A Theory of Program Size Formally Identical to Information Theory

GREGORY J. CHAITIN

IBM Thomas J. Watson Research Center, Yorktown Heights, New York

ABSTRACT. A new definition of program-size complexity is made. $H(A,B/C,D)$ is defined to be the size in bits of the shortest self-delimiting program for calculating strings A and B if one is given a minimal-size self-delimiting program for calculating strings C and D. This differs from previous definitions: (1) programs are required to be self-delimiting, i.e. no program is a prefix of another, and (2) instead of being given C and D directly, one is given a program for calculating them that is minimal in size. Unlike previous definitions, this one has precisely the formal properties of the entropy concept of information theory. For example, $H(A,B) = H(A) + H(B/A) + O(1)$. Also, if a program of length k is assigned measure 2^{-k}, then $H(A) = -\log_2$ (the probability that the standard universal computer will calculate A) $+ O(1)$.

KEY WORDS AND PHRASES: computational complexity, entropy, information theory, instantaneous code, Kraft inequality, minimal program, probability theory, program size, random string, recursive function theory, Turing machine

CR CATEGORIES: 5.25, 5.26, 5.27, 5.5, 5.6

1. *Introduction*

There is a persuasive analogy between the entropy concept of information theory and the size of programs. This was realized by the first workers in the field of program-size complexity, Solomonoff [1], Kolmogorov [2], and Chaitin [3, 4], and it accounts for the large measure of success of subsequent work in this area. However, it is often the case that results are cumbersome and have unpleasant error terms. These ideas cannot be a tool for general use until they are clothed in a powerful formalism like that of information theory.

This opinion is apparently not shared by all workers in the field (see Kolmogorov [5]), but it has led others to formulate alternative definitions of program-size complexity, for example, Loveland's uniform complexity [6] and Schnorr's process complexity [7]. In this paper we present a new concept of program-size complexity. What train of thought led us to it?

Following [8, Sec. VI, p. 7], think of a computer as decoder equipment at the receiving end of a noiseless binary communications channel. Think of its programs as code words, and of the result of a computation as the decoded message. Then it is natural to require that the programs/code words form what is called an "instantaneous code," so that successive messages sent across the channel (e.g. subroutines) can be separated. Instantaneous codes are well understood by information theorists [9–12]; they are governed by the Kraft inequality, which therefore plays a fundamental role in this paper.

This paper was written while the author was a visitor at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and was presented at the IEEE International Symposium on Information Theory, Notre Dame, Indiana, October 1974.

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

One is thus led to define the relative complexity $H(A, B/C, D)$ of A and B with respect to C and D to be the size of the shortest self-delimiting program for producing A and B from C and D. However, this is still not quite right. Guided by the analogy with information theory, one would like $H(A, B) = H(A) + H(B/A) + \Delta$ to hold with an error term Δ bounded in absolute value. But, as is shown in the Appendix, $|\Delta|$ is unbounded. So we stipulate instead that $H(A, B/C, D)$ is the size of the smallest self-delimiting program that produces A and B when it is given *a minimal-size self-delimiting program* for C and D. Then it can be shown that $|\Delta|$ is bounded.

In Sections 2–4 we define this new concept formally, establish the basic identities, and briefly consider the resulting concept of randomness or maximal entropy.

We recommend reading Willis [13]. In retrospect it is clear that he was aware of some of the basic ideas of this paper, though he developed them in a different direction. Chaitin's study [3, 4] of the state complexity of Turing machines may be of interest, because in his formalism programs can also be concatenated. To compare the properties of our entropy function H with those it has in information theory, see [9–12]; to contrast its properties with those of previous definitions of program-size complexity, see [14]. Cover [15] and Gewirtz [16] use our new definition. See [17–32] for other applications of information/entropy concepts.

2. *Definitions*

$X = \{\Lambda, 0, 1, 00, 01, 10, 11, 000, \cdots\}$ is the set of finite binary strings, and X^∞ is the set of infinite binary strings. Henceforth we shall merely say "string" instead of "binary string," and a string will be understood to be finite unless the contrary is explicitly stated. X is ordered as indicated, and $|s|$ is the length of the string s. The variables p, q, s, and t denote strings. The variables α and ω denote infinite strings. α_n is the prefix of α of length n. $N = \{0, 1, 2, \cdots\}$ is the set of natural numbers. The variables c, i, j, k, m, and n denote natural numbers. R is the set of positive rationals. The variable r denotes an element of R. We write "r.e." instead of "recursively enumerable," "lg" instead of "$\log_2$," and sometimes "$2 \uparrow (x)$" instead of "2^x." $\#(S)$ is the cardinality of the set S.

Concrete Definition of a Computer. A computer C is a Turing machine with two tapes, a program tape and a work tape. The program tape is finite in length. Its leftmost square is called the dummy square and always contains a blank. Each of its remaining squares contains either a 0 or a 1. It is a read-only tape, and has one read head on it which can move only to the right. The work tape is two-way infinite and each of its squares contains either a 0, a 1, or a blank. It has one read-write head on it.

At the start of a computation the machine is in its initial state, the program p occupies the whole program tape except for the dummy square, and the read head is scanning the dummy square. The work tape is blank except for a single string q whose leftmost symbol is being scanned by the read-write head. Note that q can be equal to Λ. In that case the read-write head initially scans a blank square. p can also be equal to Λ. In that case the program tape consists solely of the dummy square. See Figure 1.

During each cycle of operation the machine may halt, move the read head of the program tape one square to the right, move the read-write head of the work tape one square to the left or to the right, erase the square of the work tape being scanned, or write a 0 or a 1 on the square of the work tape being scanned. Then the machine changes state. The action performed and the next state are both functions of the present state and the contents of the two squares being scanned, and are indicated in two finite tables with nine columns and as many rows as there are states.

If the Turing machine eventually halts with the read head of the program tape scanning its rightmost square, then the computation is a success. If not, the computation is a failure. $C(p, q)$ denotes the result of the computation. If the computation is a failure, then $C(p, q)$ is undefined. If it is a success, then $C(p, q)$ is the string extending to the right

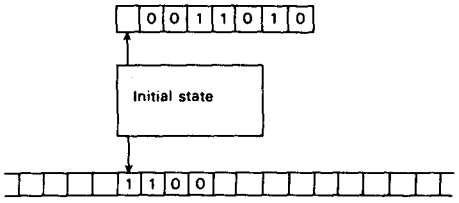

FIG. 1. The start of a computation. $p = 0011010$ and $q = 1100$.

