# ON THE MINIMUM COMPUTATION TIME OF FUNCTIONS

BY

STEPHEN A. COOK AND STÅL O. AANDERAA[1]

## I. MACHINES WITH A BOUNDED NUMBER OF ACTIVE ELEMENTS

1. **Introduction.** An underlying goal of the research reported here is to develop a theory showing that the process of multiplying decimal integers is intrinsically more difficult than the process of adding them [2]. In part I of this paper we present a precise conjecture which is a possible formulation of this proposition. The formulation is not completely satisfactory, but it certainly bears on the problem and raises interesting mathematical questions. Part II contains a proof of a weakened form of the conjecture.

There are several different ways in which multiplication seems to be more difficult than addition, and each could lead to a different formulation of the problem. For example, experience of computer designers indicates that it takes more circuitry to compute the fixed-point product of two $n$-digit integers than to find the sum. Again, even with the increased circuitry, the time required to carry out the multiplication exceeds that for addition. But still a third way in which the difficulties seem to differ is suggested not by computers, but by our centuries old experience with longhand multiplication. If a man is equipped only with paper and pencil, in general it will take him longer to multiply two numbers than to add them. It is this experience with longhand multiplication which motivates the present theory.

Thus a rough attempt to formalize our belief is embodied in the following statement: Given any algorithm for multiplying arbitrary decimal integers, and an algorithm for adding them, the number of steps required, on the average, to apply the first algorithm to a given pair of integers should substantially exceed the number of steps required by the second. Since the ordinary addition algorithm requires a number of steps proportional to the number of digits in the addends, the statement can be rephrased as follows:

1.1. No matter what algorithm is used, the number of steps, on the average,

Received by the editors July 24, 1968.

[1] Most of the results in this paper are taken from Chapters I and II of the doctoral thesis [4] of the first author. The second author contributed some central ideas to the main result (Theorem 10.1) of part II. The first author was supported in part while writing the present paper by Office of Naval Research Contract Nonr 3656 (23).

[2] See Cobham [2] for an interesting discussion of the problems in developing such a theory.

required to multiply two decimal integers is more than proportional to the number of digits in the factors.

The conjecture 1.1 leads naturally to a second question; namely, just how fast must the number of steps grow as a function of the number of digits in the factors, assuming the algorithm is as efficient as possible? A little thought shows that the ordinary method of multiplying requires a number of steps proportional to the product of the numbers of digits in the two factors. Thus, if the two factors have a common number $n$ of digits, the number of steps is proportional to $n^2$. But it turns out that the ordinary method is far from the best possible one. In fact A. L. Toom [13] and A. Schönhage [11] have each devised a different algorithm for reducing the number of steps from $n^2$ to $n^{1+\varepsilon}$, for arbitrary small $\varepsilon>0$. Schönhage showed that his method can be realized by a multi-tape Turing machine to multiply in time proportional to $n^{1+(\sqrt{2}+\varepsilon)/(\log_2 n)^{1/2}}$, and Cook [4] proved the same result for Toom's method, except the time estimate was not as sharp. Just how much further the bound can be reduced remains as an open question, although the results in part II touch on the problem.

In order to make the conjecture 1.1 precise, it is necessary to give a clear description of (i) the circumstances under which the algorithm is to be carried out, and (ii) just what constitutes a "step" during the execution of the algorithm. In other words, we must provide a mathematical model of the computer (man or machine) which is to execute the algorithm, and the model should include the notion of step. The computer should be a general purpose one, in the sense that it should be capable of executing any algorithm upon being presented with the proper program (i.e. description of the algorithm). Thus the storage (or memory) of the machine must be potentially infinite.

The class of ordinary single-tape Turing machines satisfies the above conditions, provided we take the program of a particular machine to be the specification of the tape symbols and the state transition function. So also does the class of multi-tape Turing machines, the class of iterative arrays of finite state machines [5], and the class of Shepherdson-Sturgis machines [12]; and there is no reason to believe that any two classes lead to equivalent definitions of step. The problem, then, is to decide what is the right class of machines.

Atrubin [1] has shown that there is a one-dimensional iterative array of finite state machines (cf. 5.2) which multiplies in "real time". That is, when the digits of two integers are presented to the machine at the extreme left end of the array a pair at a time, the same machine indicates the product digits at the rate of one per cycle. Thus 1.1 fails if we allow iterative arrays as executors of the algorithms. A little thought suggests a reason why. It is that the number of finite state machines in the array which are taking an active part in the computation grows without bound as a function of time. Our feeling that the number of steps per product digit should constantly increase with time depends on an assumption that the executor of the algorithm be able to change the "instantaneous description" of the computation by only a bounded amount during any one step. The iterative array can multiply in real time because it can recruit a constantly growing army of personnel to do the necessary computation[3].

2. **Bounded-activity machines.** The key to making the conjecture 1.1 precise, then, is to capture the notion of a machine whose number of active elements is bounded in time. Or, equivalently, one which is capable of changing its instantaneous description by at most a bounded amount during any one step in the computation. In the latter formulation, we must also impose the restriction that the changes depend on a bounded (but possibly variable) part of the configuration. For example, a Turing machine with a finite number of read-write heads and tapes fulfills both conditions. Here, the active elements are the heads, and the tapes are just passive storage. The conditions are satisfied just as well if the "tapes" are planar arrays of squares instead of linear arrays; and, for that matter, if they are higher-dimensional arrays. In fact, the general machine we have in mind consists of a storage structure (such as a set of tapes) together with a finite set of "heads," capable of reading and altering information stored in the structure. The behavior of the heads for a given step of the computation depends only on the symbols stored in the cells currently occupied by the heads (and on which pairs of heads currently coincide in position).

This leads to the following series of definitions, in which a notion of computation with a bounded number of active elements is introduced formally. The notion introduced is unfortunately much too general, but it does provide a framework around which to center discussion concerning the nature of the "proper model", which might involve suitable restrictions on the general model. For example, one possible restriction, discussed in the next section, is sufficient to prevent the machine from multiplying in real time.

2.1. Definition. A *storage structure* $\Lambda = \langle L, \phi_1, \ldots, \phi_p \rangle$ of *rank* $p$ consists of a countable set $L$ (the *locations* or *cells* of $\Lambda$) together with the maps $\phi_1, \ldots, \phi_p$ of $L$ into $L$, called *shift transformations.*

Thus, a multi-tape Turing machine has a storage structure of rank 2; and the two shift transformations $\phi_1$, $\phi_2$ are the left shift and the right shift. (We regard the tape as fixed and let the heads move.)

A bounded-activity machine, defined formally in 2.2, consists of a storage structure together with a "program", which specifies (i) the number of read-write

[3] On the other hand, the interesting results of Winograd [14] concerning multiplication time do not fit into the framework of the present discussion at all. Winograd is interested in the minimum possible delay time through a logic net that is designed to multiply. The net is given access to, and processes, all the digits of the factors simultaneously, and as a result the multiplication time for two $n$-digit numbers grows considerably *slower* than $n$. The time function we are interested in turns out to be more related to the number of components in the net than the delay time through the net. Winograd does not consider how the number of components of the net grows with the number of digits.

heads, (ii) a finite set $\Sigma$ of tape symbols capable of being written (or stored) in the storage cells, and (iii) a transition function which specifies the symbols to be written by the heads and the proper shift transformations to be applied, for each possible combination of symbols currently being scanned by the heads. The machine might also include a finite set of internal states, like those of a Turing machine. However, these can always be dispensed with by adding another read-write head, which never moves, but is capable of reading and writing as many symbols as there were states in the original machine. Thus we shall not include the internal states in Definition 2.2.

When two or more heads scan the same cell, provision must be made so that conflicting write orders are not given. Also, in "programming" the machine, it is convenient to have information available as to which of the heads are scanning the same square. This is the function of the partition $P$ in the definition below. Finally, the input to the machine consists of an arbitrary string of symbols (from a finite alphabet $\Gamma$), which are made available to the machine on a read-only input tape. The input and output arrangements are discussed further in §4.

Here, now, is the formal definition.

2.2. DEFINITION. A *bounded-activity machine*, or BAM, is a quintuple $\langle \Lambda, \Gamma, \Sigma, H, \Phi\rangle$, where $\Lambda = \langle L, \phi_1, \ldots, \phi_p\rangle$ is a storage structure, $\Gamma$ is a finite set of *input* symbols, $\Sigma$ is a finite set of *tape* symbols, $H$ is a positive integer (specifying the number of *heads* of the machine), and $\Phi$ is the *transition function*. The domain of $\Phi$ is the set of all $(H+2)$-tuples $\langle t, s_1, \ldots, s_H, P\rangle$, where $t \in \Gamma$, $s_i \in \Sigma$, and $P$ is a partition (i.e. an equivalence relation) on $\{1, 2, \ldots, H\}$. The range of $\Phi$ is a subset of the set of all $(2H+1)$-tuples $\langle t_1, \ldots, t_H, \psi_1, \ldots, \psi_H, s\rangle$, where $t_i \in \Sigma$, $\psi_i \in \{\phi_1, \phi_2, \ldots, \phi_p, I_L\}$, and $s \in \{0, 1\}$. Here $I_L$ is the identity map on $L$. ($s$ specifies whether or not the input tape is to be advanced.)

Thus, suppose the heads numbers $1, 2, \ldots, H$ are scanning symbols $s_1, \ldots, s_H$ respectively, and the input head is currently reading the symbol $t$, and the coincidence of heads is specified by the partition $P$ (i.e. head $i$ and head $j$ are scanning the same square if and only if $i \equiv j \pmod P$). Suppose $\Phi(\langle t, s_1, \ldots, s_H, P\rangle) = \langle t_1, \ldots, t_H, \psi_1, \ldots, \psi_H, s\rangle$. Then head $i$ will print the symbol $t_i$ and move to the cell specified by $\psi_i$, $i = 1, 2, \ldots, H$. The input tape is advanced or not according as $s$ is 1 or 0.

In order to avoid conflicting write commands, we shall require of $\Phi$ that if $i \equiv j \pmod P$, then $t_i = t_j$.

