) \asymp a$.

(b) If $r(a(n))$ is defined when $a(n)$ is, then $r(a(.)) \asymp a$.

*Proof.* By Theorem 7.1, if $r_e(a(n))$ and $r(a(n))$ are defined, $I(r_e(a(n))) \leq I(a(n)) + c$ and $I(r(a(n))) \leq I(a(n)) + c$. By Theorem 5.1(a), $I(a(n)) \leq n$. Thus $I(r_e(a(n))) \leq n + c$ and $I(r(a(n))) \leq n + c$. Applying Theorem 5.1(c), we obtain $r_e(a(n)) \leq a(n+c)$ and $r(a(n)) \leq a(n + c)$. Thus we have shown that $r_e(a(.)) \preceq a$ and $r(a(.)) \preceq a$, no matter what $C$ is.

$r(S)$, if defined, is $\geq r_e(S)$ (Theorem 7.0(e)), and thus to finish the proof it is sufficient to show that $a \preceq r_e(a(.))$ if $r_e(a(n))$ is defined when $a(n)$ is. By Theorems 7.2 and 6.5 there is a $c$ such that for all $n$, if $a(n)$ is defined, then $\max C_{a(n)} < a(n + c)$, and thus $a(n + c) \notin C_{a(n)}$. And inasmuch as for all finite sets $S$, $S \subset C_{r_e(S)}$ (Theorem 7.0(d)), it follows that $a(n + c) \in C_{r_e(a(n+c))}$.

In summary, there is a $c$ such that for all $n$, if $a(n)$ is defined, then $a(n + c) \notin C_{a(n)}$, and $a(n + c) \in C_{r_e(a(n+c))}$.

As for all $r$, $C_r \subset C_{r+1}$ (Theorem 7.0(c)), it follows that if $a(n)$ is defined then $a(n) < r_e(a(n + c))$. Thus $a \preceq r_e(a(.))$. Q.E.D.

**Theorem 7.4.** $I(K(n)) = n + O(1)$.

*Proof.* As was essentially shown in the proof of Theorem 4.3, there is a computer $C'$ such that $C'(0^{n+1} + \#(\{p \in H | \lg(p) \leq n\})) = K(n)$ and halts, for all $n$. Thus $I(K(n)) \leq n + 1 + \mathrm{sim}(C')$ for all $n$.

$K(n) = \emptyset$ can hold for only finitely many values of $n$, by Theorem 7.0(b). By Theorem 7.0(a), for all other values of $n$, $a(n) \in K(n)$, and thus, by Theorem 5.1(i), there is a $c$ such that $I(K(n)) \geq n - c$ for all $n$. Q.E.D.

**Theorem 7.5.**

(a) If $r_e(K(n))$ is defined for all $n$, then $r_e(K(.)) \asymp a$.

(b) If $r(K(n))$ is defined for all $n$, then $r(K(.)) \asymp a$.

*Proof.* By Theorem 7.1, if $r_e(K(n))$ and $r(K(n))$ are defined, $I(r_e(K(n))) \leq I(K(n)) + c$, and $I(r(K(n))) \leq I(K(n)) + c$. $I(K(n)) = n + O(1)$ (Theorem 7.4), and thus there is a $c'$ that doesn't depend on $n$ such that $I(r_e(K(n))) \leq n + c'$, and $I(r(K(n))) \leq n + c'$. Applying Theorem 5.1(c), we obtain $r_e(K(n)) \leq a(n + c')$, and $r(K(n)) \leq a(n + c')$. Thus we have shown that $r_e(K(.)) \preceq a$ and $r(K(.)) \preceq a$, no matter what $C$ is.

For all finite sets $S$ and $S'$ of strings, if $S \subset S'$, then $r_e(S) \leq r_e(S') \leq r(S')$ if these are defined (Theorem 7.0(f), (e)). As $a(n) \in K(n)$ if $a(n)$ is defined (Theorem 7.0(a)), we have $r_e(a(n)) \leq r_e(K(n)) \leq r(K(n))$ if these are defined. By Theorem 7.3(a), $r_e(a(.)) \succeq a$ if $r_e(a(n))$ is defined when $a(n)$ is, and thus $r_e(K(.)) \succeq a$ and $r(K(.)) \succeq a$ if these are defined for all $n$. Q.E.D.

**Theorem 7.6.** Suppose $r_e(a(n))$ is defined when $a(n)$ is. There is a $c$ such that the following holds for all partial recursive functions $f : N \to N$. If $n \geq I_e(f) + c$ and $f(n)$ is defined, then

(a) $a(n)$ is defined,

(b) if $a(n) \in C(p, t)$, then either $\lg(p) \geq f(n)$ or $t \geq f(n)$, and

(c) if $K(n) \subset C(p, t)$, then either $\lg(p) \geq f(n)$, or $t \geq f(n)$.

