INFORMATION AND CONTROL 8, 553–578 (1965)

One-Tape, Off-Line Turing Machine Computations*

F. C. HENNIE

Department of Electrical Engineering and Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts

This paper has two purposes. The first is to investigate the characteristics of a restricted class of Turing machines, and to develop a simple tool for describing their computations. The second is to present specific problems for which tight lower bounds can be found for the computation times required by Turing machines of this restricted class.

I. INTRODUCTION

In this paper we shall consider Turing machines that can, at any given step in their computations, do each of the following things: (a) change the tape symbols currently scanned by their reading heads, (b) shift each of their tapes one square to the left or right, (c) change their internal state, and (d) halt. Each step is assumed to require exactly one time unit for its completion. In order to investigate the total time required to compute a particular function, it is convenient to distinguish between "on-line" and "off-line" Turing machine computations. These two types of computations differ only in the way in which input data are supplied to the machine and output data are generated by the machine.

In an *on-line computation* the input data are supplied to the machine, one symbol at a time, at a special input terminal. Corresponding to each input symbol, the machine is required to produce, at a special output terminal, an appropriate output symbol. In general the machine will not

* This work was supported in part by the Joint Services Electronic Program under Contract DA 36-039-AMC-03200(E); and in part by the National Science Foundation (Grant GP-2495, the National Institutes of Health (Grant MH-04737-04), and the National Aeronautics and Space Administration (Grant NsG-496). Some of this material was presented at the Sixth Annual Symposium on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, October 6–8, 1965.

be able to do this immediately, but must spend a number of time units computing the required output symbol. However, a new symbol cannot be supplied at the input terminal until the machine has produced the output symbol that corresponds to the previous input symbol. Thus each output symbol is a function solely of the preceding input symbols. Most of the work of Yamada (1962), Hartmanis and Stearns (1965), and Rabin (1963) has used the on-line Turing machine model.

In an *off-line computation* all of the input symbols are written on one of the machine's tapes prior to the start of the computation. The results of the computation are obtained only when and if the machine halts, and may be taken to be either the pattern of symbols appearing on one of the tapes or else the internal state of the machine at the end of the computation. In this paper we shall consider only off-line computations, and in particular off-line computations having only two possible outcomes. These outcomes, represented by the symbols 0 and 1, may be thought of as being associated with the final states of the machine.

All of the tapes used by an off-line machine will be assumed to be singly-infinite, having a left end but no right end. This entails no loss of computing capability or speed, and is convenient for the analysis that is to follow. At the beginning of a computation the input pattern must be written at the left end of one of the machine's tapes that is designated for this purpose. The input pattern is finite in length, and the remainder of the tape squares are left blank. The tape is positioned so that its reading head scans the leftmost square of the input pattern, and the machine is placed in a designated starting state.

The machine then goes through a series of basic operations, as determined by its internal structure. If the machine halts in a state to which the output 0 is assigned, it is said to have *rejected* its input pattern; if it halts in a state to which the output 1 is assigned, it is said to have *accepted* its input pattern. If a machine always halts within a finite time, regardless of the input pattern with which it is presented, it is said to *recognize* the set of input patterns for which it produces outputs of 1. Such a machine may be thought of as classifying input patterns into those that are accepted and those that are rejected.

For the most part we shall be concerned with off-line machines that are guaranteed to halt, regardless of the particular input pattern supplied. The number of basic operations that such a machine requires to accept or reject an input pattern will be called the *computation time* for that input pattern. If $T(n)$ is a function such that the computation time associated with every input pattern of n symbols is less than or equal to $T(n)$, then the computations of the machine will be said to be *bounded* by $T(n)$. Similarly, the set of input patterns that the machine recognizes will be said to be *recognizable within* $T(n)$ time units.

This paper investigates the behavior of *one-tape,* off-line Turing machines, with two objectives. The first is to develop a tool for describing the computations of such machines, and the second is to apply that tool to the problem of finding good lower bounds for the times required to recognize various sets of patterns. Section II presents the idea of a "crossing sequence" and develops the properties of this concept that make it an important analytic tool for one-tape, off-line computations. Section III deals with the determination of lower bounds on computation times. In particular, Section III, A describes a set of patterns whose recognition time must exceed $C_1n^2/\log Q$, but need not exceed $C_2n^2/\log Q$, where C_1 and C_2 are appropriate constants and Q is the number of internal states in the Turing machine that is to recognize the set. Section III, B considers more general computation times of the form Kn^p, where K and p are real constants. For computation times of this form, the following result can be established. Given any two real numbers p and q in the range $1 \leqq q < p \leqq 2$, there exists a set of patterns that can be recognized within a time proportional to n^p, but that cannot be recognized within a time proportional to n^q, assuming that the recognition is to be done by a one-tape, off-line Turing machine.

II. CROSSING SEQUENCES

A. Definitions

Recall that an off-line computation begins with the pertinent input pattern prerecorded on one of the machine's tapes. We now wish to restrict our attention to machines having only one tape. Thus this tape must be used not only to record the input pattern, but also to provide space for any "scratch work" required in the course of a computation. As noted earlier, it is convenient to assume that the tape extends infinitely far to the right, but not to the left. The input pattern occupies a finite segment at the extreme left end of the tape, while the remainder of the tape is blank. A computation is started with the left-most square of the input pattern under the reading head. If desired, this square can be marked with a special symbol to keep the machine from inadvertently shifting the tape out of the reading head. In the computations to be considered in this paper, such a special end marker will not be needed, and will not be provided. To include an end marker would require no important changes in the discussion to follow.

In this paper the words "tape" and "pattern" will be used to denote two different things. The word "tape," unless otherwise qualified, refers to an infinite string of squares upon which some symbols may be written. If it is necessary to refer to a portion of a tape, the words "tape segment" will be used. In particular, if t_a denotes a finite segment at the left end of a tape, and if t_b denotes the remaining infinite segment of the same tape, the symbolism "$t_a t_b$" may be used to denote the entire infinite tape. The word "pattern," on the other hand, refers to the string of symbols that is written on a tape at a given time. Such a string must always be finite in length. For the most part it will be necessary to refer only to the pattern that appears on a Turing machine tape at the beginning of a computation. Thus we shall speak of an "initial tape" and the "input pattern" that it contains.