2.3. DEFINITIONS. Let $M = \langle \Lambda, \Gamma, \Sigma, H, \Phi\rangle$ be a BAM with storage structure $\Lambda = \langle L, \phi_1, \ldots, \phi_p\rangle$. A *storage assignment* on $M$ is a map $\alpha : L \to \Sigma$. An *instantaneous description* of $M$ is an $(H+3)$-tuple $\langle \alpha, x_1, x_2, \ldots, x_H, A, e\rangle$, where $\alpha$ is a storage assignment, $x_1, \ldots, x_H$ are members of $L$ (specifying the current locations of the heads), $A$ is a finite string on $\Gamma$ (the input string), and $e$ is a positive integer (specifying which entry of $A$ is currently being scanned). The *display* of an instantaneous description $\langle \alpha, x_1, \ldots, x_H, A, e\rangle$ is the $(H+2)$-tuple $\langle t, s_1, \ldots, s_H, P\rangle$, where $t$ is the symbol in position $e$ of the string $A$ (the display is not defined if $e$ exceeds the length of $A$), $s_i = \alpha(x_i)$, $i = 1, \ldots, H$, and $P$ is the partition on $1, \ldots, H$ defined by $i \equiv j \pmod P$ if and only if $x_i = x_j$. Suppose $I_1 = \langle \alpha, x_1, \ldots, x_H, A, e\rangle$ and $I_2 = \langle \beta, y_1, \ldots, y_H, B, f\rangle$ are two instantaneous descriptions. Let $\langle t, s_1, \ldots, s_H, P\rangle$ be the display of $I_1$ (if it exists), and suppose $\Phi(\langle t, s_1, \ldots, s_H, P\rangle) = \langle s_1', \ldots, s_H', \psi_1, \ldots, \psi_H, s\rangle$. Then $I_2$ is the *successor* of $I_1$ provided

(i) $\beta(x) = s_i'$ if $x = x_i$, $\beta(x) = \alpha(x)$ if $x \in L$ but $x \notin \{x_1, \ldots, x_H\}$,

(ii) $y_i = \psi_i(x_i)$, $i = 1, \ldots, H$,

(iii) $B = A$, and

(iv) $f = e + s$.

2.4. Definitions. A *computation* of a machine $M$ is a finite or infinite sequence $I_0, I_1, \ldots$ of instantaneous descriptions such that $I_{i+1}$ is the successor of $I_i$, $i = 0, 1, \ldots$, and either the sequence is infinite or it is finite and the last instantaneous description has no successor (i.e. the input head has run off the input tape). The *step number* of $I_t$ relative to the computation $I_0, I_1, \ldots$ is the index $t$. The *display sequence* of the computation is the sequence of displays of the successive instantaneous descriptions of the computation.

Our description of machine computations will be informal. We shall refer to the *computation at time t*, and mean the instantaneous description $I_t$. For example, the cell scanned by head $i$ at time $t$ of a computation is $x_i$, if $I_t = \langle \alpha, x_1, \ldots, x_H, A, e\rangle$.

3. **Uniform machines.** Our main result in part II, that under certain restrictions no BAM can multiply in real time, does not hold for an arbitrary BAM because it is possible in effect to build a multiplication table into the storage structure. One way to exclude this possibility is to require the storage structure to be "uniform." That is, every location should behave the same with respect to the shift transformations as every other.

3.1. Definitions. Let $\Lambda = \langle L, \phi_1, \ldots, \phi_p\rangle$ be a storage structure. An *automorphism* (or *translation*) of $\Lambda$ is a permutation $\psi$ of $L$ such that $\psi\phi_i = \phi_i\psi$, $i = 1, 2, \ldots, p$. The structure $\Lambda$ is *uniform* if for every pair of locations $x, y \in L$ there is an automorphism $\psi$ of $\Lambda$ such that $\psi(x) = y$. We shall refer to a BAM with a uniform storage structure as a *uniform BAM*.

It is interesting to note that while the shift transformations of a uniform storage structure always map onto the location set, they need not be one-one. As an example, let $L$ be the set of pairs $\langle i, j2^i\rangle$, where $i$ and $j$ are any integers such that $j \geqq 0$. Let $\phi: L \to L$ be defined by $\phi(\langle i, j2^i\rangle) = \langle i+1, [j/2]2^{i+1}\rangle$, where $[x]$ is the greatest integer not exceeding $x$. Then $\phi$ is not one-one, but it can be verified that $\langle L, \phi\rangle$ is uniform.

Uniformity is a sufficient condition on a storage structure to enable the arguments in part II to go through. Thus, for example, although an iterative array can multiply in real time, no BAM with a uniform storage structure can, provided its storage is initially "blank". Nevertheless, we need more of a restriction than uniformity before we have a suitable model of a computer, as the following result shows.

3.2. Theorem. *There is a uniform BAM which computes a nonrecursive function in real time.*

**Proof.** For a precise description of the input-output arrangement and definition of real time, see §4.

Suppose $G$ is a group with generators, $a_1, \cdots, a_n$. Let $\Gamma = \{a_1, \ldots, a_n, a_1^{-1}, \ldots, a_n^{-1}\}$, $\Delta = \{0, 1\}$, and let $f\colon \Gamma^* \to \Delta$ ($\Gamma^*$ is the set of finite strings on $\Gamma$) be defined by $f(A) = 1$ if and only if the string $A$ represents the identity element for the group $G$. We shall construct a machine which computes $f$ in real time, and hence solves the word problem for $G$. In particular, if $G$ has an unsolvable word problem, the machine will compute a nonrecursive function.

The machine $M = \langle \Lambda, \Gamma, \Sigma, H, \Phi \rangle$ is defined as follows. $\Lambda = \langle L, \phi_1, \ldots, \phi_n, \phi_1^{-1}, \ldots, \phi_n^{-1} \rangle$, where $L$ is the set of elements of $G$, $\phi_i(x) = a_i \cdot x$, and $\phi_i^{-1}(x) = a_i^{-1} \cdot x$, where $\cdot$ is the group operation of $G$, and $x \in L$. Further, $\Sigma = \{s_0\}$, $H = 2$, and $\Phi(t, s_0, s_0, P) = \langle s_0, s_0, \phi_i, I_L, 1 \rangle$ if $t = \phi_i$, and $\phi(\langle t, s_0, s_0, P \rangle) = \langle s_0, s_0, \phi_i^{-1}, I_L, 1 \rangle$ if $t = a_i^{-1}$. We assume initially both heads scan the same cell $x_0$. Thus head 2 remains at cell $x_0$ throughout the computation, while at a time $t$ head 1 is in location $x \cdot x_0$, where $x$ is the element of $G$ specified by the first $t$ input symbols. In particular, the heads coincide if and only if the input string to date represents the identity of $G$. Hence the output of the machine is a simple function of the partition $P$ on $\{1, 2\}$.

Finally, the storage structure $\Lambda$ is uniform, for given $x, y \in L$ the map $\psi$ defined by $\psi(z) = z \cdot x^{-1} \cdot y$ sends $x$ into $y$, and, as is readily verified, $\psi$ is an automorphism of $\Lambda$ (but, of course, not a group automorphism).

Thus uniformity is not a sufficient condition to guarantee that the machine computes only recursive functions. Of course this guarantee could be established by simply requiring the shift transformation $\phi_1, \ldots, \phi_p$ to be recursive, but such a requirement would represent a very small step toward finding the "right" computer model.

A simple condition on the shift transformations that forces the resulting machine to be well behaved is discussed in §6.

## 4. **Input and output arrangements.**

4.1. Notation. Let $\Gamma$ and $\Delta$ be finite alphabets. Then $\Gamma^*$ is the set of finite strings (excluding the empty string) on $\Gamma$, and $\Gamma^n$ is the set of all strings on $\Gamma$ of length $n$. If $A \in \Gamma^*$, then $|A|$ denotes the length of $A$. We are interested in having machines compute functions $f\colon \Gamma^* \to \Delta$, and for this we need the notion of an output of a BAM. Let $M$ be a BAM, let $I$ be an instantaneous description of $M$, and let $\langle t, s_1, \ldots, s_H, P \rangle$ be the display of $I$. Then the *restricted display* of $I$ is the $(H+1)$-tuple $\langle s_1, \ldots, s_H, P \rangle$. An *output* of $M$ is a map $\lambda$ from the set of all possible restricted displays of $M$ to the set $\Delta' = \Delta \cup \{0\}$, where 0 is an extra *dead* symbol not occurring in $\Delta$. The members of $\Delta$ are called *output symbols*, but 0 is not. If $D_1, D_2, \ldots$ is the sequence of restricted displays of a computation, then the *output sequence* of the computation is obtained from $\lambda(D_1)$, $\lambda(D_2), \ldots$ by deleting the dead symbols.

The function computed by a machine $M$ depends on the output function $\lambda$ and on the initial position of each head of $M$. *We shall henceforth assume these parameters have been specified when referring to the output of a machine.* We shall further assume that the tape alphabet $\Sigma$ has a distinguished symbol $s_0$ called the *blank*.

4.2. Definitions. Let $M=\langle \Lambda, \Gamma, \Sigma, H, \Phi\rangle$ be a BAM with output $\lambda$ and initial cells $x_1, \ldots, x_H$. Given $A \in \Gamma^*$, we define the *computation of $M$ with input $A$* to be the computation of $M$ whose initial instantaneous description is $\langle \alpha, x_1, \ldots, x_H, A, 1\rangle$, where $\alpha(x)=s_0$ for all cells $x$ (i.e. the storage is initially blank). Now suppose $f$: $\Gamma^* \to \Delta$, where $\Delta$ is the set of output symbols of $\lambda$. We say *$M$ computes $f$* provided first $\lambda(D_0)=0$, where $D_0$ is the restricted display of the initial instantaneous description, and second, for all $A \in \Gamma^*$ there is a number $r$ such that for all $B \in \Gamma^r$ the output sequence $d_1, d_2, \ldots$ of the computation of $M$ with input $AB$ has length at least $l=|A|$, and $d_l=f(A)$. In this case, let $T_B(A)$ be the step number (cf. 2.4) of the instantaneous description which specified $d_l$ for the computation with input $AB$. The *computation time* of $M$ at the input $A$ is $\max_{|B|=r} T_B(A)$. This time is independent of $r$, provided $r$ satisfies the property above for which it was introduced.

Suppose $T(n)$ is a nondecreasing function from positive integers to positive integers. Then *$M$ computes $f$ within time $T(n)$* provided $M$ computes $f$, and for each $n$, the computation time for no string in $\Gamma^n$ exceeds $T(n)$. We say $M$ computes $f$ in *real time* provided $M$ computes $f$ within time $T(n)$, where $T(n)=n$. Thus, in this case, $M$ must furnish an output symbol during every step of the computation except the initial step $I_0$, and we may as well also assume that the input tape is advanced during every step of the computation.

