INFORMATION AND CONTROL **13**, 114–133 (1968)

Sets Accepted by One-Way Stack Automata Are Context Sensitive

J. E. HOPCROFT* AND J. D. ULLMAN

Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey 07971

A stack automaton is a pushdown automaton that can read the interior of its pushdown list without altering it.

It is shown that a nondeterministic stack automaton with a one-way input tape can be simulated by a deterministic linear bounded automaton. Hence, each nondeterministic one-way stack language is context sensitive.

I. INTRODUCTION

A one-way stack automaton ([*1*], [*2*]) (here abbreviated as sa) can be thought of as a type of restricted nondeterministic Turing machine. The sa has the power to write or erase only at the right end of the non-blank portion of its storage tape. The storage tape is called a *stack*, and the blank immediately to the right of the rightmost nonblank symbol is called the *top of stack.*

The sa also has the power to move its stack head (storage tape head) into the nonblank portion of the stack in a read only mode. On each move of the sa, the input head either moves right or remains stationary.

We will now introduce a formal notation for the sa. The move of the machine is a function of the current state of the finite control, the symbol scanned by the stack head (and, if that symbol is blank, the rightmost nonblank) and, possibly, an input symbol. If the next move depends on the input symbol, then the input head must move right. An alternative visualization is that instead of an input tape, the sa has an input terminal, at which input symbols appear when requested by the sa, whereupon they are immediately used in determining the next move.

A one-way stack automaton is an 8-tuple $(K, T, I, \delta, \delta_b, q_0, Z_0, F)$ with the following meanings:

K is the finite set of *states*.

* Currently at Cornell University, Ithaca, New York.

T is the finite set of nonblank *stack symbols.*
I is the finite set of *input symbols.*
q_0, in K, is the *start state.*
Z_0, in T, is the *start symbol.*
F, a subset of K, is the set of *final states.*

δ is the next move mapping for the case where the stack head is not at the top of the stack. δ maps $K \times (I \cup \{\epsilon\}) \times T$ to subsets of $K \times \{-1, 0, +1\}$. ϵ represents the empty word and is here used to denote the situation where the input does not affect the next move. The significance of -1, 0 and $+1$ in the range of δ is that the stack head moves left, does not move, or moves right, respectively.

δ_b is the next move mapping for the case where the stack head is at the top of the stack. Here, the rightmost nonblank on the stack will affect the next move, so δ_b maps $K \times (I \cup \{\epsilon\}) \times T$ to subsets of $K \times [T \cup \{-1, 0, E\}]$. We assume that -1, 0 and E are not in T. Here, 0 in the range of δ_b signifies that the stack head does not move, -1 signifies that the stack head moves left without erasing the rightmost nonblank. E signifies that the stack head moves left and prints a blank over the rightmost nonblank. A nonblank stack symbol in the range of δ_b signifies that that symbol is printed over the blank at the top of the stack, the stack head then moving right.

Let $S = (K, T, I, \delta, \delta_b, q_0, Z_0, F)$ be an sa. A *move* of S is a single application of a rule of δ or δ_b, whichever is appropriate. If the second argument of δ or δ_b is ϵ, the move is an ϵ *input move.* Otherwise, it is a *non-ϵ input move.* A *configuration* of S is a combination of state, nonblank portion of the stack and position of the stack head. The configuration of S will be denoted (q, y, i), where q is the state, y is the nonblank portion of the stack, and i the number of cells from the top of the stack to the cell scanned by the stack head. That is, if S scans the blank at the top of the stack, $i = 0$. If S scans the rightmost nonblank, $i = 1$, etc.

The symbol $\underset{S}{\vdash}$ relates two configurations if the second can be derived from the first by a single move of S.

Formally, if x and y are in T^*,[1] Z in T, p and q in K, a in $I \cup \{\epsilon\}$,

[1] T^* denotes the set of finite length strings of symbols in T, including ϵ, the empty string.

