A Machine-Independent Theory of the Complexity of Recursive Functions

MANUEL BLUM

Massachusetts Institute of Technology, Cambridge, Massachusetts*

ABSTRACT. The number of steps required to compute a function depends, in general, on the type of computer that is used, on the choice of computer program, and on the input-output code. Nevertheless, the results obtained in this paper are so general as to be nearly independent of these considerations.

A function is exhibited that requires an enormous number of steps to be computed, yet has a "nearly quickest" program: Any other program for this function, no matter how ingeniously designed it may be, takes practically as many steps as this nearly quickest program.

A different function is exhibited with the property that no matter how fast a program may be for computing this function another program exists for computing the function very much faster.

1. *Introduction*

The problem is to characterize the complexity of computable functions. The theory developed here is expanded along lines suggested by Rabin's axiomatic approach [5]. The chosen axioms are the basis for determining what is or is not a legal measure of functional complexity. Perhaps the most familiar such intuitive measures are (i) the number of steps needed to compute a function, and (ii) the amount of machine tape needed for a computation. These both satisfy our axioms. Other examples are presented after the axioms.

The complexity theory offered here is machine-independent. This means that a theorem that characterizes the complexities of partial recursive functions on one class of machines equally well characterizes their complexities on almost any other class. Although at first one expects this, the claim is odd, for the complexity of a *particular* function necessarily depends on the class of machines used for the computations. Thus it often takes fewer steps to compute a function within a class of multitape machines than within a class of 1-tape machines, and in a class of machines with a base 2 input-output code one can compute certain functions (such as 2^n) in fewer steps than in a class of machines with a base 10 code. So all hope that an individual function might enjoy a unique measure of complexity, one that is independent of the class of machines, must vanish. What remains is possible, and is offered here

* Department of Mathematics and Research Laboratory of Electronics.

This work, which is based on a Ph.D. thesis submitted to the Department of Mathematics, Massachusetts Institute of Technology, May 5, 1964, was supported in part by the Joint Services Electronics Program, under Contract DA36-039-AMC-03200(E); in part by the National Science Foundation (Grant GP-2495), the National Institutes of Health (Grant MH-04737-05), the National Aeronautics and Space Administration (Grant NsG-496) and the U. S. Air Force (ASD Contract AF33(615)-1747); and in part by Project MAC, an M.I.T. research program.

instead: an axiomatic theory whose *theorems* are independent of class. An example of such a theorem will help to formulate these notions.

THEOREM 1. *To every total recursive function g there corresponds a 0-1 valued total recursive function f which is so complex that any machine that computes $f(n)$ takes more than $g(n)$ steps to do so for infinitely many inputs n.*

Although the axioms should come first, they are so intuitive that an informal proof of this theorem can be given even now. In that proof $Z_0, Z_1, Z_2, \cdots$ is a sequence of machines that computes all the partial recursive functions of the natural numbers.

PROOF. The function g is given and f must be defined—in this case by a diagonal argument. To compute $f(n)$, simulate the nth machine Z_n by input n. If Z_n stops in less than $g(n)$ steps, with some output m, make $f(n) \neq m$. Otherwise $f(n)$ can be arbitrary. Then any machine Z_i which with input i takes less than $g(i)$ steps cannot be a machine for f. But each machine appears infinitely often in the list (or else we can change the list to force this) and, in particular, each machine for f does too. So the theorem follows. Q.E.D.

This first theorem is not intended to be an impressive one. The purpose in presenting it is to exhibit a theorem that is true for any class of machines, no matter how the machines are constructed or coded. Thus $Z_0, Z_1, Z_2, \cdots$ might be the class of 1-tape machines, or it might equally well be the class of 10-tape machines. It might even boast a base 1 input code and Roman numeral output code.

We associate with each machine Z_i two functions: (i) the partial-recursive $\phi_i(n)$ which it computes and (ii) a partial recursive $\Phi_i(n)$ called its *step-counting function*. Actually, $\Phi_i(n)$ may be interpreted either as the number of steps *or* the amount of tape used by Z_i when its input is n.

In this paper we present a set of axioms, a few minor theorems and two major theorems: speed-up and compression. The speed-up and compression theorems touch on the following problem: Given a recursive function f, does there correspond to each machine that computes it another that computes it faster?

The speed-up theorem, or rather a special case of it, gives examples of 0-1 valued total recursive functions f with the property that to *every* Z_i that computes f (in $\Phi_i(n)$ steps) there corresponds another machine Z_j that does the job so much quicker (in $\Phi_j(n)$ steps) that $\Phi_i(n) > 2^{\Phi_j(n)}$ for almost all n, i.e., for all but a finite number of integers. Note that this leads to an infinite sequence $f = \phi_i = \phi_j = \phi_k = \cdots$ such that

$$\Phi_i(n) > 2^{\Phi_j(n)} > 2^{2^{\Phi_k(n)}} > \cdots .$$

The *compression* theorem is a converse of the speed-up theorem. Much like the work of Hartmanis and Stearns [2, Th. 9], it shows that two tight bounds can sandwich the number of steps needed to compute some very complex functions f. As proved in this paper, the theorem is completely general, but for the moment here is an application of it to the 1-tape machines with a b (blank), 0, 1 alphabet, a base 2 input-output code and step-counting functions defined by $\Phi_i(n)$ equals the actual number of steps to compute $\phi_i(n)$: Let ϕ_i be a partial recursive function. Then a 0-1 valued $f(n)$ exists with the same domain as ϕ_i, which is so complex that any machine computing $f(n)$ takes at least $\Phi_i(n)$ steps for almost all n, but even so at least one

machine computes $f(n)$ in less than $[\Phi_i(n)]^7$ steps. So the number of steps needed to compute f wedges between the bounds Φ_i and Φ_i^7.

2. *Axioms for Step-Counting Functions*

The axioms can be formulated without mentioning devices or machines. The first step is to postulate an effective list, $\{\phi_i\}$, of all partial recursive functions, and to require of this list that the S_n^m theorem[1] and the universal Turing machine theorem[2] hold true. (Rogers [8] calls any such list an acceptable Gödel numbering, and he proves that any two such lists are recursively isomorphic!) The second step is to associate with each ϕ_i a partial recursive function Φ_i. The set $\{\Phi_i\}$ is completely arbitrary save for two basic restrictions, the axioms:

1. $\phi_i(n)$ converges (i.e., Z_i with input n stops) $\leftrightarrow \Phi_i(n)$ converges.
2. The function

$$M(i, n, m) = \begin{cases} 1 & \text{if } \Phi_i(n) = m, \\ 0 & \text{otherwise,} \end{cases}$$

is (total) recursive.