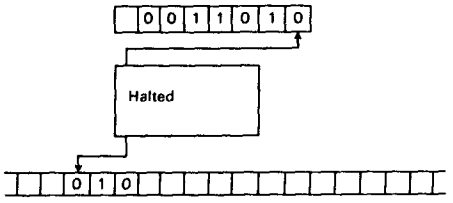

FIG. 2. The end of a successful computation. $C(p, q) = 010$.

from the square of the work tape that is being scanned to the first blank square. Note that $C(p, q) = \Lambda$ if the square of the work tape being scanned is blank. See Figure 2.

Definition of an Instantaneous Code. An instantaneous code is a set of strings S with the property that no string in S is a prefix of another.

Abstract Definition of a Computer. A computer is a partial recursive function $C : X \times X \to X$ with the property that for each q the domain of $C(\cdot, q)$ is an instantaneous code; i.e. if $C(p, q)$ is defined and p is a proper prefix of p', then $C(p', q)$ is not defined.

THEOREM 2.1. *The two definitions of a computer are equivalent.*

PROOF. Why does the concrete definition satisfy the abstract one? The program must indicate within itself where it ends since the machine is not allowed to run off the end of the tape or to ignore part of the program. Thus no program for a successful computation is the prefix of another.

Why does the abstract definition satisfy the concrete one? We show how a concrete computer C can simulate an abstract computer C'. The idea is that C should read another square of its program tape only when it is sure that this is necessary.

Suppose C found the string q on its work tape. C then generates the r.e. set $S = \{p \mid C'(p, q) \text{ is defined}\}$ on its work tape.

As it generates S, C continually checks whether or not that part p of the program that it has already read is a prefix of some known element s of S. Note that initially $p = \Lambda$.

Whenever C finds that p is a prefix of an $s \in S$, it does the following. If p is a proper prefix of s, C reads another square of the program tape. And if $p = s$, C calculates $C'(p, q)$ and halts, indicating this to be the result of the computation. Q.E.D.

Definition of an Optimal Universal Computer. U is an optimal universal computer iff for each computer C there is a constant $\text{sim}(C)$ with the following property: if $C(p, q)$ is defined, then there is a p' such that $U(p', q) = C(p, q)$ and $| p' | \leq | p | + \text{sim}(C)$.

THEOREM 2.2. *There is an optimal universal computer U.*

PROOF. U reads its program tape until it gets to the first 1. If U has read i 0's, it then simulates C_i, the ith computer (i.e. the computer with the ith pair of tables in a recursive enumeration of all possible pairs of defining tables), using the remainder of the program tape as the program for C_i. Thus if $C_i(p, q)$ is defined, then $U(0^i1p, q) = C_i(p, q)$. Hence U satisfies the definition of an optimal universal computer with $\text{sim}(C_i) = i + 1$. Q.E.D.

We somehow pick out a particular optimal universal computer U as the standard one for use throughout the rest of this paper.

Definition of Canonical Programs, Complexities, and Probabilities.

(a) The canonical program. $s^* = \min p(U(p, \Lambda) = s)$. I.e. s^* is the first element in the ordered set X of all strings that is a program for U to calculate s.

(b) Complexities.

$$H_C(s) = \min| p | (C(p, \Lambda) = s) \text{ (may be } \infty), \qquad H(s) = H_U(s),$$

$$H_C(s/t) = \min| p | (C(p, t^*) = s) \text{ (may be } \infty), \qquad H(s/t) = H_U(s/t).$$

(c) Probabilities.

$$P_C(s) = \sum 2^{-|p|} \ (C(p, \Lambda) = s), \qquad P(s) = P_U(s),$$
$$P_C(s/t) = \sum 2^{-|p|} \ (C(p, t^*) = s), \qquad P(s/t) = P_U(s/t).$$

Remark on Nomenclature. There are two different sets of terminology for these concepts, one derived from computational complexity and the other from information theory. $H(s)$ may be referred to as the information-theoretic or program-size complexity, and $H(s/t)$ may be referred to as the relative information-theoretic or program-size complexity. Or $H(s)$ and $H(s/t)$ may be termed the algorithmic entropy and the conditional algorithmic entropy, respectively. Similarly, this field might be referred to as "information-theoretic complexity" or as "algorithmic information theory."

Remark on the Definition of Probabilities. There is a very intuitive way of looking at the definition of P_C. Change the definition of the computer C so that the program tape is infinite to the right, and remove the (now impossible) requirement for a computation to be successful that the rightmost square of the program tape is being scanned when C halts. Imagine each square of the program tape except for the dummy square to be filled with a 0 or a 1 by a separate toss of a fair coin. Then the probability that the result s is obtained when the work tape is initially blank is $P_C(s)$, and the probability that the result s is obtained when the work tape initially has t^* on it is $P_C(s/t)$.

THEOREM 2.3.

(a) $H(s) \leq H_C(s) + sim(C)$,
(b) $H(s/t) \leq H_C(s/t) + sim(C)$,
(c) $s^* \neq \Lambda$,
(d) $s = U(s^*, \Lambda)$,
(e) $H(s) = |s^*|$,
(f) $H(s) \neq \infty$,
(g) $H(s/t) \neq \infty$,
(h) $0 \leq P_C(s) \leq 1$,
(i) $0 \leq P_C(s/t) \leq 1$,
(j) $1 \geq \sum_s P_C(s)$,
(k) $1 \geq \sum_s P_C(s/t)$,
(l) $P_C(s) \geq 2 \uparrow (-H_C(s))$,
(m) $P_C(s/t) \geq 2 \uparrow (-H_C(s/t))$,
(n) $0 < P(s) < 1$,
(o) $0 < P(s/t) < 1$,
(p) $\#(\{s \mid H_C(s) < n\}) < 2^n$,
(q) $\#(\{s \mid H_C(s/t) < n\}) < 2^n$,
(r) $\#(\{s \mid P_C(s) > r\}) < 1/r$,
(s) $\#(\{s \mid P_C(s/t) > r\}) < 1/r$.