*Proof.* (a) and (b) By Theorem 7.3(a), $r_e(a(.)) \succeq a$. Taking $r_e(a(.))$ to be the partial function $x(.)$ in the hypothesis of Theorem 6.3, we deduce that if $n \geq I_e(f) + c$ and $f(n)$ is defined, then $a(n)$ is defined and $r_e(a(n)) \geq f(n)$. Here $c$ doesn't depend on $f$.

Thus if $a(n) \in C(p, t)$, then by Theorem 7.0(g) it follows that either $\lg(p) \geq r_e(a(n)) \geq f(n)$ or $t \geq r_e(a(n)) \geq f(n)$.

(c) Part (c) of this theorem is an immediate consequence of parts (a) and (b) and the fact that if $a(n)$ is defined then $a(n) \in K(n)$ (see Theorem 7.0(a)). Q.E.D.

# 8. The Minimum Time Such That All Programs of Length $\leq$ N That Halt Have Done So

In this section we show that for any computer, $a \succeq$ the minimum time such that all programs of length $\leq n$ that halt have done so (Theorem 8.1). Moreover, in the case of $U$ this is true with "$\asymp$" instead of "$\succeq$" (Theorem 8.2).

The situation revealed in the proof of Theorem 8.2 can be stated in the following vague but suggestive manner. Suppose that one wishes to calculate $a(n)$ or $K(n)$ using the standard computer $U$. To do this one only needs about $n$ bits of information. But a program of length $n+O(1)$ for calculating $a(n)$ is among the programs of length $\leq n+O(1)$ that take the most time to halt. Likewise, an $(n+O(1))$-bit program for calculating $K(n)$ is among the programs of length $\leq n+O(1)$ that take the most time to halt. These are among the most difficult calculations that can be accomplished by program having not more than $n$ bits.

**Definition 8.1.** $d_C(n)$ = the least $t$ such that for all $p$ of length $\leq n$, if $C(p)$ halts, then $C(p,t)$ is halted. This is the minimum time at which all programs of length $\leq n$ that halt have done so. Although it is 0 if no program of length $\leq n$ halts, we stipulate that $d_C(n)$ is undefined in this case.

**Theorem 8.1.** $d_C \preceq a$.

*Proof.* Consider the computer $C'$ that does the following when it is given the program $p$. $C'$ simulates $C(p,t)$ for $t = 0, 1, 2, \ldots$ until $C(p,t)$ is halted. If and when this occurs, $C'$ outputs the final value of $t$, which is the time at which $C(p)$ halts. Finally, $C'$ halts.

If $d_C(n)$ is defined, then there is a program $p$ of length $\leq n$ that halts when run on $C$ and does this at time $d_C(n)$. Then $C'(p) = \{d_C(n)\}$ and halts. Thus $I(d_C(n)) \leq \lg(p) + \mathrm{sim}(C') \leq n + \mathrm{sim}(C')$. By Theorem 5.1(c), we conclude that $d_C(n)$ is, if defined, $\leq a(n + \mathrm{sim}(C'))$. Q.E.D.

**Theorem 8.2.** $d_U \asymp a$.

*Proof.* In view of Theorem 8.1, $d_U \preceq a$. Thus we need only show that $d_U \succeq a$.

Recall that $a(n)$ is defined iff $n \geq n^*$ (Theorem 7.0(b)). As $C = U$ is a universal computer, $r(a(n))$ is defined if $a(n)$ is defined. Thus

Theorem 7.3(b) applies to this choice of $C$, and $r(a(.)) \succeq a$. That is to say, there is a $c$ such that for all $n \geq n^*$, $r(a(n+c)) \geq a(n)$.

As $a \succeq a$, taking $x = a$ and $f(n) = n + c + 1$ in Theorem 6.3, we obtain the following. There is a $c'$ such that for all $n \geq n^* + c'$, $a(n) \geq f(n) = n + c + 1$. We conclude that for all $n \geq n^* + c'$, $a(n) > n + c$.

By Theorem 5.1(a), $n + c \geq I(a(n + c))$ for all $n \geq n^*$.

The preceding results may be summarized in the following chain of inequalities. For all $n \geq n^*+c'$, $r(a(n+c)) \geq a(n) > n+c \geq I(a(n+c))$.

As $r(a(n + c)) \geq I(a(n + c))$, the hypothesis of Theorem 7.0(j) is satisfied, and we conclude the following. There is a program $p$ of length $I(a(n + c)) \leq n + c$ such that $U(p) = \{a(n + c)\}$ and halts at time $\geq r(a(n + c)) \geq a(n)$. Thus for all $n \geq n^* + c'$, $d_U(n + c) \geq a(n)$.

