HIERARCHIES OF MEMORY LIMITED COMPUTATIONS

R.E. Stearns, J. Hartmanis, and P.M. Lewis II
General Electric Research Laboratory, Schenectady, N. Y.

An important goal of automata theory is a basic understanding of the computational process for various classes of problems. The theory must go beyond the notion of computability and include some measure of the difficulty of the computation and how that difficulty is related to the organization of the machine that performs the computation. The two measures of difficulty that appear particularly important are time and memory. In this paper we apply the measure of tape square requirements to a number of computer-like machine models, including the classical Turing machine. The results are similar to but stronger than the time limited results obtained in [1], [2], and [3].

The use of tape space as a complexity measure is perhaps less realistic than the time measure, but this is compensated for by the sharper results and by the many insights it can give. (See [4] for example.) It is generally easier to intuit why a problem inherently requires so much tape rather than why it requires so much time. In this paper, we restrict our attention to the so-called recognition problems, but the extension to other problems is fairly evident.

The Turing machine variations we consider are on-line or off-line (one-way or two-way) and push-down tape and ordinary tape. The input tape is separated from the working tape so as to achieve complexity less than that of linear bounded automata. The study of the rather specialized push-down cases is justified by its potential applications to the study of languages, such as those of [4].

THE BASIC MACHINE MODELS

We begin with our most general definition, that of an off-line Turing machine. This definition is specifically oriented toward recognition problems.

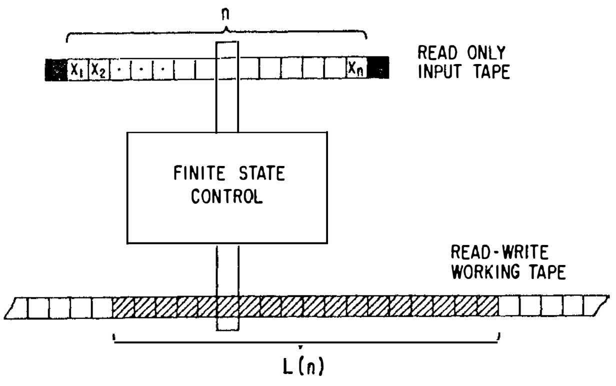

Fig. 1 Off-line Turing Machine

An <u>off-line Turing machine</u> is a machine as shown in Fig. 1 consisting of a finite control with two tapes: one a read only <u>input tape</u> and the other a read-write <u>working tape</u>. The squares of the input tape may contain either symbols from a finite <u>input alphabet</u> A or a special end marker symbol and the working tape squares may contain any symbol from a finite set K. An operation of the machine is determined by the state of the control and the symbols under each reading head. The operation consists of over-printing a new symbol on the working tape, shifting the input tape (one square right, one square left, or no shift), shifting the working tape, and changing the control state. Certain stopping states are called <u>accepting states</u> and certain other stopping states are called <u>rejecting states</u>. The machine is always started in a designated starting state with a blank working tape and the input reading head on the leftmost square of a finite length of tape with end markers on the two end squares and symbols from A on the other squares. These symbols from A read left to right constitute the <u>input word</u>. It is assumed that the control unit is designed so as to keep the reading head on the input tape. If the sequences of operations resulting from an input word carries the machine into an accepting state, we say that the word is accepted by the machine, and if the input word is carried into a rejecting state, we say that the input is rejected. If any word is neither accepted or rejected, then we say that the machine is not a recognition device.

An off-line push-down machine is an off-line Turing machine with the restriction that the machine overprint the blank symbol before any right shift of its working tape. It is customary to think of this right shift as a pop-up and a left shift as a push-down.

An on-line Turing machine is an off-line Turing machine with the additional restriction that the input tape cannot be shifted right and with the change that the disjoint accepting and rejecting states be states that occur just prior to a left shift rather than being stopping states. An input word on the input tape is accepted by the machine if and only if the machine enters an accepting state prior to shifting the last input symbol and the machine rejects the input if and only if it enters a rejecting state prior to shifting the last symbol. For this on-line model, the end markers are superfluous.

An on-line push-down machine is an on-line Turing machine with the restriction that the machine overprint the blank symbol before any right shift of its working tape. Again, it is customary to use the pop-up and push-down terminology.

A machine M of any of the four above types is said to recognize a set of finite words W if and only if each word in W is accepted and each word not in W is rejected by the machine.

