REAL TIME COMPUTATION*

BY
MICHAEL O. RABIN

ABSTRACT

We introduce a concept of real-time computation by a Turing machine. The relative strengths of one-tape versus two-tape machines is established by a new method of proofs of impossibility of actual computations.

In the formulation of computations by Turing Machines it is assumed that the problem (say a numerical value of an argument for which a function value is to be computed) is given on the machine-tape and the machine proceeds to perform its computation. No a-priori bound is imposed on the number of steps (the "time") needed for completion of the computation. The functions computable in this way are precisely all recursive functions.

It is of great interest from the point of view of a general theory of computation to gain insight into computation procedures where there is some limitation on the time allowed for computation.

One natural limitation is to require that if the problem (the input data) consists of n symbols then the computation will be performed in n basic steps, one step per input symbol. We may assume that the input sequence is entering the machine one symbol at a time and that the machine performs one of its atomic moves per input symbol. We again let our machines be Turing Machines which may, however, have more than one tape. Computations which are performed in this way will be called *real-time computations* (by a Turing Machine). Note that our systems may serve as mathematical models for computers (with auxiliary tapes) which are used for what is called "real-time" control.

If, in particular, the result of a computation on every input sequence is always 0 or 1 then we can view the machine is defining a set, namely, the set of those input sequences which yield 1. Informally we can also say that the machine recognizes for every input sequence whether it is in the set defined by the machine or not.

There are several results, notably by Yamada [2], about real-time computation by a Turing Machine. These are mainly along the lines that certain computations are possible in real time. The concept of real-time computation is generalized

Received January 24, 1964

* This research was supported in part by National Science Foundation grant GP-228 to Harvard University. This paper was written while the author was visiting at the Computation Laboratory of Harvard University during the summer of 1963.

in an interesting way by Hartmanis and Stearns in [1]. In contrast with the case of finite automata there is no neat intrinsic characterization of the class of sets which are real-time definable. In fact, rather than attempt a complete characterization we should probably contend ourselves with insight about feasibility and non-feasiblity of certain problems in real time.

Our main result is that there exists a recognition problem which can be done in real time using two tapes but cannot be done in real time using a single tape. This result has obvious implications concerning the relative strengths of computers with one or more tapes when used as real-time control devices.

In Section 6 we discuss the difficulty inherent in proofs of impossibility of certain computations. By way of illustration we give an example of a problem which somewhat unexpectedly can be done in real time on a single tape (Theorem 3).

The result about relative strength of one-tape versus two-tape real-time computation is but one example in this difficult area of assessment of "degree of difficulty of a computation." We hope that some of the ideas in our proof, especially the concept of a *bottleneck* in a computation, may generalize to apply to other problems in the same area.

1. **Real-time Turing Machines.** The model for real-time computation that we employ is the one used by J. Hartmanis and R. Stearns [1] and by Yamada [2].

A multi-tape Turing Machine over the *input alphabet* Σ is a finite automaton M having a finite set S of *states* and a *working alphabet* $W = \{\alpha_1, \cdots, \alpha_n\}$. One of the states, call it s_0, is distinguished as the *initial state* of M. A subset $F \subseteq S$ is singled out as the set of *designated final states*. The machine has k two-way infinite linear *work-tapes* $t_1, \cdots, t_k$ which are divided into squares. Furthermore, there is a reading printing head which at any given time scans one square on each of the work-tapes. M is capable of receiving inputs $\sigma \in \Sigma$. The working alphabet is always assumed to contain a blank symbol and at least one other symbol so that $2 \leqq n$.

The operation of the Machine is specified by a function

$$M(\sigma, s, \alpha_{i_1}, \cdots, \alpha_{i_k}) = (s', X_1, \cdots, X_k, \alpha_{j_1}, \cdots, \alpha_{j_k})$$

where $\sigma \in \Sigma$, $s, s' \in S$, $\alpha_i, \alpha_j \in W, X_i \in \{0, 1, -1\}$. We shall refer to this function as the *machine-table* of the k-tape Turing Machine M. The interpretation is that if the input is σ and M is in state s and is reading α_{t_r} on the tape t_r, $1 \leqq r \leqq k$, then M will go into state s', print α_{j_r} on the square scanned on t_r, and move each tape t_r one square left, or one square right, or not at all, according as to whether X_r equals 1, -1, or 0. This action of M is called an *atomic move*.