Applying Theorem 5.1(b) to this lower bound on $d_U(n + c)$, we conclude that for all $n \geq n^*$, $d_U(n + c' + c) \geq a(n + c') \geq a(n)$. Q.E.D.

# 9. Examples of Trade-Offs Between Information and Time

Consider calculating $a(n)$ using the computer $U$ and the computer $C$ defined as follows. For all programs $p$, $C(p, 0) = \{p\}$ and is halted.

Since $I(a(n)) \leq n$ (Theorem 5.1(a)), there is a program $\leq n$ bits long for calculating $a(n)$ using $U$. But inasmuch as $r(a(.)) \succeq a$ (Theorem 7.3(b)) and $d_U \preceq a$ (Theorem 8.1), this program takes "about" $a(n)$ units of time to halt (see the proof of Theorem 8.2). More precisely, with finitely many exceptions, this program takes between $a(n-c)$ and $a(n + c)$ units of time to halt.

What happens if one uses $C$ to calculate $a(n)$? Inasmuch as $C(a(n)) = \{a(n)\}$ and halts at time 0, $C$ can calculate $a(n)$ immediately. But this program, although fast, is $\lg(a(n)) = \lfloor \log_2(a(n) + 1) \rfloor$ bits long. Thus $r(a(n))$ is precisely $\lg(a(n))$ if one uses this computer.

Now for our second example. Suppose one wishes to enumerate a superset of $K(n)$, and is using the following two computers, which never halt: $C(p, t) = \{s | \lg(s) \leq t\}$ and $C'(p, t) = \{s | \lg(s) \leq \lg(p)\}$. These two computers have the property that $K(n)$, if not empty, is included in

$C(p,t)$, or is included in $C'(p,t)$, iff $t \geq \lg(a(n))$, or iff $\lg(p) \geq \lg(a(n))$, respectively. Thus for these two computers, $r_e(K(n))$, which we know by Theorem 7.5(a) must be $\asymp$ to $a$, is precisely given by the following: $r_e(K(n)) = 0$ if $a(n)$ is undefined, and $\lg(a(n))$ otherwise.

It is also interesting to slow down or speed up the computer $C$ by changing its time scale recursively. Let $f : N \to N$ be an arbitrary unbounded total recursive function with the property that for all $n$, $f(n) \leq f(n+1)$. $C^f$, the $f$ speed-up/slowdown of $C$, is defined as follows: $C^f(p,t) = \{s | f(\lg(s)) \leq t\}$. For the computer $C^f$, $r_e(K(n))$ is precisely given by the following: $r_e(K(n)) = 0$ if $a(n)$ is undefined, and $f(\lg(a(n)))$ otherwise. The fact that by Theorem 7.5(a) this must be $\asymp$ to $a$, is related to Theorem 6.4 that for any partial recursive function $f$, $f(a(.)) \preceq a$.

Now, for our third example, we consider trade-offs in calculating $K(n)$. We use $U$ and the computer $C$ defined as follows. For all $p$ and $n$, $C(p,0)$ is halted, and $n \in C(p,0)$ iff $\lg(p) > n$ and the $n$th bit of the string $p$ is a 1.

Inasmuch as $I(K(n)) = n + O(1)$ (Theorem 7.4), if we use $U$ there is a program about $n$ bits long for calculating $K(n)$. But this program takes "about" $a(n)$ units of time to halt, in view of the fact that $r(K(.)) \succeq a$ (Theorem 7.5(b)) and $d_U \preceq a$ (Theorem 8.1) (see the proof of Theorem 8.2). More precisely, with finitely many exceptions, this program takes between $a(n-c)$ and $a(n+c)$ units of time to halt.

On the other hand, using the computer $C$ we can calculate $K(n)$ immediately. But the shortest program for doing this has length precisely $1 + \max K(n) = a(n) + 1$ if $a(n)$ is defined, and has length 0 otherwise. In other words, for this computer $r(K(n)) = 0$ if $a(n)$ is undefined, and $a(n) + 1$ otherwise.

We have thus seen three examples of a drastic trade-off between information and time resources. In this setting information and time play symmetrical roles, especially in the case of the resources needed to enumerate a superset.

# 10. The Speed of Recursive Enumerations

## 10.1

We first discuss the metamathematical implications of the material in this section.