TAPE COMPLEXITY

If L(n) is a computable function of the positive integers into the positive integers, we say that L(n) is a tape function and that a machine M operates within tape L(n) if and only if the working tape head visits at most L(n) squares of the working tape for each input of length n. If a machine M recognizes a set W and operates within tape L(n), then we say that W is L(n)-tape recognizable on a machine of the same type as M. For any given machine type, we let

$$C_{L(n)} \quad \text{or} \quad C_L$$

represent the collection of all sets that are L(n)-tape recognizable by a machine of that type. The set $C_{L(n)}$ is referred to as a tape complexity class.

The tape complexity classes have many properties in common with the time complexity classes of [1]. For any L(n) and any type machine, the class C_L is recursively enumerable. Thus no C_L can contain all recursive sets. For the two Turing machine models, this fact guarantees an infinite set of distinct complexity classes. The chief purpose of this paper is to investigate these complexity classes and their ordering in more detail.

Using the notation $\lceil r\rceil$ to represent the smallest integer m such that $m \geq r$, we can now state a basic equivalence theorem.

Theorem 1: For any machine type and any positive integer N, a set W is L(n)-tape recognizable if and only if it is $\lceil L(n)/N\rceil$-tape recognizable; in other words,

$$C_{L(n)} = C_{\lceil L(n)/N\rceil}$$

Proof: Obviously, any machine M which recognizes W within tape L(n) can be redesigned into a machine M' so that one square of the working tape of M' contains the information on N working tape squares of M and such that M' recognizes W within tape $\lceil L(n)/N\rceil$.∎

Corollary 1.1: If L and Q are tape functions such that

$$\inf_{n\to\infty} \frac{L(n)}{Q(n)} > 0 \quad ,$$

then $C_Q \subseteq C_L$.

Proof: The inequality means that $L(n) \geq Q(n)/N$ for some large N and so $L(n)$-tape recognizable sets are automatically $\lfloor Q(n)/N \rfloor$- tape recognizable and are therefore $Q(n)$-tape recognizable.

Corollary 1.2: If L and Q are tape functions such that

$$\sup_{n\to\infty} \frac{L(n)}{Q(n)} < \infty ,$$

then $C_Q \supseteq C_L$.

Proof: This is the reciprocal of Corollary 2.1.

Corollary 1.3: If L and Q are tape functions such that

$$0 < \lim_{n\to\infty} \frac{L(n)}{Q(n)} < \infty ,$$

then $C_Q = C_L$.

Proof: This follows from Corollaries 1.2 and 1.3.

This last theorem and corollaries show that it is the rate of growth of $L(n)$ that determines the class $C_{L(n)}$ and we may thus permit ourselves to be a little careless in specifying tape functions. We will sometimes, for example, refer to tape function $L(n) = \log \log n$ even though we have not specified a base, even though it is not integer valued, and even though it is technically undefined for n equal to one. The exact method of rounding this function off to an integer and defining it for small n is clearly irrelevant and a change of base simply introduces a constant factor which by Corollary 1.3 leaves the complexity class unaltered.

MINIMAL TAPE FUNCTION GROWTHS

The smallest tape complexity class is that defined by the tape function $L(n) = 1$ or equivalently by any constant or bounded $L(n)$. This class is obviously the class of regular sets: those sets that can be recognized by a finite state machine. There are, however, some unbounded $L(n)$ that also give the same tape complexity class of regular sets. For each machine type, there is a minimal growth rate that must be maintained in order to recognize non-regular sets. We show this first for off-line machines.

Theorem 2: For any off-line machine M that recognizes a non-regular set and operates within tape $L(n)$, there is a number p such that for all integers j, there is a number n_j such that

$$L(n_j) \geq j \quad \text{and} \quad n_j \leq p^{p^j} .$$

Proof: For purposes of this proof, we assume without loss of generality that the reading head of the working tape never leaves a blank on a square that it visits. This amounts to treating overprinted blanks as distinct from original blanks. We define a memory configuration to be any combination of control state, string of non-blank symbols printed on the working tape, and location of the head on these symbols. If the string of symbols has length (strictly) less than j, we call the memory configuration a j-configuration. If k is the number of nonblank symbols in the working tape alphabet and q is the number of control states, the number of memory configurations with string of length exactly i is clearly $i \cdot q \cdot k^i$ and the number of j-configurations is $\sum_{i=1}^{j-1} i \cdot q \cdot k^i$ which is less than $j \cdot q \cdot k^j$.

Letting q be the number of symbols in the input alphabet, we choose p to be a number such that

$$p^{p^j} > a \cdot 2^{jqk^j} \text{ for all } j .$$

Now let n_j be the smallest integer such that M uses at least j working tape squares to process some input word

$$w = x_1 x_2 \ldots x_{n_j}$$

of length n_j. The non-regularity of the set recognized by M guarantees the existence of such a word. Because of w, we must of course have $L(n_j) \geq j$. Thus when M finally either accepts or rejects w, it is not in a j-configuration since by definition, j-configurations utilize less than j working tape squares.