REMARK. When M prints a symbol $\alpha \in W$ on a square it first of all erases the contents of that square. Printing the blank symbol of W simply means erasing the contents of the square.

The set of all finite sequences on the alphabet Σ will be denoted by Σ^*.

DEFINITION 1. A sequence $x = \sigma_1 \cdots \sigma_p \in \Sigma^*$ is said to be *accepted* by M if, when started in s_0 and with blank work tapes, M will go under the input sequence x through its atomic moves and end in a state in F (i.e., the state of M at the pth time unit is designated).

The set of all sequences accepted by M is called the se⌐ *defined* by M and is denoted by $T(M)$.

A set $T \subseteq \Sigma^*$ is called *real-time definable* (*recognizable*) if there exists a multi-tape Turing Machine M such that $T = T(M)$.

In particular T is called *k-tape real-time definable* if for some M with k work tapes, $T = T(M)$.

It is quite clear that the adjective "real-time" is appropriate for this mode of operation. If x is a sequence of length p and requires p time units to feed into M then by time p we know, by looking at the state of M, whether x is accepted. Thus, there is no time delay between receipt of data and its processing.

We restrict our attention to recognition problems. A simple analysis, however, will show that real-time computation problems can be easily reduced to recognition problems. Thus, our restriction involves no loss of generality.

2. **The set T_2.** Let $\Sigma = \{a, b, 0, 1, \alpha, \beta\}$. Words on $\{a, b\}$ will be called *ab words* and the set of *ab* words will be denoted by A. Words on $\{0, 1\}$ will be called 01 *words* and the set of 01 words will be denoted by Z.

If $x = \sigma_1\sigma_2 \cdots \sigma_{n-1}\sigma_n$ then, by definition, $x^* = \sigma_n\sigma_{n-1} \cdots \sigma_2\sigma_1$.

DEFINITION 2.

$$T_2 = \{uv\alpha u^* \,|\, u \in A, v \in Z\} \cup \{uv\beta v^* \,|\, u \in A, v \in Z\}.$$

LEMMA 1. *The set T_2 is real-time definable by a two-tape machine.*

Proof. We shall describe the mode of operation of a two-tape machine M for which $T_2 = T(M)$. The reader can verify that this mode of operation can indeed be realized by a suitable machine-table.

As the *ab* word u is coming in, M will print it on its first tape. When the 01 word v comes in, M will print it on its second tape. According as to whether the input following uv is α or β, M will start tracing back its first or second tape. M will end in a designated state if and only if the sequence w of inputs following α (or β) coincides with the sequence being traced backwards on the first (second) tape.

THEOREM 1. *The set T_2 is not real-time definable by a one-tape machine. Consequently two-tape real-time computation can do more than one-tape real-time computation.*

3. **Preliminary lemmas..** To prove that T_2 is not real-time definable by a one-tape machine assume by way of contradiction that the one-tape machine M does define T_2 in real time. Let the number of states of M be m and the number of letters in its working alphabet be n. Throughout the following Sections 3–5 M will always designate this fixed one-tape machine for which $T_2 = T(M)$.

DEFINITION 3. If M has input w then the *work space* $t(w)$ of M on w is the sequence of tape squares covered by the motion of M while having the input sequence w.

If x is a sequence of squares on the tape or a sequence of symbols then $l(x)$ will denote the *length*, i.e. the number of elements, of x.

Let x be an input sequence, by the *coding of* x we shall refer to the sequence of symbols in the squares of the work space $t(x)$, the state of M, and its position on the tape, at the end of the input x.

LEMMA 2. *There exists a numerical constant* $c > 0$ *such that for every* $u \in A$ *and every integer* $i > 0$ *there exists a* $v \in Z$ *such that* $l(v) = i$ *and* $ci \leqq l(t(uv))$.

Proof. There are 2^i sequences $v \in Z$ such that $l(v) = i$. Since the input uv may be followed by β, if $v_1 \neq v$, then uv_1 and uv must be coded differently. Otherwise, $uv\beta v^*$ and $uv_1\beta v^*$ will both be accepted by M.

