ON THE SIMPLICITY AND SPEED OF PROGRAMS FOR COMPUTING INFINITE SETS OF NATURAL NUMBERS

Journal of the ACM 16 (1969), pp. 407–422

Gregory J. Chaitin[1]
Buenos Aires, Argentina

Abstract

It is suggested that there are infinite computable sets of natural numbers with the property that no infinite subset can be computed more simply or more quickly than the whole set. Attempts to establish this without restricting in any way the computer involved in the calculations are not

entirely successful. A hypothesis concerning the computer makes it possible to exhibit sets without simpler subsets. A second and analogous hypothesis then makes it possible to prove that these sets are also without subsets which can be computed more rapidly than the whole set. It is then demonstrated that there are computers which satisfy both hypotheses. The general theory is momentarily set aside and a particular Turing machine is studied. Lastly, it is shown that the second hypothesis is more restrictive then requiring the computer to be capable of calculating all infinite computable sets of natural numbers.

Key Words and Phrases:

computational complexity, computable set, recursive set, Turing machine, constructive ordinal, partially ordered set, lattice

CR Categories:

5.22

Introduction

Call a set of natural numbers *perfect* if there is no way to compute infinitely many of its members essentially better (i.e. simpler or quicker) than computing the whole set. The thesis of this paper is that perfect sets exist. This thesis was suggested by the following vague and imprecise considerations.

One of the most profound problems of the theory of numbers is that of calculating large primes. While the sieve of Eratosthenes appears to be as simple and as quick an algorithm for calculating all the primes as is possible, in recent times hope has centered on calculating large primes by calculating a subset of the primes, those that are Mersenne numbers. Lucas's test is simple and can test whether or not a Mersenne number is a prime with rapidity far greater than is furnished by the sieve method. If there are an infinity of Mersenne primes, then it appears that Lucas

[1]Address: Mario Bravo 249, Buenos Aires, Argentina.

has achieved a decisive advance in this classical problem of the theory of numbers.[2]

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

1. General Considerations

The notation and terminology of this paper are largely taken from Davis [3].

Definition 1. A computing machine Σ is defined by a 2-ary non-vanishing computable function σ in the following manner. The natural number n is part of the output $\Sigma(p,t)$ of the computer Σ at time t resulting from the program p if and only if the nth prime[3] divides $\sigma(p,t)$. The infinite set $\Sigma(p)$ of natural numbers which the program p causes the computing machine Σ to calculate is defined to be

$$\bigcup_t \Sigma(p,t)$$

if infinitely many numbers are put out by the computer in numerical order and without any repetition. Otherwise, $\Sigma(p)$ is undefined.

Definition 2. A program complexity measure Π is a computable 1-ary function with the property that only finitely many programs p have the same complexity $\Pi(p)$.

Definition 3. The complexity $\Pi_\Sigma(S)$ of an infinite computable set S of natural numbers as computed by the computer Σ under the complexity measure Π is defined to be equal to

$$\begin{cases} \min_{\Sigma(p)=S} \Pi(p), & \text{if there are such } p, \\ \infty, & \text{otherwise.} \end{cases}$$

[2]For Lucas's test, cf. Hardy and Wright [1, Sec. 15.5]. For a history of number theory, cf. Dantzig [2], especially Sections 3.12 and B.8.

[3]The 0th prime is 2, the 1st prime is 3, etc. The primes are, of course, used here only for the sake of convenience.

I.e. $\Pi_\Sigma(S)$ is the complexity of the simplest program which causes the computer to calculate S, and if there is no such program,[4] the complexity is infinite.[5]

In this section we do not see any compelling reason for regarding any particular computing machine and program complexity measure as most closely representing the state of affairs with which number theorists are confronted in their attempts to compute large primes as simply and as quickly as possible.[6] The four theorems of this section and their extensions hold for any computer Σ and any program complexity measure Π. Thus, although we don't know which computer and complexity measure to select, as this section holds true for all of them, we are covered.

Theorem 1. For any natural number n, there exists an infinite computable set S of natural numbers which has the following properties:

(a) $\Pi_\Sigma(S) > n$.

(b) For any infinite computable set R of natural numbers, $R \subset S$ implies $\Pi_\Sigma(R) \geq \Pi_\Sigma(S)$.

Proof. We first prove the existence of an infinite computable set A of natural numbers having no infinite computable subset B such that $\Pi_\Sigma(B) \leq n$. The infinite computable sets C of natural numbers for which $\Pi_\Sigma(C) \leq n$ are finite in number. Each such C has a smallest element c. Let the (finite) set of all these c be denoted by D. We take $A = \overline{D}$.

Now let $A_0, A_1, A_2, \ldots$ be the infinite computable subsets of A. Consider the following set:

$$E = \{\Pi_\Sigma(A_0), \Pi_\Sigma(A_1), \Pi_\Sigma(A_2), \ldots\}.$$

[4]This possibility can never arise for the simple-program computers or the quick-program computers introduced later; such computers can be programmed to compute any infinite computable set of natural numbers.

[5]A more formal definition would perhaps use ω, the first transfinite ordinal, instead of ∞.

[6]In Sections 2 and 3 the point of view is different; some computing machines are dismissed as degenerate cases and an explicit choice of program complexity function is suggested.

From the manner in which A was constructed, we know that each member of E is greater than n. And as the natural numbers are well-ordered, we also know that E has a smallest element r. There exists a natural number s such that $\Pi_\Sigma(A_s) = r$. We take $S = A_s$, and we are finished. Q.E.D.

Theorem 2. For any natural number n and any infinite computable set T of natural numbers with infinite complement, there exists a computable set S of natural numbers which has the following property: $T \subset S$ and $\Pi_\Sigma(S) > n$.

Proof. There are infinitely many computable sets of natural numbers which have T as a subset, but the infinite computable sets F of natural numbers for which $\Pi_\Sigma(F) \leq n$ are finite in number. Q.E.D.

Theorem 3. For any 1-ary computable function f, there exists an infinite computable set S of natural numbers which has the following property: $\Sigma(p) \subset S$ implies the existence of a t_0 such that for $t > t_0$, $n \in \Sigma(p, t)$ only if $t > f(n)$.

Proof. We describe a procedure for computing S in successive stages (each stage being divided into two successive steps); during the kth stage it is determined in the following manner whether or nor $k \in S$. Two subsets of the computing machine programs p such that $p < [k/4]$ are considered: set A, consisting of those programs which have been "eliminated" during some stage previous to the kth; and set B, consisting of those programs not in A which cause Σ to output the natural number k during the first $f(k)$ time units of calculation.