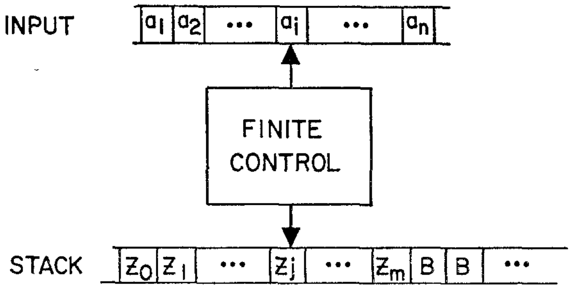
INPUT
a_1 a_2 ⋯ a_i ⋯ a_n
FINITE CONTROL
STACK
Z_0 Z_1 ⋯ Z_j ⋯ Z_m B B ⋯

FIG. 1.

$|y|^2 = i - 1 \geqq 0$, and $\delta(q, a, Z)$ contains (p, d) then we say a: $(q, xZy, i) \underset{S}{\vdash} (p, xZy, i - d)$. Let x be in T^*, Y and Z in T, p and q in K and a in $I \cup \{\epsilon\}$. If $\delta_b(q, a, Z)$ contains (p, d), $d = -1$ or 0, then we say a: $(q, xZ, 0) \underset{S}{\vdash} (p, xZ, -d)$. If $\delta_b(q, a, Z)$ contains (p, E), then we say a: $(q, xZ, 0) \underset{S}{\vdash} (p, x, 0)$. If $\delta_b(q, a, Z)$ contains (p, Y), then a: $(q, xZ, 0) \underset{S}{\vdash} (p, xZY, 0)$.

We define the use of the symbol $\overset{*}{\underset{S}{\vdash}}$ recursively by:

(1) ϵ: $Q \overset{*}{\underset{S}{\vdash}} Q$, for each configuration Q.

(2) If, for w in I^*, a in $I \cup \{\epsilon\}$ and configurations Q_1, Q_2 and Q_3, we have w: $Q_1 \overset{*}{\underset{S}{\vdash}} Q_2$ and a: $Q_2 \underset{S}{\vdash} Q_3$, then wa: $Q_1 \overset{*}{\underset{S}{\vdash}} Q_3$.

Let $Q_j = (q_j, y_j, i_j)$, $0 \leqq j \leqq m$, be configurations of S and suppose that for $1 \leqq j \leqq m$, $a_j : Q_{j-1} \underset{S}{\vdash} Q_j$. Also, let r and s be integers or ∞ and suppose that for $0 < j < m$, we have $r \geqq i_j \geqq s$. Then we say $a_1a_2 \cdots a_m : Q_0 \overset{(r,s)}{\underset{S}{\vdash}} Q_m$. In other words, Q_0 is transformed into Q_m by a sequence of configurations such that for every configuration except possibly Q_0 and Q_m, the stack head is between r and s symbols below the top of stack.

[2] $|y|$ stands for the length of y.

If, as above, we have $m \geqq 2$, $Q_0 \left|\!\frac{(\infty, 1)}{}\right. Q_m$, and $i_0 = i_m = 0$, then we write $a_1a_2 \cdots a_m : Q_0 \left|\!\frac{ss}{S}\right. Q_m$, and say that S performs a *stack scan* with input $a_1a_2 \cdots a_m$.

That is, a stack scan is a sequence of configurations beginning and ending with the stack head at the top of stack, such that the stack head is not at the top of stack in any intermediate configuration. There must be at least one intermediate configuration.

The subscript S will be dropped from $\left|\!\frac{}{S}\right.$, $\left|\!\frac{*}{S}\right.$, $\left|\!\frac{(r, s)}{S}\right.$, and $\left|\!\frac{ss}{S}\right.$ if no confusion arises.