Axiom 1 can equally well be expressed in terms of the function M by the statement: $\phi_i(n)$ converges $\leftrightarrow \exists\ m[M(i, n, m) = 1]$. A function M which so satisfies axioms 1 and 2 is called a *measure on computation.*

In what follows, $\Phi_i(n) > m$ is a shorthand for $\infty \geq \Phi_i n > m$, where $\Phi_i(n) = \infty$ is to mean that $\phi_i(n)$ diverges.

Example 1. Let $\{Z_i\}$ be the class of multitape machines, and $\{\phi_i\}$ the set of functions computed by it. Then a possible set of step-counting functions $\{\Phi_i\}$ is defined by $\Phi_i(n) = m$ if and only if Z_i with input n stops in precisely m steps (else $\Phi_i(n)$ is undefined).

Another possible set $\{\Phi_i\}$ is defined by $\Phi_i(n) = m$ if and only if Z_i with input n stops and Z_i uses precisely m squares of tape for this computation.

Example 2. If $\{\Phi_i\}$ satisfies the axioms, then so does $\{\hat{\Phi}_i\}$ defined by $\hat{\Phi}_i(n) = \Phi_i(n)2^{i+n}$.

Another possible choice of $\{\hat{\Phi}_i\}$, given that a certain ϕ_{i_0} is total recursive, is

$$\hat{\Phi}_i(n) = \begin{cases} \Phi_i(n) & \text{if } i \neq i_0, \\ 0 & \text{if } i = i_0. \end{cases}$$

Example 3. The choice $\Phi_i(n) = \phi_i(n)$ is not permissible, since it satisfies axiom 1 but not axiom 2; nor is the choice which sets $\Phi_i(n) = 0$ for all i and n permitted, since it satisfies axiom 2 but not axiom 1. It follows that the two axioms are independent.

Each step-counting function Φ_i, being partial recursive, appears somewhere in the list $\{\phi_i\}$. The second theorem asserts that there exists an effective procedure for telling where.

[1] The S_n^m theorem [9], also known as the iteration theorem, asserts the existence of a total recursive function σ such that $\phi_{\sigma(i, m)}(n) = \phi_i(m, n)$ for all i, n and m.

[2] The universal Turing machine theorem states that for any effective 1-1 onto map $\tau: N \times N \rightarrow N$, N = nonnegative integers, there is a universal machine Z_i with the property $\phi_i(\tau(x, y)) = \phi_x(y)$ for all x and y.

THEOREM 2. *There exists a total recursive function β such that $\Phi_i = \phi_{\beta(i)}$ for all i.*
PROOF. Axiom 2 indicates that $M(i, n, m) = 1 \Leftrightarrow \Phi_i(n) = m$.
Define a function f that satisfies

$$f(i, n) = m \Leftrightarrow M(i, n, m) = 1.$$

Function f is partial recursive. By the S_n^m theorem, there exists a total recursive function β such that $\phi_{\beta(i)}(n) = f(i, n)$.
This is the desired β. Q.E.D.
The third theorem makes precise the following notion: Suppose two machine classes are given, and that a function f is to be computed. Then for every machine of one class that computes it, there is a machine of the other that computes it in about the same number of steps.
THEOREM 3. *Let M and $\hat{M}$ be arbitrary measures on computation with step-counting functions $\{\Phi_i\}$ and $\{\hat{\Phi}_i\}$. Then a total recursive function g exists such that $g(n, \hat{\Phi}_i(n)) \geq \Phi_i(n)$ and $g(n, \Phi_i(n)) \geq \hat{\Phi}_i(n)$ for all i and almost all n.*
This means that $\Phi_i(n)$ and $\hat{\Phi}_i(n)$ do not differ too much from each other.
PROOF. The desired function g comes from function p defined by

$$p(i, n, m) = \begin{cases} \Phi_i(n) + \hat{\Phi}_i(n) & \text{if } \Phi_i(n) = m \text{ or } \hat{\Phi}_i(n) = m, \text{ i.e., if} \\ & M(i, n, m) = 1 \text{ or } \hat{M}(i, n, m) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Function p is total recursive: It is computable because M and $\hat{M}$ are, and it is total because $\hat{\Phi}_i(n)$ convergent $\Leftrightarrow \phi_i(n)$ convergent $\Leftrightarrow \Phi_i(n)$ convergent. The desired g is $g(n, m) = \max_{i \leq n} p(i, n, m)$. Q.E.D.
A useful tool for proving complexity theorems involves a simplified version of Kleene's recursion theorem [9]: To every total recursive function σ there corresponds an integer j such that $\phi_{\sigma(j)} = \phi_j$. An example of its use appears in this proof of the obvious—that a badly designed machine can waste a huge number of steps in computing a simple function:
THEOREM 4. *Let h and f be total recursive functions. Then there exists an index j for f such that $\Phi_j(n) > h(n)$ for all n.*
PROOF. Let

$$p(i, n) = \begin{cases} f(n) & \text{if } \Phi_i(n) > h(n), \text{ i.e., if } M(i, n, m) = 0 \\ & \text{for all } m \leq h(n), \\ 1 + \phi_i(n) & \text{otherwise.} \end{cases}$$

Function p is total recursive; hence there exists a total recursive function σ such that $p(i, n) = \phi_{\sigma(i)}(n)$ for all i and n. By the recursion theorem there exists an integer j such that $\phi_{\sigma(j)} = \phi_j$. Now it is easy to show that j is an index for f: If $\Phi_j(n) \leq h(n)$, then $\phi_j(n) = 1 + \phi_j(n)$, which is a contradiction. So $\Phi_j(n) > h(n)$ for all n. By definition of function p, it follows that $\phi_j(n) = f(n)$ for all n. Q.E.D.

3. *Speed-Up*

On a 1-tape machine with a base 10 code and with step-counting functions which count actual numbers of steps, a function like $f(n) = 0$ or $f(n) = n$ can be computed

in a quickest or a best way. There is no surprise here, for each is computable by finite automata.

On the other hand, if a function is reasonably complex, then to each machine that computes it there corresponds another much faster that computes it in just half as many steps for infinitely many n. A simple example of such a function is

$$f(n) = \begin{cases} 1 & \text{if } n \text{ is a palindrome,} \\ 0 & \text{otherwise.} \end{cases}$$

This function determines whether an integer written in base 10 reads the same forward as backward. To compute $f(n)$, $n = 372686273$, a typical machine first scans the rightmost digit 3 and deletes it, then runs down the tape to leftmost digit 3 and deletes it, backs up to rightmost digit 7 and deletes it, down to leftmost digit 7 and deletes it, and so on. After comparing opposed digits, it prints the output 1. A quicker machine scans the rightmost digits 7 and 3 simultaneously before running down the tape to leftmost digits 3 and 7, and so compares digits 2 at a time rather than 1 at a time. This quicker machine takes approximately half as many steps as the slower one for all palindromes. In fact one can show (though not easily) that no matter what machine is chosen to compute this function, another can be found which does the same job in just half as many steps, for infinitely many inputs.