The notion of "on line" computations for Turing machines is standard in the literature (see, for example, Hennie [7]). In the present context we have the following definition.

4.3. A BAM $M$ computes *on line* provided that for all $n$, and all input strings of length at least $n+1$, the input head scans the $(n+1)$th input symbol for the first time in the same step it produces the $n$th output symbol.

Equivalently, $M$ computes on line provided that for each input string $A$ and each pair $I_t$, $I_{t+1}$ of consecutive steps in the computation with input $A$, $M$ advances the input tape between $I_t$ and $I_{t+1}$ iff $M$ produces an output symbol (not the dead symbol) at $I_{t+1}$. Note that in particular, if $M$ computes a function $f$ in real time, then $M$ computes $f$ on line.

We now introduce the particular functions considered in part II.

4.4. Notation. If $\Gamma=\{0, 1, \ldots, b-1\}$, and $c_1c_2 \cdots c_n \in \Gamma^*$, then $\|c_1c_2\cdots c_n\|_b$ denotes the number $c_1+c_2b+\cdots+c_nb^{n-1}$, whose base $b$ expansion is $c_nc_{n-1}\cdots c_1$.

In each of the following examples, the function $f$ takes $\Gamma^*$ into $\Delta$.

4.5. Example (Squaring). $\Gamma=\Delta=\{0, 1, \ldots, b-1\}$. $f(c_1c_2\ldots c_n)$ is the $n$th digit in the base $b$ expansion of the number $(\|c_1c_2\cdots c_n\|_b)^2$. For instance, if $b=10$, then (since $(25)^2=625$), $f(5)=5$, $f(52)=2$, $f(520)=6$, $f(5200)=0$, etc.

4.6. Example (Multiplication).

$$\Gamma = \left\{\text{ordered pairs } \begin{pmatrix} c \\ d \end{pmatrix} \mid c, d \in \{0, 1, \ldots, b-1\}\right\}.$$

$$\Delta = \{0, 1, \ldots, b-1\}.$$

$$f\left(\begin{pmatrix} c_1 \\ d_1 \end{pmatrix}\begin{pmatrix} c_2 \\ d_2 \end{pmatrix}\cdots\begin{pmatrix} c_n \\ d_n \end{pmatrix}\right)$$

is the $n$th digit in the base $b$ expansion of $\|c_1c_2\cdots c_n\|_b\cdot\|d_1d_2\cdots d_n\|_b$.

4.7. Example (Polynomial multiplication over $Z_b$). ($Z_b$ is the ring of integers modulo $b$.)

$$\Gamma = \left\{\text{ordered pairs } \begin{pmatrix} c \\ d \end{pmatrix} \mid c, d \in \{0, 1, \ldots, b-1\}\right\}.$$

$$\Delta = \{0, 1, \ldots, b-1\}.$$

$$f\left(\begin{pmatrix} c_1 \\ d_1 \end{pmatrix}\begin{pmatrix} c_2 \\ d_2 \end{pmatrix}\cdots\begin{pmatrix} c_n \\ d_n \end{pmatrix}\right) = c_1d_n + c_2d_{n-1} + \cdots + c_nd_1,$$

where addition and multiplication are taken modulo $b$. Thus if $P(X)=c_1+c_2X+\cdots+c_nX^{n-1}$ and $Q(X)=d_1+d_2X+\cdots+d_nX^{n-1}$ then a machine computing $f$ with input

$$\begin{pmatrix} c_1 \\ d_1 \end{pmatrix}\cdots\begin{pmatrix} c_n \\ d_n \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix}\cdots\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

will put out successively the coefficients in the product $P(X)Q(X)$.

Polynomial multiplication over $Z_b$ can also be viewed as base $b$ multiplication in which the carries are ignored.

5. **Turing machines and iterative arrays.** The point of part II is to prove that there are certain operations, such as multiplication, that an iterative array of finite state machines can perform faster than any uniform BAM. In this section we consider the converse question: what can BAM's do faster than iterative arrays. Some uniform BAM's can certainly compute functions faster, since according to 3.2 they can compute a nonrecursive function in real time; something no iterative array can do at any speed. Hence we shall restrict our attention to those BAM's that closely resemble Turing machines. The results are that, while every multi-tape Turing machine (with linear tapes) can be simulated in real time by a one-dimensional iterative array, there are Turing machines with planar tapes which cannot be so simulated.

First we shall define the two kinds of machines involved—multi-tape Turing machines, in order to emphasize that they are a special case of uniform BAM; and iterative arrays, in order to make clear which of the definitions in the literature we are using. Our definition is equivalent to that of Atrubin [1] (who constructed an array which multiplies in real time), Cole [3], and Fischer [5].

5.1. DEFINITION. A multi-tape Turing machine (with $m$ work tapes and an input tape) is a BAM whose storage structure $\Lambda = \langle L, \phi_1, \phi_2\rangle$ is as follows: $L$ is the set of all pairs of integers $\langle i, j\rangle$ such that $1 \leqq j \leqq m$; $\phi_1(\langle i, j\rangle) = \langle i-1, j\rangle$ (left shift); and $\phi_2(\langle i, j\rangle) = \langle i+1, j\rangle$ (right shift).

Thus $j$ is the tape number, and $i$ indicates the position on the tape. Normally we assume there is exactly one head per tape, in which case the initial position of head $j$ is $\langle 0, j\rangle$, $j = 1, 2, \ldots, m$. However, occasionally (as in Theorem 5.3) we allow more than one head per tape, in which case the initial position of each extra head coincides with that of one of the first $m$ heads. The storage structure is uniform since, given a pair $x, y \in L$, we can define an automorphism $\psi$ of $\Lambda$ by interchanging the tape containing $x$ with the tape containing $y$ and then sliding one tape left and one tape right the appropriate number of squares. That is, if $x = \langle i_1, j_1\rangle$ and $y = \langle i_2, j_2\rangle$, then $\psi(\langle i, j\rangle) = \langle i, j\rangle$ if $j \neq j_1, j_2$; $\psi(\langle i, j_1\rangle) = \langle i - i_1 + i_2, j_2\rangle$; and $\psi(\langle i, j_2\rangle) = \langle i - i_2 + i_1, j_1\rangle$. Then $\psi$ obviously commutes with the two shift transformations.

Note that a set of internal states is not necessary, as explained before Definition 2.2.

Turing machines with $n$-dimensional tapes can be defined similarly to 5.1, by letting $L$ be the set of $(n+1)$-tuples $\langle i_1, \ldots, i_n, j\rangle$ of integers such that $1 \leqq j \leqq m$. The shift transformations $\phi_1, \ldots, \phi_{2n}$ are defined by $\phi_k(\langle i_1, \ldots, i_n, j\rangle) = \langle i_1, \ldots, i_k + 1, \ldots, i_n, j\rangle$ and $\phi_{n+k}(\langle i_1, \ldots, i_n, j\rangle) = \langle i_1, \ldots, i_k - 1, \ldots, i_n, j\rangle$, $1 \leqq k \leqq n$.

Our discussion of iterative arrays will be informal.

5.2. DEFINITION. A one-dimension iterative array consists of an infinite sequence $M_1, M_2, \ldots,$ of identical finite state machines together with an initial finite state machine $M_0$. The state transition functions are such that

(i) the state of $M_i$, $i > 0$, at time $t+1$ depends exactly on the states of $M_{i-1}$, $M_i$, and $M_{i+1}$ at time $t$,

(ii) the state of $M_0$ at time $t+1$ depends exactly on the states of $M_0$ and $M_1$ and the input to $M_0$ at time $t$, and

(iii) the machines $M_1, M_2, \ldots$ are in a quiescent state $q_0$ at time $t = 0$, and the transitions are such that $M_i$ will remain in state $q_0$ at time $t+1$ if $M_{i-1}$, $M_i$, and $M_{i+1}$ are in state $q_0$ at time $t$, $(i > 0)$.

The output of the array at time $t$ is a function only of the state of $M_0$ at time $t$, and is defined similarly to the output of a BAM (cf. 4.1). The input to the array and the computation time of the array are defined as in the case of BAM's.

5.3. THEOREM. *Suppose $f$: $\Gamma^* \rightarrow \Delta$ is computable in time $T(n)$ by a multi-tape Turing machine; possibly with several heads per tape (but with linear tapes). Then $f$ is computable in time $T(n)$ by a one-dimensional iterative array.*

We shall omit the proof. Cole [3, pp. 5–30 to 5–33] proved a very similar result, and an outline of the proof appears in [4, p. 20].

By adapting results of Hennie [7], we can show the previous theorem fails when the "tapes" of the Turing machine are allowed to be two dimensional. Hennie defines a function $f$: $\Gamma^* \to \Delta$, where $\Gamma=\{0, 1, 2\}$ and $\Delta=\{0, 1\}$ [7, p. 36] for which he shows that any multi-tape Turing machine which computes $f$ *on line* (cf. 4.3) must have computation time at least $T(n)=Cn^2/(\log n)^2$. He then states that there is a two-dimensional Turing machine which computes $f$ in time only $Cn^{3/2}$. Thus the two-dimensional Turing machine computes $f$ faster than can any one-dimensional iterative array, *subject to the on-line restriction.* However, it is possible to modify the function $f$ by including a dummy symbol $d$ in its domain alphabet $\Gamma$, and by making $f(A)=0$ unless $A$ includes appropriately long strings of $d$'s between every pair of symbols from $\{0, 1, 2\}$. Just enough $d$'s should be included so that the two-dimensional Turing machine has enough time to compute $f$ at the 0–1–2 part of a string before the next symbol from $\{0, 1, 2\}$ comes along. In this way we can find a two-dimensional Turing machine which computes $f^*$, the modified version of $f$, in real time. On the other hand, no one-dimensional array can compute $f^*$ in real time, for otherwise a modified version of the array (which would artificially insert the missing $d$'s) could compute $f$ in time $Cn^{3/2}$, which we know is impossible. Therefore, we have

5.4. THEOREM. *Theorem* 5.3 *fails when the Turing machine is allowed to have two-dimensional tapes.*

## 6. **Abelian machines.**

6.1. DEFINITION. A storage structure $\Lambda$ is *abelian* iff each shift transformation is a permutation and any two shift transformations commute. A BAM is *abelian* iff its storage structure is abelian.