When describing the computations performed by a one-tape, off-line machine, it is convenient to think of the tape as remaining fixed and the control unit, or reading head, as moving back and forth. The zig-zag line in Fig. 1 shows a typical path that the head might trace out on its tape during the course of a computation. We will usually think of this path as being traced out on the input pattern itself, and ignore the manner in which the pattern changes during the course of the computation. On the other hand, we will be very much interested in the internal state that the machine assumes at each step in the computation, and may wish to label the path with these states.

Now consider two adjacent squares on a tape, say those marked x and y in Fig. 1, and note the points in the computation at which the head crosses the boundary between the two squares. Since every computation starts with the head at the left end of its tape, each odd-numbered crossing between two given adjacent squares must be a crossing from left to right; similarly, each even-numbered crossing must be from right to left.

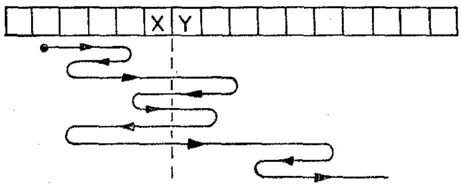

FIG. 1. The path of a computation

Let $S(i)$ denote the state of the machine at the time of the ith crossing between the two squares in question. Then the sequence $C = S(1)$, $S(2), \cdots, S(i), \cdots$ will be called the *crossing sequence* that the given Turing machine generates on the boundary between the two squares.[1] Informally, this crossing sequence describes the way in which the machine "carries information," by means of its internal states, from one side of the boundary to the other.

Specifying the crossing sequences associated with all of the boundaries that the machine reaches in the course of its computation is equivalent to specifying every basic step in that computation. In particular the total computation time is equal to the total number of crossings that the machine makes. If the number of crossings in a given sequence is referred to as the *length* of that sequence, then the total computation time is equal to the sum of the lengths of all the crossing sequences. Evidently a machine will halt for a given initial tape iff it generates on that tape a finite number of non-empty crossing sequences, each of which is of finite length.

B. Basic Properties

Although we are primarily interested in machines that must always eventually halt, regardless of the initial tape with which they are presented, let us temporarily relax this restriction. Then an off-line machine can react to an initial tape in three ways: it can halt and accept the tape, it can halt and reject the tape, or it can continue computing forever. Two initial tapes will be said to be *treated identically* by a given machine iff they are both accepted, or both rejected, or if they both cause the machine to compute forever.

THEOREM 1. *Let t_at_b be an initial tape consisting of a finite segment t_a, followed on the right by an infinite segment t_b. Note that t_a need* not *coincide with that portion of the tape that is initially nonblank. Similarly, let t_ct_d be an initial tape consisting of a finite segment t_c followed by an infinite segment t_d. Assume that t_at_b and t_ct_d are treated identically by a given Turing machine M. Let C_1 and C_2 be the crossing sequences that this machine generates on the boundary between t_a and t_b, and on the boundary between t_c and t_d, respectively. If C_1 and C_2 are identical, then*

(A) *The initial tapes t_at_b and t_at_d must be treated identically by the given machine M.*

[1] A crossing sequence corresponds to the notion of a "scheme" used by Rabin (1963).

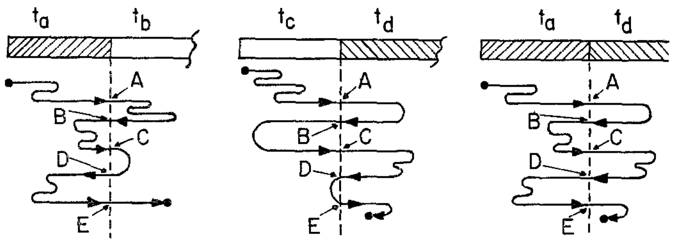

FIG. 2. Illustration of Theorem 1

(B) *In the computations performed on t_at_b and t_at_d, the crossing sequences generated at corresponding boundaries in the t_a portions of the tapes must be identical.*

(C) *In the computations performed on t_at_d and t_ct_d, the crossing sequences generated at corresponding boundaries in the t_d portions of the tapes must be identical.*

In other words, the t_c portion of the tape t_ct_d can be replaced by t_a without changing the final disposition of the tape or the crossing sequences that appear in the segments t_a and t_d. Figure 2(a) and (b) shows typical paths that a hypothetical Turing machine might trace out on tapes t_at_b and t_ct_d. If the crossing sequence generated on the boundary between t_a and t_b is the same as that generated on the boundary between t_c and t_d, as shown, then the computation performed on the tape t_at_d must follow the path shown in Fig. 2(c). This picture makes the theorem almost "intuitively obvious," and a rigorous proof will not be given.

A useful corollary follows almost directly from Theorem 1:

COROLLARY 1. *If for an initial tape $t_1t_2t_3$ the crossing sequence that a given machine M generates on the boundary between t_1 and t_2 is identical to the crossing sequence that it generates on the boundary between t_2 and t_3, then*

(A) *Machine M must treat initial tapes $t_1t_2t_3$ and t_1t_3 identically.*

(B) *Machine M must treat initial tapes $t_1t_2t_3$ and $t_1t_2t_2t_3$ identically.*

In other words, if the same crossing sequence appears on both sides of some tape segment, that segment can be removed, and the neighboring segments joined together, without affecting the final disposition of the tape. Alternatively, an extra copy (or copies) of the given segment can be sandwiched in without affecting the final disposition of the tape.

Proof: Part (A). Let t_a, t_b, t_c, and t_d represent the tape segments t_1, t_2t_3, t_1t_2, and t_3, respectively. Then t_at_b and t_ct_d are identical tapes (namely

$t_1t_2t_3$) and will certainly be treated identically by M. Thus according to the Theorem, t_at_b and t_at_d must be treated identically. But $t_at_b = t_1t_2t_3$ and $t_at_d = t_1t_3$.

Part (B). Let $t_a = t_1t_2$, $t_b = t_3$, $t_c = t_1$ and $t_d = t_2t_3$, and again apply the Theorem. Q. E. D.

Although Theorem 1 and Corollary 1 are sufficient to establish the time bounds discussed in the next section, it is worthwhile to investigate in greater detail the ideas involved in the theorem. By so doing, we can develop a better understanding of the relationship between the crossing sequence concept and the more familiar internal state concept as it is applied to finite-state machines.

Theorem 1 describes a situation in which it is possible to replace a segment t_c at the left end of an initial tape t_ct_d by another segment t_a without affecting the outcome of the computation. Specifically, this can be done if there exists a second tape t_at_b that is treated identically to t_ct_d and for which the machine generates the same crossing sequence at the right end of the t_a segment as it does at the right end of the t_c segment. Segments such as t_a and t_c that appear at the left end of a tape will be referred to as "left-end" segments. Consider now the relationship that must hold between two left-end tape segments t_1 and t_2 if t_2 can *always* be substituted for t_1, regardless of the complete pattern in which the latter appears. Evidently such a substitution will be possible only if every crossing sequence that can appear at the end of t_1 can appear at the end of t_2.