Consider a particular formal system, and a particular r.e. set of strings $R$. Suppose that a proposition of the form "$s \in R$" is a theorem of this formal system iff it is true, i.e. iff the string $s$ is an element of $R$. Define $e(n)$ to be the least $m$ such that all theorems of the formal system of the form "$s \in R$" with $\lg(s) \leq n$ have proofs of size $\leq m$. By using Theorem 3.1(a) we can draw the following conclusions from the results of this section. First, $e \preceq a$ for any $R$. Second, $e \asymp a$ iff

$$I(\{s \in R | \lg(s) \leq n\}) = n + O(1). \qquad (*)$$

Thus r.e. sets $R$ for which $e \asymp a$ are the ones that require the longest proofs to show that "$s \in R$," and this is the case iff $R$ satisfies $(*)$. It is shown in this section that the r.e. set of strings $\{p | p \in U(p)\}$ has property $(*)$, and the reader can show without difficulty that $H$ and $P$ are also r.e. sets of strings that have property $(*)$. Thus we have three examples of $R$ for which $e \asymp a$.

## 10.2

Now we begin the formal exposition, which is couched exclusively in terms of computers.

Consider an r.e. set of strings $R$ and a particular computer $C^*$ and $p^*$ such that $C^*(p^*) = R$. How quickly is $R$ enumerated? This is, what is the time $e(n)$ that it takes to output all elements of $R$ of length $\leq n$?

**Definition 10.1.** $R_n = \{s \in R | \lg(s) \leq n\}$. $e(n) =$ the least $t$ such that $R_n \subset C^*(p^*, t)$.

We shall see that the rate of growth of the total function $e(n)$ can be related to the growth of the complexity of $R_n$. In this way we shall show that some r.e. sets $R$ are the most difficult to enumerate, i.e. take the most time.

**Theorem 10.1.**[8] There is a $c$ such that for all $n$, $I(R_n) \leq n + c$.

[8]This theorem, with a different proof, is due to Loveland [37, p. 64].

*Proof.* $0 \leq \#(R_n) \leq 2^{n+1} - 1$, for there are precisely $2^{n+1} - 1$ strings of length $\leq n$. Consider $p$, the $\#(R_n)$-th string of length $n+1$; i.e. $p = 0^{n+1} + \#(R_n)$. This string has both $n$ $(= \lg(p) - 1)$ and $\#(R_n)$ $(= p - 0^{\lg(p)})$ coded into it. When this string $p$ is its program, the computer $C$ generates the r.e. set $R$ by simulating $C^*(p^*)$, until it has found $\#(R_n)$ strings of length $\leq n$ in $R$. $C$ then outputs this set of strings, which is $R_n$, and halts. Thus $I(R_n) \leq \lg(p) + \mathrm{sim}(C) = n + 1 + \mathrm{sim}(C)$. Q.E.D.

**Theorem 10.2.**

(a) There is a $c$ such that for all $n$, $e(n) \leq a(I(R_n) + c)$.

(b) $e \preceq a$.

*Proof.* (a) Consider the computer $C$ that does the following. Given a description $p$ of $R_n$ as its program, the computer $C$ first simulates running $p$ on $U$ in order to determine $R_n$. Then it simulates $C^*(p^*, t)$ for $t = 0, 1, 2, \ldots$ until $R_n \subset C^*(p^*, t)$. $C$ then outputs the final value of $t$, which is $e(n)$, and halts.

This shows that $\leq \mathrm{sim}(C)$ bits need be added to the length of a description of $R_n$ to bound the length of a description of $e(n)$; i.e. if $U(p) = R_n$ and halts, then $C(p) = \{e(n)\}$ and halts, and thus $I(e(n)) \leq \lg(p) + \mathrm{sim}(C)$. Taking $p$ to be a minimal description of $R_n$, we have $\lg(p) = I(R_n)$, and thus $I(e(n)) \leq I(R_n) + \mathrm{sim}(C)$. By Theorem 5.1(c), this gives us $e(n) \leq a(I(R_n) + \mathrm{sim}(C))$. Part (a) of the theorem is proved with $c = \mathrm{sim}(C)$.

(b) By part (a) of this theorem, $e(n) \leq a(I(R_n) + c)$. And by Theorem 10.1, $I(R_n) \leq n + c'$ for all $n$. Applying Theorem 5.1(b), we obtain $e(n) \leq a(I(R_n) + c) \leq a(n + c' + c)$ for all $n$. Thus $e \preceq a$. Q.E.D.

**Theorem 10.3.** If $a \preceq e$, then there is a $c$ such that $I(R_n) \geq n - c$ for all $n$.

*Proof.* By Theorem 7.0(b) and the definition of $\preceq$, if $a \preceq e$, then there is a $c_0$ such that for all $n \geq n^*$, $a(n) \leq e(n + c_0)$. And by Theorem 10.2(a), there is a $c_1$ such that $e(n + c_0) \leq a(I(R_{n+c_0}) + c_1)$ for all $n$. We conclude that for all $n \geq n^*$, $a(n) \leq a(I(R_{n+c_0}) + c_1)$.