STEP 1. Put k in S if and only if B is empty.

STEP 2. Eliminate all programs in B (i.e. during all future stages they will be in A).

The above constructs S. That S contains infinitely many natural numbers follows from the fact that up to the kth stage at most $k/4$ programs have been eliminated, and thus at most $k/4$ natural numbers less than or equal to k can fail to be in S.[7]

[7] I.e. the Schnirelman density $d(S)$ of S is greater than or equal to $3/4$. It follows from $d(S) \geq 3/4$ that S is basis of the second order; i.e. every natural number can be expressed as the sum of two elements of S. Cf. Gelfond and Linnik [4, Sec. 1.1]. We conclude that the mere fact that a set is a basis of the second order for the natural numbers does not provide a quick means for computing infinitely many of its members.

It remains to show that $\Sigma(p) \subset S$ implies the existence of a t_0 such that for $t > t_0$, $n \in \Sigma(p,t)$ only if $t > f(n)$. Note that for $n \geq 4p+4$, $n \in \Sigma(p,t)$ only if $t > f(n)$. For a value of n for which this failed to be the case would assure p's being in A, which is impossible. Thus given a program p such that $\Sigma(p) \subset S$, we can calculate a point at which the program has become slow and will remain so; i.e. we can calculate a permissible value for t_0. In fact, $t_0(p) = \max_{j<4p+4} f(j)$. Q.E.D.

The following theorem and the type of diagonal process used in its proof are similar in some ways to Blum's exposition of a theorem of Rabin in [5, pp. 241–242].

Theorem 4. For any 1-ary computable function f and any infinite computable set T of natural numbers with infinite complement, there exists an infinite computable set S of natural numbers which is a superset of T and which has the following property: $\Sigma(p) = S$ implies the existence of a t_0 such that for $t > t_0$, $n \in \Sigma(p,t)$ only if $t > f(n)$.

Proof. First we define three functions: $a(n)$ is equal to the nth natural number in $\overline{T}$; $b(n)$ is equal to the smallest natural number j greater than or equal to n such that $j \in T$ and $j+1 \notin T$; and $c(n)$ is equal to $\max_{n \leq k \leq b(n)} f(k)$. As proof, we give a process for computing $S \cap \overline{T}$ in successive stages; during the kth stage it is determined in the following manner whether or not $a(k) \in S$. Consider the computing machine programs $0, 1, 2, \ldots, k$ to fall into two mutually exclusive sets: set A, consisting of those programs which have been eliminated during some stage previous to the kth; and set B, consisting of all others.

Step 1. Determine the set C consisting of the programs in B which cause the computing machine Σ to output during the first $c(a(k))$ time units of calculation any natural numbers greater than or equal to $a(k)$ and less then or equal to $b(a(k))$.

Step 2. Check whether C is empty. Should $C = \emptyset$, we neither eliminate programs nor put $a(k)$ in S; we merely proceed to the next (the $(k+1)$-th) stage. Should $C = \emptyset$, however, we proceed to step 3.

Step 3. We determine p_0, the smallest natural number in C.

Step 4. We ask, "Does the program p_0 cause Σ to output the number $a(k)$ during the first $c(a(k))$ time units of calculation?" According as the answer is "no" or "yes" we do or don't put $a(k)$ in S.

Step 5. Eliminate p_0 (i.e. during future stages p_0 will be in A).

The above constructs S. We leave to the reader the verification that

the constructed S has the desired properties. Q.E.D.

We now make a number of remarks.

Remark 1. We have actually proved somewhat more. Let U be any infinite computable set of natural numbers. Theorems 1 and 3 hold even if it is required that the set S whose existence is asserted be a subset of U. And if in Theorems 2 and 4 we make the additional assumption that T is a subset of U, and $U \cap \overline{T}$ is infinite, then we can also require that S be a subset of U.

The above proofs can practically be taken word for word (with obvious changes which may loosely be summarized by the command "ignore natural numbers not in U") as proofs for these extended theorems. It is only necessary to keep in mind the essential point, which in the case of Theorem 3 assumes the following form. If during the kth stage of the diagonal process used to construct S we decide whether to put in S the kth element of U, we are still sure that $\Sigma(p) \subset S$ is impossible for all the p which were eliminated before. For if $\Sigma(p) \subset U$, then p is eliminated as before; while if $\Sigma(p)$ has elements not in U, then it is clear that $\Sigma(p) \subset S$ is impossible, for S is a subset of U.

Remark 2. In Theorems 1 and 2 we see two possible extremes for S. In Theorem 1 we contemplate an arbitrarily complex infinite computable set of natural numbers that has the property that there is no way to compute infinitely many of its members which is simpler than computing the whole set. On the other hand, in Theorem 2 we contemplate an infinite computable set of natural numbers that has the property that there is a way to compute infinitely many of its members which is very much simpler than computing the whole set. Theorems 3 and 4 are analogous to Theorems 1 and 2, but Theorem 3 does not go as far as Theorem 1. Although Theorem 3 asserts the existence of infinite computable sets of natural numbers which have no infinite subsets which can be computed quickly, it does not establish that no infinite subset can be computed more quickly than the whole set. In this generality we are unable to demonstrate a Theorem 3 truly analogous to Theorem 1, although an attempt to do so is made in Remark 5.

Remark 3. The restriction in the conclusions of Theorems 3 and 4 that t be greater than t_0 is necessary. For as Arbib remarks in [6, p. 8], in some computers Σ any finite part of S can be computed very quickly by a table look-up procedure.

Remark 4. The 1-ary computable function f of Theorems 3 and 4 can go to infinity very quickly indeed with increasing values of its argument. For example, let $f_0(n) = 2^n$, $f_{k+1}(n) = f_k(f_k(n))$. For each k, $f_{k+1}(n)$ is greater than $f_k(n)$ for all but a finite number of values of n. We may now proceed from finite ordinal subscripts to the first transfinite ordinal by a diagonal process: $f_\omega(n) = \max_{k\le n} f_k(n)$. We choose to continue the process up to ω^2 in the following manner, which is a natural way to proceed (i.e. the fundamental sequences can be computed by simple programs) but which is by no means the only way to get to ω^2. i and j denote finite ordinals.

$$\begin{aligned} f_{\omega i+j+1}(n) &= f_{\omega i+j}(f_{\omega i+j}(n)), \\ f_{\omega(i+1)}(n) &= \max_{k\le n} f_{\omega i+k}(n), \\ f_{\omega^2}(n) &= \max_{k\le n} f_{\omega k}(n). \end{aligned}$$

Taking $f = f_{\omega^2}$ in Theorem 3 yields an S such that any attempt to compute infinitely many of its elements requires an amount of time which increases almost incomprehensibly quickly with the size of the elements computed.