Suppose $\Lambda=\langle L, \phi_1, \dots, \phi_p\rangle$ is an abelian storage structure. Then $\{\phi_1, \dots, \phi_p\}$ generates an abelian group $G$ of permutations on $L$. (Each member of $G$ is an automorphism of $\Lambda$.) We say $\Lambda$ is *connected* iff the set $L$ is an orbit under $G$; that is, iff for each pair $x, y \in L$ there is $\phi \in G$ so $\phi(x)=y$. Note that in this case $y$ is not necessarily accessible from $x$ by applying a finite sequence of shift transformations, but rather by applying a finite sequence of shift transformations and inverses of shift transformations. Obviously connected abelian storage structures are uniform.

6.2. LEMMA. *Let* $\Lambda=\langle 1, \phi_1, \dots, \phi_p\rangle$ *be a connected abelian storage structure, and let $G$ be the permutation group generated by* $\{\phi_1, \dots, \phi_p\}$. *Then for each pair* $x, y \in L$ *there is exactly one* $\phi \in G$ *such that* $\phi(x)=y$.

**Proof.** Suppose $\phi_1(x)=\phi_2(x)=y$, where $\phi_1, \phi_2 \in G$. Let $z$ be any element of $L$, and let $\psi \in G$ be such that $\psi(x)=z$. Then $\phi_1(z)=\phi_1\psi(x)=\psi\phi_1(x)=\psi\phi_2(x)=\phi_2\psi(x)$ $=\phi_2(z)$. Since $z$ is arbitrary, $\phi_1=\phi_2$.

We wish to show that abelian BAM's are similar to Turing machines with multi-dimensional tapes (cf. after 5.1) in the sense that the two kinds of machines compute functions at about the same rate. In particular, it will follow that only recursive functions are computable by abelian BAM's. This is true even though the shift transformation of such machines need not be recursive.

It is apparent that a Turing machine with several $n$-dimensional tapes is already an abelian BAM, so our task is to prove the following theorem.

6.3. THEOREM. *Suppose $f\colon \Gamma^* \to \Delta$, and suppose $M_1$ is an abelian BAM which computes $f$ within time $T(n)$. Then there is a Turing machine $M_2$ with multidimensional tapes such that $M_2$ computes $f$ within time $KT(n)$ for some integer $K$.*

**Proof.** Suppose $M_1=\langle \Lambda_1, \Gamma, \Sigma_1, H, \Phi_1\rangle$, where $\Lambda_1=\langle L_1, \phi_1, \dots, \phi_p\rangle$ is abelian. Let $G$ be the permutation group generated by $\{\phi_1, \dots, \phi_p\}$. Although $L_1$ may have infinitely many orbits under $G$, initially the finite set of heads can scan locations in only finitely many orbits, and a given head must remain in the same orbit throughout any computation. Hence we may as well assume $L_1$ has only finitely many orbits under $G$. The Turing machine $M_2$ to be constructed will have one multidimensional tape for each orbit. We will carry out the construction for the case of one orbit only, since the general case requires more complicated notation but no new ideas. Thus we shall assume $\Lambda_1$ is connected.

By the structure theorem for finitely generated abelian groups, $G$ is the direct sum of subgroups $G_I$ and $G_F$, where $G_I$ is a finitely generated free abelian group and $G_F$ is finite. Thus there are homomorphisms $\pi_I\colon G \to G_I$ and $\pi_F\colon G \to G_F$ such that for each $\phi \in G$, $\phi=\pi_I(\phi)\pi_F(\phi)$. Fix an element $x_0 \in L_1$. Then there is a natural one-one correspondence $\gamma$ of $G$ onto $L_1$ defined by $\gamma(\phi)=\phi(x_0)$. The map $\gamma$ is one-one by 6.2. Let $\pi_I', \pi_F'\colon L_1 \to L_1$ be defined by $\pi_I'=\gamma\pi_I\gamma^{-1}$ and $\pi_F'=\gamma\pi_F\gamma^{-1}$.

The storage structure for $M_2$ is given by $\Lambda_2=\langle L_2, \psi_1, \dots, \psi_n, \psi_1^{-1}, \dots, \psi_n^{-1}\rangle$, where $L_2=\pi_I'(L_1)$ and $\{\psi_1, \dots, \psi_n\}$ is a set of free generators for $G_I$. The structure $\Lambda_2$ is isomorphic to the $n$-dimensional tape described after 5.1 (since we have only one tape, the parameter $j$ may be deleted). The isomorphism sends the $n$-tuple $\langle i_1, \dots, i_n\rangle$ of integers to $\psi_1^{i_1}\psi_2^{i_2}\cdots\psi_n^{i_n}(x_0)$.

We will assume for convenience that the machine $M_2$ to be constructed is equipped with a finite state control like that in a conventional Turing machine. As explained before, this device can be eliminated by adding an extra head which always scans the same location and records the current state there.

We let $M_2=\langle \Lambda_2, \Gamma, \Sigma_2, H, \Phi_2, Q\rangle$, where $\Gamma$ and $H$ are the same as for $M_1$ and $\Lambda_2$ is as defined above. The symbol set $\Sigma_2=\Sigma_1^{G_F}$; the set of maps $\rho\colon G_F \to \Sigma_1$. The finite set $Q$ of internal states is given by $Q=G_F^H \times S^H$, where $S$ is the set of strings of length $K-1$ or less on the alphabet $\{\psi_1, \dots, \psi_n, \psi_1^{-1}, \dots, \psi_n^{-1}\}$. The integer $K$ will be specified later.

Before describing $\Phi_2$ formally, we shall first describe the computations of $M_2$ in terms of the computations of $M_1$. To each instantaneous description $I=\langle \alpha, x_1, \dots, x_H, A, e\rangle$ of $M_1$ we associate the instantaneous description $\delta(I)=\langle \alpha', \pi_I'(x_1), \dots, \pi_I'(x_H), A, e, q\rangle$ of $M_2$, where the storage assignment $\alpha'\colon L_2 \to \Sigma_2$

is given by $\alpha'(x)(\phi)=\alpha(\phi(x))$, $x \in L_2$, $\phi \in G_F$, and the state $q \in Q$ is $\langle\langle \pi_F\gamma^{-1}(x_1), \ldots, \pi_F\gamma^{-1}(x_H)\rangle, \langle \varepsilon, \ldots, \varepsilon\rangle\rangle$, where $\varepsilon$ is the empty string. Notice that $I$ can be uniquely recovered from $\delta(I)$ (i.e. $\delta$ is one-one). If $I_1, I_2, \ldots$ are the successive instantaneous descriptions of a computation of $M_1$, then the corresponding computation of $M_2$ will consist of $\delta(I_1)$, $\delta(I_2), \ldots$ occurring successively, but with each pair separated by up to $K-1$ intermediate steps. In general, $M_2$ cannot go from $\delta(I_i)$ to $\delta(I_{i+1})$ in a single step because each shift transformation $\phi_i$ of $M_1$ must be represented by a string of more than one shift transformation $\psi_j$ of $M_2$. The purpose of the second component $S^H$ of $Q$ is to keep track of the remaining shifts $\psi_j$ necessary to achieve the current shift $\phi_i$ of $M_1$.

To describe $\Phi_2$ formally, let us associate with each shift transformation $\phi_i$ a string $\sigma(\phi_i)=\eta_1\eta_2\cdots\eta_l \in S$ of elements of $\{\psi_1, \ldots, \psi_n, \psi_1^{-1}, \ldots, \psi_n^{-1}\}$ whose composition is $\pi_I(\phi_i)$. The number $K$ is one plus the length of the longest $\sigma(\phi_i)$. Now suppose $D=\langle t, \rho_1, \ldots, \rho_H, P, q\rangle$ is a display of $M_2$, and suppose $q=\langle\langle \xi_1, \ldots, \xi_H\rangle, \langle B_1, \ldots, B_H\rangle\rangle$. Let us write the string $B_i=\eta_i C_i$, where $\eta_i$ is the first member of $B_i$ if $B_i \neq \varepsilon$, and $\eta_i=\varepsilon$ if $B_i=\varepsilon$. If not all $\eta_i$ are empty, we define $\Phi_2(D)=\langle \rho_1, \ldots, \rho_H, \eta_1', \ldots, \eta_H', 0, q'\rangle$, where $\eta_i'$ is $\eta_i$ if $\eta_i \neq \varepsilon$ and $\eta_i'=I_{L_2}$ (the identity transformation) if $\eta_i=\varepsilon$, and $q'=\langle\langle \xi_1, \ldots, \xi_H\rangle, \langle C_1, \ldots, C_H\rangle\rangle$.

Now suppose $B_i=\varepsilon$, $i=1, 2, \ldots, H$. Then the above display $D$ for $M_2$ comes from an instantaneous description $I$ such that the corresponding instantaneous description $\delta^{-1}(I)$ of $M_1$ has the associated display $D_1=\langle t, \rho_1(\xi_1), \ldots, \rho_H(\xi_H), P\rangle$. Suppose $\Phi_1(D_1)=\langle t_1, \ldots, t_H, \theta_1, \ldots, \theta_H, s\rangle$. Then $\Phi_2(D)=\langle \rho_1', \ldots, \rho_H', I_{L_2}, \ldots, I_{L_2}, s, q\rangle$, where $\rho_i'(\phi)=\rho_i(\phi)$ if $\phi \neq \xi_i$ and $\rho_i'(\xi_i)=t_i$, and $q=\langle\langle \xi_1\pi_F(\theta_1), \ldots, \xi_H\pi_F(\theta_H)\rangle, \langle \sigma(\theta_1), \ldots, \sigma(\theta_H)\rangle\rangle$.

This completes our description of $M_2$ except for the output function $\lambda_2$, which is constructed from the output function $\lambda_1$ of $M_1$ in the obvious manner.

## II. A Negative Result

7. **Discussion.** Now, with the apparatus of BAM's, the conjecture 1.1 can be made precise in several possible ways. Perhaps the simplest formulation would state that if $f$ is the base $b$ multiplication function defined in 4.6, $b \geqq 2$, and $c$ is a positive constant, then no BAM computes $f$ within time $T(n)=cn$. However, this proposition is false, as the following theorem shows.