PROOF. These are immediate consequences of the definitions. Q.E.D.

Definition of Tuples of Strings. Somehow pick out a particular recursive bijection $b : X \times X \to X$ for use throughout the rest of this paper. The 1-tuple $\langle s_1 \rangle$ is defined to be the string s_1. For $n \geq 2$ the n-tuple $\langle s_1, \cdots, s_n \rangle$ is defined to be the string $b(\langle s_1, \cdots, s_{n-1} \rangle, s_n)$.

Extensions of the Previous Concepts to Tuples of Strings ($n \geq 1$, $m \geq 1$).

$$H_C(s_1, \cdots, s_n) = H_C(\langle s_1, \cdots, s_n \rangle),$$
$$H_C(s_1, \cdots, s_n/t_1, \cdots, t_m) = H_C(\langle s_1, \cdots, s_n \rangle / \langle t_1, \cdots, t_m \rangle),$$
$$H(s_1, \cdots, s_n) = H_U(s_1, \cdots, s_n),$$
$$H(s_1, \cdots, s_n/t_1, \cdots, t_m) = H_U(s_1, \cdots, s_n/t_1, \cdots, t_m),$$
$$P_C(s_1, \cdots, s_n) = P_C(\langle s_1, \cdots, s_n \rangle),$$
$$P_C(s_1, \cdots, s_n/t_1, \cdots, t_m) = P_C(\langle s_1, \cdots, s_n \rangle / \langle t_1, \cdots, t_m \rangle),$$
$$P(s_1, \cdots, s_n) = P_U(s_1, \cdots, s_n),$$
$$P(s_1, \cdots, s_n/t_1, \cdots, t_m) = P_U(s_1, \cdots, s_n/t_1, \cdots, t_m).$$

Definition of the Information in One Tuple of Strings About Another ($n \geq 1$, $m \geq 1$).

$$I_C(s_1, \cdots, s_n : t_1, \cdots, t_m) = H_C(t_1, \cdots, t_m) - H_C(t_1, \cdots, t_m/s_1, \cdots, s_n),$$
$$I(s_1, \cdots, s_n : t_1, \cdots, t_m) = I_U(s_1, \cdots, s_n : t_1, \cdots, t_m).$$

Extensions of the Previous Concepts to Natural Numbers. We have defined H, P, and I for tuples of strings. This is now extended to tuples each of whose elements may either be a string or a natural number. We do this by identifying the natural number n with the nth string ($n = 0, 1, 2, \cdots$). Thus, for example, "$H(n)$" signifies "H (the nth element of X)," and "$U(p, \Lambda) = n$" stands for "$U(p, \Lambda) =$ the nth element of X."

3. *Basic Identities*

This section has two objectives. The first is to show that H and I satisfy the fundamental inequalities and identities of information theory to within error terms of the order of unity. For example, the information in s about t is nearly symmetrical. The second objective is to show that P is approximately a conditional probability measure: $P(t/s)$ and $P(s, t)/P(s)$ are within a constant multiplicative factor of each other.

The following notation is convenient for expressing these approximate relationships. $O(1)$ denotes a function whose absolute value is less than or equal to c for all values of its arguments. And $f \approx g$ means that the functions f and g satisfy the inequalities $cf \geq g$ and $f \leq cg$ for all values of their arguments. In both cases $c \in N$ is an unspecified constant.

THEOREM 3.1.

(*a*) $H(s, t) = H(t, s) + O(1)$,
(*b*) $H(s/s) = O(1)$,
(*c*) $H(H(s)/s) = O(1)$,
(*d*) $H(s) \leq H(s, t) + O(1)$,
(*e*) $H(s/t) \leq H(s) + O(1)$,
(*f*) $H(s, t) \leq H(s) + H(t/s) + O(1)$,
(*g*) $H(s, t) \leq H(s) + H(t) + O(1)$,
(*h*) $I(s : t) \geq O(1)$,
(*i*) $I(s : t) \leq H(s) + H(t) - H(s, t) + O(1)$,
(*j*) $I(s : s) = H(s) + O(1)$,
(*k*) $I(\Lambda : s) = O(1)$,
(*l*) $I(s : \Lambda) = O(1)$.

PROOF. These are easy consequences of the definitions. The proof of Theorem 3.1(f) is especially interesting, and is given below in full. Also, note that Theorem 3.1(g) follows immediately from Theorem 3.1(f, e), and Theorem 3.1(i) follows immediately from Theorem 3.1(f) and the definition of I.

Now for the proof of Theorem 3.1(f). We claim that there is a computer C with the following property. If $U(p, s^*) = t$ and $|p| = H(t/s)$ (i.e. if p is a minimal-size program for calculating t from s^*), then $C(s^*p, \Lambda) = \langle s, t \rangle$. By using Theorem 2.3(e, a) we see that $H_C(s, t) \leq |s^*p| = |s^*| + |p| = H(s) + H(t/s)$, and $H(s, t) \leq H_C(s, t) + \mathrm{sim}(C) \leq H(s) + H(t/s) + O(1)$.

It remains to verify the claim that there is such a computer. C does the following when it is given the program s^*p on its program tape and the string Λ on its work tape. First it simulates the computation that U performs when given the same program and work tapes. In this manner C reads the program s^* and calculates s. Then it simulates the computation that U performs when given s^* on its work tape and the remaining portion of C's program tape. In this manner C reads the program p and calculates t from s^*. The entire program tape has now been read, and both s and t have been calculated. C finally forms the pair $\langle s, t \rangle$ and halts, indicating this to be the result of the computation. Q.E.D.

Remark. The rest of this section is devoted to showing that the "$\leq$" in Theorems 3.1(f) and 3.1(i) can be replaced by "$=$." The arguments used to do this are more probabilistic than information-theoretic in nature.

THEOREM 3.2 (Extension of the Kraft inequality condition for the existence of an instantaneous code).

Hypothesis. Consider an effectively given list of finitely or infinitely many "requirements" $\langle s_k, n_k \rangle$ ($k = 0, 1, 2, \cdots$) *for the construction of a computer. The requirements are said to be "consistent" if* $1 \geq \sum_k 2 \uparrow (-n_k)$, *and we assume that they are consistent. Each requirement* $\langle s_k, n_k \rangle$ *requests that a program of length* n_k *be "assigned" to the result* s_k. *A computer* C *is said to "satisfy" the requirements if there are precisely as many programs* p *of length* n *such that* $C(p, \Lambda) = s$ *as there are pairs* $\langle s, n \rangle$ *in the list of requirements. Such a* C *must have the property that* $P_C(s) = \sum 2 \uparrow (-n_k)$ ($s_k = s$) *and* $H_C(s) = \min n_k$ ($s_k = s$).