In processing w, the machine M may cross and recross many times any particular input symbol x_i on its input tape. For each $i \leq n_j$, let S_i be the set of all configurations achieved by M while processing w and while the reading head is at the symbol x_i.

In order to bound n_j, we will first show that $S_r = S_s$ for $r < s$ implies that $x_r \neq x_s$. For assume to the contrary that $S_r = S_s$ and $x_r = x_s$ for some $r < s$, and consider what happens when the machine processes the word

$$w' = x_1 \dots x_r x_{s+1} \dots x_{n_j} \quad .$$

The length of w' is less than n_j and so all the memory configurations that occur in processing w' must be j-configurations because of the minimal nature of n_j. We now show that the j-configurations for any x_i of w' will be j-configurations from the set S_i (which recall was defined for w). Assume to the contrary that in processing w', M is reading x_i and for the first time enters a j-configuration that is not in S_i. This could only conceivably occur when the reading head had just crossed from x_r to x_{s+1} or from x_{s+1} to x_r. Suppose the machine is crossing from x_r to x_{s+1}. By assumption, at x_r its configuration is in S_r. But since $S_r = S_s$ the machine is in a configuration that could also have occurred at x_s in w. Furthermore since $x_r = x_s$, the machine must go into one of the configurations that occur at x_s in processing w. This configuration is by definition in S_s, which proves the assertion that all j-configurations occurring from processing w' are S_i configurations. But this means that M does not stop while processing w' since the only stopping configuration in the processing of w is not a j-configuration (because at least j squares are needed and thus the S_i contain no stopping configuration). Since M must stop for all inputs by definition, this is a contradiction which proves the assertion that $S_r = S_s$, implies that $x_r \neq x_s$.

Since the S_i have to be distinct for two occurrences of the same input symbol, the input length n_j must be no greater than the number of subsets of j-configurations times the number of input symbols. But this number is less than $a2^{jqk^j}$ and so $n_j \leq p^{p^j}$ by choice of p.

Corollary 2.1: For off-line type machines, if L(n) is a tape function such that

$$\lim_{n\to\infty} \frac{L(n)}{\log \log n} = 0 \quad ,$$

then C_L contains only regular sets; that is $C_{L(n)} = C_{\text{constant}}$

Proof: If C_L contains a non-regular set, then $L(n_j) \geq j \geq \log_p \log_p n_j$ or

$$\frac{L(n_j)}{\log_p \log_p n_j} \geq 1$$

for all n_j given by the theorem and so the limit cannot go to zero.

Corollary 2.2: For machines of an off-line type, if $C_{L(n)}$ contains a non-regular set, then

$$\sup_{n\to\infty} \frac{L(n)}{\log \log n} > 0$$

Proof: This is the converse of Corollary 2.1.

Corollary 2.3: For machines of an off-line type, if L(n) is a monotone tape function such that $C_{L(n)}$ contains a non-regular set, then

$$C_{L(n)} \supseteq C_{\log\log n}$$

Proof: Letting $j = [\log_p \log_p n]$, we have $n \geq n_j$ and so

$$L(n) \geq L(n_j) \geq j = [\log_p \log_p n],$$

hence

$$\inf_{n\to\infty} \frac{L(n)}{\log_p \log_p (n)} \geq 1$$

and the result follows by Corollary 1.1.

Theorem 3: For on-line Turing machines, if L(n) is a tape function, then either

$$C_{L(n)} = C_{\text{constant}} \text{ or } C_{L(n)} \supseteq C_{\log n},$$

depending on whether or not

$$\inf_{n\to\infty} \frac{L(n)}{\log n} = 0 \quad .$$

Proof: Assume that there is an on-line Turing machine M which operates on tape L(n) and recognizes a non-regular set. We make all the conventions we did in the first paragraph of the proof of Theorem 2 except that we choose p such that

$$p^j > a \cdot j \cdot k^j$$

for all j. For any given n, let

$$w = x_1 \ldots x_m$$

be a shortest word such that the processing of w by M marks at least $j = \log n$ tape squares. In processing w, M cannot mark j working tape squares before the reading head reaches x_m, because than a shorter subsequence of w would use j squares. Machine M cannot repeat a configuration on x_r and x_s such that $x_r = x_s$ and $r < s$ because then

$$w' = x_1 \ldots x_r x_{s+1} \ldots x_n$$

would be a shorter word using j working tape squares. Therefore, m - 1 is no greater than the number of j-configurations times the size of the input alphabet. Therefore $m \leq p^j \leq n$ and w extended to a word of length n also requires $j = \log_p n$ tape squares. Therefore

$$\frac{L(n)}{\log_p n} > 1 \text{ for all } n$$

and the result follows.