7.1. Theorem. *Let $\Gamma$ and $\Delta$ be finite alphabets, and suppose $f$: $\Gamma^* \to \Delta$. Then there is some BAM which computes $f$ in real time.*

**Proof.** We shall assume, for the sake of readability, that $\Delta=\{1, 2\}$. Suppose $\Gamma=\{a_1, \ldots, a_k\}$. Let $\Gamma'=\{a_1', \ldots, a_k'\}$ be a disjoint copy of $\Gamma$ and let $h$: $\Gamma^* \cup \{\varepsilon\} \to \Gamma'^* \cup \{\varepsilon\}$ be the semigroup isomorphism defined by $h(a_i)=a_i'$, $i=1, 2, \ldots, k$, where $\varepsilon$ is the empty string. Let $\Lambda=\langle L, \phi_1, \ldots, \phi_{2k}\rangle$ be the storage structure in which $L=\Gamma^* \cup \Gamma'^* \cup \{\varepsilon\}$, and for $i=1, 2, \ldots, k$ and for all $A \in \Gamma^* \cup \{\varepsilon\}$,

$$\begin{aligned} \phi_i(A) &= \phi_i(h(A)) = Aa_i, \\ \phi_{k+i}(A) &= \phi_{k+i}(h(A)) = Aa_i \quad \text{if } f(A) = 1, \\ &= h(A)a_i' \quad \text{if } f(A) = 2. \end{aligned}$$

Let $M=\langle \Lambda, \Gamma, \Sigma, H, \Phi\rangle$ be the BAM in which $\Lambda$ and $\Gamma$ are as above, $\Sigma=\{s_0\}$, $H=2$ and the transition function $\lambda$ is given by

$$\begin{aligned} \lambda(s_0, s_0, P) &= 1 \quad \text{if } P \text{ indicates both heads coincide,} \\ &= 2 \quad \text{otherwise.} \end{aligned}$$

Initially both heads are assumed to be scanning the location $\varepsilon$. It is easy to see that $M$ computes $f$ in real time.

Thus, in particular, some BAM can multiply decimal integers in real time. Examination of the construction shows that this machine has the multiplication tables, in effect, built into its storage structure. Hence, in order to formulate a reasonable negative proposition concerning multiplication it is necessary to place some regularity condition on the storage structure. Uniformity (cf. 3.1) is one natural condition. A second condition is the following.

7.2. Definition. A storage structure $\Lambda=\langle L, \phi_1, \ldots, \phi_p\rangle$ has *polynomial-limited accessibility* provided there are constants $K$ and $n$ such that the number of locations accessible in $t$ steps from any given location does not exceed $Kt^n$. Here a location $y$ is *accessible* from a location $x$ in $t$ steps provided there is a sequence $\psi_1, \psi_2, \ldots, \psi_t$ of shift transformations such that $y=\psi_1\psi_2\cdots\psi_t(x)$.

Note that the storage structures of multi-tape Turing machines have polynomial-limited accessibility, even if the tapes are multi-dimensional. In fact, the storage structure of any BAM whose shift transformations commute has this property. There are, however, storage structures which are uniform but do not have polynomial-limited accessibility. An example is the structure obtained from a finitely generated free group with two or more generators. (Here the location set $L$ is the set of elements of the group, and a shift transformation is right multiplication by one of the generators.)

Thus a reasonable formulation of the conjecture 1.1 might be made in terms of BAM's whose storage structures are uniform and/or have polynomial-limited accessibility. Unfortunately we have not been able to prove any such proposition without adding the further restriction that the machine computes on line. Our main theorem, 10.1, does apply to BAM's which are either uniform or have polynomial-limited accessibility, and which compute on line.

8. **Other work.** Theorem 10.1 may be far from the best possible negative result for multiplication, but it does provide a basis for distinguishing the computing power of multitape Turing machines and one-dimensional iterative arrays. (Compare our 10.3 and Atrubin's result in [1], where he shows that a one-dimensional iterative array can multiply in real time.) This answers a question posed, for example, by McNaughton [9, p. 404].

Other negative results in the literature concerning time-limited Turing machine computations either apply equally well to iterative arrays, or apply only to Turing machines with one tape. For example, Hartmanis and Stearns [6] exhibit a context-free language and give a simple proof that no multi-tape Turing machine can recognize it in real time. The same proof also applies to iterative arrays. Hennie [7] exploits this method of proof to find a language which requires at least time $n^2/(\log n)^2$ for recognition by a Turing machine operating on line. Again, his techniques yield a similar result for iterative arrays. The relevant results in Hennie [8] and Rabin [10] apply only to Turing machines with a single tape.

9. **Complex functions.** The three functions squaring, multiplication, and polynomial multiplication (cf. 4.5–4.7) share a certain type of complexity which is exploited in the proof that they cannot be computed in real time. We shall now isolate this property, and call functions which possess it "complex."

Suppose $f$: $\Gamma^* \rightarrow \Delta$, where $\Gamma$ and $\Delta$ are finite alphabets with at least two symbols each. The relevant property can be described roughly by saying that for any machine that computes $f$ in real time, the outputs during any time interval $I_2$ depend heavily on the inputs during the previous time interval $I_1$. Now let $C$ and $X$ be strings with a common length $l$ on $\Gamma$, let $E$ be a string of arbitrary length on $\Gamma$, and suppose $C=c_1c_2\cdots c_l$; $c_i \in \Gamma$. If we think of $C$ as the input to the machine during $I_2$, $X$ the input during $I_1$, and $E$ the input previous to $I_1$, the above statement can be rephrased to say that the sequence of values $f(EXc_1), f(EXc_1c_2), \ldots,$ $f(EXc_1c_2\cdots c_l)$ depends heavily on $X$. To make this precise, we must define an equivalence relation $\equiv_{E,C}$ on the values of $X$; namely if $X_1, X_2 \in \Gamma^l$, then $X_1 \equiv_{E,C} X_2$ if and only if $f(EX_1c_1\cdots c_i)=f(EX_2c_1\cdots c_i)$, $i=1,2,\ldots,l$. The equivalence classes must be small in order for $f$ to be complex. Actually, this need only be true for certain $E$; and in the following precise definition such $E$ will be written as the concatenation of two strings $A$ and $B$.

9.1. Definition. The function $f$ is *complex*, provided there are numbers $l_0$, $\theta$, $0<\theta<1$, such that for all sufficiently large integers $N$ there is a string $A$ on $\Gamma$ of length $N$ such that for all $B$, $C \in \Gamma^*$ with $l_0 \leqq |C| \leqq N/2$, no equivalence class under $\equiv_{AB,C}$ has more than $b^{\theta l}$ members, where $b$ is the number of elements in $\Gamma$, and $l=|C|$.

9.2. Lemma. *Squaring, multiplication, and polynomial multiplication are complex functions.*

**Proof.** We shall prove this for multiplication. The argument for squaring is very similar, and the argument for polynomial multiplication is almost the same, except there is no worry about carries.

Let $f$ be the multiplication function defined in 4.6. Given $N$ and $l$ with $2l \leqq N$, we wish to consider strings $ABXC$ on the alphabet of pairs of digits from $\{0, 1, \ldots,$ $b-1\}$, where $|A|=N$, and $|X|=|C|=l$, and $|B|$ is arbitrary. Let $M=|B|$. The string $ABXC$ encodes two integers in base $b$ notation. Let these integers be $P_1$ and $P_2$. Then

$$P_i = A_i + b^N B_i + b^{N+M} X_i + b^{N+M+l} C_i, \qquad i = 1, 2,$$

where $A_1$ and $A_2$ are the two integers encoded by $A$, $B_1$ and $B_2$ are the two integers encoded by $B$, and similarly for $X_i$ and $C_i$. Now choose $A$ so

$$A_1 = b + b^2 + b^4 + \cdots + b^{2^n} = \sum_{i=0}^{n} b^{2^i} \quad (n = [\log N]), \qquad A_2 = 0.$$

Then

$$(9.3) \qquad P_1 P_2 = b^{N+M} A_1 X_2 + A_1(b^N B_2 + b^{N+M+l} C_2) + 2B_1 B_2 b^{2N} + b^{N+M+2l} D,$$

for some integer $D$ (recall $N \geqq 2l$).

Our strategy is to show that given the integers $B_1$, $B_2$ and $C_2$ and digit numbers $N+M+l+1$ through $N+M+2l$ (where digit "number one" is the lowest order digit) of $P_1P_2$, we can recover, with slight uncertainty, about a quarter of the digits of $X_2$. Hence we have narrowed the possibilities for $X$, which is the same as limiting the size of the equivalence class of $X$ under $\equiv_{AB,C}$.

Given the $l$ product digits mentioned in the preceding paragraph, as well as $B_2$ and $C_2$, it is evident from (9.3) that we can determine digit numbers $N+M+l+1$ through $N+M+2l$ of $b^{N+M}A_1X_2$, and hence digit numbers $l+1$ through $2l$ of the product $A_1X_2$. Call these latter digits $d_{l+1}, d_{l+2}, \ldots, d_{2l}$.

Now let $l = 2^k + m$, where $0 \leqq m \leqq 2^k$, and suppose the digits of $X_2$ (from low order to high order) are $x_1, x_2, \ldots, x_l$. We claim that if

$$r = \|x_{m+2} x_{m+3} \cdots x_{2m}\|_b \quad \text{and} \quad s = \|d_{l+2^k+2} \cdots d_{l+2^k+m}\|_b$$

(cf. 4.4 for notation) then either $r = s$ or $r+1 = s$[4]. To see this, write

$$\begin{aligned} A_1 X_2 &= X_2(b + b^2 + b^4 + \cdots + b^{2^n}) \\ &= r b^{2^{k+1}+N+1} + \{b^{2^{k+1}} \|x_1 x_2 \cdots x_{m+1}\|_b + X_2(b + b^2 + \cdots + b^{2^k})\} + t b^{2^{k+1}+2m}, \end{aligned}$$

for some integer $t$. The last term contributes nothing to $s$, and, letting $B$ abbreviate the expression between the braces, we have

$$B < b^{2^{k+1}+m+1} + b^{2^k+l+1}$$

(note that $2^{k+1} + m = 2^k + l$), so $B$ contributes only a carry of 0 or 1 to $s$. This establishes the claim.

Thus we are able to narrow down the possible values of the $(m-1)$-tuple $\langle x_{m+2}, \ldots, x_{2m} \rangle$ to two. A similar argument enables us to narrow the possible values of the $(2^{k-1} - m - 1)$-tuple $\langle x_{2^{k-1}+m+2} \cdots x_{2^k} \rangle$ to two, provided $m < 2^{k-1} - 1$. Hence, in any case, we can narrow to four the possible values of a string of $2^{k-1} - 2$ digits. Since $X$ encodes a total of $2l$ digits (the base $b$ notation for both $X_1$ and $X_2$),

[4] Except if $r = b^{m-1} - 1$, then possibly $s = 0$.

this leaves undetermined a total of at most $4b^{2l-(2^{k-1}-2)}$ members of the equivalence class of $X$ under $\equiv_{AB,C}$. Since $l<2^{k+1}$ we have $2l-2^{k-1}+4<(9/10)(2l)$, provided $l\geqq 64$, and therefore the equivalence class has at most $(b^2)^{0.9l}$ members for $l>64$. Thus we can take $l_0=64$ and $\theta=0.9$, and Definition 9.1 will be satisfied for the case of multiplication (recall $\Gamma$ has $b^2$ elements for multiplication).