By Theorems 6.5 and 5.1(b), there is a $c_2$ such that if $a(m)$ is defined and $m \leq n - c_2$, then $a(m) < a(n)$. As we have shown in the first

paragraph of this proof that for all $n \geq n^*$, $a(n) \leq a(I(R_{n+c_0}) + c_1)$, it follows that $I(R_{n+c_0}) > n - c_2$.

In other words, for all $n \geq n^*$, $I(R_{n+c_0}) > (n+c_0) - c_0 - c_1 - c_2$. And thus for all $n$, $I(R_n) \geq n - c_0 - c_1 - c_2 - M$, where $M = \max_{n<n^*+c_0} n - c_0 - c_1 - c_2$ if this is positive, and 0 otherwise. The theorem is proved with $c = c_0 + c_1 + c_2 + M$. Q.E.D.

**Theorem 10.4.**

If there is a $c$ such that $I(R_n) \geq n - c$ for all $n$, then

(a) there is a $c'$ such that if $t \geq e(n)$, then $I(t) > n - c'$, and

(b) $e \succeq a$.

*Proof.* By Theorem 5.1(f) it follows from (a) that $e(n) > a(n - c')$ if $a(n - c')$ is defined. Hence $e(n + c') \geq a(n)$ if $a(n)$ is defined, i.e. $e \succeq a$. Thus to complete the proof we need only show that (a) follows from the hypothesis.

We consider the case in which $t \geq e(n)$ and $n \geq I(t) = n - k$, for if $I(t) > n$ then any $c'$ will do.

There is a computer $C$ that does the following when it is given the program $0^{\lg(k)}1kp$, where $p$ is a minimal description of $t$. First, $C$ determines $\lg(p) + k = I(t) + k = (n - k) + k = n$. Second, $C$ simulates running $p$ on $U$ in order to determine $U(p) = \{t\}$. $C$ now uses its knowledge of $n$ and $t$ in order to calculate $R_n$. To do this $C$ first simulates running $p^*$ on $C^*$ in order to determine $C^*(p^*, t)$, and finally $C$ outputs all strings in $C^*(p^*, t)$ that are of length $\leq n$, which is $R_n$, and halts.

In summary, $C$ has the property that if $t \geq e(n)$, $I(t) = n - k$, and $p$ is a minimal description of $t$, then $C(0^{\lg(k)}1kp) = R_n$ and halts, and thus

$$\begin{aligned} & I(R_n) \\ \leq\ & \lg(0^{\lg(k)}1kp) + \mathrm{sim}(C) \\ =\ & \lg(p) + 2\lg(k) + \mathrm{sim}(C) + 1 \\ =\ & I(t) + 2\lg(k) + \mathrm{sim}(C) + 1 \\ =\ & n - k + 2\lg(k) + \mathrm{sim}(C) + 1. \end{aligned}$$

Taking into account the hypothesis of this theorem, we obtain the following for all $n$: if $t \geq e(n)$ and $I(t) = n - k$, then $n - c \leq I(R_n) \leq n - k + 2\lg(k) + \mathrm{sim}(C) + 1$, and thus $c + \mathrm{sim}(C) + 1 \geq k - 2\lg(k)$. As $\lg(k) = \lfloor \log_2(k+1) \rfloor$, this implies that there is a $c'$ such that for all $n$, if $t \geq e(n)$ and $I(t) = n - k$, then $k < c'$. We conclude that for all $n$, if $t \geq e(n)$, then either $I(t) > n$ or $I(t) = n - k > n - c'$. Thus in either case $I(t) > n - c'$. Q.E.D.

**Theorem 10.5.** If $R = \{p | p \in U(p)\}$, then there is a $c$ such that $I(R_n) > n - c$ for all $n$.

*Proof.* Consider the following computer $C$. When given the program $p$, $C$ first simulates running $p$ on $U$ until $U$ halts. If and when it finishes doing this, $C$ then outputs each string $s \notin U(p)$, and never halts.

If the program $p$ is a minimal description of $R_n$, then $C$ enumerates a set that cannot be enumerated by any program $p'$ run on $U$ having $\leq n$ bits. The reason is that if $\lg(p') \leq n$, then $p' \in C(p)$ iff $p' \notin R_n$ iff $p' \notin U(p')$. Thus $\leq \mathrm{sim}(C)$ bits need be added to the length $I(R_n)$ of a minimal description $p$ of $R_n$ to bound the length of an e-description of the set $C(p)$ of e-complexity $> n$; i.e. $n < I_e(C(p)) \leq \lg(p) + \mathrm{sim}(C) = I(R_n) + \mathrm{sim}(C)$. Hence $n < I(R_n) + c$, where $c = \mathrm{sim}(C)$. Q.E.D.