Theorem 4: For an on-line push-down machine and tape function L(n), either

$$C_{L(n)} = C_{\text{constant}} \text{ or } C_{L(n)} = C_n ,$$

depending on whether or not

$$\inf_{n\to\infty} \frac{L(n)}{n} = 0 \quad .$$

Proof: Assume that there is an on-line push-down machine M that operates on tape L(n) and recognizes a non-regular set. There must be some input word

$$w = x_1 \ldots x_m$$

that puts the machine into a configuration with p symbols pushed-down for some p > aqk where a is the size of the input alphabet, q the number of control states, and k the size of the working tape alphabet. Call these symbols $k_1 \ldots k_p$. With each k_i, we associate the control state q_i and the input symbol x_{j_i} that occurred just before k_i was printed. Because of the size of p, there must be some r and s such that $r < s$, $k_r = k_s$, $x_{j_r} = x_{j_s}$, and $q_r = q_s$. If $j_r = j_s$, the machine cycles and if $j_r < j_s$, then

$$w_h = x_1 \ldots x_{j_r} (x_{j_r+1} \ldots x_{i_s})^h$$

is an input word of length $j_r + (j_s - j_r)h$ that requires $s + (s-r)h$ tape squares.

Thus L(n) must grow linearly as the theorem claims.

Very similar reasoning shows that the machine cannot push down more than qk tape symbols for a given input letter without cycling and thus any non-regular set which is recognizable, must be L(n) = n tape recognizable.

This last theorem completely disposes of the on-line push-down case. The hierarchy consists of two classes, C_1 and C_n. These classes are distinct because some push-down machines recognize non-regular sets.

For the on-line Turing machine, it is clear that the set

$$\{0^k 1^k \mid k \text{ an integer}\}$$

is log n-tape recognizable and so the bound of Theorem 3 is exact.

We now wish to find sets which are log log n-tape recognizable on the off-line machines. Let b_i represent the binary expansion of integer i on alphabet {0,1} and let w_k on alphabet {0,1,s} be given by

$$w_k = b_o \ s \ b_1 \ s \ldots s \ b_k \quad .$$

(e.g. w_4 = 0 s 1 s 10 s 11 s 100) To recognize whether a given input sequence is in $\{w_k\}$, treat it as a sequence of binary numbers placed between s markers, and check the first number to see if it is b_o. If it is, start checking the sucessive numbers to see if they are one more than the preceding. If and when a number is encountered that fails this test, quit the computation and reject the sequence. To check any pair of consecutive numbers to see if the right-hand number is one larger, it is sufficient to compare corresponding bits one at a time in order and remember the carry variable with the finite control. On the off-line Turing machine, the digits may be compared one at a time going back and forth between the numbers using the working tape to record the position (i.e. first, second, etc.) of the digit being compared. If the left-hand number has length ℓ, than this takes log ℓ working squares. On the other hand, it is easily verified that $\ell \leq$ log n where n is the length of the input word. Hence our scheme takes log log n on the off-line Turing machine and the lower bound of Corollary 2.3 is exact for the off-line Turing machine.

To get an example for the off-line push-down machine, we first observe that the sequence w_k can be recognized on an off-line push-down machine using L(n) = log n. The digits in each b_i are stored on the push-down tape and then compared digit by digit with b_{i+1}. We now modify w_k as follows: We add two new symbols, z and u (for zero and unit) to the input alphabet {0,1,s} and use these to mark the position of a digit. Thus we define b_i^a to be the binary form of i written in {0,1} with the digits addressed with consecutive numbers written in {z,u}. For example,

$$b_9^a = \text{uzz } 1 \text{ uu } 0 \text{ uz } 0 \text{ u } 1 \quad .$$

We define

$$w_k^a = b_o^a \ s \ b_1^a \ldots s \ b_k^a$$

and consider how efficiently we can recognize $\{w_k^a\}$. Expanding on the push-down observations of the previous paragraph, it is easy to verify that one can first check the consistency of the addresses (using log ℓ squares) and then use these addresses for a bit by bit comparison as in the previous case, thus doing all the processing in log log n tape squares.

MAXIMAL TAPE FUNCTION GROWTHS

Since the Turing machines can recognize all recursive sets, no maximal C_L can be found for these machines. On the other hand, we have seen in Theorem 4 that C_n contains all the sets that can be recognized by an on-line push-down machine. The off-line push-down case is settled by the next result.