## 10. The main theorem.

10.1. Theorem. *Suppose $f$: $\Gamma^* \to \Delta$ is a complex function, and suppose $M$ is a bounded-activity machine which computes $f$ on line within time $T(n)$. Then, provided the storage structure of $M$ is either uniform* 3.1 *or has polynomial-limited accessibility* 7.2,

$$T(n) > \delta n \log n/(\log\log n)^2 \quad (10.2)$$ [5]

*for some $\delta>0$ and arbitrarily large values of $n$.*

10.3. Corollary. *No multi-tape Turing machine can multiply in real time. The same holds even if the machine has multi-dimensional tapes and several read-write heads per tape.*

The corollary follows from the theorem by Lemma 9.2 and the fact that these generalized Turing machines have uniform storage structures.

**Proof of 10.1.** Assume the theorem is false. Then there is a complex function $f$: $\Gamma^*\to\Delta$ and a bounded activity machine $M=\langle\Lambda, \Gamma, \Sigma, H, \Phi\rangle$ with a storage structure $\Lambda$ which is either uniform or has polynomial-limited accessibility which computes $f$ on line within time $T(n)$, where $T(n)$ violates the inequality (10.2) for all $\delta>0$ and all $n>B=B(\delta)$. In particular, setting $\delta=1$, we have

$$T(n) \leqq n\log n/(\log\log n)^2, \qquad n > B(1). \quad (10.4)$$

Given a large positive integer $r$, let $N=r^r$ and let $A$ be the string on $\Gamma$ of length $N$ with the properties guaranteed by the definition of complex (cf. 9.1). In §§11–14 we shall show the existence of a string $D$ of length $N$ such that

$$T(AD) > \delta\, 2N\log(2N)/(\log\log(2N))^2, \quad (10.5)$$

provided $r$ is sufficiently large, where the positive number $\delta$ depends only on the machine $M$, and where $T(AD)$ is the time the machine $M$ takes to process the string $AD$ (i.e. the number of steps $M$ takes to give all $2N$ outputs associated with $AD$). This contradiction will establish the theorem. (The inequality (10.5) is proved as (14.4), where $\delta=\delta_2/4$.)

It remains to construct the string $D$. In the remainder of the paper the function $f$ and machine $M$ are as above. We assume $A$ and $D$ are strings of length $N$ on the input alphabet $\Gamma$. We think of $A$ and $N$ as being fixed, and will talk as if the parameters of the computation depend only on $D$.

[5] All logarithms in part II are natural logarithms.

11. **The notion of overlap.** It will be useful to number the entries of $D$ from 1 to $N$ and consider them to be divided into input intervals of various lengths. Thus an *input interval* is a sequence of consecutive integers chosen from the set $\{1, 2, \ldots, N\}$. If $I$ is the input interval $\{k+1, \ldots, k+l\}$, then for each choice of the string $D$, there is a string assigned to the interval $I$; namely the substring of $D$ consisting of symbols in position $k+1, \ldots, k+l$. Also associated with the input interval $I$ and the string $D$ is the *time interval* $I^\tau$, which refers to the sequence of steps the machine $M$ undergoes with input $AD$ while scanning the input interval $I$. Precisely, $I^\tau$ is the subsequence of the computation of $M$ with input $AD$ consisting of those instantaneous descriptions $\langle \alpha, x_1, \ldots, x_H, AD, e\rangle$ for which $e$ is in $I$. Since $M$ computes on line, all output symbols except the last one corresponding to the input $I$ are designated during $I^\tau$.

We are now in a position to define overlap.

11.1. Definition. Suppose $I$ and $J$ are adjacent (in the obvious sense) input intervals, and suppose the machine $M$ is given the input $AD$. Let $S$ be the set of locations which are scanned by some head during the time interval $I^\tau$, and let $T$ be the set scanned by some head during $J^\tau$. Then the *overlap* caused by the input $D$ during input intervals $I$ and $J$, denoted $\omega^D(I, J)$, is the number of locations in $S \cap T$. The set $S \cap T$ will be denoted by $\Omega^D(I, J)$.

Now choose positive integers $r$ and $m$, and suppose $N = r^m$. For each $i$, $i = 1, 2, \ldots, m$, we divide the entries of $D$ into $r^{m-i}$ successive intervals $I_{i1}, I_{i2}, \ldots, I_{ir}m-i$ of length $r^i$ each. Thus

$$I_{ij} = \{(j-1)r^i+1, (j-1)r^i+2, \ldots, (j-1)r^i+r^i\}.$$

Let

$$\Omega^D_{ij} = \Omega^D(I_{i,j-1}, I_{ij}), \qquad j > 1, \tag{11.2}$$

and

$$\omega^D_{ij} = \omega^D(I_{i,j-1}, I_{ij}), \qquad j > 1. \tag{11.3}$$

11.4. Lemma. *Let $T(AD)$ be the total time taken to process the input $AD$, and let $H$ be the number of heads of the machine. Then*

$$T(AD) \geqq \frac{1}{H}\Bigl(\sum_{j \not\equiv 1 (\text{mod } r)} \omega^D_{ij}\Bigr),$$

*where the sum is taken over all pairs $i, j$ with $1 \leqq i \leqq m$, $1 \leqq j \leqq r^{m-i}$, and $j$ not of the form $1+rk$.*

**Proof.** First note that $\omega^D_{ij}$ is the number of elements in $\Omega^D_{ij}$, and $\Omega^D_{ij}$ is the set of locations which are scanned during both the adjacent time intervals $I^\tau_{i,j-1}$ and $I^\tau_{ij}$. The idea is to count the steps in the computation by associating with every location in $\Omega^D_{ij}$ a different step. The association is such that even the same location $x$ will be assigned different steps if $x$ appears in different overlap sets $\Omega^D_{ij}$ (provided

$j \not\equiv 1 \pmod r$)). Actually, the association is not quite one-one, since a total of $H$ distinct locations may be scanned in any one step. This accounts for the factor $1/H$ in the above inequality.

To make the argument precise, let $\pi$ be the set of pairs $(h, t)$, where $h$ is an integer, $1 \leqq h \leqq H$, and $t$ is a step in the second "half" of the computation the machine undergoes with input $AD$; that is a step while the machine is processing the string $D$. For each pair $i, j$ with $1 \leqq i \leqq m$, $1 \leqq j \leqq r^{m-i}$, and $j \equiv 1 \pmod r$, define the function $f_{ij}$: $\Omega_{ij}^D \rightarrow \pi$ as follows: For each $x \in \Omega_{ij}^D$, let $t$ be the first step of the interval $I_{ij}^\tau$ such that some head scans $x$, let $h$ be the smallest-numbered head which scans $x$ at time $t$, and define $f_{ij}^D(x) = (h, t)$. Thus, although the square $x$ is scanned several times during the pair of adjacent intervals corresponding to the overlap set, we count only the first time of the second interval.

Since a given head scans only one square at a given time, each of the functions $f_{ij}^D$ is one-one. Further, if $(i, j) \neq (k, l)$, then the ranges of the two functions $f_{ij}^D$ and $f_{kl}^D$ are disjoint. This is so because either (i) the time intervals $I_{ij}^\tau$ and $I_{kl}^\tau$ do not overlap at all, or (ii) say $I_{ij}^\tau$ is a subinterval of $I_{kl}^\tau$ (but not the first subinterval, since $j \not\equiv 1 \pmod r$)), in which case a given head cannot scan a location $x$ at a time $t$ which is the first-scanned time in *both* $I_{ij}^\tau$ and $I_{kl}^\tau$ and still satisfy the condition that $x$ is in the overlap set $\Omega_{ij}^D$ of the smaller interval.

Since each of the functions is one-one, and their ranges are pairwise disjoint, we can conclude $|\pi| \geqq \sum_{j \not\equiv 1 (\mathrm{mod}\, r)} \omega_{ij}^D$ (where $|\pi|$ is the number of elements in $\pi$), from which the lemma follows immediately.

12. **Consequences of the restrictions on** $\Lambda$. We now wish to construct a string $D$ such that for many values of $i, j$, the overlap $\omega_{ij}^D$ will be large, and hence, according to 11.4, the computation time $T(AD)$ will be large. This is done roughly as follows. Let $I$ and $J$ be two adjacent input intervals of length $l$, and suppose the input string is fixed except during the interval $I$, when it is a variable $X$. The first step is to assume that $X$ is restricted to some subset $S_\omega$ of $\Gamma^l$ such that the overlap $\omega^D(I, J)$ does not exceed the value $\omega$. The restriction on the overlap puts an upper bound on the number of behaviors (i.e. display sequences) possible during $J^\tau$, and hence on the number of possible output strings during $J^\tau$. This upper bound is stated as the inequality (12.3) of Lemma 12.1. (Condition (ii) of the lemma embodies the restriction on the overlap, while condition (iii) states a necessary restriction on the processing time to ensure the upper bound holds.) On the other hand, since $f$ is a complex function, the output during $J^\tau$ depends heavily on $X$, and hence the set $S_\omega$ must be small for small values of $\omega$. In other words, for most values of $X$, $\omega$ is large. This statement is made precise in Lemma 13.1.

12.1. LEMMA. *Suppose* $2N > B(1)$, *so that* (10.4) *holds with* $n = 2N$. *Suppose* $\omega$ *and* $T$ *are integers,* $0 \leqq \omega \leqq T$, $B$ *and* $C$ *are strings on* $\Gamma$ *whose combined length does not exceed* $N - |C|$ (*we shall call such a pair* $(B, C)$ *an* admissible *pair*), *and suppose* $C = c_1 c_2 \cdots c_l$, $c_i \in \Gamma$. *Let* $I, J$ *be the input intervals corresponding to* $X$ *and* $C$ *in the* *input string $D=BXCF$ for any strings $X$, $F$ on $\Gamma$ such that $|X|=l$ and $|D|=N$. Let $S(\omega, T, B, C)$ be the set of output strings (for fixed $A$, variable $X$)*

$$f(ABXc_1)f(ABXc_1c_2)\cdots f(ABXC) \tag{12.2}$$

*such that*

(i) $X\in\Gamma^l$,

(ii) $\omega^D(I,J)\leqq\omega$ *for* $D=BXCF$, *and*

(iii) *the length of neither of the time intervals* $I^\tau$, $J^\tau$ *exceeds* $T$.