What is needed at this point is a precise means of determining whether a given finite crossing sequence can appear at the right of a given left-end tape segment. This cannot be done by considering all the tapes that contain the given left-end segment, since in general it would be necessary to investigate an infinite number of computations in order to be sure that a given crossing sequence could *not* appear at the end of a given segment.

Instead, for any given Turing machine, *finite* left-end tape segment t, and finite crossing sequence, $C = S(1), S(2), \cdots, S(\lambda)$, we may perform the following experiment.

1. Begin the experiment by placing the machine in its designated initial state and causing it to scan the leftmost square of t.

2. If, when the machine leaves the right-hand end of t for the ith time ($i < \lambda$) it is in state $S(i)$, put the machine in state $S(i + 1)$ and send it back onto the rightmost square of t.

3. If the machine halts within t, or gets stuck in some periodic be-

havior within t, or leaves t in such a way that rule 2 does not apply, stop the experiment.

Evidently such an experiment must terminate within a finite number of steps. If at the end of the experiment the crossing sequence developed at the right-hand end of t is exactly C, the segment t will be said to *support* the crossing sequence C at its right-hand end. If the length of C is odd, then the experiment ends with the machine outside the segment t. In this case C will be called a *transient* crossing sequence for t. If the length of C is even, then the experiment ends with the machine inside the segment t. If the machine halted and produced an output of 1, the sequence C will be called an *accepting* sequence for t; otherwise it will be called a *nonaccepting* sequence for t.

Now suppose that every finite crossing sequence that is supported by a given left-end tape segment t_1 is also supported by some other left-end segment t_2. Furthermore, suppose that every nontransient sequence that is accepting for t_1 is also accepting for t_2. Then consider any tape in which t_1 appears as a left-end segment and on which the machine performs a finite computation. By virtue of reasoning similar to that of Theorem 1, the segment t_2 can be substituted for t_1 without changing the disposition of the original tape or the crossing sequences that appear on the portion of the tape to the right of t_1.

Finally, suppose that (a) every crossing sequence that is supported by t_1 is also supported by t_2, and vice versa, and (b) every nontransient sequence that is accepting for t_1 is also accepting for t_2, and vice versa. Then the left-end segments t_1 and t_2 can be freely interchanged in any finite computation without affecting the outcome of the computation. Classifying left-end tape segments according to the crossing sequences that they support and according to which of these sequences are accepting may thus be thought of as a generalization of the classification of input strings according to the states to which they take a given finite-state machine.

C. Simple Applications

It is now instructive to consider the special case in which the crossing sequences generated by a given machine never exceed a certain length, say K. In other words, the machine never spends more than K time units in any one square of its tape. Loosely speaking, if a machine never visits any tape square more than K times, there is only a finite number of different sequences of things that it can "do" in any one square. Furthermore, there is only a finite number of distinct subsets of these sequences. If the subset of sequences that the machine might generate in a given square is known, then the subset of sequences that it might generate in the next square to the right can be determined. Therefore the machine might just as well be designed to visit each square just once, working from left to right, and to keep track of the corresponding subsets of sequences as it goes. This line of reasoning suggests the following theorem:

THEOREM 2. *If every crossing sequence in every computation performed by a given one-tape, off-line Turing machine contains at most K members, then there exists a finite-state machine that recognizes precisely the same set of input patterns as the given Turing machine.*

Proof: The proof is accomplished by showing that the set of patterns that the given machine recognizes can be represented as the union of a number of classes of a finite, right-invariant equivalence relation.

For any given Turing machine, and for any given finite value of K, there is at most a finite number of distinct crossing sequences whose length does not exceed K. Furthermore, the number of subsets of these crossing sequences is finite. Every finite left-end tape segment can then be classified according to the crossing sequences of length K or less that it supports at its right-hand end, and according to which of these sequences are transient, which are accepting, and which are nonaccepting.

Now suppose that two finite left-end segments, t_1 and t_2, belong to the same class, as just described. Then consider any left-end tape segment of the form t_1t_x, where t_x is finite, and determine the crossing sequences of length K or less that it supports. Evidently any crossing sequence that is supported by t_1t_x will also be supported by t_2t_x and vice versa, since in the experiment described in Section II, B t_1 can be replaced by t_2 without changing the crossing sequences in the t_x portion. Furthermore, any nontransient sequence that is accepting for t_1t_x will also be accepting for t_2t_x, and any nontransient sequence that is nonaccepting for t_1t_x will also be nonaccepting for t_2t_x. Thus t_1t_x and t_2t_x fall in the same class of left-end tape segments. Since this is true for any t_x, the classification described above is a finite, right-invariant equivalence classification.

We must now show that the set of acceptable input patterns is composed of the union of a number of the equivalence classes. Suppose that pattern p is accepted by the given Turing machine. That is, when p appears at the left end of a tape, followed by an indefinitely long segment of blank tape, that tape is accepted by the machine. If q is another pattern that falls in the same equivalence class as p, then q must also be accepted by the machine, for we have seen that members of a given equivalence class can be interchanged without affecting the disposition of a tape in which they appear as left-end segments. Therefore if one member of a certain equivalence class is acceptable, every member of that class must be acceptable. Similarly, if one member of a certain class is not acceptable, then no member of that class is acceptable. It follows that the set of acceptable tape patterns must be composed of the union of certain of the equivalence classes. This in turn implies that the set of acceptable patterns can be recognized by a finite-state machine. Q. E. D.

Theorem 2 states that if a certain Turing machine never spends more than a fixed number of time units on any one square of its tape it can be replaced by a finite-state machine. Next suppose that the *maximum* time that a machine spends in each square is not limited, but that the *average* time per square of the input pattern is limited. In other words, suppose that the machine is guaranteed to complete its computations within Kn time units, where n is the length of the input pattern. In such a case, the machine may generate very long crossing sequences at some boundaries, as long as it generates enough short sequences elsewhere. Nevertheless, it is possible to show that any Turing machine of this type can also be replaced by a finite-state machine.