Theorem 5: For off-line push-down machines, if L(n) is a tape function, then

$$C_{L(n)} \subseteq C_n \quad .$$

Proof: We need to show that C_n contains the set recognized by any given off-line push-down machine M. Suppose that at some point in processing an input of length n, machine M has symbols $k_1 \ldots k_m$ printed on its working tape. With each symbol k_i, we associate the control state q_i and the square of the input tape which occurred just after k_i was pushed-down for the first time. If $k_r = k_s$ and $q_r = q_s$ for some $r \neq s$, then the associated input squares must be distinct, for otherwise M would return to that square over and over again, each time with additional tape symbols, and would never stop. Thus we must have

$$m \leq q \cdot k \cdot (n+2)$$

where q is the number of control states, k the size of the working tape alphabet, and n+2 the number of input tape squares (counting the end markers). Thus M recognizes its set within tape $L(n) = q\ k\ (n+2)$ and so this set is $L(n) = n$-tape computable by Corollary 1.3.

LIMIT THEOREMS

We have seen that, for each machine type, there is a quantum jump between the tape requirements for regular sets and those for non-regular sets. We shall now show that above the initial jump (but below C_n in the push-down case) a very slight increase in limiting behavior of a given tape function defines a new tape-complexity class. The exact statement of these results requires the concept of "constructability," which we discuss with each machine type.

A tape function L(n) is called constructable on an off-line Turing machine if and only if there is an off-line Turing machine M which always stops, operates within tape L(n), and for each n takes L(n) working tape squares for some input of length n. (The condition that M stops is actually superfluous.) Because of Theorem 1 and its corollaries, we informally extend the term "constructable" to include those functions Q(n) with the same limiting behavior as constructable L(n). This means, for example, that $L(n) = \log \log n$ is constructable because we previously discussed a machine (to recognize $\{w_k\}$) which operated within tape log log n and took "exactly" log log n for some inputs of length n. Similarly, each off-line Turing machine that recognizes some set also defines a corresponding tape function. The "real-time countable" functions of [5] and [1] are also constructable because the counters do not take enough operations to use more tape than they count out. If R(n) is real-time countable, one can first get log log n by the previous scheme and then count off R(log log n) working tape squares. Thus R(log log n) is constructable. Since the real-time countable functions are known to include all the common monotone increasing functions obtained from multiplication, exponentation and function concatenation, the hierarchy of constructable functions above log log n is very rich indeed. We now show that the hierarchy of constructable functions correspond to the tape-complexity class hierarchy (for off-line Turing machines) within the equivalence of Corollaries 1.1 and 1.2.

Theorem 6: For off-line Turing machines, if L(n) is a constructable tape function, then there exists a set W which is L(n)-tape computable and is not Q(n)-tape computable for all tape functions Q(n) such that

$$\inf_{n\to\infty} \frac{Q(n)}{L(n)} = 0 \quad .$$

Proof: The proof is by a diagonalization method. We exhibit a machine M which operates on L(n) tape such that given any other machine that uses Q(n) tape (as in the theorem statement), there is some input sequence for which M simulates this other machine operating on that sequence and gives a different output. Thus the set W is the set recognized by M.

Let M' be a machine that constructs L(n) and has an input alphabet A'. Consider a new alphabet A formed from A' in the following way: For each symbol x in A', there are two symbols x_0 and x_1 in A.

Each input word from A may be interpreted as a word from A' (by dropping subscripts) or as a binary number (reading only the subscripts). We now design a machine M which operates in three phases.

Phase 1: Machine M reads the input from A as a word from A' and behaves like M' until M' stops. During the process, it leaves a mark in the extreme left and right working tape squares it visits. Under all circumstances, M will confine itself to the squares between the markers, thus insuring that M operates within tape L(n).

Phase 2: Reading the input as a binary number i, M prepares on its working tape a description of the i-th off-line Turing machine over input alphabet A. This takes a certain number of squares D_i and if there is not enough room between the end markers on the working tape, the machine stops and enters a rejecting state.

Phase 3: If the description of M_i has been successfully completed, the machine M begins to simulate M_i using the input tape as the input and a special track of the working tape as the working tape of M_i. The working tape symbols of M_i are coded into binary and in general it will take some k_i squares of M to represent the information in one square of M_i. A third track of the working tape is used to count in base three the operations of M_i that have been simulated. The simulation is continued until one of three situations occur:

a) The simulation requires more working tape than is available between the end markers. At this point, M enters a rejecting state.

b) The counting of the operations of M_i requires more working space than is available between the end markers. At this point, machine M enters a rejecting state. Note that this rule insures that M always stops.

c) Machine M_i stops. Machine M enters a rejecting state if M_i enters an accepting state and M enters an accepting state if M_i enters a rejecting state.