Conclusion. There are computers that satisfy these requirements. Moreover, if we are given the requirements one by one, then we can simulate a computer that satisfies them. Hereafter we refer to the particular computer that the proof of this theorem shows how to simulate as the one that is "determined" by the requirements.

PROOF. (a) First we give what we claim is the (abstract) definition of a particular computer C that satisfies the requirements. In the second part of the proof we justify this claim.

As we are given the requirements, we assign programs to results. Initially all programs for C are available. When we are given the requirement $\langle s_k , n_k \rangle$ we assign *the first available program* of length n_k to the result s_k (first in the ordering which X was defined to have in Section 2). As each program is assigned, it and all its prefixes and extensions become unavailable for future assignments. Note that a result can have many programs assigned to it (of the same or different lengths) if there are many requirements involving it.

How can we simulate C? As we are given the requirements, we make the above assignments, and we simulate C by using the technique that was given in the proof of Theorem 2.1 for a concrete computer to simulate an abstract one.

(b) Now to justify the claim. We must show that the above rule for making assignments never fails, i.e. we must show that it is never the case that all programs of the requested length are unavailable. The proof we sketch is due to N. J. Pippenger.

A geometrical interpretation is necessary. Consider the unit interval [0, 1). The kth program of length n $(0 \leq k < 2^n)$ corresponds to the interval $[k2^{-n}, (k + 1)2^{-n})$. Assigning a program corresponds to assigning all the points in its interval. The condition that the set of assigned programs must be an instantaneous code corresponds to the rule that an interval is available for assignment iff no point in it has already been assigned. The rule we gave above for making assignments is to assign that interval $[k2^{-n}, (k + 1) \cdot 2^{-n})$ of the requested length 2^{-n} that is available that has the smallest possible k. Using this rule for making assignments gives rise to the following fact.

Fact. The set of those points in [0, 1) that are unassigned can always be expressed as the union of a finite number of intervals $[k_i 2 \uparrow (-n_i), (k_i + 1)2 \uparrow (-n_i))$ with the following properties: $n_i > n_{i+1}$, and $(k_i + 1)2 \uparrow (-n_i) \leq k_{i+1} 2 \uparrow (-n_{i+1})$. I.e. these intervals are disjoint, their lengths are *distinct* powers of 2, and they appear in [0, 1) in order of increasing length.

We leave to the reader the verification that this fact is always the case and that it implies that an assignment is impossible only if the interval requested is longer than the total length of the unassigned part of [0, 1), i.e. only if the requirements are inconsistent. Q.E.D.

THEOREM 3.3. (Recursive "estimates" for H_C and P_C). *Consider a computer C.*

(a) The set of all true propositions of the form "$H_C(s) \leq n$" is r.e. Given t^* one can recursively enumerate the set of all true propositions of the form "$H_C(s/t) \leq n$".

(b) The set of all true propositions of the form "$P_C(s) > r$" is r.e. Given t^* one can recursively enumerate the set of all true propositions of the form "$P_C(s/t) > r$".

PROOF. This is an easy consequence of the fact that the domain of C is an r.e. set. Q.E.D.

Remark. The set of all true propositions of the form "$H(s/t) \leq n$" is not r.e.; for if it were r.e., it would easily follow from Theorems 3.1(c) and 2.3(q) that Theorem 5.1(f) is false, which is a contradiction.

THEOREM 3.4. *For each computer C there is a constant c such that (a) $H(s) \leq -\lg P_C(s) + c$, (b) $H(s/t) \leq -\lg P_C(s/t) + c$.*

PROOF. It follows from Theorem 3.3(b) that the set T of all true propositions of the form "$P_C(s) > 2^{-n}$" is r.e., and that given t^* one can recursively enumerate the set T_t of all true propositions of the form "$P_C(s/t) > 2^{-n}$". This will enable us to use Theorem 3.2 to show that there is a computer C' with these properties:

(1) $H_{C'}(s) = \lceil -\lg P_C(s) \rceil + 1$, $P_{C'}(s) = 2 \uparrow (-\lceil -\lg P_C(s) \rceil)$,

(2) $H_{C'}(s/t) = \lceil -\lg P_C(s/t) \rceil + 1$, $P_{C'}(s/t) = 2 \uparrow (-\lceil -\lg P_C(s/t) \rceil)$.

Here $\lceil x \rceil$ denotes the least integer greater than x. By applying Theorem 2.3(a, b) to (1) and (2), we see that Theorem 3.4 holds with $c = \mathrm{sim}(C') + 2$.

How does the computer C' work? First of all, it checks whether it has been given Λ or t^* on its work tape. These two cases can be distinguished, for by Theorem 2.3(c) it is impossible for t^* to be equal to Λ.

(a) If C' has been given Λ on its work tape, it enumerates T and simulates the computer determined by all requirements of the form

(3) $\langle s, n+1 \rangle$ ("$P_C(s) > 2^{-n}$" $\in T$).

Thus $\langle s, n \rangle$ is taken as a requirement iff $n \geq \lceil -\lg P_C(s) \rceil + 1$. Hence the number of programs p of length n such that $C'(p, \Lambda) = s$ is 1 if $n \geq \lceil -\lg P_C(s) \rceil + 1$ and is 0 otherwise, which immediately yields (1).

However, we must check that the requirements (3) are consistent. $\sum 2^{-|p|}$ (over all programs p we wish to assign to the result s) $= 2 \uparrow (-\lceil -\lg P_C(s) \rceil) < P_C(s)$. Hence $\sum 2^{-|p|}$ (over all p we wish to assign) $< \sum_s P_C(s) \leq 1$ by Theorem 2.3(j). Thus the hypothesis of Theorem 3.2 is satisfied, the requirements (3) indeed determine a computer, and the proof of (1) and Theorem 3.4(a) is complete.