The *initial configuration* of S is $(q_0, Z_0, 0)$. The *language accepted by S by final state*, denoted $\mathfrak{I}(S)$, is the set of words w such that w: $(q_0, Z_0, 0) \overset{*}{\vdash} (q, y, i)$ for some q in F, y in T^* and integer i. The *language accepted by S by empty stack*, denoted $\mathfrak{N}(S)$, is the set of words w such that w: $(q_0, Z_0, 0) \overset{*}{\vdash} (q, \epsilon, 0)$, for some q in K.

II. BASIC LEMMAS

We begin by giving an informal proof of the equivalence of acceptance by final state and empty stack. The proof is a simple generalization of the analogous result for pushdown automata, as given in [3].

LEMMA 1. $L = \mathfrak{I}(S_1)$ *for some* sa S_1 *if and only if* $L = \mathfrak{N}(S_2)$ *for some* sa S_2.

Proof. To construct S_2 from S_1, one can specify an sa which first prints a new symbol X on its stack, prints the start symbol of S_1, then simulates S_1. If S_1 enters a final state, S_2 moves its stack head to the top of stack and erases its stack. The symbol X ensures that S_2 will not "accidentally" erase its stack.

Given S_2, we can construct an sa S_1 as follows. S_1 prints a new symbol Y on its stack, prints the start symbol of S_2 and proceeds to simulate S_2. If S_1's stack head is ever subsequently at the top of stack and finds Y to be the right-most nonblank stack symbol, S_1 enters an accepting state.

From this point on, we shall consider only the languages accepted by empty stack. Lemma 1 provides the justification for so doing. We shall next prove a property of an sa that may be assumed without loss of generality. This property will be called property A.

Property A. For no p, q and r in K, y in T^* and w in I^* does ϵ: $(q, y, 0) \overset{ss}{\vdash} (p, y, 0)$ and w: $(p, y, 0) \overset{*}{\vdash} (r, \epsilon, 0)$. (As a consequence, no stack scans may be performed on ϵ input in a computation of S leading to acceptance of some word by empty stack.)

LEMMA 2. *If L is accepted by an* sa, *then $L = \mathfrak{N}(S)$, for some* sa *having property A.*

Proof. Let $L = \mathfrak{N}(S')$, where $S' = (K', T', I, \delta', \delta_b', q_0, Z_0', \varphi)$.[3] Let $\mathfrak{A}$ be the set of subsets of pairs of states of S'. That is, $\mathfrak{A} = 2^{K' \times K'}$. To each string y in $(T')^*$ we can *associate* α_y, an element of $\mathfrak{A}$, having the property that $[q, p]$ is in α_y if and only if ϵ: $(q, y, 1) \overset{(\infty, 1)}{\underset{S'}{\vdash}} (p, y, 0)$ That is, with ϵ inputs only, configuration $(q, y, 1)$ can go to configuration $(p, y, 0)$ by a sequence of configurations in which the stack head remains below the top of stack until the last move.

Observe that for any Z in T, α_{yZ}, the element of $\mathfrak{A}$ associated with yZ, is determined by α_y and δ'.[4] The value of y is unimportant; α_y tells us what we need to know.

S will simulate S'. If $Z_1Z_2 \cdots Z_k$ is the stack of S', then the stack of S will be $[Z_1, \alpha^{(1)}][Z_2, \alpha^{(2)}] \cdots [Z_k, \alpha^{(k)}]$, where for each i, $1 \leqq i \leqq k$, $\alpha^{(i)} = \alpha_{Z_1Z_2\ldots Z_i}$. Suppose that in the simulation of S', S finds symbol $[Z, \alpha]$ to be the rightmost nonblank on its stack, while S's stack head is at the top of stack, and S has determined that S' is in state q_1. Also, suppose that $\delta_b'(q_1, \epsilon, Z)$ contains $(q_2, -1)$ and α contains $[q_2, q_3]$. Then S may, with ϵ input, go from state q_1 to q_3 in one move, leaving its stack head fixed. Thus, no stack scans with ϵ input are needed by S. In addition, when the stack head of S leaves the top of stack, the finite control of S keeps track of whether a non-ϵ input is used before the stack head returns to the top of stack. If not, no further moves can be made by S upon reaching the top of stack.