Before proving this fact, note that we may safely restrict our attention to Turing machines that never leave the input portions of their tapes. For suppose that M is a machine that completes all of its computations within Kn time units. Such a machine certainly does not visit more than $(K - 1)n$ squares of blank tape. Now design a new machine M', whose tapes are divided into K levels, or "tracks," as shown in Fig. 3. The symbols that may appear in the squares of the new tapes represent ordered combinations of K of the symbols from the original tapes, one symbol for each track.

The top track of each new tape is used to record the input pattern, while the remaining $K - 1$ tracks account for the $(K - 1)n$ squares of blank tape that machine M might use. The machine M' starts out by working only with the symbols that appear in the top track of its tape, behaving exactly as M would at the beginning of its computation. However, if M should move off the input portion of its tape onto the blank portion, M' turns around and works backwards on the second track of its tape. If M should move beyond the nth square of blank tape, M' simply turns around again and works toward the right on the third track

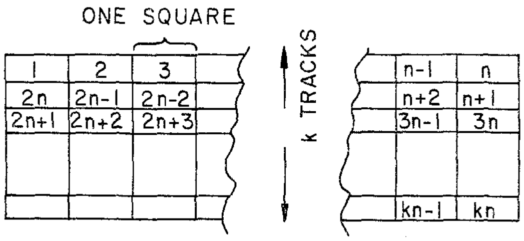

FIG. 3. Multitrack tape

of its tape. In effect, the blank portion of the tape from machine M has been folded, in zig-zag fashion, onto the input portion of the tape. Although the details will not be given, it should be clear that it is always possible to design a machine M' that recognizes the same set of input patterns as the given machine M.

THEOREM 3. *If a one-tape, off-line Turing machine performs all of its computations within the time bound $T(n) = Kn$, where K is a constant, then there exists a finite-state machine that recognizes precisely the same set of input patterns as does the given Turing machine.*

Proof: The proof consists in showing that if the total computation time is limited by Kn, then the time spent in any one square must be limited by $2KQ^K + K$, where Q is the number of internal states. The desired conclusion then follows immediately from Theorem 2.

Assume that the given Turing machine has been constructed in such a way that it never leaves the input portions of its tapes, and that it has Q internal states. Then suppose that in the process of performing its computation on some input pattern the machine does generate a crossing sequence whose length is greater than $2KQ^K + K$. In particular, let n_0 be the shortest input pattern length for which such a "long" crossing sequence is generated.

Now choose some input pattern of length n_0 for which a long crossing sequence is generated and examine the computation that the given machine performs on this pattern. Let s denote the number of crossing sequences in this computation that are shorter than K. Recalling that the total computation time T is equal to the sum of the lengths of all the crossing sequences in the computation, we have:

$$Kn_0 \geqq T(n_0) > 2KQ^K + K + (n_0 - 1 - s)K$$

Solving for s gives

$$s > 2Q^K$$

But the number of distinct crossing sequences whose length is less than K is

$$\sum_{i=1}^{K-1} Q^i \leqq Q^K - 1$$

since the given machine has only Q states. Therefore among the more than $2Q^K$ boundaries that have "short" crossing sequences, there must be at least *three* that have identical crossing sequences. Call these boundaries b_1, b_2, and b_3, and call the boundary on which the "long" crossing sequence appears b_0.

At least two of the boundaries b_1, b_2, and b_3 must lie on the same side of b_0. Suppose that b_1 and b_2 are so located, as shown in Fig. 4. Now form a new input pattern by removing the portion of the original pattern between b_1 and b_2 and joining the end pieces together. According to Corollary 1, the crossing sequences that the given machine generates in performing its computation on the new tape must be identical to the crossing sequences that it generated at corresponding boundaries on the old tape. In particular, since b_0 lay outside the portion of the old tape between b_1 and b_2, the computation on the new tape will contain a crossing sequence whose length exceeds $2KQ^K + K$.

The length of the new pattern is certainly less than n_0. But n_0 was determined to be the shortest length of an input pattern for which such a "long" crossing sequence is generated. Therefore the assumption that some finite input pattern yields a crossing sequence longer than $2KQ^K + K$ leads to a contradiction and must be false. Consequently no crossing sequence generated by the given machine is longer than $2KQ^K + K$,

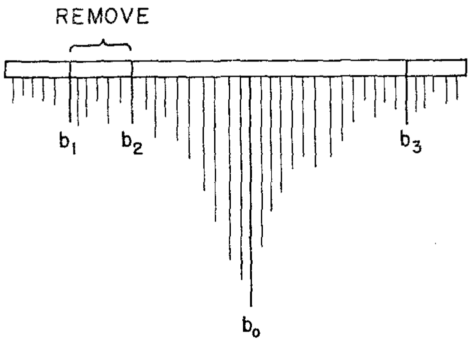

FIG. 4. Illustration of Theorem 3

regardless of the particular finite input pattern supplied to the machine. Then according to Theorem 2 there must exist a finite-state machine that recognizes the same set of input patterns as the given Turing machine. Q. E. D.

Restating Theorem 3 slightly gives the following interpretation. Suppose that a certain set of input patterns can be recognized by a one-tape, off-line Turing machine that completes each of its computations in a time that is directly proportional to the length of the particular input pattern supplied. Then the same set of patterns can be recognized by a one-tape, off-line Turing machine that completes each of its computations in a time that is exactly equal to the length of the pattern supplied. This latter Turing machine simply moves from left to right across its tape, behaving like a finite-state machine. Thus no increase in the computational capabilities of one-tape, off-line Turing machines is achieved by increasing the allowed computation time from n to Kn.

It is interesting to compare this result with related results for other computing situations. Using the techniques of Hartmanis and Stearns (1965) it is possible to show that if a certain set of patterns can be recognized by a one-tape, off-line Turing machine that completes its computations within time $KT(n)$, then it can also be recognized by a machine that completes its computations within time $T(n)$, *provided* $T(n) > n^2$. In other words, if the computation time is large enough to begin with (greater than n^2), increasing the computation time by a multiplicative factor does not increase the computational capabilities of a one-tape, off-line machine. It seems reasonable to conjecture that this is true for all functions $T(n)$. However, the notion of a crossing sequence does not seem to be useful in establishing this result for $T(n)$ greater than n, and the methods of Hartmanis and Stearns do not seem to be useful for $T(n)$ less than or equal to n^2. At present, then, it is not known whether increasing the computation time from $T(n)$ to $KT(n)$ increases the capabilities of a one-tape, off-line Turing machine when $T(n)$ lies in the range $n^2 \geqq T(n) > n$.