(b) If C' has been given t^* on its work tape, it enumerates T_t and simulates the computer determined by all requirements of the form

(4) $\langle s, n+1 \rangle$ ("$P_C(s/t) > 2^{-n}$" $\in T_t$).

Thus $\langle s, n \rangle$ is taken as a requirement iff $n \geq \lceil -\lg P_C(s/t) \rceil + 1$. Hence the number of programs p of length n such that $C'(p, t^*) = s$ is 1 if $n \geq \lceil -\lg P_C(s/t) \rceil + 1$ and is 0 otherwise, which immediately yields (2).

However, we must check that the requirements (4) are consistent. $\sum 2^{-|p|}$ (over all programs p we wish to assign to the result s) $= 2 \uparrow (-\lceil -\lg P_C(s/t) \rceil) < P_C(s/t)$. Hence $\sum 2^{-|p|}$ (over all p we wish to assign) $< \sum_s P_C(s/t) \leq 1$ by Theorem 2.3(k). Thus the hypothesis of Theorem 3.2 is satisfied, the requirements (4) indeed determine a computer, and the proof of (2) and Theorem 3.4(b) is complete. Q.E.D.

THEOREM 3.5.

(a) For each computer C there is a constant c such that $P(s) \geq 2^{-c} P_C(s)$, $P(s/t) \geq 2^{-c} P_C(s/t)$.

(b) $H(s) = -\lg P(s) + O(1)$, $H(s/t) = -\lg P(s/t) + O(1)$.

PROOF. Theorem 3.5(a) follows immediately from Theorem 3.4 using the fact that $P(s) \geq 2 \uparrow (-H(s))$ and $P(s/t) \geq 2 \uparrow (-H(s/t))$ (Theorem 2.3(l, m)). Theorem 3.5(b) is obtained by taking $C = U$ in Theorem 3.4 and also using these two inequalities. Q.E.D.

Remark. Theorem 3.5(a) extends Theorem 2.3(a, b) to probabilities. Note that Theorem 3.5(a) is not an immediate consequence of our weak definition of an optimal universal computer.

Theorem 3.5(b) enables one to reformulate results about H as results concerning P, and vice versa; it is the first member of a trio of formulas that will be completed with Theorem 3.9(e, f). These formulas are closely analogous to expressions in information theory for the information content of *individual* events or symbols [10, Secs. 2.3, 2.6, pp. 27–28, 34–37].

THEOREM 3.6.

(a) $\#(\{p \mid U(p, \Lambda) = s \;\&\; |p| \leq H(s) + n\}) \leq 2 \uparrow (n + O(1))$.

(b) $\#(\{p \mid U(p, t^*) = s \;\&\; |p| \leq H(s/t) + n\}) \leq 2 \uparrow (n + O(1))$.

PROOF. This follows immediately from Theorem 3.5(b). Q.E.D.

THEOREM 3.7. $P(s) \approx \sum_t P(s, t)$.

PROOF. On the one hand, there is a computer C such that $C(p, \Lambda) = s$ if $U(p, \Lambda) = \langle s, t \rangle$. Thus $P_C(s) \geq \sum_t P(s, t)$. Using Theorem 3.5(a), we see that $P(s) \geq 2^{-c} \sum_t P(s,t)$.

On the other hand, there is a computer C such that $C(p, \Lambda) = \langle s, s \rangle$ if $U(p, \Lambda) = s$.

Thus $\sum_t P_C(s, t) \geq P_C(s, s) \geq P(s)$. Using Theorem 3.5(a), we see that $\sum_t P(s, t) \geq 2^{-c}P(s)$. Q.E.D.

THEOREM 3.8. *There is a computer C and a constant c such that $H_C(t/s) = H(s, t) - H(s) + c$.*

PROOF. The set of all programs p such that $U(p, \Lambda)$ is defined is r.e. Let p_k be the kth program in a particular recursive enumeration of this set, and define s_k and t_k by $\langle s_k, t_k \rangle = U(p_k, \Lambda)$. By Theorems 3.7 and 3.5(b) there is a c such that $2 \uparrow (H(s) - c) \sum_t P(s, t) \leq 1$ for all s. Given s^* on its work tape, C simulates the computer C_s determined by the requirements $\langle t_k, | p_k | - | s^* | + c \rangle$ for $k = 0, 1, 2, \cdots$ such that $s_k = U(s^*, \Lambda)$. Recall Theorem 2.3(d, e). Thus for each p such that $U(p, \Lambda) = \langle s, t \rangle$ there is a corresponding p' such that $C(p', s^*) = C_s(p', \Lambda) = t$ and $| p' | = | p | - H(s) + c$. Hence $H_C(t/s) = H(s, t) - H(s) + c$.

However, we must check that the requirements for C_s are consistent. $\sum 2 \uparrow (- | p' |)$ (over all programs p' we wish to assign to any result t) $= \sum 2 \uparrow (- | p | + H(s) - c)$ (over all p such that $U(p, \Lambda) = \langle s, t \rangle$) $= 2 \uparrow (H(s) - c) \sum_t P(s, t) \leq 1$ because of the way c was chosen. Thus the hypothesis of Theorem 3.2 is satisfied, and these requirements indeed determine C_s. Q.E.D.

THEOREM 3.9.

(a) $H(s, t) = H(s) + H(t/s) + O(1)$.

(b) $I(s : t) = H(s) + H(t) - H(s, t) + O(1)$.

(c) $I(s : t) = I(t : s) + O(1)$.

(d) $P(t/s) \approx P(s, t)/P(s)$.

(e) $H(t/s) = \lg P(s)/P(s, t) + O(1)$.

(f) $I(s : t) = \lg P(s, t)/P(s)P(t) + O(1)$.

PROOF. Theorem 3.9(a) follows immediately from Theorems 3.8, 2.3(b), and 3.1(f). Theorem 3.9(b) follows immediately from Theorem 3.9(a) and the definition of $I(s : t)$. Theorem 3.9(c) follows immediately from Theorems 3.9(b) and 3.1(a). Theorem 3.9(d, e) follows immediately from Theorems 3.9(a) and 3.5(b). Theorem 3.9(f) follows immediately from Theorems 3.9(b) and 3.5(b). Q.E.D.