Let $l(t(uv)) \leqq k$ for all $v \in Z$, $l(v) = i$. Then there are at most $n^k \cdot k \cdot m$ different codings of the inputs uv. Hence $2^i \leqq n^k \cdot k \cdot m$. If i is large this forces k to be large so that we may assume that $km \leqq n^k$ (we assume $2 \leqq n$). Thus $2^i \leqq n^{2k}$ and hence

$$\frac{1}{2}\,\frac{ln\,2}{ln\,n}\,i \leqq k.$$

Thus we may take $c_1 = \frac{1}{2}(ln\,2)/(ln\,n)$. This c_1 will do for all i larger than some i_0; for a suitable smaller c the lemma will hold for all i.

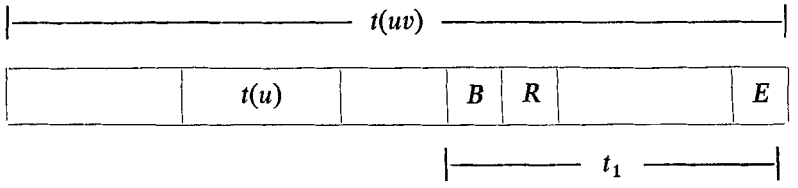
LEMMA 3. *There exists an integer* $d > 0$ *(depending only on* M*) such that for every* $u \in A$ *and every integer* $i \geqq l(u)$ *there exist a sequence* $v \in Z$, $l(v) = i$, *such that* a) $ci \leqq l(t(uv))$, b) *no more than* $\frac{1}{5}$*th of the squares of* $t(uv)$ *are covered by* M *more than* d *times.*

Proof. Let us choose a sequence $v \in Z$, $l(v) = i$, for which a) holds. Let d_1 be a number such that more than $\frac{1}{5}$th of the squares of $t(uv)$ are covered by M more than d_1 times. Then the total number of moves of M exceeds $d_1 \frac{1}{5} l(t(uv)) \geqq \frac{1}{5} d_1 ci$. But since M operates in real time the number of moves of M by the input uv is exactly $l(u) + l(v) \leqq 2i$. Thus, $\frac{1}{5} d_1 ci \leqq 2i$ and $d_1 \leqq 10/c$. The number $d = [(10/c) + 1]$ satisfies b).

4. **Bottleneck squares.** The proof of our main theorem rests on the idea that in working on certain input sequences the machine M develops bottleneck squares

on its work-tape through which information cannot flow in sufficient quantity. The idea of a bottleneck is made precise in the following:

DEFINITION 3. Let $u \in A, v \in Z$. A square B on $t(uv)$ is called a *bottleneck square* of $t(uv)$ if 1) under input uv the machine passes through B no more than d times (where d is as in previous Lemma 3), 2) B lies outside the work space $t(u)$, 3) the length of the section of $t(uv)$ determined by B which does not contain $t(u)$ exceeds $l(u) + 1$.



$l(u) + 1 < l(t_1)$

Figure 1

LEMMA 4. *For every $u \in A$ there exists a $v \in Z$ such that the tape $t(uv)$ has a bottleneck square.*

Proof. Let i be an integer such that $5l(u) + 5 < ci$ and also $l(u) \leqq i$. By Lemma 3 there exists a sequence $v \in Z$ such that $ci \leqq l(t(uv))$ and fewer than $\frac{1}{5}$th of the squares of $t(uv)$ are covered more than d times. Now $l(t(u)) \leqq l(u) + 1 < (ci/5) \leqq l(t(uv))/5$. Dividing $t(uv)$ into 5 equal parts (there is a trivial modification of the argument if $l(t(uv))$ is not divisible by 5) we see that either on the left or on the right end of $t(uv)$ there is an interval of length $\frac{2}{5}\, l(t(uv))$ which does not contain any squares of $t(u)$. In this interval consider the $\frac{1}{5}$th of $t(uv)$ which does not run to the end. Since fewer than $\frac{1}{5}$th squares of $t(uv)$ are covered more than d times by M, there is a square B in this $\frac{1}{5}$th of $t(uv)$ which is covered at most d times.