Matters are slightly different for *two*-tape, off-line machines. Again, going from $T(n)$ to $KT(n)$ does not increase computational capabilities so long as $T(n) > n^2$. However, going from n to Kn apparently *does* increase the computational capabilities of two-tape, off-line machines. Exactly where in the range from n to n^2 the use of a multiplicative constant ceases to make a difference is not yet known, but it is clear that the addition of a second tape does change the computational characteristics of an off-line machine. Further understanding of the differences between one- and two-tape machines will be developed in the next section.

III. TIME BOUNDS FOR ONE-TAPE, OFF-LINE COMPUTATIONS

A. The Recognition of Set S

We now consider the problem of determining upper and lower bounds on the time required to recognize certain sets of input patterns on one-tape, off-line machines. We shall first exhibit a set of patterns for which the minimum computation time, $T(n)$, can be shown to lie in the range $C_1n^2/\log Q \leqq T(n) \leqq C_2n^2/\log Q$, where n is the length of the input pattern and Q is the number of internal states of the machine that performs the computation. Thus for this particular set of patterns the optimum computation time grows as n^2, and can be determined within a multiplicative constant. Appropriate modifications of this set lead to sets whose optimum computation times grow as other powers of n in the range between 1 and 2, and whose optimum computation times can also be determined within multiplicative constants.

The patterns to be discussed in this section are based on the alphabet $\{0, 1, 2\}$. It is convenient to think of such patterns as being composed of *blocks* of 0's and 1's, separated by *blocks* of 2's. Thus the pattern 010112022201022 contains a total of six blocks. Now let the set S be defined as the set of patterns on the alphabet $\{0, 1, 2\}$ that satisfy each of the following conditions:

1. The pattern consists of exactly three blocks: a block of 0's and 1's, followed by a block of 2's, followed by a block of 0's and 1's.
2. The lengths of the three blocks are equal.
3. The pattern of 0's and 1's appearing in the first block is identical to the pattern of 0's and 1's appearing in the third block.

Thus 020, 110222110, and 001102222200110 are members of S, while 002, 1122110, 1212, and 010222011 are not.

It is not difficult to design a one-tape, off-line Turing machine that recognizes the set S. Perhaps the most natural approach is to have the machine make a series of passes across the input pattern from left to right. On each pass the machine compares one symbol in the first block with the corresponding symbol in the third block, and "checks off" one symbol in the second block. In this way the machine will eventually determine whether the pattern consists of three equal-length blocks, and whether the first and third blocks contain identical patterns.

The total time required for such a computation is equal to the product of the number of passes and the number of steps per pass. If the input pattern is acceptable, each pass requires a round-trip travel time of $\frac{4}{3}n$ steps, and $n/3$ passes must be performed. Thus we might expect the total computation time to be about $\frac{4}{9}n^2$. Actually, a slightly longer time may be required to handle certain nonacceptable input patterns. Although the construction will not be given, there is an eight-state off-line machine that recognizes the set S by means of the procedure described above. This machine can be shown to complete its computations within the time $T(n) = \frac{4}{9}n^2 + n$.

Faster computations will result if the machine is designed to compare two or more symbols from the first and third blocks during a single pass. In particular, if k symbols are compared on each pass, the number of passes—and hence the total computation time—will be approximately divided by k. Such a scheme will, of course, require extra internal states. Without going into details, we may simply note that k symbols can be compared on each pass by a machine that uses at most $(k + 1)2^{k+1}$ states. The corresponding computation time can be shown to be at most

$$\frac{2n^2}{\log Q} + 4n \qquad \text{(for } Q \geqq 8\text{)}$$

where Q is the total number of internal states and the logarithm is taken to the base two. Thus the set S can be recognized by a one-tape, off-line Turing machine within a computation time whose growth for large n is directly proportional to n^2 and inversely proportional to $\log Q$, the number of bits of internal memory.

Let us now try to find a good lower bound for the fastest possible computation time, in terms of the number of states and the length of the input pattern. Suppose that we are presented with a one-tape, off-line Turing machine that has Q internal states and does in fact recognize the set S. In the discussion to follow we shall consider only the acceptable input patterns of some arbitrarily chosen length n, where n is a multiple of three. There are, of course, exactly $2^{n/3}$ such patterns. For each of these patterns the machine will go through a well-defined computation, generating a finite crossing sequence at each of the $n - 1$ boundaries within the pattern. Which of these crossing sequences must be distinct, and which may be identical?

Since we are considering only acceptable patterns, Theorem 1 and Corollary 1 imply that two crossing sequences can be identical only if the result of joining the tape segment on the left of one crossing sequence to the tape segment on the right of the other is to form another acceptable pattern. A little thought shows that this will happen only if the two crossing sequences appear at corresponding positions within the first (or third) blocks of two different computations, and if the patterns appearing to the left (right) of the two sequences are identical. In such a case, cutting the two initial tapes at the boundaries in question and swapping their left (right) ends does not change the patterns at all. For any other locations of the two crossing sequences, cutting the initial tape or tapes at the boundaries involved and rejoining the ends will yield at least one nonacceptable pattern. Hence the crossing sequences that appear on such boundaries must be distinct.

In particular, all of the crossing sequences that appear on boundaries within the center blocks of acceptable patterns of length n must be distinct. For if we consider two boundaries within the center block of the same pattern, removing the segment between the boundaries would result in a pattern without enough 2's to be acceptable. If we consider two boundaries within the center blocks of different patterns, swapping ends will result in a pattern in which the first and third blocks do not match. If the crossing sequences on the two boundaries were identical, either Corollary 1 or Theorem 1 would be violated.

We can now use this fact to derive a lower bound on the maximum time that the given machine must spend on some acceptable pattern of length n. Because the argument to be used does not take into consideration the crossing sequences that appear in the first and third blocks, it cannot be expected to yield the best possible bound. However, consideration of all the crossing sequences requires a rather lengthy argument and results in an improvement of only a factor of two in computation time.

Let s denote the number of distinct crossing sequences whose length does not exceed $\lambda = (n/3 \log Q) - 1$, where again Q is the number of states in the given machine and the logarithm is taken to the base two. Note that the number of distinct crossing sequences of length one is Q, the number of length two is Q^2, and so on. Therefore

$$s = \sum_{i=1}^{\lambda} Q^i < Q^{\lambda+1} = Q^{n/3\log Q} = 2^{n/3} \qquad (\text{for } Q \geqq 2)$$

But the number of acceptable patterns of length n is exactly $2^{n/3}$, which is greater than s. Since no one crossing sequence can appear in the center portion of two different computations, we see that there are not enough of the "short" crossing sequences to have one in the center portion of each computation. That is, there must be some computation performed on an acceptable pattern of length n for which the length of every crossing sequence in the center portion exceeds λ.