**Theorem 10.6.**

(a) $e \preceq a$ and $\exists c\ \forall n\ I(R_n) \leq n + c$.

(b) $e \succeq a$ iff $\exists c\ \forall n\ I(R_n) \geq n - c$.

(c) If $R = \{p | p \in U(p)\}$, then $e \asymp a$ and $I(R_n) = n + O(1)$.

*Proof.* (a) is Theorems 10.2(b) and 10.1. (b) is Theorem 10.4(b) and 10.3. And (c) follows immediately from parts (a) and (b) and Theorem 10.5. Q.E.D.

# Appendix. Examples of Universal Computers

In this Appendix we use the formalism of Rogers.[9] In particular, $P_x$ denotes the $x$th Turing machine, $\varphi_x^{(2)}$ denotes the partial recursive func-

[9] See [38, pp. 13–15, 21, 70].

tion $N \times N \to N$ that $P_x$ calculates, and $D_x$ denotes the $x$th finite set of natural numbers. Here the index $x$ is an arbitrary natural number.

First we give a more formal definition of computer than in Section 2.

A partial recursive function $c : N \times N \to N$ is said to be adequate (as a defining function for a computer $C$) iff it has the following three properties:

(a) it is a total function;

(b) $D_{c(p,t)} \subset D_{c(p,t+1)}$;

(c) if the natural number 0 is an element of $D_{c(p,t)}$, then $D_{c(p,t)} = D_{c(p,t+1)}$.

A computer $C$ is defined by means of an adequate function $c : N \times N \to N$ as follows.

(a) $C(p,t)$ is halted iff the natural number 0 is an element of $D_{c(p,t)}$.

(b) $C(p,t)$ is the set of strings $\{n|n+1 \in D_{c(p,t)}\}$; i.e. the $n$th string is in $C(p,t)$ iff the natural number $n+1$ is an element of $D_{c(p,t)}$.

We now give an name to each computer. The natural number $i$ is said to be an adequate index iff $\varphi_i^{(2)}$ is an adequate function. If $i$ is an adequate index, $C^i$ denotes the computer whose defining function is $\varphi_i^{(2)}$. If $i$ isn't an adequate index, then "$C^i$" isn't the name of a computer.

We now define a universal computer $U$ in such a way that it has the property that if $i$ is an adequate index, then $U(0^i1p) = C^i(p)$ and halts iff $C^i(p)$ halts. In what follows $i$ and $t$ denote arbitrary natural numbers, and $p$ denotes an arbitrary string. $U(0^i, t)$ is defined to be equal to $\emptyset$ and to be halted. $U(0^i1p, t)$ is defined recursively. If $t \geq 1$ and $U(0^i1p, t-1)$ is halted, then $U(0^i1p, t) = U(0^i1p, t-1)$ and is halted. Otherwise $U(0^i1p, t)$ is the set of strings $\{n|n+1 \in W\}$ and is halted iff $0 \in W$. Here

$$W = \bigcup_{t'<t_0} D_{\varphi_i^{(2)}(p,t')},$$

and $t_0$ is the greatest natural number $\leq t$ such that if $t' < t_0$ then $P_i$ applied to $\langle p, t' \rangle$ yields an output in $\leq t$ steps.

The universal computer $U$ that we have just defined is, in fact, effectively universal: to simulate the computation that $C^i$ performs when it is given the program $p$, one gives $U$ the program $p' = 0^i 1 p$, and thus $p'$ can be obtained from $p$ in an effective manner. Our second example of a universal computer, $U'$, is not effectively universal, i.e. there is no effective procedure for obtaining $p'$ from $p$.[10]

$U'$ is defined as follows:

$$\begin{cases} U'(\Lambda, t) &= \emptyset \text{ and is halted,} \\ U'(0p, t) &= U(p,t) - \{1\} \text{ and is halted iff } U(p,t) \text{ is, and} \\ U'(1p, t) &= U(p,t) \cup \{1\} \text{ and is halted iff } U(p,t) \text{ is.} \end{cases}$$

I.e. $U'$ is almost identical to $U$, except that it eliminates the string 1 from the output, or forces the string 1 to be included in the output, depending on whether the first bit of its program is 0 or not. It is easy to see that $U'$ cannot be effectively universal. If it were, given any program $p$ for $U$, by examining the first bit of the program $p'$ for $U'$ that simulates it, one could decide whether or not the string 1 is in $U(p)$. But there cannot be an effective procedure for deciding, given any $p$, whether or not the string 1 is in $U(p)$.