Obviously, machine M recognizes some set W within tape L(n) and all that remains is to be shown that W cannot be recognized within any Q(n) which satisfies the theorem. Suppose to the contrary that some M_j recognizes W within tape Q(n) where Q(n) is some function satisfying the limit equation. Clearly there is a large integer N such that

$$L(N) \geq \log_2 j \quad ,$$

$$D_j \leq L(N) \quad ,$$

$$k_j \cdot Q(N) \leq L(N) \quad ,$$

and

$$g_j \cdot N \cdot L(N) \cdot 2^{L(N)} < 3^{L(N)} \quad ,$$

where g_j is the number of control states of M_j. Let w' be a word from A' which has length N and for which M' visits L(N) working tape squares. By adding subscripts, convert w' to a word w from A such that the binary number represented is j. This is possible because $L(n) \geq \log_2 j$ and because sufficient zero's can be added on the left of the binary representation of j to get a length N binary representation. Now consider what happens when input w is applied to M.

In phase 1, L(N) tape squares are marked off. Phase 2 can be completed because $D_j \leq L(N)$. In phase 3, situation a) cannot occur because M_j only requires $k_j \cdot Q(N)$ tape squares which is less than L(N). Situation b) cannot occur because M_j does not cycle and cannot therefore take more than $L(N) \cdot g_j \cdot 2^{L(N)}$ operations before it stops, and M can count up to $3^{L(N)}$ operations. Whenever situation c) occurs, M accepts those words which M_j rejects and rejects those that M_j accepts, and this would be contrary to the assumption that M_j recognizes w. Thus there is no possibility that M_j recognizes w within Q(n) and so the theorem is proved.

A tape function L(n) is called constructable on an on-line Turing machine if and only if there is an on-line Turing machine M which always stops, operates within tape $L(n)$, and for each n takes exactly $L(n)$ working tape squares for all inputs of length n. Again, we informally call $Q(n)$ "constructable" if it has the same limiting behavior as $L(n)$.This means, for example, that $L(n) = \log n$ is constructable because counting the number of inputs in an appropriate base b causes a machine to visit exactly $[\log_b n]$ squares. Similarly to the off-line case, if $R(n)$ is a real-time countable function, function $R(\log n)$ is constructable. Thus we have a rich hierarchy of constructable functions above log n which we now show corresponds to the tape-complexity hierarchy for on-line Turing machines.

Theorem 7: For on-line Turing machines, if $L(n)$ is a constructable tape function, then there exists a set W which is $L(n)$-tape computable and is not $Q(n)$-tape computable for all tape functions $Q(n)$ such that

$$\inf_{n\to\infty} \frac{Q(n)}{L(n)} = 0 \quad .$$

Proof: Since the theorem holds trivially for bounded $L(n)$, we assume that $L(n)$ is unbounded. We know from the proof of Theorem 4 that any machine that constructs an unbounded $L(n)$ must have $L(n) \geq \log n$ for some base. This fact enables us to again use a diagonalization proof. We exhibit an on-line machine M which uses $L(n)$ tape and such that given any machine which uses $Q(n)$ tape, there is some input sequence for which M simulates this machine and gives a different output. Because of the on-line character of M, the simulation proceeds somewhat differently.

The input alphabet of $M = \{0,1\}$. After its i^{th} input M must do three things.

Phase 1: Machine M initiates the machine that constructs $L(n)$ and marks off $L(i)$ squares on the working tape useing special end markers. Subsequent machine operations before the next input will be confined to this space.

Phase 2: The machine updates its input storage. The past input is represented (if possible) on the working tape by a number and a word, each stored in individual tracks. The number represents the number of initial zeros of the input sequence and the word consists of the first one and all subsequent inputs. The number of initial zeros can always be stored in some base b such that $L(n) \geq \log_b n$. If the word doesn't fit the alloted working tape space, the machine enters a state which rejects the present and all subsequent inputs.

Phase 3: Treating the stored word as a binary number j, the machine treats this as a description of the j-th binary input on-line Turing machine M_j, and simulates M_j as it would process the input sequence already applied to M. Using a special binary track of the working tape to simulate the working tape of M_j, some k_j squares of M will be sufficient to represent the information in one square of M_j. The number of operations of M_j for each input symbol are counted in base three on another track of the working tape. The simulation is continued until one of three situations occur:

a) The simulation requires more working tape than is available between the end markers. At this point, M enters a rejecting state and goes on to the next input symbol.

b) The counting of the operations of M_j requires more working tape than is available between the end markers. At this point M enters a rejecting state and goes on to the next input symbol. This rule prevents M from processing an input symbol indefinitely.

c) Machine M_j completes its processing of the input word. Machine M rejects the input if M_j accepts it and accepts the input if M_j rejects it. Machine M then goes on to the next input symbol.