The amount of time that the given machine must spend on this one computation is equal to the sum of the lengths of the crossing sequences that appear in that computation. In particular, the amount of time the machine spends in the center block of the pattern is at least $(n/3)(\lambda + 1)$, since there are $n/3$ boundaries in this block and each must have a crossing sequence of length $\lambda + 1$ or more. Thus the total computation time is at least $n^2/9 \log Q$.

If all the crossing sequences in the computations performed on acceptable patterns of length n are taken into account, this lower bound can be doubled. Thus if n is a multiple of three, there must be some acceptable pattern of length n upon which the machine spends at least $2n^2/9 \log Q$ time units. In conclusion, if the set S is to be recognized by a Q-state, one-tape, off-line Turing machine, then the associated computation time must exceed $2n^2/9 \log Q$ when n is a multiple of three, but need not exceed $(2n^2/\log Q) + 4n$. In particular, for $Q = 8$, the computation time must exceed $2n^2/27$, whereas we know that it is possible to construct an eight-state machine whose computation time does not exceed $12n^2/27 + n$. Thus for large n the optimum computation time of an eight-state machine has been determined within a factor of six.

It is instructive to try to determine, at least on an informal basis, why the upper and lower bounds differ by as much as a factor of six. In order to do this, it is appropriate to think of the task of a one-tape machine as that of carrying data back and forth across the various boundaries of its tape. The shorter the computation time is to be made, the more data each individual member of a crossing sequence must carry. Of course, fixing the number of internal states also fixes the maximum "amount" of data that can be transported by a single crossing sequence member. If the shortest possible computation time is to be achieved, each crossing sequence member must be used to its full capacity. That is, for any choice of an internal state and integer i, it must be possible to find a crossing sequence in which the given state appears in the ith position.

Returning to the method described above for recognizing the set S, we see that the members of the crossing sequences are not being used at their fullest data-carrying capacity. In particular, whenever the machine moves from right to left, it is simply returning from the completion of one pass to the starting point of the next, and is carrying almost no useful information. Thus the data-carrying capacity of all the even-numbered crossing sequence members is almost entirely wasted. From this fact alone we would expect the computation time required by the present method to be about twice that given by the lower bound.

The efficiency of the realization is further reduced by other "inter-symbol" constraints that the machine imposes on its crossing sequences. These other constraints arise primarily from the machine's need to keep track of which block of the input pattern its reading head is currently scanning. Thus the machine's "housekeeping" duties interfere with its primary job of transporting information about the input pattern, and this interference substantially reduces the machine's computing efficiency. The only way of combating this problem is to change the method used to recognize the set S. With a modified computing scheme and a greatly expanded tape alphabet, it is possible to design an eight-state machine that comes within about a factor of two of achieving the computation time given by the lower bound. Thus, *in this particular example*, a relatively high data-carrying efficiency can be obtained, and the lower bound previously given is a relatively good one.

It must be remembered, however, that the argument used to obtain the lower bound on computation time does not take into account the fact that the crossing sequences involved will have to be generated by a one-tape Turing machine. Since the Turing machine model itself imposes constraints on the crossing sequences that can appear in a given computation, arguments similar to those of this section cannot be expected to lead to good lower bounds for all recognition problems.

The problem of recognizing the set S is interesting for several reasons. First, it illustrates the application of the crossing-sequence concept to the problem of determining minimum computation times. Second, it provides a specific example of a computing problem for which reasonably close upper and lower time bounds can be obtained. Third, it provides some insight into the relationships among crossing sequences, particular computing schemes, and the "efficiencies" of these schemes. Finally, it provides information about the relative speeds of one- and two-tape machines, as discussed below.

We next consider the problem of designing a *two*-tape, off-line Turing machine that recognizes the set S. Such a machine is to begin its computation with the input pattern written on one of its tapes, called the *input tape*. The second tape, called the *extra tape*, is to be completely blank at the beginning of the computation. One simple way of carrying out the desired computation consists of the following three steps:

Step 1. The machine moves both its tapes from right to left, copying the first block of the input pattern onto the extra tape. The total time required for this step is equal to the length of the first block.

Step 2. The machine continues to move the input tape from right to left but now moves the extra tape from left to right, so that it passes back over the pattern that it has just recorded on that tape. In this way the machine is able to compare the length of the second block (on the input tape) with the length of the first block (on the extra tape). If these lengths are not the same, the computation is immediately stopped. If the lengths are the same, the machine will end up scanning the first symbol of the third block on the input tape, and the first symbol of the first block on the extra tape. In any event the time required for this step will not exceed the number of symbols in the second block of the input pattern.

Step 3. The machine moves both tapes from right to left, comparing the pattern that appears in the third block (on the input tape) with the pattern that appears in the first block (on the extra tape). The time required for this step will not exceed the number of symbols in the third block of the input pattern. Thus if n is the length of the entire input pattern, the total time required for the computation is at most n, and the computation time is certainly bounded by the function $T(n) = n$.

The present example is one for which the fastest computation time that can be achieved with a one-tape, off-line machine is necessarily proportional to the square of the fastest computation time that can be achieved with a two-tape, off-line machine. On the other hand, we know from the work of Hartmanis and Stearns (1965) that if a given two-tape machine completes its computations within $T_2(n)$ time units, there must exist a one-tape machine that completes its computations within $C[T_1(n)]^2$, where C is a constant. In other words, going from a two-tape machine to a one-tape machine need never require more than a squaring of the computation time. But we now have an example in which the squaring is necessary. Thus the Hartmanis-Stearns "square law" cannot be improved upon in general.

B. Growth Rates for Other Sets

The set S provides an example of a recognition problem whose computation time necessarily grows in proportion to n^2, and for which a computation time that grows in proportion to n^2 can be realized. Thus we may think of n^2 as the "growth rate" associated with the recognition of S by a one-tape, off-line Turing machine. In this section we shall briefly consider recognition problems with other growth rates, in particular growth rates of the form n^p, where p is a rational number between one and two.