More generally, the above process may be continued through to any constructive ordinal.[8] For example, there are more or less natural manners to reach ϵ_0, the first epsilon-number; the territory up to it is very well charted.[9]

The above is essentially a constructive version of remarks by Borel [8] in an appendix on a theorem of P. du Bois-Reymond. These remarks are partly reproduced in Hardy [9].

Remark 5. Remark 4 suggests the following approach to the speed of programs. For any constructive ordinal α there is a computable 2-ary function f (by no means unique) with the property that the set of 1-ary functions f_k defined by $f_k(n) = f(k, n)$ is a representative of α when ordered in such a manner that a function g comes before a function h if and only if $g(n) < h(n)$ holds for all but a finite number of values of n. We now associate an ordinal $\mathrm{Ord}_\Sigma(S)$ with each infinite computable set S of natural numbers in accordance with the following rules:

[8] Cf. Davis [3, Sec. 11.4] for a definition of the concept of a constructive ordinal number.

[9] Cf. Fraenkel [7, pp. 207–208].

(a) $\mathrm{Ord}_\Sigma(S)$ equals the smallest ordinal $\beta < \alpha$ such that f_{k_0}, the βth element of the set of functions f_k, has the following property: There exists a program p and a time t_0 such that $\Sigma(p) = S$ and for $t > t_0$, $n \in \Sigma(p, t)$ only if $t \leq f_{k_0}(n)$.

(b) If (a) fails to define $\mathrm{Ord}_\Sigma(S)$ (i.e. if the set of ordinals β is empty), then $\mathrm{Ord}_\Sigma(S) = \alpha$.

Then for any constructive ordinal α we have the following analogue to Theorem 1.

Theorem 1′. Any infinite computable set T of natural numbers has an infinite computable subset S with the following properties:

(a) $\mathrm{Ord}_\Sigma(S) \leq \mathrm{Ord}_\Sigma(T)$.

(b) For any infinite computable set R of natural numbers, $R \subset S$ implies $\mathrm{Ord}_\Sigma(S) \leq \mathrm{Ord}_\Sigma(R)$.

Proof. Let $T_0, T_1, T_2, \ldots$ be the infinite computable subsets of T. Consider the following set of ordinal numbers less than or equal to α:

$$\{\mathrm{Ord}_\Sigma(T_0), \mathrm{Ord}_\Sigma(T_1), \mathrm{Ord}_\Sigma(T_2), \ldots\}.$$

As the ordinal numbers less than or equal to α are well-ordered, this set has a smallest element β. There exists a natural number s such that $\mathrm{Ord}_\Sigma(T_s) = \beta$. We take $S = T_s$. Q.E.D.

However, we must admit that this approach to the speed of programs does not seem to be a convincing support for the thesis of this paper.

2. Connected Sets, Simple-Program Computers, and Quick-Program Computers

The principal results of subsections 2.A and 2.B, namely, Theorems 6 and 8, hold only for certain computers, but we argue that all other computing machines are degenerate cases of computers which in view of their unnecessarily restricted capabilities do not merit consideration.

In this section and the next we attempt to make plausible the contention that some connected sets (defined below) may well be considered to be perfect sets. In subsection 2.A we study the complexity of subsets of connected sets, and in subsection 2.B we study the speed of programs for computing subsets of connected sets. The treatments are analogous but we find the second more convincing, because in the first treatment one explicit choice is made for the program complexity measure Π. $\Pi(p)$ is taken to be $[\log_2(p+1)]$.

The concept of a connected set is analogous to the concept of a retraceable set, cf. Dekker and Myhill [10].

Definition 4. A connecting function γ is a one-to-one onto mapping carrying the set of all finite sets of natural numbers onto the set of all natural numbers. The monotonicity conditions $\gamma(V \cup W) \geq \gamma(W)$ must be satisfied and there must be a 1-ary computable function g such that $\gamma(W) = g(\prod_{n \in W} p_n)$, where p_n denotes the nth prime.[2] Let $S = \{s_0, s_1, s_2, \ldots\}$ $(s_0 < s_1 < s_2 < \cdots)$ be a computable set of natural numbers with m members $(0 \leq m \leq \aleph_0)$. From a connecting function γ we define a secondary connecting function Γ as follows:[10]

$$\Gamma(S) = \bigcup_{k<m} \{\gamma(\bigcup_{j \leq k} \{s_j\})\}.$$

A γ-connected set is defined to be any infinite computable set of natural numbers which is in the range of Γ.

Remark 6. Consider a connecting function γ. Note that any two γ-connected sets which have an infinite intersection must be identical. In fact, two γ-connected sets which have an element in common must be identical up to that element.

The following important results concerning γ-connected sets are established by the methods of Section 1 and thus hold for any computer Σ and complexity measure Π. For any natural number n there exists a γ-connected set S such that $\Pi_\Sigma(S) > n$. This follows from the fact that there are infinitely many γ-connected sets, while the infinite computable sets H of natural numbers such that $\Pi_\Sigma(H) \leq n$ are only finite in number. Theorem 3 remains true if we require that the set S whose

[10] Thus $\Gamma(S)$ always has the same number of elements as S, be S empty, finite or infinite.

existence is asserted be a γ-connected set. S may be constructed by a procedure similar to that of the proof of Theorem 3; during the kth stage instead of deciding whether or not $k \in S$, it is decided whether or not $k \in \Gamma^{-1}(S)$. These two results should be kept in mind while appraising the extent to which the theorems of this section and the next corroborate the thesis of this paper.

2.A. Simplicity

In this subsection we make one explicit choice for the program complexity measure Π. We consider programs to be finite binary sequences as well as natural numbers:

	Programs										
Binary Sequence	Λ	0	1	00	01	10	11	000	001	010	011...
Natural Number	0	1	2	3	4	5	6	7	8	9	10...

Henceforth, when we denote a program by a lowercase (uppercase) Latin letter, we are referring to the program considered as a natural number (binary sequence). Next we define the complexity of a program P to be the number of bits in P (i.e. its length). I.e. the complexity $\Pi(p)$ of a program p is equal to $[\log_2(p+1)]$, the greatest integer not greater than the base-2 logarithm of $p+1$.