This indicates a result that is generally true: There exists a total recursive function f with the property that to every index i for f there corresponds another index j for f such that $\Phi_i(n) > 2\Phi_j(n)$ for almost all n. Can a stronger theorem be proved? In particular, does there exist a function f with the property that to every index i for f there corresponds an index j for f such that

$$\Phi_i(n) > \Phi_j(n)^{\Phi_j(n)}$$

for almost all n? The next theorem asserts that such a function exists.

THEOREM 5 (SPEED-UP). *Let r be a total recursive function of 2 variables. Then there exists a total recursive function f taking values 0 and 1 with the property that to every index i for f there corresponds another index j for f such that $\Phi_i(n) > r(n, \Phi_j(n))$ for almost all n.*

PROOF. Simple intuition lies behind this theorem. It comes from a direct proof that to every total recursive h there corresponds a total recursive f with the property that if i is any index for f, then $\Phi_i(n) > h(n)$ for almost all n. This direct proof computes $f(n)$ by canceling the first uncanceled function $\phi_i(n)$ which is in the set $\{\phi_0(n), \cdots, \phi_n(n)\}$ and whose step-counting function Φ_i has the property $\Phi_i(n) \leq h(n)$. It then makes $f(n) \neq \phi_i(n)$. To prevent the folly of looking no farther than the ith function ϕ_i at later times, the canceled ϕ_i is passed up in computing f for larger n.

A minor change in this construction makes f into the one wanted for the theorem: Instead of canceling the first uncanceled element of the set $\{\phi_0(n), \cdots, \phi_n(n)\}$ with the property $\Phi_i(n) \leq h(n)$, we cancel the first one with the property $\Phi_i(n) \leq h(n - i)$. Then make $f(n) \neq \phi_i(n)$ as before. That defines f. In the proof, this computation of f is $f = \phi_{l(0,0,l)}$, where l is an index for h.

The intuition enters in seeing what makes this f work. The machine for $\phi_{l(0,0,l)}$ goes through the set $\{\phi_0(n), \cdots, \phi_n(n)\}$ and cancels the first uncanceled ϕ_i for which $\Phi_i(n) \leq \phi_l(n - i)$. A quicker way to compute $f(n)$ uses the fact that it is not really

necessary to scan the whole set $\{\phi_0(n), \cdots, \phi_n(n)\}$. Given a fixed number u, it is sufficient for large n to scan only the set $\{\phi_u(n), \cdots, \phi_n(n)\}$. The reason is that for large enough n, each of the elements $\phi_0(n), \cdots, \phi_{u-1}(n)$ is either already canceled or will never be canceled; so none of these will be canceled during the computation of $f(n)$. These may therefore be bypassed for sufficiently large n, say $n \geq v$. Such a shortcut computation of $f(n)$, which scans the smaller set $\{\phi_u(n), \cdots, \phi_n(n)\}$, takes fewer steps; it is known here as $f = \phi_{t(u,v,l)}$. In general, the larger u, the faster is the corresponding machine that computes f. The increase is sufficient to prove the theorem.

Only one small problem arises. These computations involving u and v are special ways to compute f, and it might be thought that some strange and much quicker way to compute f has escaped us. Such is not the case, for if j is any index for f, then as a consequence of the definition of f, $\Phi_j(n) > \phi_l(n - j)$ for almost all n, which puts a lower bound on the number of steps that it takes to compute $f(n)$. This lower bound is used to show that any procedure for computing f is no better than one of those above involving fixed numbers u and v.

The proof of the theorem follows a plan suggested by Michael Arbib, which clarifies and condenses the present author's original proof. We define a crucial total recursive function $t(u, v, l)$ so that ϕ_l total implies $\phi_{t(u,v,l)}$ total with values 0 and 1. Then prove:

LEMMA 1. *If ϕ_l is total, then for each u there exists a v such that $\phi_{t(u,v,l)} = \phi_{t(0,0,l)}$.*

LEMMA 2. *If ϕ_l is total, and if we define $f = \phi_{t(0,0,l)}$, then for each index i for f, $\Phi_i(n) > \phi_l(n - i)$ for almost all n.*

LEMMA 3. *There exists a total recursive function ϕ_l such that for all u and v, $\phi_l(n - u + 1) \geq r(n, \Phi_{t(u,v,l)}(n))$ for almost all n.*

Then for each index i for $f = \phi_{t(0,0,l)}$ and for almost all n,

$$\Phi_i(n) > \phi_l(n - i) \qquad \text{(Lemma 2)}$$

$$\geq r(n, \Phi_{t(i+1,v,l)}(n)) \quad \text{for all } v \qquad \text{(take } u = i+1 \text{ in Lemma 3)}$$

$$= r(n, \Phi_j(n)) \quad \text{(by Lemma 1, choose } v \text{ so that } f = \phi_{t(i+1,v,l)} = \phi_j),$$

and the theorem follows. Q.E.D.

We now turn to the construction of t and the proof of the three lemmas.

1. *Construction of t.* Function $\phi_{t(u,v,l)}$ is defined in terms of the set (see Figure 1)

$$C_{uv} = \{\phi_k(m) \mid (m < v \text{ and } k \leq m) \text{ or } (m \geq v \text{ and } u \leq k \leq m)\}$$

Compute $\phi_{t(u,v,l)}(n)$ as follows:

(i) If $v < u$, set $\phi_{t(u,v,l)}(n) = \phi_{t(u,u,l)}(n)$ and use (ii).

(ii) If $v \geq u$, proceed as follows:

(α) Compute $\phi_l(0), \cdots, \phi_l(n)$ if $n < v$, but compute only $\phi_l(0), \cdots, \phi_l(n - u)$ if $n \geq v$. If any of these diverge, let $\phi_{t(u,v,l)}(n)$ diverge. Otherwise go to (β).

(β) Circle all $\phi_k(n)$ in $C_{uv} \cap$ column n such that $\Phi_k(n) \leq \phi_l(n - k)$. If none are circled, set $\phi_{t(u,v,l)}(n)$ equal to zero. Otherwise, go to (γ).