Remark. We thus have at our disposal essentially the entire formalism of information theory. Results such as these can now be obtained effortlessly:

$$H(s_1) \leq H(s_1/s_2) + H(s_2/s_3) + H(s_3/s_4) + H(s_4) + O(1),$$

$$H(s_1, s_2, s_3, s_4) = H(s_1/s_2, s_3, s_4) + H(s_2/s_3, s_4) + H(s_3/s_4) + H(s_4) + O(1).$$

However, there is an interesting class of identities satisfied by our H function that has no parallel in information theory. The simplest of these is $H(H(s)/s) = O(1)$ (Theorem 3.1(c)), which with Theorem 3.9(a) immediately yields $H(s, H(s)) = H(s) + O(1)$. This is just one pair of a large family of identities, as we now proceed to show.

Keeping Theorem 3.9(a) in mind, consider modifying the computer C used in the proof of Theorem 3.1(f) so that it also measures the lengths $H(s)$ and $H(t/s)$ of its subroutines s^* and p, and halts indicating $\langle s, t, H(s), H(t/s) \rangle$ to be the result of the computation instead of $\langle s, t \rangle$. It follows that $H(s, t) = H(s, t, H(s), H(t/s)) + O(1)$ and $H(H(s), H(t/s)/s, t) = O(1)$. In fact, it is easy to see that $H(H(s), H(t), H(t/s), H(s/t), H(s, t)/s, t) = O(1)$, which implies $H(I(s : t)/s, t) = O(1)$. And of course these identities generalize to tuples of three or more strings.

4. *A Random Infinite String*

The undecidability of the halting problem is a fundamental theorem of recursive function theory. In algorithmic information theory the corresponding theorem is as follows: The base-two representation of the probability that U halts is a random (i.e. maximally complex) infinite string. In this section we formulate this statement precisely and prove it.

THEOREM 4.1 (Bounds on the complexity of natural numbers).

(a) $\sum_n 2^{-H(n)} \leq 1$.

Consider a recursive function $f : N \to N$.

(b) *If* $\sum_n 2^{-f(n)}$ *diverges, then* $H(n) > f(n)$ *infinitely often.*

(c) *If* $\sum_n 2^{-f(n)}$ *converges, then* $H(n) \leq f(n) + O(1)$.

PROOF.

(a) By Theorem 2.3(l, j), $\sum_n 2^{-H(n)} \leq \sum_n P(n) \leq 1$.

(b) If $\sum_n 2^{-f(n)}$ diverges, and $H(n) \leq f(n)$ held for all but finitely many values of n, then $\sum_n 2^{-H(n)}$ would also diverge. But this would contradict Theorem 4.1(a), and thus $H(n) > f(n)$ infinitely often.

(c) If $\sum_n 2^{-f(n)}$ converges, there is an n_0 such that $\sum_{n \geq n_0} 2^{-f(n)} \leq 1$. By Theorem 3.2 there is a computer C determined by the requirements $\langle n, f(n) \rangle$ $(n \geq n_0)$. Thus $H(n) \leq f(n) + \mathrm{sim}(C)$ for all $n \geq n_0$. Q.E.D.

THEOREM 4.2 (Maximal complexity finite and infinite strings).

(a) $max\, H(s)(|\, s\,| = n) = n + H(n) + O(1)$.

(b) $\#(\{s \mid |\, s\,| = n \,\&\, H(s) \leq n + H(n) - k\}) \leq 2 \uparrow (n - k + O(1))$.

(c) *Imagine that the infinite string* α *is generated by tossing a fair coin once for each of its bits. Then, with probability one,* $H(\alpha_n) > n$ *for all but finitely many* n.

PROOF. Consider a string s of length n. By Theorem 3.9(a), $H(s) = H(n, s) + O(1) = H(n) + H(s/n) + O(1)$. We now obtain Theorem 4.2(a, b) from this estimate for $H(s)$.

There is a computer C such that $C(p, |\, p\,|^*) = p$ for all p. Thus $H(s/n) \leq n + \mathrm{sim}(C)$, and $H(s) \leq n + H(n) + O(1)$. On the other hand, by Theorem 2.3(q), fewer than 2^{n-k} of the s satisfy $H(s/n) < n - k$. Hence fewer than 2^{n-k} of the s satisfy $H(s) < n - k + H(n) + O(1)$. Thus we have obtained Theorem 4.2(a, b).

Now for the proof of Theorem 4.2(c). By Theorem 4.2(b), at most a fraction of $2 \uparrow (-H(n) + c)$ of the strings s of length n satisfy $H(s) \leq n$. Thus the probability that α satisfies $H(\alpha_n) \leq n$ is $\leq 2 \uparrow (-H(n) + c)$. By Theorem 4.1(a), $\sum_n 2 \uparrow (-H(n) + c)$ converges. Invoking the Borel-Cantelli lemma, we obtain Theorem 4.2(c). Q.E.D.

Definition of Randomness. A string s is random iff $H(s)$ is approximately equal to $|\, s\,| + H(|\, s\,|)$. An infinite string α is random iff $\exists c\ \forall n\ H(\alpha_n) > n - c$.

Remark. In the case of infinite strings there is a sharp distinction between randomness and nonrandomness. In the case of finite strings it is a matter of degree. To the question "How random is s?" one must reply indicating how close $H(s)$ is to $|\, s\,| + H(|\, s\,|)$.

C. P. Schnorr (private communication) has shown that this complexity-based definition of a random infinite string and P. Martin-Löf's statistical definition of this concept [7, pp. 379–380] are equivalent.

Definition of Base-Two Representations. The base-two representation of a real number $x \in (0, 1]$ is that unique infinite string $b_1 b_2 b_3 \cdots$ *with infinitely many* 1's such that $x = \sum_{k=1}^{\infty} b_k 2^{-k}$.

Definition of the Probability ω *that* U *Halts.* $\omega = \sum_s P(s) = \sum 2^{-|p|}$ ($U(p, \Lambda)$ is defined).

By Theorem 2.3(j, n), $\omega \in (0, 1]$. Therefore the real number ω has a base-two representation. Henceforth ω denotes both the real number and its base-two representation. Similarly, ω_n denotes a string of length n and a rational number $m/2^n$ with the property that $\omega > \omega_n$ and $\omega - \omega_n \leq 2^{-n}$.

THEOREM 4.3 (Construction of a random infinite string).