We now introduce the simple-program computers. Computers similar to them have been used in Solomonoff [11], Kolmogorov [12], and in [13].

Definition 5. A simple-program computer Σ has the following property: For any computer Ξ, there exists a natural number $c_{\Sigma\Xi}$ such that $\Pi_\Sigma(S) \leq \Pi_\Xi(S) + c_{\Sigma\Xi}$ for all infinite computable sets S of natural numbers.

To the extent that it is plausible to consider all computer programs to be binary sequences, it seems plausible to consider all computers which are not simple-program computers as unnecessarily awkward degenerate cases which are unworthy of attention.

Remark 7. Note that if Σ and Ξ are two simple-program computers, then there exists a natural number c_Ξ^Σ which has the following property: $|\Pi_\Sigma(S) - \Pi_\Xi(S)| \leq c_\Xi^\Sigma$ for all infinite computable sets S of

natural numbers. In fact we can take

$$c_{\Xi}^{\Sigma} = \max(c_{\Sigma\Xi}, c_{\Xi\Sigma}).$$

Theorem 5. For any connecting function γ, there exists a simple-program computer Σ^γ which has the following property: For any γ-connected set S and any infinite computable subset R of S,

$$\Pi_{\Sigma^\gamma}(S) \leq \Pi_{\Sigma^\gamma}(R).$$

Proof. Taking for granted the existence of a simple program computer Σ^* (cf. Theorem 9), we construct the computer Σ^γ from it as follows:

$$\left\{ \begin{array}{lcl} \Sigma^\gamma(\Lambda, t) & = & \emptyset, \\ \Sigma^\gamma(P0, t) & = & \Sigma^*(P, t), \\ \Sigma^\gamma(P1, t) & = & \bigcap_{t'<t} \overline{\Sigma^\gamma(P1, t')} \cap \Gamma(\bigcup_{n \in \Sigma^*(P,t)} \gamma^{-1}(n)). \end{array} \right. \tag{1}$$

As Σ^* is a simple-program computer, so is Σ^γ, for $\Sigma^\gamma(P0, t) = \Sigma^*(P, t)$. Σ^γ also has the following very important property: For all programs $P0$ for which $\Sigma^\gamma(P0)$ is a subset of some γ-connected set S, $\Sigma^\gamma(P1) = S$. Moreover, $\Sigma^\gamma(P1)$ cannot be a proper subset of any γ-connected set. In summary, given a P such that $\Sigma^\gamma(P)$ is a proper subset of a γ-connected set S, then by changing the rightmost bit of P to a 1 we get a program P' with the property that $\Sigma^\gamma(P') = S$. This implies that for any infinite computable subset R of a γ-connected set S,

$$\Pi_{\Sigma^\gamma}(S) \leq \Pi_{\Sigma^\gamma}(R).$$

Q.E.D.

In view of Remark 7, the following theorem is merely a corollary to Theorem 5.

Theorem 6. Consider a simple-program computer Σ. For any connecting function γ, there exists a natural number c_γ which has the following property: For any γ-connected set S and any infinite computable subset R of S, $\Pi_\Sigma(S) \leq \Pi_\Sigma(R) + c_\gamma$. In fact, we can take[11]

$$c_\gamma = 2 \max(c_{\Sigma\Sigma^\gamma}, c_{\Sigma^\gamma\Sigma}).$$

[11] That

$$c_\gamma = c_{\Sigma\Sigma^\gamma} + c_{\Sigma^\gamma\Sigma}$$

will do follows upon taking a slightly closer look at the matter.

2.B. Speed

This treatment runs parallel to that of subsection 2.A.

Definition 6. A quick-program computer Σ has the following property: For any computer Ξ, there exists a 1-ary computable function $s_{\Sigma\Xi}$ such that for all programs p for which $\Xi(p)$ is defined, there exists a program p' such that $\Sigma(p') = \Xi(p)$ and

$$\bigcup_{t' \leq t} \Xi(p, t') \subset \bigcup_{t' \leq s_{\Sigma\Xi(t)}} \Sigma(p', t')$$

for all but a finite number of values of t.

Theorem 7. For any connecting function γ, there exists a quick-program computer Σ^γ which has the following property: For any program P such that $\Sigma^\gamma(P)$ is a proper subset of a γ-connected set S, there exists a program P' such that $\Sigma^\gamma(P') = S$ and $\Sigma^\gamma(P, t) \subset \Sigma^\gamma(P', t)$ for all t. In fact, P' is just P with the 0 at its right end changed to a 1, as the reader has no doubt guessed.

Proof. Taking for granted the existence of a quick-program computer Σ^* (cf. Theorem 9), we construct Σ^γ from it exactly as in the proof of Theorem 5. I.e. Σ^γ is defined, as before, by eqs. (1). The remainder of the proof parallels the proof of Theorem 5. Q.E.D.

Theorem 7 yields the following corollary in a manner analogous to the manner in which Theorem 5 yields Theorem 6.

Theorem 8. Consider a quick-program computer Σ. For any connecting function γ there exists a 1-ary computable function s_γ which has the following property: For any program p such that $\Sigma(p)$ is a subset of a γ-connected set S, there exists a program p' such that

$$\begin{aligned} \Sigma(p') &= S, \\ \bigcup_{t' \leq t} \Sigma(p, t') &\subset \bigcup_{t' \leq s_\gamma(t)} \Sigma(p', t') \end{aligned}$$

for all but a finite number of values of t. In fact we can take

$$s_\gamma(n) = s_{\Sigma\Sigma^\gamma}(s_{\Sigma^\gamma\Sigma}(n)).$$

Remark 8. Arbib and Blum [14] base their treatment of program speed upon the idea that if two computers can imitate act by act the

computations of the other, and not take too many time units of calculation to imitate the first several time units of calculation of the other, then these computers are essentially equivalent. The idea used to derive Theorem 8 from Theorem 7 is similar: Any two quick-program computers (and in particular Σ^γ and Σ) can imitate act by act each other's computations and are thus in a sense equivalent.

In order to clarify the above, let us formally define within the framework of Arbib and Blum a concept analogous to that of the quick-program computer. In what remains of this remark we use the notation and terminology of Arbib and Blum, not that of this paper. However, in order to prove that this analogous concept is not vacuous, it is necessary to make explicit an assumption which is implicit in their framework. For any machine M there exists a total recursive function m such that $m(i,x,t) = 2y$ if and only if $\phi_i^M(x) = y$ and $\Phi_i^M(x) = t$.