There are at least $l(t(uv))/5 \geqq (ci/5) > l(u) + 1$ squares between B and the end of $t(uv)$. Thus B is a bottleneck square.

5. **Proof of main theorem.** Let $u \in A$ and $v \in Z$ be such that $t(uv)$ has a bottleneck square B. To fix ideas let us assume that B is to the right of $t(u)$. As the input uv is coming in, there is a first time that M enters the right-most square E of $t(uv)$ (see Figure 1). Let $w \in Z$ be the initial section of v such that uw is the sequence leading to the first visit of M at E. Thus $t(uv)$ and $t(uw)$ have the same right-hand end square E and B is also a bottleneck square of $t(uw)$.

Denote the square immediately to the right of B by R. By a *passage* of M *through* B we mean either a move of M from B to R or a move from R to B. The *state of M during a passage* is the state M has when it reaches R in the first case, and the state M has when it reaches B in the second case.

Note that passages of M through B do not include atomic moves of M in which it starts on B and stays on B.

Under the input uw the machine M will first cover the tape $t(u)$ and then, under the w portion of the input, move to the square E. Let $p_1, p_2, \cdots, p_r$, be the consecutive passages through B ($r = 1$ is not excluded). The passage p_1 is a move from B to R, p_2 is a move from R to B, etc. Let the state of M during the passage p_i be $s_i, 1 \leqq i \leqq r$. The *scheme* of the bottleneck square B is the $r+1$ tuple $(e, s_1, \cdots, s_r)$ where e is 1 if B is to the right of $t(u)$ and e is -1 if B is to the left of $t(u)$, and $s_1, \cdots, s_r$ are as above. The notions of passage and state during a passage are modified in an obvious way when B is to the left of $t(u)$.

Now the number r of passages through B is at most d. Thus, there are at most N

$$N = 2 \cdot m + 2 \cdot m^2 + \cdots + 2 \cdot m^d$$

different schemes of bottleneck squares, where m is the number of states of M.

Let g be a number such that $N < 2^g$. For each $u \in A$, $l(u) = g$, let $v \in Z$ be a 01 sequence such that $t(uv)$ has a bottleneck square B_u and let w denote the section of v leading to the first visit of M to the end E_u of $t(uv)$. There must be two different sequences $u_1, u_2 \in A, l(u_1) = l(u_2) = g$, such that the bottleneck squares B_{u_1} and B_{u_2} have the same scheme, say $(1, s_1, \cdots, s_r)$. Note that $e = 1$ which means that B_{u_i} is to the right of $t(u_i), i = 1, 2$.

Let

$$u_1 w_1 = u_1 \varepsilon_1 \cdots \varepsilon_{n_1} \cdots \varepsilon_{n_2} \cdots \varepsilon_{n_r} \cdots \varepsilon_{n_{r+1}}$$

$$u_2 w_2 = u_2 \delta_1 \cdots \delta_{m_1} \cdots \delta_{m_2} \cdots \delta_{m_r} \cdots \delta_{m_{r+1}}$$

where $\varepsilon, \delta \in \{0, 1\}$, ε_{n_1} is the input when M visits B_{u_1} during the first passage, ε_{r_2} is the input when M visits B_{u_1} during the second passage, and so on up to ε_{n_2}; similarly for $\delta_{m_1}, \delta_{m_2}, \cdots$, in the second sequence $u_2 w_2$. After receiving the input $\varepsilon_{n_{r+1}}$ ($\delta_{m_{r+1}}$) M visits for the first time the right-hand end-square $E_{u_1}(E_{u_2})$.