As a first example, we will examine a set of input patterns whose recognition time grows as $n^{3/2}$. This set is similar to the set S, the essential difference between the two lying in the relative lengths of the three blocks. The new set, designated set R, consists of just those patterns that satisfy each of the following conditions:

1. The pattern consists of exactly three blocks: a block of 0's and 1's, followed by a block of 2's, followed by a block of 0's and 1's.
2. Let n denote the length of the entire pattern and let x denote the length of the first block. Then
   (A) x is a power of 2, and
   (B) n equals x^2.
3. The patterns that appear in the first and third blocks are identical (which implies that the first and third blocks have the same length).

Thus the pattern 0111222222220111 belongs to the set R, while 011122220111 does not, because its entire length is not equal to the square of the length of its first block.

The set R may conveniently be recognized by a machine whose computations consist of two consecutive stages. In the first stage, the machine determines whether conditions 1 and 2 are satisfied; in the second stage it determines whether condition 3 is satisfied. Condition 1 can be checked very simply by making a single pass across the tape, which requires only n time units. Condition 2(A) is equivalent to requiring that $\log x$ be an integer, while condition 2(B) is equivalent to requiring that $\log x = \frac{1}{2} \log n$. The machine will therefore be designed to check condition (2) by computing the logarithms of x and n.

A machine can compute the integer part of the base-two logarithm of the length of a block of tape by making a series of passes across that block. On each pass it "marks" with some special symbol the first, third, fifth, etc. previously unmarked squares, as illustrated in Fig. 5. The number of passes required to mark all the squares will then be one greater than the integer part of the logarithm of the length of that block. Furthermore, the length of the block will be a power of two (and its

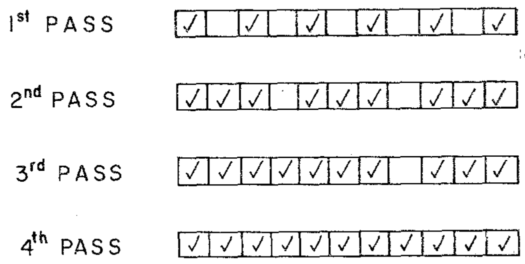

FIG. 5. Method of determining logarithms

logarithm will be an integer) iff the rightmost square in the block is the last square to be marked.

In order for a machine to determine whether condition 2 is met, it must carry out two marking operations, one on the entire pattern, the other on the first block alone. These two operations can be distinguished by the use of different marking symbols. It is most convenient to design the machine so that it makes two passes across the entire tape for each pass across the first block. Then condition 2 will be met iff the two marking operations are completed on the same pass, and in each case the last symbol is not marked until the last pass. Each pass requires at most $2n$ time units, and the maximum number of passes required is proportional to the logarithm of n. Thus the total time needed to check conditions 1 and 2 is at most $\alpha_1 n \log n$, where α_1 is a constant.

Finally, condition 3 can be checked in a manner similar to that used in the recognition of S. The machine again makes a series of passes across the tape, on each pass comparing one symbol in the first block with one symbol in the third block. The time required for a single pass is less than $2n$, while the number of passes is $x = n^{1/2}$. Thus the time required to check condition 3 is less than $2n^{3/2}$, and the time required for the entire computation is less than $2n^{3/2} + \alpha_1 n \log n$, which in turn is less than $\alpha_2 n^{3/2}$. In other words, the set R can be recognized within a computation time that grows only as fast as $n^{3/2}$.

That a growth rate of $n^{3/2}$ is necessary for the recognition of R can be established by a technique similar to that used in the preceding section. If we are not concerned with the actual constant of proportionality, it is sufficient to consider only the crossing sequences that are generated in the center blocks of the acceptable patterns of length n. Note that there are exactly $2^{n^{1/2}}$ different acceptable patterns of length n (assuming n to be a power of four). On the other hand, the number of distinct crossing sequences whose lengths do not exceed $\mu = (n^{1/2}/\log Q) - 1$ is

$$\sum_{i=1}^{\mu} Q^i < Q^{\mu+1} = 2^{n^{1/2}} \qquad \text{(for } Q \geqq 2\text{)}$$

Thus there are not as many "short" sequences as there are computations and some computation must have only crossing sequences of length greater than μ in its center block. Since the number of boundaries in the center block is at least $\frac{1}{2}n$, the total time required for this computation is at least

$$T(n) = (\mu + 1)\frac{n}{2} = \frac{n^{3/2}}{2 \log Q}$$

In other words, for any fixed number of states, the computation time required to recognize the set R must grow in proportion to $n^{3/2}$.

The definition of set R can now be modified so as to describe sets having different growth rates in the range from n to n^2, exclusive. The following conditions define a set of patterns whose associated growth rate is $n^{1+q/r}$, where q and r are integers and q is less than r.

1. Each pattern consists of exactly three blocks: a block of 0's and 1's, followed by a block of 2's, followed by a block of 0's and 1's.

2. Let n denote the length of the entire pattern and let x denote the length of the first block. Then

(A) x is a power of 2^q,

(B) $x = n^{q/r}$.

3. The patterns that appear in the first and third blocks are identical.

A machine that recognizes such a set can be designed in much the same way as the machine that recognizes R. As before, condition 1 is quite easy to check. Conditions 2(A) and 2(B) together are equivalent to requiring that

$$\frac{1}{q}\log x = \frac{1}{r}\log n = \text{integer}$$

Therefore when the machine computes the logarithms of x and n it should make a series of r passes across the entire pattern, followed by a series of q passes across the first block, followed by a series of r passes across the entire pattern, etc. Condition 2 will then be met iff

(a) The marking operation for the entire pattern is completed on the first pass after some integral number, k, of series of r passes.

(b) The marking operation for the first block is completed on the first pass after the same integral number k, of series of q passes.

(c) The last step of each marking operation is the marking of the last square in the appropriate tape segment; i.e., both $\log x$ and $\log n$ are integers.

Condition (3) can be checked in the usual manner. The total computation time is at most

$$\alpha_4 n \log n + \alpha_5 n(n^{q/r}) \leqq \alpha_6 n^{1+q/r}$$

That a growth rate of $n^{1+q/r}$ is necessary for the recognition of the given set can be shown by arguments similar to those used for the set R.

C. General Results

The results developed in the preceding two sections can be summarized in:

Theorem 4. *For any rational number p in the range $1 < p \leqq 2$ there exists a set of input patterns S_p such that*

(a) S_p can be recognized within a computation time that is less than $C_1(p, Q)n^p$, and

(b) S_p cannot be recognized within a computation time that is less than $C_2(p, Q)n^p$

where C_1 and C_2 are functions only of p and the number of internal states available for the computation.