(γ) Cancel the first entry $\phi_k(n)$ in column n which is circled and which has the property that $\phi_k(m)$, $m < n$, has not been canceled. If none of

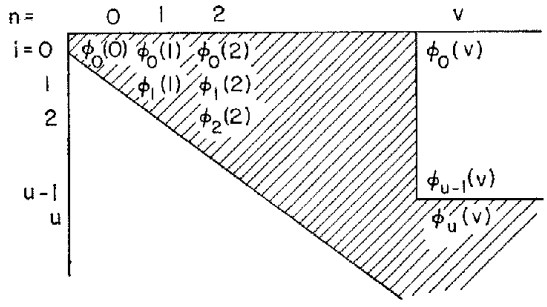

FIG. 1

the entries in $C_{uv} \cap$ column n are canceled, set $\phi_{t(u,v,l)}(n)$ equal to zero. Otherwise go to (δ).

(δ) If $\phi_k(n)$ is canceled, set

$$\phi_{t(u,v,l)}(n) = \begin{cases} 0 & \text{if } \phi_k(n) \neq 0, \\ 1 & \text{if } \phi_k(n) = 0. \end{cases}$$

This definition implies that if ϕ_l is total, then $\phi_{t(u,v,l)}$ is total for all u and v.

2. *Proof of Lemma* 1. In the computation of $\phi_{t(0,0,l)}$, the number of entries above row u that are canceled must be finite, since a row contains at most one canceled entry. Let column v be immediately to the right of the rightmost canceled entry above row u. Then it is clear from the figure that $\phi_{t(0,0,l)}(n) = \phi_{t(u,v,l)}(n)$ for all n. Q.E.D.

3. *Proof of Lemma* 2. Let i be an index for f, and suppose to the contrary that $\Phi_i(n) \leq \phi_l(n - i)$ for infinitely many $n = n_1, n_2, n_3, \cdots$. Then it is seen that, in the computation of $\phi_{t(0,0,l)}(n)$, $\phi_i(n_k)$ must be canceled for some k, and so $\phi_{t(0,0,l)}(n_k) \neq \phi_i(n_k)$, which implies that $f(n_k) \neq \phi_i(n_k)$. Contradiction. Q.E.D.

4. *Proof of Lemma* 3. Choose a recursive 1-1 map of the integers onto the set of all 1-tuples, 2-tuples, 3-tuples, $\cdots$ of integers. Let $\langle\langle a_0, \cdots, a_n\rangle\rangle$ denote the integer which maps onto $(a_0, \cdots, a_n)$. Define

$$g(n, u, v, i, z) = \begin{cases} \Phi_{t(u,v,i)}(n) & \text{if } n \geq u + v \text{ and } z = \langle\langle\Phi_i(0), \cdots, \Phi_i(n - u)\rangle\rangle, \\ 0 & \text{if not.} \end{cases}$$

Using the M-function, we see that g is total recursive. It thus suffices to find a total recursive function ϕ_l such that, for almost all n,

$$\phi_l(n - u + 1) \geq r[n, g(n, u, v, l, \langle\langle\Phi_l(0), \cdots, \Phi_l(n - u)\rangle\rangle)]. \quad (3.1)$$

Define p as follows:

$$p(i, 0) = 0$$

$$p(i, z + 1) = \max_{\substack{0 \leq v \leq z \\ 0 \leq u \leq z}} r[z + u, g(z + u, u, v, i, \langle\langle\Phi_i(0), \cdots, \Phi_i(z)\rangle\rangle)]. \quad (3.2)$$

The function p is partial recursive, so there exists a total recursive function σ such that $p(i, z) = \phi_{\sigma(i)}(z)$. By the recursion theorem, there exists an index l such that

$\phi_{\sigma(l)}(z) = \phi_l(z)$ for all z. Hence

$$\phi_l(z) = p(l, z). \tag{3.3}$$

Thus $\phi_l(0) = p(l, 0) = 0$. Now assume $\phi_l(n)$ converges for $n \leq z$. On inspecting (3.2) and recalling (3.3), we see that $\phi_l(z + 1)$ converges, since g and r are total recursive, and convergence of $\phi_l(n)$ for $n \leq z$ implies convergence of $\Phi_l(n)$ for $n \leq z$. Hence ϕ_l is total recursive.

Fix u and v. Then for $n \geq \max [2u, u + v]$, we have $v \leq n - u$, $u \leq n - u$ and so, putting $z = n - u$ and $i = l$ in (3.2), we obtain (3.1). Q.E.D.

For example, let $r(n, m) = 2^m$. Then the speed-up theorem asserts the existence of a total recursive function f such that to every index i for f there corresponds another index j for f such that $\Phi_i(n) > 2^{\Phi_j(n)}$ for almost all n. Hence it also asserts that to index j for f there corresponds an index k for f such that $\Phi_i(n) > 2^{2^{\Phi_k(n)}}$, and so on. Unfortunately, the speed-up theorem does not provide an effective procedure for going from a machine for f to a speedier one. In fact it cannot, as Corollary 1 shows:

COROLLARY 1. *Let r be a total recursive function and f the corresponding function of the speed-up theorem. If r is sufficiently large, there can be no total recursive function κ such that: (1) κ enumerates only indices of f, (2) to every index i for f there corresponds an index $\kappa(j)$ for f such that $\Phi_i(n) > r(n, \Phi_{\kappa(j)}(n))$ for almost all n.*

PROOF. Let π be a total recursive function with the property that for each j there exist infinitely many n such that $\pi(n) = j$. By the S_n^m theorem, there exists a total recursive function σ such that

$$\phi_{\sigma(i)}(n) = \phi_{\phi_i(\pi(n))}(n) \tag{3.4}$$

for all i and n. For this σ, it is easy to show that there exists a total recursive function g such that

$$\Phi_{\sigma(i)}(n) = g(n, i, \Phi_i(\pi(n)), \Phi_{\phi_i(\pi(n))}(n)),$$

and therefore there exists a total recursive function h such that

$$\Phi_{\sigma(i)}(n) < h(n, \Phi_{\phi_i(\pi(n))}(n)) + h(i, \Phi_i(\pi(n))) \tag{3.5}$$

for all i and n. Let $\kappa = \phi_k$ enumerate only indices of f. Then $\phi_{\phi_k(\pi(n))}(n) = \phi_{\kappa(\pi(n))}(n)$, and so by (3.4), $\phi_{\sigma(k)}(n) = \phi_{\kappa(\pi(n))}(n)$, so $\sigma(k)$ is an index for f. By (3.5),