(a) *There is a recursive function* $w : N \to R$ *such that* $w(n) \leq w(n + 1)$ *and* $\omega = lim_{n \to \infty} w(n)$.

(b) ω *is random.*

(c) *There is a recursive predicate* $D : N \times N \times N \to \{true, false\}$ *such that the k-th bit of* ω *is a* 1 *iff* $\exists i\ \forall j\ D(i, j, k)$ $(k = 0, 1, 2, \cdots)$.

PROOF. $\{p \mid U(p, \Lambda)$ is defined$\}$ is r.e. Let p_k $(k = 0, 1, 2, \cdots)$ denote the kth p in a particular recursive enumeration of this set. Let $w(n) = \sum_{k \leq n} 2 \uparrow (-|\, p_k\,|)$. $w(n)$ tends monotonically to ω from below, which proves Theorem 4.3(a).

In view of the fact that $\omega > \omega_n \geq \omega - 2^{-n}$ (see the definition of ω), if one is given ω_n one can find an m such that $\omega \geq w(m) > \omega_n \geq \omega - 2^{-n}$. Thus $\omega - w(m) < 2^{-n}$, and $\{p_k \mid k \leq m\}$ contains all programs p of length less than or equal to n such that $U(p, \Lambda)$ is defined. Hence $\{U(p_k, \Lambda) \mid k \leq m \ \& \ |p_k| \leq n\} = \{s \mid H(s) \leq n\}$. It follows there is a computer C with the property that if $U(p, \Lambda) = \omega_n$, then $C(p, \Lambda)$ equals the first string s such that $H(s) > n$. Thus $n < H(s) \leq H(\omega_n) + \text{sim}(C)$, which proves Theorem 4.3(b).

To prove Theorem 4.3(c), define D as follows: $D(i, j, k)$ iff $j \geq i$ implies the kth bit of the base-two representation of $w(j)$ is a 1. Q.E.D.

Appendix. The Traditional Concept of Relative Complexity

In this Appendix programs are required to be self-delimiting, but the relative complexity $H(s/t)$ of s with respect to t will now mean that one is directly given t, instead of being given a minimal-size program for t.

The standard optimal universal computer U remains the same as before. H and P are redefined as follows:

$H_C(s/t) = \min |p| \ (C(p, t) = s)$ (may be ∞),
$H_C(s) = H_C(s/\Lambda)$,
$H(s/t) = H_U(s/t)$,
$H(s) = H_U(s)$,

$P_C(s/t) = \sum 2^{-|p|} \ (C(p, t) = s)$,
$P_C(s) = P_C(s/\Lambda)$,
$P(s/t) = P_U(s/t)$,
$P(s) = P_U(s)$.

These concepts are extended to tuples of strings and natural numbers as before. Finally, $\Delta(s, t)$ is defined as follows:

$$H(s, t) = H(s) + H(t/s) + \Delta(s, t).$$

THEOREM 5.1.

(a) $H(s, H(s)) = H(s) + O(1)$,
(b) $H(s, t) = H(s) + H(t/s, H(s)) + O(1)$,
(c) $-H(H(s)/s) - O(1) \leq \Delta(s, t) \leq O(1)$,
(d) $\Delta(s, s) = O(1)$,
(e) $\Delta(s, H(s)) = -H(H(s)/s) + O(1)$,
(f) $H(H(s)/s) \neq O(1)$.

PROOF. (a) On the one hand, $H(s, H(s)) \leq H(s) + c$ because a minimal-size program for s also tells one its length $H(s)$, i.e. because there is a computer C such that $C(p, \Lambda) = \langle U(p, \Lambda), |p| \rangle$ if $U(p, \Lambda)$ is defined. On the other hand, obviously $H(s) \leq H(s, H(s)) + c$.

(b) On the one hand, $H(s, t) \leq H(s) + H(t/s, H(s)) + c$ follows from Theorem 5.1 (a) and the obvious inequality $H(s, t) \leq H(s, H(s)) + H(t/s, H(s)) + c$. On the other hand, $H(s, t) \geq H(s) + H(t/s, H(s)) - c$ follows from the inequality $H(t/s, H(s)) \leq H(s, t) - H(s) + c$ analogous to Theorem 3.8 and obtained by adapting the methods of Section 3 to the present setting.

(c) This follows from Theorem 5.1(b) and the obvious inequality $H(t/s, H(s)) - c \leq H(t/s) \leq H(H(s)/s) + H(t/s, H(s)) + c$.

(d) If $t = s$, $H(s, t) - H(s) - H(t/s) = H(s, s) - H(s) - H(s/s) = H(s) - H(s) + O(1) = O(1)$, for obviously $H(s, s) = H(s) + O(1)$ and $H(s/s) = O(1)$.

(e) If $t = H(s)$, $H(s, t) - H(s) - H(t/s) = H(s, H(s)) - H(s) - H(H(s)/s) = -H(H(s)/s) + O(1)$ by Theorem 5.1(a).

(f) The proof is by *reductio ad absurdum*. Suppose on the contrary that $H(H(s)/s) < c$ for all s. First we adapt an idea of A. R. Meyer and D. W. Loveland [6, pp. 525–526] to show that there is a partial recursive function $f : X \to N$ with the property that if $f(s)$ is defined it is equal to $H(s)$ and this occurs for infinitely many values of s. Then we obtain the desired contradiction by showing that such a function f cannot exist.

Consider the set K_s of all natural numbers k such that $H(k/s) < c$ and $H(s) \leq k$. Note that $\min K_s = H(s)$, $\#(K_s) < 2^c$, and given s one can recursively enumerate K_s. Also, given s and $\#(K_s)$ one can recursively enumerate K_s until one finds all its elements,

and, in particular, its smallest element, which is $H(s)$. Let $m = \limsup \#(K_s)$, and let n be such that $|s| \geq n$ implies $\#(K_s) \leq m$.

Knowing m and n one calculates $f(s)$ as follows. First one checks if $|s| < n$. If so, $f(s)$ is undefined. If not, one recursively enumerates K_s until m of its elements are found. Because of the way n was chosen, K_s cannot have more than m elements. If it has less than m, one never finishes searching for m of them, and so $f(s)$ is undefined. However, if $\#(K_s) = m$, which occurs for infinitely many values of s, then one eventually realizes all of them have been found, including $f(s) = \min K_s = H(s)$. Thus $f(s)$ is defined and equal to $H(s)$ for infinitely many values of s.