We come now to the main point of our argument, In the sequence $u_1 w_1$ replace, for each *odd* $1 \leqq i \leqq r-2$, the segment $\varepsilon_{n_i+1} \cdots \varepsilon_{n_{i+1}-1}$ by the sequence $\delta_{m_i+1} \cdots \delta_{m_{i+1}-1}$. Furthermore, replace $\varepsilon_{n_r+1} \cdots \varepsilon_{n_{r+1}}$ by $\delta_{m_r+1} \cdots \delta_{m_{r+1}}$. Call the resulting sequence $u_1 w_1'$. Note that all the changes were made in the w_1 portion of $u_1 w_1$. Now, $u_1 w_1$ and $u_2 w_2$ have the same scheme of states in the passages of M through B_{u_1} and B_{u_2}, respectively, and our changes in $u_1 w_1$ were made only in the inputs *between* visits to B_{u_1}, while M was on the right of B_{u_1}, or after the last visit to B_{u_1}. One can see by finite induction over $1 \leqq i \leqq r+1$ that $u_1 w_1'$ again has the same scheme $(1, s_1, s_2, \cdots, s_r)$ and that at each input ε_{n_j}, j odd and $2 \leqq j \leqq r+1$, the portion of the tape right of B_{u_1} is identical with the portion of the tape $t(u_2 w_2)$ right of B_{u_2} at input δ_{m_j} and the states of M at the corresponding inputs are the same.

The work spaces $t(u_1w_1')$ and $t(u_2w_2)$ have squares B_{u_1} and B_{u_2}, respectively, with the following properties. The work space $t(u_i)$ is completely to the left of B_{u_i}, $i = 1,2$. The portions of $t(u_1w_1')$ and $t(u_2w_2)$ beyond B_{u_1} and B_{u_2} are strictly longer than $l(u_1) = l(u_2) = g$. By the previous paragraph, at the end of the inputs u_1w_1' and u_2w_2 M is at the end-squares E_1 and E_2 of the respective work spaces and the portions of tape from B_{u_1} to E_1 and from B_{u_2} to E_2 as well as the states of M at E_1 and E_2 are identical. Assume now that both u_1w_1' and u_2w_2 are followed by the input αu_1^*. We have, since $u_1' \neq u_2$,

$$u_1w_1'\alpha u_1^* \in T_2, \quad u_2w_2\alpha u_1^* \notin T_2.$$

But $l(\alpha u_1^*) = g + 1$ is less than the distance from E_i to B_{u_i}, $i = 1,2$. Since M operates in real time and makes one move per input, it will stay, throughout the input portion αu_1^*, to the right of B_{u_i}. Thus, M will start in both cases in the same state and will move through identicaly printed portions of tape. It will therefore be in the same state at the ends of $u_1w_1'\alpha u_1^*$ and $u_2w_2\alpha u_1$ and hence cannot accept one and reject the other; a contradiction.

An analysis of the previous proof shows that we can derive from it explicit information as to the point where any given one-tape machine will fail to decide correctly whether a sequence x is in T_2. We state the result without detailed proof since the proof is already contained in our previous work.

THEOREM 2. *Let M have m states and n letters in its working alphabet. Let $c = \frac{1}{2}(ln 2)/(ln n)$, $d = [(10/c) + 1]$, and $N = 2m + 2m^2 + \cdots + 2m^d$. Let g be the smallest integer such that $N < 2^g$ and let i be the smallest number such that $5g < ci$ and $im \leqq n$. Then there exists a sequence $x \in \Sigma^*$ such that $l(x) \leqq 2g + i + 1$ and M accepts x even though $x \notin T_2$ or M rejects x even though $x \in T_2$.*

6. **General remarks on proofs of impossibility.** We would like to compare briefly the result in Theorem 1 and its proof with other results concerning impossibility of computations by various mathematical machines.

The proofs of impossibility in the literature fall into two classes. In some situations where we want to show about two classes A and B of mathematical machines or computational procedures that there is a function computable by a procedure in B, but not computable by any of the procedures in A, we can use the diagonal method. The class B is rich enough to contain a single procedure which in some sense is universal with respect to A and this procedure is utilized to diagonalize over all procedures in A. The proof that there exists a general recursive function which is not primitive recursive is a case in point. The class of general recursive functions contains a function which enumerates all primitive recursive functions and this function is used in a diagonalization argument.

The second method, applicable mainly when considering mathematical machines, consists in showing that the machines in the class A cannot store enough information to perform a certain task. Consider, for example, the set $T = \{0^n10^n | n = 1,2,\cdots\}$ where 0^n stands for a sequence of n symbols 0. To show that T is not definable by any finite automation M, we observe that as soon as n exceeds the number m of states of M, the automaton cannot remember how many 0's were in the sequence 0^n and consequently cannot always decide whether in 0^n10^k the equality $k = n$ holds.