$$\Phi_{\sigma(k)}(n) < h(n, \Phi_{\kappa(\pi(n))}(n)) + h(k, \Phi_k(\pi(n))) \tag{3.6}$$

for all n. For each j and infinitely many n, $\pi(n) = j$, and so for each j and infinitely many n,

$$\Phi_{\sigma(k)}(n) < h(n, \Phi_{\kappa(j)}(n)) + h(k, \Phi_k(j)). \tag{3.7}$$

But h as defined in (3.5) is independent of r. Therefore r can be chosen as large as one likes without changing h. In particular, let r be any function that satisfies the inequality, $r(n, m) \geq n + h(n, m)$ for all n and m. If we let $m = \Phi_{\kappa(j)}(n)$, eq. (3.7) gives

$$\Phi_{\sigma(k)}(n) < r(n, \Phi_{\kappa(j)}(n)) - n + h(k, \Phi_k(j))$$

for infinitely many n. This in turn implies that for every j there exist infinitely many

n such that

$$\Phi_{\sigma(k)}(n) < r(n, \Phi_{\kappa(j)}(n)).$$

As stated just before eq. (3.6), $f = \phi_{\sigma(k)}$, so this is a contradiction to part 2 of this corollary. Q.E.D.

It may be supposed of the r-speed-up $\Phi_i(n) > r(n, \Phi_j(n))$ that r must be an exceedingly small function in comparison with Φ_j. The next theorem asserts, however, that r may in fact be as large as Φ_j.

THEOREM 6 (SUPER SPEED-UP). *Let g be a total recursive function. Then there exists a 0-1 valued total recursive function f such that (1) if i is an index for f, then $\Phi_i(n) > g(n)$ for almost all n; (2) to every index i for f there corresponds an index j for f such that $\Phi_i(n) > \Phi_j(\Phi_j(n))$ for almost all n.*

PROOF. The proof of this theorem is like that of the speed-up theorem, so it is not given here in detail but only indicated. It depends on the following lemma:

LEMMA. *There exists a sequence $\{p_s\}_{s=0}^{\infty}$ of monotonically increasing total recursive functions p_s such that (1) for each s and almost all n, $p_s(n) > g(n)$; (2) for each s and almost all n, $p_s(n) > p_{s+1}(p_{s+1}(n))$; (3) for each s and all n, $p_s n \geq p_{s+1}(n)$.* The proof of this lemma is left to the reader.

To compute $f(n)$, cancel the first uncanceled element ϕ_i which is in the set $\{\phi_0(n), \cdots, \phi_n(n)\}$ and whose step-counting function Φ_i has the property $\Phi_i(n) \leq p_i(n)$. Then make $f(n) \neq \phi_i(n)$. It follows that if i is an index for f, then for almost all n,

$$\Phi_i(n) > p_i(n). \tag{3.8}$$

By the lemma, $p_i(n) > g(n)$ for almost all n, so part 1 of this theorem is proved.

To compute $f(n)$ more quickly, it suffices to fix a number u and for large enough n to scan only the set $\{\phi_u(n), \cdots, \phi_n(n)\}$. The number of steps needed to compute $f(n)$ in this way can be determined: It can be shown, by using part 3 of the lemma, that there exists a total recursive function h which is independent of the sequence $\{p_s\}$ such that a machine which computes $f(n)$ in this way (i.e., by scanning only the set $\{\phi_u(n), \cdots, \phi_n(n)\}$) takes less than $h(n, p_u(n))$ steps for almost all n. This means that for every number u there exists a machine Z_j which computes f so quickly that for almost all n,

$$h(n, p_u(n)) > \Phi_j(n). \tag{3.9}$$

Since the function h is independent of the choice of sequence $\{p_s\}$, the function g may be made so large in the lemma that $p_{u-1}(n) > h(n, p_u(n))$ for almost all n. Then by taking $u = i + 2$, this yields that for almost all n,

$$p_{i+1}(n) > h(n, p_{i+2}(n)). \tag{3.10}$$

It follows that if Z_i computes f, then for almost all n,

$\Phi_i(n) > p_i(n)$ (by eq. (3.8))

$> p_{i+1}(p_{i+1}(n))$ (Lemma, part 2)

$> p_{i+1}(h(n, p_{i+2}(n)))$ (by eq. (3.10) and the fact that p_{i+1} is monotonically increasing)

$> p_{i+1}(\Phi_j(n))$ where $f = \phi_j$ ($u = i + 2$ in eq. (3.9))

$> h(\Phi_j(n), p_{i+2}(\Phi_j(n)))$ (by eq. (3.10))

$> \Phi_j(\Phi_j(n))$ where $f = \phi_j$ ($u = i + 2$ in eq. (3.9)).

Q.E.D.

4. *Compression*

The next theorem, which is similar to one by Rabin [5], proves the existence of enormously complex 0-1 valued functions. As such it resembles Theorem 1, but it extends that result in three important directions. Whereas Theorem 1 asserts that there are 0-1 valued functions $f(n)$ which take more than $g(n)$ steps to compute for infinitely many n, the next theorem asserts this for almost all n. It also allows g and f to be just partial recursive, instead of total recursive. Finally, it provides a recursive γ which maps indices of g's into indices of f's. Theorem 7 is not a trivial generalization, and the compression theorem leans heavily upon it, especially upon the existence of γ.

THEOREM 7. (1) *To every partial recursive function g there corresponds a* 0-1 *valued partial recursive function f with the same domain as g such that if j is any index for f, then $\Phi_j(n) > g(n)$ for almost all n.* (2) *There exists a total recursive function γ which takes an index for any g into an index for the corresponding f.*

PROOF. If it is known in advance that g is total recursive, then a very simple procedure will serve to compute f. To compute $f(n)$, cancel the first uncanceled element ϕ_j which is in the set $\{\phi_0(n), \cdots, \phi_n(n)\}$ and whose step-counting function has the property $\Phi_j(n) \leq g(n)$. Then make $f(n) \neq \phi_j(n)$. In this way, $f(n)$ is forced to be different from those functions ϕ_j for which $\Phi_j(n) \leq g(n)$; hence, it must happen that if k is an index for f, then $\Phi_k(n) > g(n)$ for almost all n.

This procedure for computing $f(n)$ requires that it be known which functions in the set $\{\phi_0, \cdots, \phi_n\}$ were canceled during the computation of $f(0), \cdots, f(n-1)$. So to compute $f(n)$ with this procedure, it is necessary first to compute $f(0)$, then $f(1)$, and so on to $f(n-1)$. This is easy enough if g is total, but if not, $g(m)$ may diverge for some m and then so must $f(m)$ and there may be no way of telling that $f(m)$ diverges. If $m < n$, this makes it impossible to compute $f(n)$. To bypass this difficulty, we compute $f(n)$ in such a way that $f(0)$ through $f(n-1)$ need not be computed first. The idea is to enumerate in some order all pairs of nonnegative integers so that given any nonnegative integers n and m, the pair (n, m) is certain to be produced eventually.