Definition AB. A quick-program machine M is a machine with the following property. Consider any machine N. There exists a total recursive function f_{NM} increasing in both its variables such that $N \geq_{(f_{NM})} M$; i.e. M is at least as complex as N (modulo (f_{NM})). Here, a two-variable function enclosed in parentheses denotes the monoid with multiplication $*$ and identity $e(x,y) = y$, which is generated by the function.

Then by (ii) of Theorem 2 [14] we have

Theorem AB. Consider two quick-program machines M and N. There exists a total recursive function g_{NM} increasing in both of its variables such that $N \equiv_{(g_{NM})} M$; i.e. N and M are (g_{NM})-equivalent.

Remark 9. In an effort to make this subsection more comprehensible, we now cast it into the framework of lattice theory, cf. Birkhoff [15].

Definition L1. Let Σ_1 and Σ_2 be computing machines. Σ_1 **im** Σ_2 (Σ_1 can be imitated by Σ_2) if and only if there exists a 1-ary computable function f which has the following property: For any program p for which $\Sigma_1(p)$ is defined, there exists a program p' such that $\Sigma_2(p') = \Sigma_1(p)$ and

$$\bigcup_{t' \leq t} \Sigma_1(p,t') \subset \bigcup_{t' \leq f(t)} \Sigma_2(p',t')$$

for all but a finite number of values of t.

Lemma L1. The binary relation **im** is reflexive and transitive.

Definition L2. Let Σ_1 and Σ_2 be computing machines. Σ_1 **eq** Σ_2 if and only if Σ_1 **im** Σ_2 and Σ_2 **im** Σ_1.

Lemma L2. The binary relation **eq** is an equivalence relation.

Definition L3. L is the set of equivalence classes induced by the equivalence relation **eq**. For any computer Σ, (Σ) is the equivalence class of Σ, i.e. the set of all computers Σ' such that Σ' **eq** Σ. For any $(\Sigma_1), (\Sigma_2) \in L$, $(\Sigma_1) \leq (\Sigma_2)$ if and only if Σ_1 **im** Σ_2.

Lemma L3. L is partially ordered by the binary relation $\leq$.

Lemma L4. Consider a computer which cannot be programmed to compute any infinite set of natural numbers, e.g. the computer Σ_0 defined by $\Sigma_0(p,t) = \emptyset$. Denote by 0 the equivalence class of this computer; i.e. denote by 0 the computers which compute no infinite sets of natural numbers. 0 bounds L from below; i.e. $0 \leq A$ for all $A \in L$.

Lemma L5. Consider a quick-program computer, e.g. the computer Σ^* of Theorem 9. Denote by 1 the equivalence class of this computer; i.e. denote by 1 the quick-program computers. 1 bounds L from above; i.e. $A \leq 1$ for all $A \in L$.

Lemma L6. Let Σ_1 and Σ_2 be computers. Define the computer Σ_3 as follows: $\Sigma_3(\Lambda, t) = \emptyset$, $\Sigma_3(P0, t) = \Sigma_1(P, t)$, $\Sigma_3(P1, t) = \Sigma_2(P, t)$. (Σ_3) is the l.u.b. of (Σ_1) and (Σ_2).

Lemma L7. Let Σ_1 and Σ_2 be computers. Define the computer Σ_3 as follows: Consider the sets

$$S_1 = \bigcup_{t' \leq t} \Sigma_1(K(p), t'),$$

$$S_2 = \bigcup_{t' \leq t} \Sigma_2(L(p), t'),$$

where $(K(p), L(p))$ is the pth ordered pair in an effective enumeration of the ordered pairs of natural numbers (cf. Davis [3, pp. 43–45]). If Σ_1 and Σ_2 output in size order and without repetitions the elements of, respectively, S_1 and S_2, and $S_1 \subset S_2$ or $S_2 \subset S_1$, then

$$\Sigma_3(p,t) = S_1 \cap S_2 \cap \overline{\bigcup_{t' < t} \Sigma_3(p, t')}.$$

Otherwise, $\Sigma_3(p,t) = \emptyset$. (Σ_3) is the g.l.b. of (Σ_1) and (Σ_2).

Theorem L. L is a denumerable, distributive lattice with zero element and one element.

We may describe the g.l.b. and l.u.b. operations of this lattice as follows. The l.u.b. of two computers is the slowest computer which is faster than both of them, and the g.l.b. of two computers is the fastest computer which is slower than both of them.

3. A Simple, Quick-Program Computer

This section is the culmination of this paper. A computer is constructed which is both a simple-program computer and a quick-program computer.

If it is believed that programs are essentially binary sequences and that the only natural measure of the complexity of a program considered as a binary sequence is its length, then apparently the conclusion would have to be drawn that only simple, quick-program computers are worthy of attention, all other computers being degenerate cases.

It would seem to follow that the connected sets indeed corroborate this paper's thesis. For there is a simple, quick-program computer which best represents mathematically the possibilities open to number theorists in their attempts to calculate large primes. We do not know which it may happen to be, but we do know (cf. Remark 6) that there are connected sets which are very complex and which must be computed very slowly when one is using this computer. In view of Theorems 6 and 8 it would seem to be appropriate to consider these connected sets to be perfect sets. Thus our quest for perfect sets comes to a close.

Theorem 9. There exists a simple, quick-program computer, namely Σ^*.

Proof. We take it for granted that there is a computer $\Sigma^\$$ which can compute every 2-ary computable function f in the following sense: There exists a binary sequence P_f and a 2-ary computable function $\#_f$ increasing in its second argument such that

$$\{f(n,m)\} = \Sigma^\$(B(n)P_f, \#_f(n,m))$$

for all natural numbers n and m. Moreover, $\Sigma^\$(B(n)P_f, t)$ is nonempty only if there exists an m such that $t = \#_f(n,m)$. Here B is the function

carrying each natural number into its associated binary sequence, as in Section 2.

From $\Sigma^\$$ we now construct the computer Σ^*: $n \in \Sigma^*(p,t)$ if and only if $\Sigma^\$(p,t)$ has only a single element, this element is not zero, and the nth prime divides it.[2]

We now verify that Σ^* is a simple, quick-program computer. Consider a computer Ξ. We give the natural number $c_{\Sigma^*\Xi}$ explicitly: $c_{\Sigma^*\Xi}$ is the length of P_ξ. We also give the 1-ary computable function $s_{\Sigma^*\Xi}$ explicitly:

$$s_{\Sigma^*\Xi}(n) = \max_{k \le n} \#_\xi(k,n).$$

Here ξ is, of course, the 2-ary computable function which defines the computer Ξ as in Definition 1. Q.E.D.