# Added in Proof

The following additional references have come to our attention.

Part of Gödel's analysis of Cantor's continuum problem [39] is highly relevant to the philosophical considerations of Section 1. Cf. especially [39, pp. 265, 272].

Schwartz [40, pp. 26–28] first reformulates our Theorem 4.1 using the hypothesis that the formal system in question is a consistent extension of arithmetic. He then considerably extends Theorem 4.1 [40, pp. 32–34]. The following is a paraphrase of these pages.

Consider a recursive function $f : N \to N$ that grows very quickly, say $f(n) = n!!!!!!!!!!$. A string $s$ is said to have *property* $f$ if the fact

[10] The definition of $U'$ is an adaptation of [38, p. 42, Exercise 2-11].

that $p$ is a description of $\{s\}$ either implies that $\lg(p) \geq \lg(s)$ or that $U(p)$ halts at time $> f(\lg(s))$. Clearly a 1000-bit string with property $f$ is very difficult to calculate. Nevertheless, a counting argument shows that there are strings of all lengths with property $f$, and they can be found in an effective manner [40, Lemma 7, p. 32]. In fact, the first string of length $n$ with property $f$ is given by a recursive function of $n$, and is therefore of complexity $\leq \log_2 n + c$. This is thus an example of an extreme trade-off between program size and the length of computation. Furthermore, an argument analogous to the demonstration of Theorem 4.1 shows that proofs that specific strings have property $f$ must necessarily be extremely tedious (if some natural hypotheses concerning $U$ and the formal system in question are satisfied) [40, Theorem 8, pp. 33–34].

[41, Item 2, pp. 12–20] sheds light on the significance of these results. Cf. especially the first unitalicized paragraphs of answers numbers 4 and 8 to the question "What is programming?" [41, pp. 13, 15–16]. Cf. also [40, Appendix, pp. 63–69].

# Index of Symbols

Section 2: $\lg(s)$ $\max S$ $\#(S)$ $X^*$ $N$ $C(p,t)$ $C(p)$ $U$ $\mathrm{sim}(C)$ $I(S)$ $I_e(S)$

Section 3: $\langle C, p\rangle$ $\langle U, p\rangle$

Section 4: $H$ $P$

Section 5: $\preceq$ $\succeq$ $\asymp$ $a(n)$

Section 6: $b(n)$

Section 7: $K(n)$ $r(S)$ $r_e(S)$ $C_r$ $n^*$

Section 8: $d_C(n)$

Section 10: $R_n$ $e(n)$

# References

[1] Chaitin, G. J. Information-theoretic aspects of Post's construction of a simple set. On the difficulty of generating all binary strings of complexity less than $n$. (Abstracts.) *AMS Notices 19* (1972), pp. A-712, A-764.

[2] Chaitin, G. J. On the greatest natural number of definitional or information complexity $\leq n$. There are few minimal descriptions. (Abstracts.) *Recursive Function Theory: Newsletter,* no. 4 (1973), pp. 11–14, Dep. of Math., U. of California, Berkeley.

[3] von Neumann, J. Method in the physical sciences. *J. von Neumann—Collected Works, Vol. VI,* A. H. Taub, Ed., MacMillan, New York, 1963, No. 36, pp. 491–498.

[4] von Neumann, J. The mathematician. In *The World of Mathematics, Vol. 4,* J. R. Newman, Ed., Simon and Schuster, New York, 1956, pp. 2053–2063.

[5] Bell, E. T. *Mathematics: Queen and Servant of Science.* McGraw-Hill, New York, 1951, pp. 414–415.

[6] Weyl, H. Mathematics and logic. *Amer. Math. Mon. 53* (1946), 1–13.

[7] Weyl, H. *Philosophy of Mathematics and Natural Science.* Princeton U. Press, Princeton, N.J., 1949, pp. 234–235.

[8] Turing, A. M. Solvable and unsolvable problems. In *Science News,* no. 31 (1954), A. W. Heaslett, Ed., Penguin Books, Harmondsworth, Middlesex, England, pp. 7–23.

[9] Nagel, E., and Newman, J. R. *Gödel's Proof.* Routledge & Kegan Paul, London, 1959.

[10] Davis, M. *Computability and Unsolvability.* McGraw-Hill, New York, 1958.

[11] Quine, W. V. Paradox. *Scientific American 206,* 4 (April 1962), 84–96.

[12] Kleene, S. C. *Mathematical Logic.* Wiley, New York, 1968, Ch. V, pp. 223–282.