Corollary 4. *If p and q are two real numbers in the range $1 \leqq q < p \leqq 2$, then there exists a set of input patterns that can be recognized within a computation time that is proportional to n^p, but that cannot be recognized within a computation time that is proportional to n^q.*

Proof: First choose r to be a rational number such that $q < r < p$. Then according to Theorem 4 there must exist a set of patterns that can be recognized within $C_1(r, Q)n^r$ time units, but that cannot be recognized within $C_2(r, Q)n^r$ time units. Since $C_1(r, Q)n^r < C_1(r, Q)n^p$, this set can certainly be recognized within a computation time that is proportional to n^p. Now assume that there exists some Q-state machine that recognizes the same set within Cn^q time units. But for sufficiently large n, $Cn^q < C_2(r, Q)n^r$, contradicting the assumption. Thus the set cannot be recognized within a computation time that is proportional to n^q.

Q. E. D.

As pointed out before, it seems reasonable to conjecture that if a certain set of input patterns can be recognized by a one-tape, off-line Turing machine within a computation time $KT(n)$, then it can also be recognized within the computation time $T(n)$, provided $T(n) \geqq n$. In other words, it seems reasonable to suppose that any computation can be speeded up by a constant factor through the use of additional internal states. According to Theorem 4, however, this is the best speed-up that can be obtained for an arbitrary recognition problem, at least in the range of computation times from n to n^2.

Furthermore, the allowable computation time need not be increased very much in order to provide one-tape, off-line machines with the ability to recognize new sets. According to Corollary 4, increasing any time bound of the form $n^p (1 \leqq p < 2)$ by a factor of $n^\epsilon (\epsilon > 0)$ increases the computing capabilities of one-tape machines. Thus the hierarchy of complexity classes (Hartmanis and Stearns, 1965) is very densely packed, at least in the range from n to n^2.

IV. CONCLUSION

Crossing sequences provide a convenient means of describing the manner in which a one-tape, off-line Turing machine carries "information" from one part of its tape to another. Their role may be compared with that of the internal states of a finite-state machine. The occurrence of a particular state in a finite-state machine specifies the one class, out of a finite number of classes, that contains the input sequence that the machine has received so far. The occurrence of a particular crossing sequence in the computation performed by a Turing machine specifies two classes: one that contains the portion of the initial tape pattern that lies to the left of the sequence, and one that contains the portion of the initial tape pattern that lies to the right of the sequence. In each case, the specification is of one class out of an infinite number of classes.

Thus both internal states and crossing sequences provide some information about the input pattern with which their respective machines are supplied. Whereas the internal state supplies information only about the pattern that appears on one "side" of its point of occurrence (the past), the crossing sequence supplies information about the patterns that appear on both sides of its point of occurrence. Furthermore, the state of a finite-state machine can denote only a finite number of classes of past histories, while the crossing sequences of a Turing machine can denote an infinite number of pairs of classes of patterns.

When working with finite-state machines, it is often possible to conclude that two specific input sequences must lead the machine to different internal states in order that the machine behave properly in the future. In much the same way, it is often possible to conclude that the crossing sequences that appear at specific points in one or more computations must be distinct in order that the Turing machine that produces them behave properly for all possible input patterns. The knowledge that the states that result from certain input sequences must be distinct enables us to place a lower bound on the number of states required by a finite-state machine. The knowledge that certain crossing sequences must be distinct similarly enables us to place a lower bound on the time required by the computations of a one-tape Turing machine. Unfortunately, such a bound is not usually as easy to obtain as the analogous bound on the number of states of a finite-state machine.

The time bounds arrived at through consideration of crossing sequences are not necessarily very close ones. There are two reasons for this. First, it is not always feasible to take into account all the distinctions that must exist among the various crossing sequences that a given Turing machine generates. Thus it may be necessary to ignore part of the machine's task when deriving a lower bound on the computation time. Second, the arguments used to obtain the bounds assume that the machine uses its crossing sequences to carry "information" at maximum efficiency. In practice, this is not always possible, either because the machine must return "empty-handed" from some remote portion of its tape, or because it must spend considerable time organizing, or encoding, the data to be transmitted into a form in which it can be used in another part of the tape.

It is this last problem, that of encoding data into a usable format, that prevents the concept of a crossing sequence from being very useful for problems that require computation times greater than n^2. For consider the extreme case in which all the crossing sequences in all the computations performed on all the input patterns of length n or less must be distinct. Even in this case, the arguments used in the preceding sections require an average computation time that grows only as fast as n^2. This is the maximum growth rate required, under ideal conditions, to distribute to each tape square complete information about the symbols initially appearing in every other tape square. The catch is, of course, that simply distributing this data is not enough. It must be supplied in such a form that it can be correctly interpreted by the machine at the point at which the decision concerning acceptance or rejection is made. Since this decision must be made by what amounts to a finite-state mechanism (whose inputs are the states of the Turing machine and whose internal states are the tape symbols), the format of the crossing sequences is often severely constrained. In such cases the crossing sequences may need to be much longer than the length dictated by data-transmission considerations alone, and the total computation time may need to be much greater than that dictated by the arguments of Section III.

Although the idea of a crossing sequence can be extended to apply to off-line machines with two tapes, it does not seem to be useful for obtaining time bounds for such machines. Perhaps the most appropriate way of regarding the computations of a two-tape machine is to think of the reading head as moving in a plane, the coordinates of a point in this plane indicating the location of the head on each of the two tapes. It is then possible to define a crossing sequence as the sequence of internal states in which the machine enters and leaves a given square in the plane. Unfortunately, there seems to be no direct counterpart of Theorem 1 or Corollary 1 that applies to the planar situation, and hence the entire method of finding lower bounds falls apart.

In spite of its limited range of application, the concept of a crossing sequence can be used to obtain strong results about certain Turing machine computations. Indeed, the class of one-tape, off-line Turing machines is one of the few classes of Turing machines for which such strong statements as Theorem 4 can be made. Most important, this class of Turing machines is the only one for which any concept that approaches the power and usefulness of the finite-internal-state concept is presently available.

Acknowledgment

The author wishes to thank Dr. Juris Hartmanis of the General Electric Research Laboratory for his critical reading of the manuscript.

Received: January 12, 1965

REFERENCES

Yamada, H. (1962), Real-time computation and recursive functions not real-time computable, *IRE Trans. Electron Computers* **EC-11**, 753–760.

Hartmanis, J., and Stearns, R. E. (1965), On the computational complexity of algorithms. *Trans. Am. Math. Soc.*, in press.

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