Appendix A. A Turing Machine

The contents of this appendix have yet to be fitted into the general framework which we attempted to develop in Sections 1–3.

Definition A. Δ is a Turing machine. Δ's "tape" is a quarter-plane or quadrant divided into squares. It has a single scanner which scans one of the squares. If the scanner runs off the quadrant, Δ halts. Δ can perform any one of the following operations: quadrant one square left (L), right (R), up (U), or down (D); or the scanner can overprint a 0 (0), a 1 (1), or erase (E) the square of the quadrant being scanned. The programs of Δ are tables with three columns headed "blank," "0," and "1," and consecutively numbered rows. Each place in the table must have an ordered pair; the first member of the pair gives the operation to be performed and the second member gives the number of the next row of the table to be obeyed. As program complexity measure Ω, we take the number of rows in the program's table. One operation (L, R, U, D, 0, 1, or E) is performed per unit time. The computing machine Δ begins calculating with its scanner on the corner square, with the quadrant completely erased, and obeying the last row of its program's table. The Turing machine outputs a natural number n when the binary sequence which represents n in base-2 notation appears at the bottom of the quadrant, starting in the corner square, ending in

the square being scanned, and with Δ obeying the next to last row of its program's table.

Theorem A1. For any connecting function γ there exists a natural number c_γ and a 1-ary computable function s_γ which have the following property: For any program p for which $\Delta(p)$ is a subset of a γ-connected set S, there is a program p' such that

(a) $\Delta(p') = S$, $\Omega(p') = \Omega(p) + c_\gamma$;[12] and

(b) for all natural numbers t,

$$\bigcup_{t' \le t} \Delta(p, t') \subset \bigcup_{t' \le t + s_\gamma(n)} \Delta(p', t'),$$

where n stands for the largest element of the left-hand side of the relation, if this set is not empty (otherwise, n stands for 0).

Proof. p' is obtained from p in the following manner. c_γ rows are added to the table defining the program p. All transfers to the next to the last row in the program p are replaced by transfers to the first row of the added section. The new rows of the table use the program p as a subroutine. They make the program p think that it is working as usual, but actually p is using neither the quadrant's three edge rows nor the three edge columns; p has been fooled into thinking that these squares do not exist because the new rows moved the scanner to the fourth square on the diagonal of the quadrant before turning control over to p for the first time by transferring to the last row of p. This protected region is used by the new rows to do its scratch-work, and also to keep permanent records of all natural numbers which it causes Δ to output.

Every time the subroutine thinks it is making Δ output a natural number n, it actually only passes n and control to the new rows. These proceed to find out which natural numbers are in $\Gamma(\gamma^{-1}(n))$. Then

[12] This implies

$$\Omega_\Delta(S) \le \Omega_\Delta(\Delta(p)) + c_\gamma.$$

I.e.

$$\Omega_\Delta(S) \le \Omega_\Delta(R) + c_\gamma$$

for any infinite computable subset R of the γ-connected set S.

the new rows eliminate those elements of $\Gamma(\gamma^{-1}(n))$ which Δ put out previously. Finally, they make Δ output those elements which remain, move the scanner back to what the subroutine last thought was its position, and return control to the subroutine. Q.E.D.

Remark A. Assuming that only the computer Δ and program complexity measure Ω are of interest, it appears that we have before us some connected sets which are in a very strong sense perfect sets. For, as was mentioned in Remark 6, there are γ-connected sets which Δ must compute very slowly. For such sets, the term $s_\gamma(n)$ in (b) above is negligible compared with t.

Theorem A2. Consider a simple-program computer Σ and the program complexity measure $\Pi(p) = [\log_2(p+1)]$. Let $S_0, S_1, S_2, \ldots$ be a sequence of distinct infinite computable sets of natural numbers. Then we may conclude that

$$\lim_{k\to\infty} \frac{\Pi_\Sigma(S_k)}{2\Omega_\Delta(S_k)\log_2\Omega_\Delta(S_k)}$$

exists and is in fact unity.

Proof. Apply the technique of [16, Pt. 1].

Of course,

Theorem A3. Δ is a quick-program computer.

Appendix B. A Lattice of Computer Speeds

The purpose of this appendix is to study L^*, the lattice of speeds of computers which calculate all infinite computable sets of natural numbers. L^* is a sublattice (in fact, a filter) of the lattice L of Remark 9. It will be shown that L^* has a rich structure: every countable partially ordered set is imbeddable in L^*.[13] Thus to require a computer to be a quick-program computer is more than to require that it be able to compute all infinite computable sets of natural numbers.

[13] An analogous result is due to Sacks [17, p. 53]. If P is a countable partially ordered set, then P is imbeddable in the upper semilattice of degrees of recursively enumerable sets. Cf. also Sacks [17, p. 21].

Definition B1. L^* is the sublattice of L consisting of the (Σ) such that Σ can be programmed to compute all infinite computable sets of natural numbers.

In several respects the following theorem is quite similar to Theorem 9 of Hartmanis and Stearns [18] and to Theorem 8 of Blum [19]. The diagonal process of the proof of Theorem 3 is built into a computer's circuits.

Theorem B1. There exists a quick-program computer Σ_1 with the property that for any 1-ary computable function f and any infinite computable set U of natural numbers, there exists a 1-ary computable function g and an infinite computable set S of natural numbers such that

(a) $S \subset U$,

(b) $g(n) > f(n)$ for all but a finite number of values of n,

(c) there exists a program p such that $\Sigma_1(p) = S$ and $n \in \Sigma_1(p, g(n)+1)$ for all $n \in S$,

(d) for all programs p' such that $\Sigma_1(p') \subset S$, $n \in \Sigma_1(p', t)$ only if $t > g(n)$, with the possible exception of a finite number of values of n.