Obviously, machine M recognizes some set W within tape $L(n)$ and all that remains is to show that W satisfies the theorem. Supposing to the contrary that some M_j recognizes W within tape $Q(n)$ where $Q(n)$ is some function satisfying the limit equation, we find an N such that

$$L(N-[\log_2 j]) \geq \log_2 j \quad ,$$

$$k_j \cdot Q(N) \leq L(N) \quad ,$$

and

$$L(N) \cdot 2^{L(N)} < 3^{L(N)} \quad .$$

Consider what M does after applying $N-[\log_2 j]$ zeros followed by the binary representation of j. The first inequality insures that there is enough room to store the input, the second insures that there is enough space to simulate M_j and the third insures that the operations of M_j do not overflow the counter.

Following the reasoning used to prove Theorem 6, it is clear that M must in fact treat this input opposite to M_j and the theorem is proved by contradiction.

For the off-line push-down machines, we know of no way to implement a diagonalization proof as was done for the Turing machines, but we can still demonstrate a hierarchy by finding sets that satisfy the next lemma.

Lemma: If W is a set of words on alphabet A and L(n) is a function of integers into integers such that for each n, there is a word w_n in W and a symbol a in A such that

1) the last L(n) symbols in w_n are a,

2) w_n followed by any sequence of additional a's is not in W;

then if W is Q(n)-tape computable on an off-line push-down machine M, there exists a constant c such that $Q(n) \geq c \cdot L(n)$.

Proof: There are more details to the proof than space permits us to give, but the basic idea is as follows. Letting q be the number of states of the control of M and k the number of working tape symbols, it can be shown that, as M processes w_n and scans the section of the input tape containing the L(n) a's, the machine cannot go more than $q \cdot k^{qk} Q(n)$ operations without its behavior becoming "periodic" with period less than or equal to $q \cdot k^{qk}$. This behavior could be either periodically repeating a state-working tape memory configuration where the length of the tape is the same after each period, or periodically increasing where the number of tape symbols is longer after each period. If the machine always became periodically repeating as it crossed the a's, it could not distinguish between w_n and w_n followed by $(q \cdot k^{qk})!$ a's because these extra a's contain the period. The only alternatives are that either M not become periodic in which case

$$q\, k^{qk} Q(n) \geq L(n)$$

or M becomes periodically increasing. But if M becomes periodically increasing we can bound the number of operations since the tape length cannot exceed Q(n). After at most $q \cdot k^{qk} Q(n)$ operations, the periodic increase must begin; and thence, after at most every $q \cdot k^{qk}$ operations the length of the tape must increase by at least 1. Since there can be at most Q(n) increases, the number of operations is at most

$$q \cdot k^{qk} \cdot Q(n) + q \cdot k^{qk} \cdot Q(n)$$

$$= 2 \cdot q \cdot k^{qk} \cdot Q(n)$$

But this number of operations must be at least enough to scan the L(n) a's at least once, and so we must have

$$2 \cdot q \cdot k^{qk} \cdot Q(n) \geq L(n)$$

Thus in either case, the lemma is seen to be true if we choose $1/c = 2 \cdot q \cdot k^{qk}$.

We say that a tape function L(n) is constructable on an off-line push-down machine if there is a set W which satisfies the lemma and is L(n)-tape recognizable on an off-line push-down machine. Stated less formally, L(n) is constructable if it is possible for some input of length n to recognize that exactly L(n) ones appear at the end of the word and if the recognition can be done within tape L(n). Although these constructable functions have not been investigated in much detail, we know that they contain the various rational powers of n, log n, and log log n that lie between n and log log n and so a fairly rich structure is assured. We now carry the structure over to the complexity hierarchy.

Theorem 8: For off-line push-down machines, if L(n) is a constructable function, then there exists a set W which is L(n)-tape computable and is not Q(n)-tape computable for all tape functions Q(n) such that

$$\inf_{n\to\infty} \frac{Q(n)}{L(n)} = 0$$

Proof: We choose W to be the set that makes L(n) constructable and if W is also Q(n)-tape computable, the lemma insures us that $Q(n) \geq c \cdot L(n)$ for some c and so $\inf_{n\to\infty} \frac{Q(n)}{L(n)} \geq c$.