Theorem 1 does not lend itself to either of the above methods. Two-tape real-time Turing Machines are not strong enough to diagonalize over the set of all one-tape real-time Turing Machines. The information storing capacity of a single tape, however, is equal to that of two tapes. Thus, neither method applies. As a rule, the proofs of non-feasibility of certain computations where the class of algorithms falls in this in-between range, not strong enough to use diagonalization but not weak enough to allow a straightforward information capacity argument, are rather hard. We have to resort to a fine analysis of the computational procedures available and it is difficult to survey all the possibilities.

Thus, there is a need for new techniques for handling this kind of problem. It should be remarked that many of the most interesting questions about non-feasibility of computations fall in this in-between area.

7. **A set which is one-tape recognizable.** When we look at the set T_2 we do have a strong intuitive feeling that T_2 is not real-time recognizable by a one-tape machine. For if we consider the way in which the input uv, $u \in A$, $v \in Z$, was coded by the two-tape machine (Proof of Lemma 1), we see that on one tape M we will have to code u as it comes in, and then v as it comes in. By the time M finishes coding v it is far from $t(u)$, if now αu^* comes in, then M is not able to compare it with u. These observations, however, are far from a proof because they apply only to the straightforward way of coding uv, and in a proof of impossility we must take into account all conceivable codings.

The following is an example of a set T_1 which is very similar to the set T_2 and to which the above suggestion of proof of impossibility equally applies. It turns out that by use of a more complicated coding, T_1 is recognizable by a one-tape machine. Let $\sum = \{a,0,\alpha,\beta\}$ and let

$$T_1 = \{a^n0^m\alpha a^n \mid n,m = 1,2\cdots\} \cup \{a^n0^m\beta 0^m \mid n,m = 1,2,\cdots\}.$$

Note that T_1 has the same structure as T_2 except that we do not use b and 1.

THEOREM 3. *The set T_1 is recognizable in real time by a one-tape machine.*

Proof. We shall outline the operation of the machine M without giving all details of its structure.

As the sequence a^n is coming in M prints a string x of a's. However, the machine will print an a and move to the right only for every second input a. Thus, at the end of a^n the work tape contains a sequence x of $n/2$ a's and M is at the right end of x.

As soon as the first 0 input comes in, M prints a 0 at the right end of x. During the 0^m input M will move the whole sequence x leftwards from the square containing 0. This will be done by erasing an a on the right, moving all the way to the left end of x and printing an a, moving back all the way to the right end of x, etc. All this is done at the rate of one atomic move per input 0.

We now distinguish two cases. If an input α comes in and is followed by a^k, then M has to check whether the sequence x (which by now has been moved from the original place) has length $k/2$ (or perhaps $(k/2)-1$ in case M is in the stage of moving inside x from right to left), for this would check whether $n = k$ and hence whether $a^n 0^m \alpha a^k \in T_1$. Let the square where M is inside x be E. When α comes in, M will mark E, move one square to the right for each a input till it comes to the right end of x. It will then move clear to the left end of x and then back to E. We have $n = k$ if and only if by the time M had k inputs 0 it is back at E.

If an input β comes in and is followed by 0^k, then M has to check whether k equals the number m of previous 0 inputs. This is done by moving the x sequence to the right, back to the 0 symbol on the tape. The moving to the right is done by reversal of the procedure used before for moving to the left. x will return to the original position under k inputs 0 if and only if $k = m$; i.e., if and only if $a^n 0^m \beta 0^k \in T_1$.

In all other cases, where the whole input sequence does not have the form $a^n 0^m \alpha a^k$ or $a^n 0^m \beta 0^k$, it is easy to arrange for M to reject the sequence. Thus, M accepts precisely the input sequences in T_1.

BIBLIOGRAPHY

1. Hartmanis, J. and Stearns, R. E., On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* to appear.

2. Yamada, H., 1961, Real-time computation and recursive functions not real-time computable, *IRE Trans. On Computers*, EC-10, pp. 753–760.

THE HEBREW UNIVERSITY OF JERUSALEM