Proof. Let Σ be a quick-program computer. We construct Σ_1 from it. $\Sigma_1(\Lambda, t) = \emptyset$, $\Sigma_1(P0, t) = \Sigma(P, t)$, $\Sigma_1(P1, 0) = \emptyset$, and $\Sigma_1(P1, t+1)$ is a subset of $\Sigma(P, t)$. For each element $n^{\#}$ of $\Sigma(P, t)$, it is determined in the following manner whether or not $n^{\#} \in \Sigma_1(P1, t+1)$. Define m, n_k $(0 \leq k \leq m)$, m', and t_k $(0 \leq k \leq m)$ as follows:

$$\bigcup_{t' \leq t} \Sigma(P, t') = \{n_0, n_1, n_2, \ldots, n_m\}\,(n_0 < n_1 < n_2 \cdots < n_m),$$

$$n^{\#} = n_{m'},$$

$$n_k \in \Sigma(P, t_k)\,(0 \leq k \leq m).$$

Define $A(i, j)$ (the predicate "the program j is eliminated during the ith stage"),[14] A (the set of programs eliminated before the m' th stage),

[14] During the ith stage of this diagonal process it is decided whether or not the ith element of $\Sigma(P)$ is in $\Sigma_1(P1)$.

and A' (the set of programs eliminated before or during the m'th stage) as follows:

$$A(i,j) \text{ iff } j < [i/4] \text{ and } n_i \in \bigcup_{t' \leq t_i} \Sigma_1(j,t'),$$

$$A = \{j | A(i,j) \text{ for some } i < m'\},$$

$$A' = \{j | A(i,j) \text{ for some } i \leq m'\}.$$

$n^{\#} \in \Sigma_1(P1, t+1)$ iff $A' = A$.

That the above indeed constructs Σ_1 follows from the fact that each of the t_k is less than $t+1$, and thus $\Sigma_1(P1, t+1)$ is defined only in terms of $\Sigma_1(p', t')$, for which t' is less than $t+1$. I.e. that $\Sigma_1(p,t)$ is defined follows by induction on t. Also, Σ_1 is a quick-program computer, for $\Sigma_1(P0, t) = \Sigma(P, t)$.

We now define the function g and the set S, whose existence is asserted by the theorem. By one of the extensions of Theorem 3, there exists a program P which has the following properties:

1. $\Sigma(P) \subset U$.

2. For all but a finite number of values of n, $n \in \Sigma(P,t)$ only if $t > f(n)$.

$S = \Sigma_1(P1)$. That S is infinite follows from the fact that at most $k/4$ of the first k elements of $\Sigma(P)$ fail to be in S. $g(n)$ is defined for all $n \in \Sigma(P)$ by $n \in \Sigma(P, g(n))$. It is irrelevant how $g(n)$ is defined for $n \notin \Sigma(P)$, as long as $g(n) > f(n)$.

Part (a) of the conclusion follows from the fact that $\Sigma_1(P1, t+1) \subset \Sigma(P,t) \subset U$ for all t. Part (c) follows from the fact that if $n \in \Sigma_1(P1)$, then

$$n \in \Sigma_1(P1, g(n)+1).$$

Part (d) follows from the fact that if $\Sigma_1(p')$ is defined and n is the first element of $\Sigma_1(p') \cap \Sigma(p)$ which is greater than or equal to the $(4p'+4)$-th element[15] of $\Sigma(P)$ and which is contained in a $\Sigma_1(p', t)$ such that $t \leq g(n)$, then n is not an element of S. Q.E.D.

Corollary B1. On the hypothesis of Theorem B1, not only do the g and S whose existence is asserted have the properties (a) to (d), they

[15] I.e. it is greater than or equal to $n_{4p'+4}$.

also, as follows immediately from (c) and (d), have the property that for any quick-program computer Σ:

(e) There exists a program p_2 such that $\Sigma(p_2) = S$ and $n \in \Sigma(p_2, t)$ with $t \leq s_{\Sigma\Sigma_1}(g(n) + 1)$ for all but a finite number of values of $n \in S$.

(f) For all programs p_3 such that $\Sigma(p_3) \subset S$, $n \in \Sigma(p_3, t)$ only if $s_{\Sigma_1\Sigma}(t) > g(n)$, with the possible exception of a finite number of values of n.

Remark B. Theorem B1 is a "no speed-up" theorem; i.e. it contrasts with Blum's speed-up theorem (cf. [5, 6, 19]). Each S whose existence is asserted by Theorem B1 has a program for Σ_1 to compute it which is as fast as possible. I.e. no other program for Σ_1 to compute S can output more than a finite number of elements more quickly. Thus it is not possible to speed up every program for computing S by the computer Σ_1. And, as is pointed out by Corollary B1, this also holds for any other quick-program computer, but with the slight "fogginess" that always results in passing from a statement about one particular quick-program computer to a statement about another.

Definition B2. Let S be a computable set of natural numbers and let Σ be a computer. Σ^S denotes the computer which can compute only subsets of S, but which is otherwise identical to Σ. I.e.

$$\Sigma^S(p,t) = \begin{cases} \Sigma(p,t), & \text{if } \bigcup_{t' \leq t} \Sigma(p,t') \subset S, \\ \emptyset, & \text{otherwise.} \end{cases}$$

Theorem B2. There is a computer Σ_0 such that $(\Sigma_0) \in L^*$ and $(\Sigma_0) < 1$. Moreover, for any computable sets T and R of natural numbers,

(a) if $T - R$ and $R - T$ are both infinite, then l.u.b. $(\Sigma_0), (\Sigma_1^T)$ and l.u.b. $(\Sigma_0), (\Sigma_1^R)$ are incomparable members of L^*, and

(b) if $T \subset R$ and $R - T$ is infinite, then the first of these two members of L^* is less than the second.

Proof. Σ_0 is constructed from the computer Σ_1 of Theorem B1 as follows. $n \in \Sigma_0(p,t)$ if and only if there exist t' and t'' with $\max(t',t'') = t$ such that

$$n \in \Sigma_1(p,t'),\ s_{t'} \in \Sigma_1(p,t'')$$

where

$$\bigcup_{t_3 \le t} \Sigma_1(p,t_3) = \{s_0, s_1, s_2, \ldots\}\ (s_0 < s_1 < s_2 < \cdots),$$

and for no $n_1 \ge n_2$, $t_1 < t_2 \le t$ is it simultaneously the case that $n_1 \in \Sigma_1(p,t_1)$ and $n_2 \in \Sigma_1(p,t_2)$. Note that for all p, $\Sigma_1(p) = \Sigma_0(p)$, both sides of the equation being undefined if one of them is.

Sets S whose existence is asserted by Theorem B1 which must be computed very slowly by Σ_1 must be computed very much more slowly indeed by Σ_0, and thus Σ_1 **im** Σ_0 cannot be the case. Moreover, within any infinite computable set U of natural numbers, there are such sets S.