After each pair (n, m) is produced, take the index i for g and determine whether or not $\Phi_i(n) = m$. This may be done effectively using the measure function, M. If $\Phi_i(n) \neq m$, go on to the next pair in the list. On the other hand, if $\Phi_i(n) = m$, then compute $f(n)$ as follows: Cancel the first element ϕ_j in the set $\{\phi_0, \cdots, \phi_n\}$ which was not canceled during the enumeration of pairs up to the present pair (n, m) and whose step-counting function Φ_j has the property $\Phi_j(n) \leq g(n)$. Perhaps no ϕ_j is canceled. If so, set $f(n) = 1$. On the other hand, if ϕ_j is canceled, set $f(n) \neq \phi_j(n)$. This defines f.

The proof of Corollary 1 to Theorem 8 requires an extremely detailed statement of the above construction of f. This statement is provided by the following indented paragraphs, which the reader may skip.

The procedure for going from an index i for g and a number n to a value of $f(n)$ is as follows:

First compute $g(n)$, $(g = \phi_i)$. If it diverges, let $f(n)$ diverge. If it converges, $f(n)$ must be made to converge. To compute it, a procedure is set up so that all functions ϕ_j are canceled whose step-counting functions Φ_j have the property that $\Phi_j(n) \leq g(n)$ for infinitely many n, and the corresponding pairs (j, n) are adjoined to a list L. The list L grows in stages and at each stage at most one pair (j, n) is adjoined to L. The growth of L at stage x is described next. A recursive code that maps $N \times N$ onto N is used to rewrite x in the form $\langle p, q \rangle = x$.

Stage $x = \langle p, q \rangle$. If $\Phi_i(p) \neq q$, $(g = \phi_i)$, then bypass stage x and go to stage $x+1$. On the other hand, if $\Phi_i(p) = q$, then cancel the first uncanceled function $\phi_j(p)$, if there be any, which is in the set $\{\phi_0(p), \cdots, \phi_p(p)\}$ and whose step-counting function has the property $\Phi_j(p) \leq g(p)$. To cancel $\phi_j(p)$ means to add (j, p) to list L. After canceling $\phi_j(p)$, go to stage $x+1$.

Now, to compute $f(n)$, look down the list L for a pair (s, n) ending in n. It corresponds to a function ϕ_s for which $\Phi_s(n) \leq g(n)$. If such a pair (s, n) exists, make $f(n) \neq \phi_s(n)$. On the other hand, if (s, n) does not appear in L for any s, set $f(n) = 1$. Note that an effective procedure can determine whether or not (s, n) is in L for some s. That is because the pair (s, n) can appear only in stage $\langle n, m \rangle$, where $\Phi_i(n)$ converges to m.

By construction, if j is any index for f, then $\Phi_j(n) > g(n)$ for almost all n. Hence part (1) of this theorem is immediate. Part (2) follows from the proof of (1) because that proof shows how to construct an f for every g: The proof of (1) furnishes a procedure for going from the index i for g and an integer n to the value of $f(n)$. With i fixed, a machine that follows this procedure computes $f(n)$, and the index of this machine, which is easily determined, is the desired index $\gamma(i)$ for f. Q.E.D.

This last theorem serves as a lemma to the compression theorem, which comes next. For virtually all machines and codes, the compression theorem sets upper and lower bounds on the number of steps needed to compute a function $f(n)$, the lower bound being a partial recursive $\mathcal{G}(n)$ and the upper bound being a total recursive function of it, $h(n, \mathcal{G}(n))$. As a bonus, it demonstrates a curious relationship between measures on computation and bounds on complexity. This relation becomes clearer with the following definition.

Definition. Let $\mathfrak{M}$ be a 0-1 valued total recursive function of three variables with the property that for every i and n there is at most one m such that $\mathfrak{M}(i, n, m) = 1$. Set $\mathcal{G}_i(n) =$ the unique m, if any, such that $\mathfrak{M}(i, n, m) = 1$, and let $\mathcal{G}_i(n)$ diverge if m does not exist. The set $\{\mathcal{G}_i\}$ of partial recursive functions so defined is called a *measured set.*

Example 1. A measured set of functions is the set $\{\Phi_i\}$ of all step-counting functions.

Example 2. A measured set of functions, $\{\mathcal{G}_i\}$, which in fact includes the real-time computable functions [Yamada, 10] is defined by

$$\mathcal{G}_i(n) = \begin{cases} \phi_i(n) & \text{if } \phi_i(n) \text{ converges and } \Phi_i(n) < i \cdot \phi_i(n), \\ \text{divergent} & \text{otherwise.} \end{cases}$$

Example 3. A measured set of functions cannot include all total recursive functions. To see this, suppose to the contrary that $\{\mathcal{G}_i\}$ does contain all total recursive functions. Let function f be defined by

$$f(n) = \begin{cases} 0 & \text{if } \mathfrak{M}(n, n, 1) = 1, \text{ i.e., if } \mathcal{G}_n(n) = 1, \\ 1 & \text{otherwise.} \end{cases}$$

This f is total recursive because $\mathfrak{M}$ is. But $f \notin \{\varphi_i\}$, for if $f = \varphi_i$, then $f(i) = 0 \Leftrightarrow \varphi_i(i) = 1 \Leftrightarrow f(i) = 1$, which is a contradiction.

THEOREM 8 (COMPRESSION). *$\{\varphi_i\}$ is a measured set $\Leftrightarrow$ there exists a total recursive function h such that the following is true: To each partial recursive function φ_i there corresponds a 0-1 valued partial recursive function f with the same domain as φ_i such that (1) if j is any index for f then $\varphi_i(n) < \Phi_j(n)$ for almost all n; (2) there exists an index k for f such that $\Phi_k(n) < h(n, \varphi_i(n))$ for almost all n; (3) there exists a total recursive function τ which maps the subscript i for any φ_i into the index k above for the corresponding f.*

PROOF. $\Rightarrow$: There exists a total recursive function ξ such that $\varphi_i = \phi_{\xi(i)}$ for all i. The proof of the existence of ξ is like that of Theorem 2, so it is left to the reader.