It remains to show that such an f is impossible. As the length of s increases, $H(s)$ tends to infinity, and so f is unbounded. Thus given n and $H(n)$ one can calculate a string s_n such that $H(n) + n < f(s_n) = H(s_n)$, and so $H(s_n/n, H(n))$ is bounded. Using Theorem 5.1(b) we obtain $H(n) + n < H(s_n) \leq H(n, s_n) + c' \leq H(n) + H(s_n/n, H(n)) + c'' \leq H(n) + c'''$, which is impossible for $n \geq c'''$. Thus f cannot exist, and our initial assumption that $H(H(s)/s) < c$ for all s must be false. Q.E.D.

Remark. Theorem 5.1 makes it clear that the fact that $H(H(s)/s)$ is unbounded implies that $H(t/s)$ is less convenient to use than $H(t/s, H(s))$. In fact, R. Solovay (private communication) has announced that $\max H(H(s)/s)$ taken over all strings s of length n is asymptotic to $\lg n$. The definition of the relative complexity of s with respect to t given in Section 2 is equivalent to $H(s/t, H(t))$.

ACKNOWLEDGMENTS. The author is grateful to the following for conversations that helped to crystallize these ideas: C. H. Bennett, T. M. Cover, R. P. Daley, M. Davis, P. Elias, T. L. Fine, W. L. Gewirtz, D. W. Loveland, A. R. Meyer, M. Minsky, N. J. Pippenger, R. J. Solomonoff, and S. Winograd. The author also wishes to thank the referees for their comments.

REFERENCES

(Note. Reference [33] is not cited in the text.)

1. SOLOMONOFF, R. J. A formal theory of inductive inference. *Inform. and Contr. 7* (1964), 1–22, 224–254.
2. KOLMOGOROV, A. N. Three approaches to the quantitative definition of information. *Problems of Inform. Transmission 1*, 1 (Jan.–March 1965), 1–7.
3. CHAITIN, G. J. On the length of programs for computing finite binary sequences. *J. ACM 13*, 4 (Oct. 1966), 547–569.
4. CHAITIN, G. J. On the length of programs for computing finite binary sequences: Statistical considerations. *J. ACM 16*, 1 (Jan. 1969), 145–159.
5. KOLMOGOROV, A. N. On the logical foundations of information theory and probability theory. *Problems of Inform. Transmission 5*, 3 (July–Sept. 1969), 1–4.
6. LOVELAND, D. W. A variant of the Kolmogorov concept of complexity. *Inform. and Contr. 15* (1969), 510–526.
7. SCHNORR, C. P. Process complexity and effective random tests. *J. Comput. and Syst. Scis. 7* (1973), 376–388.
8. CHAITIN, G. J. On the difficulty of computations. *IEEE Trans. IT-16* (1970), 5–9.
9. FEINSTEIN, A. *Foundations of Information Theory*. McGraw-Hill, New York, 1958.
10. FANO, R. M. *Transmission of Information*. Wiley, New Nork, 1961.
11. ABRAMSON, N. *Information Theory and Coding*. McGraw-Hill, New York, 1963.
12. ASH, R. *Information Theory*. Wiley-Interscience, New York, 1965.
13. WILLIS, D. G. Computational complexity and probability constructions. *J. ACM 17*, 2 (April 1970), 241–259.
14. 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. *Russ. Math. Survs. 25*, 6 (Nov.–Dec. 1970), 83–124.
15. COVER, T. M. Universal gambling schemes and the complexity measures of Kolmogorov and Chaitin. Rep. No. 12, Statistics Dep., Stanford U., Stanford, Calif., 1974. Submitted to *Ann. Statist.*
16. GEWIRTZ, W. L. Investigations in the theory of descriptive complexity. Ph.D. Thesis, New York University, 1974 (to be published as a Courant Institute rep.).

17. WEISS, B. The isomorphism problem in ergodic theory. *Bull. Amer. Math. Soc. 78* (1972), 668–684.
18. RÉNYI, A. *Foundations of Probability*. Holden-Day, San Francisco, 1970.
19. FINE, T. L. *Theories of Probability: An Examination of Foundations*. Academic Press, New York, 1973.
20. COVER, T. M. On determining the irrationality of the mean of a random variable. *Ann. Statist. 1* (1973), 862–871.
21. CHAITIN, G. J. Information-theoretic computational complexity. *IEEE Trans. IT-20* (1974), 10–15.
22. LEVIN, M. Mathematical Logic for Computer Scientists. Rep. TR-131, M.I.T. Project MAC, 1974, pp. 145–147, 153.
23. CHAITIN, G. J. Information-theoretic limitations of formal systems. *J. ACM 21*, 3 (July 1974), 403–424.
24. MINSKY, M. L. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N. J., 1967, pp. 54, 55, 66.
25. MINSKY, M., AND PAPERT, S. *Perceptrons: An Introduction to Computational Geometry*. M.I.T. Press, Cambridge, Mass., 1969, pp. 150, 153.
26. SCHWARTZ, J. T. On Programming: An Interim Report on the SETL Project. Installment I: Generalities. Lecture Notes, Courant Institute, New York University, New York, 1973, pp. 1–20.
27. BENNETT, C. H. Logical reversibility of computation. *IBM J. Res. Develop. 17* (1973), 525–532.
28. DALEY, R. P. The extent and density of sequences within the minimal-program complexity hierarchies. *J. Comput. and Syst. Scis.* (to appear).
29. CHAITIN, G. J. Information-theoretic characterizations of recursive infinite strings. Submitted to *Theoretical Comput. Sci.*
30. ELIAS, P. Minimum times and memories needed to compute the values of a function. *J. Comput. and Syst. Scis.* (to appear).
31. ELIAS, P. Universal codeword sets and representations of the integers. *IEEE Trans. IT* (to appear).
32. HELLMAN, M. E. The information theoretic approach to cryptography. Center for Systems Research, Stanford U., Stanford, Calif., 1974.
33. CHAITIN, G. J. Randomness and mathematical proof. *Sci. Amer. 232*, 5 (May 1975), in press.

RECEIVED APRIL 1974; REVISED DECEMBER 1974