[13] Gödel, K. On the length of proofs. In *The Undecidable,* M. Davis, Ed., Raven Press, Hewlett, N.Y., 1965, pp. 82–83.

[14] Cohen, P. J. *Set Theory and the Continuum Hypothesis.* Benjamin, New York, 1966, p. 45.

[15] Arbib, M. A. Speed-up theorems and incompleteness theorems. In *Automata Theory,* E. R. Cainiello, Ed., Academic Press, New York, 1966, pp. 6–24.

[16] Ehrenfeucht, A., and Mycielski, J. Abbreviating proofs by adding new axioms. *AMS Bull. 77* (1971), 366–367.

[17] Pólya, G. Heuristic reasoning in the theory of numbers. *Amer. Math. Mon. 66* (1959), 375–384.

[18] Einstein, A. Remarks on Bertrand Russell's theory of knowledge. In *The Philosophy of Bertrand Russell,* P. A. Schilpp, Ed., Northwestern U., Evanston, Ill., 1944, pp. 277–291.

[19] Hawkins, D. Mathematical sieves. *Scientific American 199,* 6 (Dec. 1958), 105–112.

[20] Kolmogorov, A. N. Logical basis for information theory and probability theory. *IEEE Trans. IT-14* (1968), 662–664.

[21] Martin-Löf, P. Algorithms and randomness. *Rev. of Internat. Statist. Inst. 37* (1969), 265–272.

[22] Loveland, D. W. A variant of the Kolmogorov concept of complexity. *Inform. and Contr. 15* (1969), 510–526.

[23] Chaitin, G. J. On the difficulty of computations. *IEEE Trans. IT-16* (1970), 5–9.

[24] Willis, D. G. Computational complexity and probability constructions. *J. ACM 17,* 2 (April 1970), 241–259.

[25] Zvonkin, A. K., and Levin, L. A. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Math. Surveys 25,* 6 (Nov.-Dec. 1970), 83–124.

[26] Schnorr, C. P. *Zufälligkeit und Wahrscheinlichkeit — Eine algorithmische Begründung der Wahrscheinlichkeitstheorie.* Springer, Berlin, 1971.

[27] Fine, T. L. *Theories of Probability—An Examination of Foundations.* Academic Press, New York, 1973.

[28] Chaitin, G. J. Information-theoretic computational complexity. *IEEE Trans. IT-20* (1974), 10–15.

[29] DeLong, H. *A Profile of Mathematical Logic.* Addison-Wesley, Reading, Mass., 1970, Sec. 28.2, pp. 208–209.

[30] Davis, M. Hilbert's tenth problem is unsolvable. *Amer. Math. Mon. 80* (1973), 233–269.

[31] Post, E. Recursively enumerable sets of positive integers and their decision problems. In *The Undecidable,* M. Davis, Ed., Raven Press, Hewlett, N.Y., 1965, pp. 305–307.

[32] Minsky, M. L. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, N.J., 1967, Sec. 12.2–12.5, pp. 222–232.

[33] Shoenfield, J. R. *Mathematical Logic.* Addison-Wesley, Reading, Mass., 1967, Sec. 1.2, pp. 2–6.

[34] Mendelson, E. *Introduction to Mathematical Logic.* Van Nostrand Reinhold, New York, 1964, pp. 29–30.

[35] Russell, B. Mathematical logic as based on the theory of types. In *From Frege to Gödel,* J. van Heijenoort, Ed., Harvard U. Press, Cambridge, Mass., 1967, pp. 150–182.

[36] Lin, S., and Rado, T. Computer studies of Turing machine problems. *J. ACM 12,* 2 (April 1965), 196–212.

[37] Loveland, D. W. On minimal-program complexity measures. Conf. Rec. of the ACM Symposium on Theory of Computing, Marina del Rey, California, May 1969, pp. 61–65.

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

[39] Gödel, K. What is Cantor's continuum problem? In *Philosophy of Mathematics,* Benacerraf, P., and Putnam, H., Eds., Prentice-Hall, Englewood Cliffs, N.J., 1964, pp. 258–273.

[40] Schwartz, J. T. A short survey of computational complexity theory. Notes, Courant Institute of Mathematical Sciences, NYU, New York, 1972.

[41] Schwartz, J. T. *On Programming: An Interim Report on the SETL Project. Installment I: Generalities.* Lecture Notes, Courant Institute of Mathematical Sciences, NYU, New York, 1973.

[42] Chaitin, G. J. A theory of program size formally identical to information theory. Res. Rep. RC4805, IBM Res. Center, Yorktown Heights, N.Y., 1974.

Received October 1971; Revised July 1973