Let γ be the function of Theorem 7, and let $f = \phi_{\gamma\xi(i)}$. Then for every index j for f, Theorem 7 insures that

$$\Phi_j(n) > \phi_{\xi(i)}(n) (= \varphi_i(n)),$$

as in (1). Now set

$$p(i, n, m) = \begin{cases} \Phi_{\gamma\xi(i)}(n) & \text{if } \varphi_i(n) = m, \text{ i.e., if } \mathfrak{M}(i, n, m) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Define $h(n, m) = 1 + \max_{i \leq n} p(i, n, m)$. Then h is total recursive, and $h(n, \varphi_i(n)) > \Phi_{\gamma\xi(i)}(n)$ for almost all n, which means that this h satisfies (2). As for the recursive τ in (3), the function $\tau = \gamma\xi$ maps the subscript i for φ_i into the correct index $\gamma\xi(i)$ for f. This completes the proof in this direction.

$\Leftarrow$: A proof that $\{\varphi_i\}$ is a measured set must show that $\mathfrak{M}$ is total recursive. The proof that $\mathfrak{M}(i, n, m)$ is total recursive is just a procedure for computing it which makes use of the fact that for every i and almost all n,

$$h(n, \varphi_i(n)) > \Phi_{\tau(i)}(n) > \varphi_i(n).$$

To compute $\mathfrak{M}(i, n, m)$, first determine whether $\Phi_{\tau(i)}(n) < h(n, m)$. This can be decided because h is total recursive. If $\Phi_{\tau(i)}(n) \geq h(n, m)$, then $\varphi_i(n) \neq m$, so set $\mathfrak{M}(i, n, m) = 0$. If $\Phi_{\tau(i)}(n) < h(n, m)$, then $\Phi_{\tau(i)}(n)$ converges, and since $\Phi_{\tau(i)}(n) > \varphi_i(n)$, so does $\varphi_i(n)$. In this case it is possible to determine whether $\varphi_i(n) = m$ and so to decide whether to set $\mathfrak{M}(i, n, m) = 0$ or 1. Hence $\mathfrak{M}$ is total. Q.E.D.

In [2], Hartmanis and Stearns prove a beautiful special form of this theorem. Theirs has to do with the multitape machines that print an arbitrary finite number of symbols. Assume these machines use a base 2 input-output code. Also, assume $\{\varphi_i\}$ to be the set of real-time computable functions. Finally, interpret Φ_i as the actual number of moves made by the ith machine over its tapes. Under these conditions the theorem states that h exists and is approximately given by $h(n, m) = m^2$. For example, let $\varphi_i(n) = 2^{2^n}$. Then a bound on the minimum number of steps needed to compute the corresponding f ($f = \phi_j$) is given by

$$2^{2^n} < \Phi_j(n) < 2^{2^{n+1}}.$$

This means that $f(n)$ can be computed by a multitape machine in less than $2^{2^{n+1}}$ steps, but it cannot be computed in less than 2^{2^n} steps, no matter how many additional symbols, tapes or states are added to the machine.

Our proof of the compression theorem is constructive. Hence, it may be used to find h for any class of machines and any measured set $\{\varphi_i\}$. The first corollary is a

case in point. Its proof carefully follows the proof of Theorem 8. In a way, it illustrates the goodness of the theoretical bounds given there.

COROLLARY 1. *Let $\{Z_i\}$ be the class of 1-tape machines that print and erase the symbols b(blank), 0 or 1 and which have a base 2 input-output code. Let $\mathcal{S}_i(n) = \Phi_i(n)$ be the number of steps taken by Z_i with input n. Then the function h of Theorem 8 is total recursive, and its value is given by $h(n, m) < (n + m)^7$. This means that to every step-counting function Φ_i there corresponds a 0-1 valued partial recursive function f with the same domain as Φ_i such that (1) if j is any index for f, then for almost all n, $\Phi_i(n) < \Phi_j(n)$; (2) function f has the index $\tau(i)$ such that for almost all n, $\Phi_i(n) < \Phi_{\tau(i)}(n) < [n + \Phi_i(n)]^7$.*

PROOF. Since $\mathfrak{M} = M$ is the measure on computation, it is total recursive. Hence, by the compression theorem, h exists and is total recursive. The proof of Theorem 8 is an outline for determining the function h given in this corollary. When the details of that proof are filled in, a bound on h is obtained. Filling in the details of that proof is a chore, not difficult but tedious.

To begin, function h is defined in the proof of Theorem 8 in terms of the function

$$p(i, n, m) = \begin{cases} \Phi_{\tau(i)}(n) & \text{if } \Phi_i(n) = m, \\ 0 & \text{otherwise,} \end{cases}$$

where τ represents the function $\gamma\xi$ in that proof. Then

$$h(n, m) = 1 + \max_{i \leq n} p(i, n, m)$$

$$\leq 1 + \max_{i} \{\Phi_{\tau(i)}(n) \mid i \leq n \text{ and } \Phi_i(n) = m\}.$$

So the problem of finding an upper bound on $h(n, m)$ reduces to that of finding an upper bound on $\Phi_{\tau(i)}(n)$. To simplify the calculations, an upper bound is first found on $\hat{\Phi}_{\tau(i)}(n)$ which equals the number of steps needed to compute $f(n)$ on a 10-tape machine. Then the value of $\Phi_{\tau(i)}(n)$ is determined from that of $\hat{\Phi}_{\tau(i)}(n)$. Hartmanis and Stearns [2] show that $\Phi_{\tau(i)}(n) \leq [\hat{\Phi}_{\tau(i)}(n)]^2$ in the case of 1-tape and 10-tape machines that print and erase an arbitrarily large number of symbols. The machines considered in this corollary, however, can only print and erase the blank, 0 and 1, and for this reason the relation is actually $\Phi_{\tau(i)}(n) \leq 10[\hat{\Phi}_{\tau(i)}(n)]^2$, where the 10 corresponds to the 10 tapes of the 10-tape machine.

The computation of $f(n)$ is described in the proof of Theorem 7. What follows here is a bit-by-bit reiteration of the constructive part of that proof, together with bracketed statements giving the number of steps needed to do each bit of the computation. The symbol c is used throughout to denote a constant. To compute $f(n)$:

A. Compute $\Phi_i(n) = m$ and go to stage $0 = \langle 0, 0 \rangle$. Number of steps needed to do this is $[\alpha \leq c + c \log n + cm \log m]$. Then inductively,

B. At stage $x = \langle p, q \rangle$,

1. Determine whether $\Phi_i(p) = q$: $[\beta_1 \leq c + c \log p + cq \log q]$.

Assume that $\Phi_i(p) = q$.