*Then*

$$M(\omega, T, B, C) \leqq K_1(NT)^{K_2}\binom{K_3T}{\omega} \tag{12.3}$$

*where $M(\omega, T, B, C)$ is the number of elements in $S(\omega, T, B, C)$, and $K_1$, $K_2$, $K_3$ are constants depending only on the machine (recall $N=|A|=|D|$).*

**Proof.** The proof is divided into two cases; one for each of the alternative conditions on the storage structure $\Lambda$.

*Case* I. Assume $\Lambda$ has polynomial-limited accessibility. Suppose at most $Kt^n$ cells are accessible from any one cell within $t$ steps, and suppose the strings $B$ and $C$ are given.

The output string (12.2) is designated by the machine during $J^\tau$, except the last symbol $f(ABXC)$ is not designated until immediately after $J^\tau$. Now we claim that the display sequence during $J^\tau$ (and hence all entries of (12.2) except possibly the last) is completely determined (independent of $X$, given $A$, $B$, $C$ and the machine $M$) from the following information:

(i) The position of each of the heads at the first step of $J^\tau$, and

(ii) for each $x\in\Omega^D(I,J)$ the triple $\langle s_x, t_x, h_x\rangle$, where $s_x$ is the tape symbol stored at $x$ at the first step of $J^\tau$, $t_x$ is the number of the first step of $J^\tau$ in which $x$ is scanned, and $h_x$ is one of the heads (say the one with the smallest number) scanning $x$ at step number $t_x$. (The location $x$ itself need not be specified.)

In fact, not only the display sequence is determined, but also the location of each head at each step of $J^\tau$ is determined. This assertion follows by a straightforward induction, using the facts that the instantaneous description just before $I^\tau$ is known, and the inputs during $J^\tau$ are known to constitute $C$.

Now, assuming the length of $I^\tau$ does not exceed $T$, the position of a given head at the beginning of $J^\tau$ must be one of at most $KT^n$ cells. Hence, part (i) of the above information is determined by specifying one of at most $(KT^n)^H$ possibilities.

Also, assuming $\omega^D(I,J)\leqq\omega$ and the length of $J^\tau$ does not exceed $T$, the information in part (ii) can be specified by selecting a subset of $\omega$ elements from the set of triples $\{\langle s,t,h\rangle \mid s\in\Sigma, 1\leqq t\leqq T, 1\leqq h\leqq H\}$. (If $\omega^D(I,J)<\omega$, extra harmless triples $\langle s,t,h\rangle$ can be added so that the subset always has exactly $\omega$ elements.) Hence there are at most

$$\binom{\sigma TH}{\omega}$$

possibilities for part (ii), where $\sigma$ is the number of elements of $\Sigma$. Therefore, the maximum number of output strings (12.2) for fixed $B$, $C$, is

$$(KT^n)^H \binom{\sigma TH}{\omega} d,$$

where $d$ is the number of symbols in $\Delta$ (i.e. the number of possibilities for $f(ABXC)$). The inequality (12.3) follows.

*Case* II. Assume now that $\Lambda$ is uniform. First we need a lemma.

12.4. Lemma (for uniform case). *If $\psi_1$ and $\psi_2$ are two translations of a uniform storage structure $\Lambda = \langle L, \phi_1, \ldots, \phi_p \rangle$, and $\psi_1(x) = \psi_2(x)$ for some $x \in L$, then $\psi_1(y) = \psi_2(y)$ for each $y$ accessible from $x$. (Cf. 7.2 for a definition of accessible.) Since the identity map is always a translation, it follows that if $\psi_1(x) = x$, then $\psi_2(y) = y$ for all $y$ accessible from $x$.*

**Proof.** Suppose $\psi_1(x) = \psi_2(x)$, and $y = \phi(x)$, where $\phi$ is a shift transformation. Then $\psi_1(y) = \psi_1\phi(x) = \phi\psi_1(x) = \phi\psi_2(x) = \psi_2\phi(x) = \psi_2(y)$. The lemma now follows by induction on the number of shifts required to obtain $y$ from $x$.

To handle Case II, assume again that $B$ and $C$ are given, but $X$ is variable. This time we cannot supply the information in (i) of the previous case, because the lack of a polynomial bound on the accessibility leads to too many possible head positions. However, it is possible to construct the display sequence during $J^\tau$ even without knowing the head positions at the beginning of $J^\tau$, provided the following information is available:

(i) For each head $h$, the step number of $J^\tau$ and location $x$ of $L$ (if such exist) at which $h$ first scans a cell that has been scanned previously to $I^\tau$ (i.e. during the time of the input string $AB$),

(ii) same as (ii) for Case I, and

(iii) for each pair of heads $h_1$, $h_2$, the step numbers $t_1(h_1, h_2)$ and $t_2(h_1, h_2)$, which are defined as follows. Let $S(h_1, h_2)$ be the set of locations scanned by $h_1$ during $J^\tau$ and then by $h_2$ at the same or a later time of $J^\tau$, and let $x(h_1, h_2)$ be the member of $S(h_1, h_2)$ first scanned by $h_1$ (if $S(h_1, h_2)$ is nonempty). Then $t_i(h_1, h_2)$ is the number of the earliest step of $J^\tau$ at which $h_i$ scans $x(h_1, h_2)$, $i = 1, 2$.

Although the display sequence during $J^\tau$ is determined by (i)–(iii), the locations of the heads are not necessarily determined. However, suppose $C_a$ and $C_b$ are two computations with the same parameters (i)–(iii), and suppose for $C_a$ the heads $1, \ldots, H$ scan cells $x_{1t}, \ldots, x_{Ht}$ at step number $t$, while for $C_b$ the heads scan cells $y_{1t}, \ldots, y_{Ht}$ at step number $t$. Let $\psi_i$ be a translation of $\Lambda$ such that $\psi_i(x_{it_0}) = y_{it_0}$, $i = 1, \ldots, H$, where $t_0$ is the number of the first step in $J^\tau$. Then, for each step number $t$ of a step of $J^\tau$,

(12.5)
(1) $\psi_i(x_{it}) = y_{it}$, $\quad i = 1, \ldots, H$, and
(2) the displays of $C_a$ and $C_b$ are the same at step number $t$.

The statements of (12.5) are proved jointly by induction on $t$. Lemma 12.4 and the specification (iii) are applied to show $\psi_{h_1}(x)=\psi_{h_2}(x)$ for each $x\in S(h_1, h_2)$.

It remains to count the possible specifications (i)–(iii). According to (10.4), the total computation time of the input string $AD$ does not exceed $2N\log(2N)$, so the total number of locations scanned during the computation cannot exceed $H\cdot 2N\log(2N)$. By 12.1, (iii), the number of steps in $J^\iota$ does not exceed $T$. Hence the number of possibilities for (i) is bounded by $(2NTH\log(2N))^H$. Simple arguments show that the numbers of possibilities for (ii) and (iii) are bounded respectively by

$$\binom{\sigma TH}{\omega} \quad \text{and} \quad (T^2)^{H^2}.$$

Thus, the maximum possible number of output strings (12.2) for fixed $B$, $C$ is

$$(2NTH\log(2N))^H\binom{\sigma TH}{\omega}(T^2)^{H^2}d,$$

where, as in Case I, $d$ is the number of symbols of $\Delta$. The inequality (12.3) follows.

13. **Consequences of the complexity of** $f$. Since $f$ is complex, there are numbers $l_0$ and $\theta$ satisfying 9.1. Let $r$, $m$ be positive integers such that $r>l_0$ and $N=r^m$ is sufficiently large within the meaning of 9.1. Let the string $A$ be as in 9.1, and for $i=1, 2,\ldots, m$ let $l_i=r^i$.

13.1. Lemma. *Let $\omega_1, \omega_2,\ldots, \omega_m$ and $T_1, T_2,\ldots, T_m$ be nonnegative integers, and let $b$ be the number of elements in $\Gamma$. Then there is a string $D$ in $\Gamma^N$ such that for each interval $I_{ij}, j>1$ (cf.* §11 *for notation) if we write $D=BXCF$, with $X$ and $C$ the strings associated with $I_{i,j-1}$ and $I_{ij}$, respectively, and $C=c_1c_2\cdots c_{l_i}$, then the string $f(ABXc_1)f(ABc_1c_2)\cdots f(ABXC)$ is not a member of $S(\omega_i, T_i, B, C)$, provided*

$$M(\omega_i, T_i, B, C) \leqq b^{\varepsilon l_i}/2mN, \tag{13.2}$$

*where $\varepsilon=1-\theta$. Here $S(\omega, T, B, C)$ and $M(\omega, T, B, C)$ are as in Lemma* 12.1.

**Proof.** Let us count the strings $D$ in $\Gamma^N$ which *do not* satisfy the conditions of the lemma. If $D$ is such, then there is some $i$, some pair of strings $B$, $C$ with $|C|=l_i$, and some pair $X$, $F$ such that $D=BXCF$, and the output sequence associated with $C$ is in $S(\omega_i, T_i, B, C)$, and (13.2) holds. There are $m$ choices for $i$, and, for each $i$, no more than $Nb^{N-l_i}$ choices for the triple $B$, $C$, $F$. Further, given $B$, $C$, and a member $Y$ of $S(\omega_i, T_i, B, C)$ there are, according to Definition 9.1, at most $b^{\theta l_i}$ choices for $X$ which give rise to $Y$ as the output associated with $C$. Finally, there are at most $b^{\varepsilon l_i}/2mN$ members of $S(\omega_i, T_i, B, C)$. Therefore, there are at most

$$\sum_{i=1}^{m} Nb^{N-l_i}b^{\theta l_i}\frac{b^{\varepsilon l_i}}{2mN}=\frac{b^N}{2}$$

strings $D$ on $\Gamma^N$ which do not satisfy the lemma. But there are $b^N$ members of $\Gamma^N$, and hence at least one (and in fact half) satisfy the conditions of the lemma.

14. **Calculation of the overlap.** As in earlier sections, we assume $N=r^m$, the numbers $\theta$, $l_0$, $N$ and the string $A$ all satisfy Definition 9.1 for the function $f$, and $l_i=r^i$, $i=1, 2, \ldots, m$. We further assume $r=m\geqq 3$. The constants $B_1$, $B_2$, $B_3$, $\delta_1$, $\delta_2$, and $\delta_3$ introduced here depend only on the numbers $\theta$ and $l_0$, the constants $K_1$, $K_2$, and $K_3$ of (12.3), the constant $B(1)$ of (10.4), and the number $b$ of elements of $\Gamma$. We assume $K_3$ is an integer.