SUMMARY

We have seen that bounding the tape required to recognize a set on any of the four machine types can be used as a complexity measure for the recognition problem. For three of the models, this leads to a rich hierarchy of tape complexity classes, each representing a different degree of difficulty. The structure of these hierarchies may be summarized as follows:

1) For each model, the "simplest" complexity class is C_{constant} which is the class of regular sets.

2) Each machine type has a lower bound $\mathcal{L}(n)$ which bounds the non-regular complexity classes in the sense that

a) $\lim \frac{L(n)}{\mathcal{L}(n)} = 0$ implies that C_L contains only regular sets

and

b) $C_{L(n)} \supseteq C_{\mathcal{L}(n)}$ for all monotone L(n) such that C_L contains non-regular sets.

The function $\mathcal{L}(n)$ for each type is:

off-line Turing machines	log log n
off-line push-down machines	log log n
on-line Turing machines	log n
on-line push-down machines	n

3) For the Turing machine cases, there is an infinite hierarchy of classes above $\mathcal{L}(n)$ in which the slightest increase in the limiting behavior of one tape function gives a new tape function that defines a new complexity class (whenever the new function is constructable).

4) For the off-line push-down machines, there is a maximal complexity class, namely $C_{L(n)=n}$, which contains all sets that can be recognized by an off-line push-down machine. Between the functions L(n) = n and $\mathcal{L}(n)$ = log log n, suitable slight increases in limiting behavior again defines a new class.

5) For the on-line push-down machine, the hierarchy consists of two sets, $C_{L(n)=n}$ and C_{constant}.

REFERENCES

1. J. Hartmanis and R.E. Stearns, "On the Computational Complexity of Algorithms," Transactions of the American Mathematical Society, 117 (May 1965) pp. 285-306.

2. J. Hartmanis and R.E. Stearns, "Computational Complexity of Recursive Sequences," Proc. Fifth Annual Sympos. on Switching Theory and Logical Design, Princeton, N.J. 1964.

3. F.C. Hennie and R.E. Stearns, "Two-tape Simulation of Multi-tape Turing Machines," General Electric Report 65-RL-4020E.

4. P.M. Lewis II, R.E. Stearns, and J. Hartmanis, "Memory Bounds for the Recognition of Context Free and Context Sensitive Languages," these Proceedings.

5. H. Yamada, "Real-time Computation and Recursive Functions not Real-time Computable," IRE Trans. EC-11 (1962) 753-760.

6. A.M. Turing, "On Computable Numbers, with Applications to the Entscheidungs Problem," Proc. London Math. Soc. (22) 42 1937, 230-265.

7. J. Myhill, "Linear Bounded Automata," WADD Tech. Note 60-165, Rep. No. 60-22, Univ. of Pennsylvania, June 1960.

8. R.W. Ritchie, "Classes of Predictably Computable Functions," Trans. Amer. Math. Soc. 106 (1963) 139-173.

9. J. Hartmanis, P.M. Lewis II and R.E. Stearns, "Classifications of Computations by Time and Memory Requirements," Proceedings of IFIP Congress 65, Vol. 1, pp. 31-35.

<u>Errata For Paper on Next Page (Ref. 4)</u>

The languages L_4, L_a, L_b, L_c, L_d, L_f, and L_g are incorrectly stated. They should be

$$L_4 = \overrightarrow{b_1}\ s\ \left(\overleftarrow{b_j}\ s\ \overrightarrow{b_{j+1}}\ s\right)^*\ (0 + 1)^*\ s$$

$$L_a = \overrightarrow{b_1^\alpha}\ s\ \left(\overleftarrow{b_j^\alpha}\ s\ \overrightarrow{b_{j+1}^\alpha}\ s\right)^*\ (0 + 1 + a + b)^*\ ss\ (1 + s)^*$$

$$L_b = \left(\overrightarrow{b_k^\alpha}\ s\ \overleftarrow{b_{k+1}^\alpha}\ s\right)^*\ ss\ (1 + s)^*$$

$$L_c = (0 + 1 + s + a + b)^*\ (0 + 1 + s)\ (a + b)^q\ ss\ 1^q\ s\ (1^\ell\ s\ 1^{\ell-1}\ s)^*\ 1^*\ sss\ (1 + s)^*$$

$$L_d = (0 + 1 + s + a + b)^*\ (a + b)\ s\ (1^p\ s\ 1^{p-1}\ s)^*\ ss\ (1 + s)^*$$

$$L_f = (0 + 1 + s + a + b)^*\ sss\ 1\ s\ (1^t\ s\ 1^{2t}\ s)^*\ 1^*\ s$$

$$L_g = (0 + 1 + s + a + b)^*\ sss\ (1^v\ s\ 1^{2v}\ s)^*$$