We now show in greater detail that $(\Sigma_0) < (\Sigma_1) = 1$ by a reductio ad absurdum of Σ_1 **im** Σ_0. Suppose Σ_1 **im** Σ_0. Then by definition there exists a 1-ary computable function h such that for any program p for which $\Sigma_1(p)$ is defined, there exists a program p' such that $\Sigma_0(p') = \Sigma_1(p)$ and

$$\bigcup_{t' \le t} \Sigma_1(p,t') \subset \bigcup_{t' \le h(t)} \Sigma_0(p',t')$$

for all but a finite number of values of t.

In Theorem B1 we now take $f(n) = \max(n, \max_{k \le n} h(k))$. We obtain g and S satisfying

1. $g(n) > n$,
2. $g(n) > \max_{k \le n} h(k)$ for all but finitely many n. From (1) it follows that for all but a finite number of values of $n \in S$, the fastest program for Σ_1 to compute S outputs n at time $t' = g(n) + 1$, while the fastest program for Σ_0 to compute S outputs n at time

$$t'' = g(s_{g(n)+1}) + 1 = g(s_{t'}) + 1 > s_{t'} + 1 \ge t' + 1.$$

Here, as usual, $S = \{s_0, s_1, s_2, \ldots\}$ $(s_0 < s_1 < s_2 < \cdots)$. Note that s_k, the kth element of S, must be greater than or equal to k:

3. $s_k \geq k$.

By the definition of (Σ_1) **im** (Σ_0) we must have

$$h(g(n)+1) \geq g(s_{g(n)+1})+1,$$

for all but finitely many $n \in S$. By (2) this implies

$$h(g(n)+1) > \max_{k \leq s_{g(n)+1}} h(k)+1;$$

hence $g(n)+1 > s_{g(n)+1}$ for all but finitely many $n \in S$. Invoking (3) we obtain $g(n)+1 > s_{g(n)+1} \geq g(n)+1$, which is impossible. Q.E.D.

A slightly different way of obtaining the following theorem was announced in [20].

Theorem B3. Any countable partially ordered set is order-isomorphic with a subset of L^*. That is, L^* is a "universal" countable partially ordered set.

Proof. We show that an example of a universal partially ordered set is C, the computable sets of natural numbers ordered by set inclusion. Thus the theorem is established if we can find in L^* an isomorphic image of C. This isomorphic image is obtained in the following manner. Let S be a computable set of natural numbers. Let S' be the set of all odd multiples of 2^n, where n ranges over all elements of S. The isomorphic image of the element S of C is the element l.u.b. $(\Sigma_0), (\Sigma_1^{S'})$ of L^*. Here Σ_0 is the computer of Theorem B2, Σ_1 is the computer of Theorem B1, and "$\Sigma_1^{S'}$" is written in accordance with the notational convention of Definition B2.

It only remains to prove that C is a universal partially ordered set. Sacks [17, p. 53] attributes to Mostowski [21] the following result: There is a universal countable partially ordered set $A = \{a_0, a_1, a_2, \ldots\}$ with the property that the predicate $a_n \leq a_m$ is computable. We finish the proof by constructing in C an isomorphic image $A' = \{A_0, A_1, A_2, \ldots\}$ of A [as follows:

$$A_i = \{k | a_k \leq a_i\}.$$

It is easy to see that $A_i \subset A_j$ if and only if $a_i \leq a_j$.] Q.E.D.

Corollary B2. L^* has exactly $\aleph_0$ elements.

References

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

[2] Dantzig, T. *Number, the Language of Science.* Macmillan, New York, 1954.

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

[4] Gelfond, A. O., and Linnik, Yu. V. *Elementary Methods in Analytic Number Theory.* Rand McNally, Chicago, 1965.

[5] Blum, M. Measures on the computation speed of partial recursive functions. Quart. Prog. Rep. 72, Res. Lab. Electronics, MIT, Cambridge, Mass., Jan. 1964, pp. 237–253.

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

[7] Fraenkel, A. A. *Abstract Set Theory.* North-Holland, Amsterdam, The Netherlands, 1961.

[8] Borel, É. *Leçons sur la Theorie des Fonctions.* Gauthier-Villars, Paris, 1914.

[9] Hardy, G. H. *Orders of Infinity.* Cambridge Math. Tracts, No. 12, U. of Cambridge, Cambridge, Eng., 1924.

[10] Dekker, J. C. E., and Myhill, J. Retraceable sets. *Canadian J. Math. 10* (1958), 357–373.

[11] Solomonoff, R. J. A formal theory of inductive inference, Pt. I. *Inform. Contr. 7* (1964), 1–22.

[12] Kolmogorov, A. N. Three approaches to the definition of the concept "amount of information." *Problemy Peredachi Informatsii 1* (1965), 3–11. (Russian)

[13] Chaitin, G. J. On the length of programs for computing finite binary sequences: statistical considerations. *J. ACM 16,* 1 (Jan. 1969), 145–159.

[14] Arbib, M. A., and Blum, M. Machine dependence of degrees of difficulty. *Proc. Amer. Math. Soc. 16* (1965), 442–447.

[15] Birkhoff, G. *Lattice Theory.* Amer. Math. Soc. Colloq. Publ. Vol. 25, Amer. Math. Soc., Providence, R. I., 1967.

[16] Chaitin, G. J. On the length of programs for computing finite binary sequences. *J. ACM 13,* 4 (Oct. 1966), 547–569.

[17] Sacks, G. E. *Degrees of Unsolvability.* No. 55, Annals of Math. Studies, Princeton U. Press, Princeton, N. J., 1963.

[18] Hartmanis, J., and Stearns, R. E. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. 117* (1965), 285–306.

[19] Blum, M. A machine-independent theory of the complexity of recursive functions. *J. ACM 14,* 2 (Apr. 1967), 322–336.

[20] Chaitin, G. J. A lattice of computer speeds. Abstract 67T-397, *Notices Amer. Math. Soc. 14* (1967), 538.

[21] Mostowski, A. Über gewisse universelle Relationen. *Ann. Soc. Polon. Math. 17* (1938), 117–118.

[22] Blum, M. On the size of machines. *Inform. Contr. 11* (1967), 257–265.

[23] Chaitin, G. J. On the difficulty of computations. Panamerican Symp. of Appl. Math., Buenos Aires, Argentina, Aug. 10, 1968. (to be published)

Received October, 1966; Revised December, 1968