2. Extract the indices $s_1, \cdots, s_t$ of all uncanceled functions in the set $\{\phi_0(p), \cdots, \phi_p(p)\}$: $[\beta_2 \leq c(p + 1)(p + q)(\log (p + q))]$.

3. Pick the smallest index s_k such that $\Phi_{s_k}(p) < q$ and adjoin (s_k, p) to the list.

This requires that the following quantities be determined:

$s(i)$ $=_{df}$ number of steps needed to print the instructions of machine Z_i;

$r(p)$ $=_{df}$ $\sum_{i=0}^{p} s(i)$, $\quad [r(p) < cp^2 \log p]$;

$d(p)$ $=_{df}$ $\sum_{i=0}^{p} d_i$, where $d_i\Phi_i(n)$ is the number of steps needed to simulate the computation of Z_i with input n. $[d(p) < cp\ (\log p)^2]$, $[\beta_3 \leq r(p) + c \cdot d(p)(q \log q) + cqp]$.

C. Go from stage $n = \langle p, q\rangle$ to stage $n+1$ if $\langle p, q\rangle \neq \langle n, m\rangle$: $[\gamma \leq c \log p + c \log q + c]$.

D. If no function $\Phi_s(n)$ is canceled in stage $\langle n, m\rangle$, let $f(n) = 0$. Otherwise, $\Phi_s(n)$ is canceled in stage $\langle n, m\rangle$, and so let $f(n) = 1 \dot{-} \phi_s(n)$. $[\delta \leq c]$.

Total number of steps required to compute $f(n)$ on a 10-tape machine is

$$\hat{\Phi}_{r(i)}(n) \leq \left\{\alpha + \sum_{q=0}^{n+m} \sum_{p=0}^{n+m-q} (\gamma + \beta_1) + (\beta_2 + \beta_3)|_{p+q=n+m} \cdot (1 + n + m) + \delta\right\}$$

$$< c + c[\log (1 + n + m)]^3 \cdot (1 + n + m)^3.$$

But $\Phi_{r(i)}(n) \leq 10[\hat{\Phi}_{r(i)}(n)]^2$. Therefore a permissible bound on h is $h(n, m) < (n + m)^7$. Q.E.D.

Definition. The statement *there exist arbitrarily large total recursive functions r with property A* means that to each total recursive function s there corresponds an r with property A such that $r(n) > s(n)$ for all n. For example, there exist arbitrarily large step-counting functions (Theorem 4).

Definition. To say that *all sufficiently large total recursive functions r have property A* is to say that there exists a total recursive function s such that if $r(n) > s(n)$ for all n then r has property A. For example, all sufficiently large total recursive functions r take at least 2^n steps to compute for almost all n. (This follows from Theorem 3 and the fact that a standard Turing machine whose step-counting function counts actual numbers of steps takes 2^n steps just to print the value of $r(n)$.)

COROLLARY 2. *There exist arbitrarily large total recursive functions r, and associated with each r a 0-1 valued total recursive function f, such that (1) if i is any index for f, then $\Phi_i(n) > r(n)$ for almost all n; (2) there exists an index k for f such that $r(n + 1) > \Phi_k(n) > r(n)$ for almost all n.*

The proof of this corollary is left to the reader.

One might suppose that to all sufficiently large total recursive functions h there correspond total recursive functions f such that (1) if i is any index for f, then $h(n) < \Phi_i(n)$ for almost all n; (2) there exists an index k for f such that $h(n) < \Phi_k(n) < h[h(n)]$ for almost all n. But this is false unless some constraint binds h to a measured set. In fact, there exist arbitrarily large total recursive functions h such that for every index j, either $\Phi_j(n) < h(n)$ or $h[h(n)] < \Phi_j(n)$ for infinitely many n. Disappointing though this may be, a weak form of this compression does hold. One can show that to each sufficiently large total recursive function h there corresponds a total recursive function f such that (1) if i is any index for f, then for infinitely many n, $h(n) < \Phi_i(n)$; (2) for some index k for f, $h(n) < \Phi_k(n) < h[h(n)]$ for infinitely many n. To prove this statement, first show that to each sufficiently large total recursive function h there corresponds an index j such

that $h(n) < \Phi_j(n) < h[h(n)]$ for infinitely many n. Then invoke the compression theorem to obtain the result.

ACKNOWLEDGMENTS. The author expresses his sincere appreciation to Dr. Warren S. McCulloch, Professor Marvin Minsky, and Dr. Juris Hartmanis for their enthusiasm and support of this work, to Dr. Michael Arbib, Professor William Kilmer, and the referee for this and also for their help in revising the manuscript, and to Professor Hartley Rogers, Jr., for his superb lectures and notes on recursive function theory.

REFERENCES

1. ARBIB, M. A., AND BLUM, M. Machine dependence of degrees of difficulty. *Proc. Amer. Math. Soc. 16*, 3 (June 1965), 442–447.
2. HARTMANIS, J., AND STEARNS, R. E. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. 117*, 5 (May 1965), 285–306.
3. —— AND ——. Computational complexity of recursive sequences. Proc. 5th Annual Symp. on Switching Theory and Logical Design, Princeton, N. J., 1964.
4. MYHILL, J. Linear bounded automata. WADD Tech. Note 60–165, U. of Pennsylvania Rep. No. 60–22 (June 1960).
5. RABIN, M. O. Degree of difficulty of computing a function and a partial ordering of recursive sets. Tech. Rep. No. 2, Hebrew U., Jerusalem, Israel (April 1960).
6. ——. Real time computation. *Israel J. Math. 1* (1963), 203–211.
7. RITCHIE, R. W. Classes of predictably computable functions. *Trans. Amer. Math. Soc. 106*, 1 (June 1963), 139–173.
8. ROGERS, H., JR. Gödel numberings of partial recursive functions. *J. Symbolic Logic 23*, 3 (Sept. 1958), 331–341.
9. ——. Recursive functions and effective computability. McGraw-Hill, New York (in press).
10. YAMADA, H. Real-time computation and recursive functions not real-time computable. *IRE Trans. EC-11*, 66 (Dec. 1962), 753–760.
11. COBHAM, A. The intrinsic computational complexity of functions. Proc. 1964 Int. Congress on Logic, Methodology and Philosophy of Science. North-Holland, Amsterdam, 1965, 24–30.
12. COOKE, S. A. On the minimum computation time of functions. Bell Labs. Rep. BL-41, 1966.
13. WINOGRAD, S. On the time required to perform multiplication. IBM Res. Rep. RC-1564, 1966.

RECEIVED OCTOBER, 1965; REVISED JUNE, 1966