Now let

$$(14.1) \qquad T_i = [9l_i \log(2N)], \qquad i = 2, 3, \ldots, r,$$

where $[x]$ is the greatest integer not exceeding $x$. Let $\omega_i$ be the least positive integer such that

$$(14.2) \qquad K_1(NT_i)^{K_2}\binom{K_3T_i}{\omega_i} \geqq \frac{b^{\varepsilon l_i}}{2rN}, \qquad i = 2, 3, \ldots, r,$$

where $K_1$, $K_2$, $K_3$ are as in (12.3), and $\varepsilon=1-\theta$. To ensure that $\omega_i$ exists, note that Lemma 12.1 remains true if the values of $K_1$, $K_2$, and $K_3$ are increased. Hence we may assume $K_1, K_2\geqq 1$ and $K_3\geqq 2b$. Then, setting $\omega=bl_i$ and noting $T_i>l_i$, we have

$$K_1(NT_i)^{K_2}\binom{K_3T_i}{\omega} > \binom{2bl_i}{bl_i} > 2^{bl_i} > b^{l_i} > \frac{b^{\varepsilon l_i}}{2rN}.$$

Thus $\omega_i$ exists, and $\omega_i<K_3T_i$. (We shall use the last fact in Lemma 14.5.)

Now we claim there is a constant $B_1\geqq 2$ such that

$$(14.3) \qquad K_1(NT_i)^{K_2}\binom{K_3T_i}{\omega_i^*} < \frac{b^{\varepsilon l_i}}{2rN}, \qquad i = 2, 3, \ldots, r,$$

for all $r>B_1$, where $\omega_i^*=\omega_i-1$. If $\omega_i\geqq 2$, the inequality follows from our definition of $\omega_i$. If $\omega_i=1$, we argue as follows. Note that the variables $N$, $T_i$, and $l_i$ are all functions of $r$. If we set $\omega_i^*=0$ and multiply both sides of (14.3) by $2rN$, then the right-hand side is bounded below by $b^{\varepsilon r^2}$, and the left-hand side is bounded above by $r^{ar}$ for some constant $a$. Choose $B_1\geqq 2$ so $r^{ar}<b^{\varepsilon r^2}$ for all $r>B_1$. Then (14.3) holds for $r>B_1$ when $\omega_i=1$.

Now let $B_2=\max(B_1, \log B(1))$ where $B(1)$ is as in (10.4). Assume $r>B_2$, so $2N>B(1)$ (cf. Lemma 12.1) and (14.3) holds. Now by Lemma 12.1, for each $i$, $2\leqq i\leqq r$, and each admissible pair $(B, C)$ of strings with $|C|=l_i$, we have

$$M(\omega_i^*, T_i, B, C) \leqq K_1(NT_i)^{K_2}\binom{K_3T_i}{\omega_i^*}.$$

It follows by (14.3) that the inequality (13.2) holds with $\omega_i$ replaced by $\omega_i^*$, $i=2, 3,\ldots, r$. Thus by Lemma 13.1 there is a string $D$ in $\Gamma^N$ such that for every pair of adjacent intervals $I_{i,j-1}$, $I_{ij}$ $(i, j\geqq 2)$, the output string (12.2) is not in $S(\omega_i^*, T_i, B, C)$. Hence, by definition of $S$, one of the conditions (ii), (iii) of Lemma 12.1 must fail for $I=I_{i,j-1}$, $J=I_{ij}$, $\omega=\omega_i^*$, and $T=T_i$.

Now by (10.4) we have

$$T(AD) \leqq (2N \log (2N))/(\log \log (2N))^2,$$

and by (14.1) we have $T_i \geqq 8l_i \log (2N)$, $i=2, 3, \ldots, r$. Therefore, we can conclude that for each $i$, at least one-fourth of the pairs $(I_{i,j-1}, I_{ij})$, $1<j\leqq N/l_i$, satisfy condition (iii) and hence violate condition (ii). Thus, the set $\gamma_i$ of all indices $j$ such that $\omega_{ij}^D > \omega_i^*$ contains at least one-fourth the possible values of $j$. We have $\omega_{ij}^D \geqq \omega_i^* + 1 = \omega_i$, $j \in \gamma_i$, and $|\gamma_i| \geqq \frac{1}{4} r^{m-i}$, where $|\gamma_i|$ is the number of elements in $\gamma_i$. These two inequalities, together with 11.4, yield

$$T(AD) \geqq \frac{1}{H}\left(\frac{1}{4} - \frac{1}{r}\right) \sum_{i=2}^{r-1} r^{r-i}\omega_i.$$

By Lemma 14.5 below, we have for $r > \max (B_2, B_3)$

$$T(AD) \geqq \frac{1}{H}\left(\frac{1}{4} - \frac{1}{r}\right) \sum_{i=2}^{r-1} \frac{\delta_1 r^r}{\log r} \geqq \frac{\delta_2 r^{r+1}}{\log r},$$

where $\delta_2$ is a positive constant. But $N=r^r$, so

$$T(AD) \geqq \frac{\delta_2 N \log N}{(\log \log N)^2} > \frac{\delta_2}{4} \cdot \frac{2N \log (2N)}{(\log \log (2N))^2} \tag{14.4}$$

for arbitrarily large values of $N$, which establishes (10.5) with $\delta = \delta_2/4$.

It remains to prove

14.5. Lemma. *For each integer $r \geqq 3$ let $N=r^r$, and for $i=2, 3, \ldots, r$ let $l_i = r^i$, let $T_i$ be given by* (14.1), *and let $\omega_i = \omega_i(r)$ be a positive integer satisfying* (14.2), *where $K_1, K_2, K_3, b$, and $\varepsilon$ are positive constants. Assume $\omega_i < K_3 T_i$, and that $K_3$ is an integer. Then there are positive constants $\delta_1$ and $B_3$ such that $\omega_i > \delta_1 r^i/\log r$, $i=2, \ldots, r$, for all $r > B_3$.*

**Proof.** By one form of Stirling's approximation, $m! > (m/e)^m$ for all positive $m$, so the binomial coefficient satisfies

$$\binom{n}{m} \leqq \frac{n^m}{m!} < \left(\frac{ne}{m}\right)^m$$

for positive integers $m$, $n$, with $n > m$. Applying this to (14.2) and taking logarithms, we obtain

$$\log K_1 + K_2 \log (NT_i) + \omega_i(\log (eK_3 T_i) - \log \omega_i) \geqq \varepsilon l_i \log b - \log (2rN).$$

Substituting $9l_i \log (2N)$ for $T_i$ by (14.1), $r^i$ for $l_i$, $r^r$ for $N$, and solving for $\omega_i$ we obtain

$$\omega_i > \frac{\delta_3 r^i - C_1 r \log r}{C_2 \log r + i \log r - \log \omega_i} \tag{14.6}$$

for $r \geqq 2$ and some positive constants $\delta_3$, $C_1$, and $C_2$. By assumption, $\omega_i \geqq 1$, so

$\log \omega_i \geqq 0$. Hence (14.6) remains valid with the term $\log \omega_i$ deleted. Taking logarithms of the modified (14.6) we obtain

$$\log \omega_i > \log (\delta_3 r^i - C_1 r \log r) - \log (C_2 \log r + i \log r). \tag{14.7}$$

In general, $\log (x+y) \leqq \log x + \log y$ for $x, y \geqq 2$, and if $x > 2y \geqq 4$, then $x - y > x/y$, so $\log (x-y) > \log x - \log y$. Thus (14.7) yields

$$\log \omega_i > i \log r - C_3 \log r, \tag{14.8}$$

for some constant $C_3 > 0$, $2 < i < r$, and sufficiently large $r$. If we substitute the right side of (14.8) for $\log \omega_i$ in (14.6), we obtain

$$\omega_i > \frac{\delta_3 r^i - C_1 r \log r}{(C_2 + C_3) \log r} = \frac{r^i}{(C_2 + C_3) \log r} \left( \delta_3 - \frac{C_1 \log r}{r^{i-1}} \right),$$

and the lemma follows.

## References

1. A. J. Atrubin, *A one-dimensional real-time iterative multiplier*, IEEE Trans. Electronic Computers EC-14, No. 3 (1965), 394–399.

2. A. Cobham, *The intrinsic computational difficulty of functions*, Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science, North-Holland, Amsterdam, 1965, pp. 24–30.

3. S. N. Cole, *Real-time computation by iterative arrays of finite-state machines*, Doctoral Thesis, and Report BL-36, Computation Laboratory, Harvard Univ., Cambridge, Mass., 1964.

4. S. A. Cook, *On the minimum computation time of functions*, Doctoral Thesis, Harvard Univ., Cambridge, Mass., 1966.

5. P. C. Fischer, *Generation of primes by a one-dimensional real-time iterative array*, J. Assoc. Comput. Mach. **12** (1965), 388–394.

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

7. F. C. Hennie, *On-line Turing machine computations*, IEEE Trans. Electronic Computers EC-15, No. 1 (1966), 35–44.

8. ———, *One-tape, off-line Turing machine computations*, Information and Control **8** (1965), 553–578.

9. R. McNaughton, "The theory of automata, a survey" in *Advances in computers*, Vol. 2, F. L. Alt, editor, Academic Press, New York, 1961, pp. 379–421.

10. M. O. Rabin, *Real-time computation*, Israel J. Math. **1** (1963), 203–211.

11. A. Schönhage, *Multiplikation grosser Zahlen*, Computing (Arch. Elektron. Rechnen) **1** (1966), 182–196.

12. J. C. Shepherdson, and H. H. Sturgis, *Computability of recursive functions*, J. Assoc. Comput. Mach. **10** (1963), 217–255.

13. A. L. Toom, *The complexity of a scheme of functional elements realizing the multiplication of integers*, Dokl. Akad. Nauk SSSR **150** (1963), 496–498 = Soviet Math. Dokl. **4** (1963), 714–716.

14. S. Winograd, *On the time required to perform multiplication*, J. Assoc. Comput. Mach. **14** (1967), 793–802.

COMPUTER CENTER, UNIVERSITY OF CALIFORNIA,
BERKELEY, CALIFORNIA
UNIVERSITY OF OSLO,
OSLO, NORWAY