A formal construction of $S = (K, T, I, \delta, \delta_b, q_0, Z_0, \varphi)$ follows. $K = K' \cup \{\bar{q} \mid q \text{ is in } K'\}$. $T = T' \times \mathfrak{A}$. $Z_0 = [Z_0', \alpha_{Z_0'}]$. For all q, p and r in K', Y and Z in T', α in $\mathfrak{A}$ and a in I, δ and δ_b are defined as follows.

(1) If $(q, 0)$, (q, E) or (q, Y) is in $\delta_b'(p, a, Z)$, then $(q, 0)$, (q, E)

[3] φ denotes the empty set.

[4] This computation is effective. We shall not so prove, since effectiveness of the construction is not crucial to the theoretical result.

or $(q, [Y, \beta])$, respectively, is in $\delta_b(p, a, [Z, \alpha])$. β is that element of $\mathfrak{A}$ such that if α is associated with some y in T^*, then β is associated with yY. The above holds also if a is replaced by ϵ. (S simulates S' while S' remains at the top of the stack. If S' prints a symbol, S also prints the element of $\mathfrak{A}$ associated with the new stack contents.)

(2) If $(q, -1)$ is in $\delta_b'(p, a, Z)$, then $\delta_b(p, a, [Z, \alpha])$ contains $(q, -1)$. If $(q, -1)$ is in $\delta_b'(p, \epsilon, Z)$, then $\delta_b(p, \epsilon, [Z, \alpha])$ contains $(\bar{q}, -1)$. (S also simulates the moves of S' when S' leaves the top of stack. If S' does so on ϵ input, S enters a barred state.)

(3) $\delta(q, a, [Z, \alpha]) = \delta(\bar{q}, a, [Z, \alpha]) = \delta'(q, a, Z)$; $\delta(q, \epsilon, [Z, \alpha]) = \delta'(q, \epsilon, Z)$; $\delta(\bar{q}, \epsilon, [Z, \alpha]) = \{(\bar{p}, d) \mid (p, d) \text{ is in } \delta'(q, \epsilon, Z)\}$. ($S$ simulates S' below the top of stack. The bar is removed from the state if and only if S makes a move on a non-ϵ input.)

(4) If $\delta_b'(q, \epsilon, Z)$ contains $(p, -1)$ and α contains $[p, r]$, then $\delta_b(q, \epsilon, [Z, \alpha])$ contains $(r, 0)$. (S simulates stack scans of S' performed on ϵ input by making a single move at the top of stack.)

(5) No other moves are allowed to S. A proof that $\mathfrak{N}(S) = \mathfrak{N}(S')$ is straightforward.

III. SUMMARIES OF STACKS

Given an sa S, we wish to find an equivalent sa S', which accepts each word of length n in $\mathfrak{N}(S)$ by a sequence of moves in which the stack does not grow longer than cn, for some constant c. An sa having that property can be simulated by a nondeterministic linear bounded automaton (lba) ([*4*,] [*5*], [*6*]), and hence $\mathfrak{N}(S)$ is context sensitive. (To show that S' can be simulated by a deterministic lba is more difficult, but a proof can be given after we have the desired S'.)

We shall show that if S has property A, and the stack of S grows "too long," then certain substrings of that stack can be replaced by a single symbol on the stack of S'. These single symbols will hold all the information S' needs to simulate S in this sequence of moves.

Let S be an sa, P be a sequence of moves of S leading to an empty stack and $Z_1Z_2 \cdots Z_k$ be S's stack contents at some time during the execution of the moves in sequence P. We say the symbol Z_i, $1 \leqq i \leqq k$ is *useful in sequence* P if either:

(1) Z_i is scanned by S's stack head when a move on a non-ϵ input is made in sequence P, or

(2) Z_i is the rightmost nonblank on the stack when S's stack head is at the top of stack and a move on a non-ϵ input is made in sequence P.

We say the symbol Z_i, $2 \leqq i \leqq k$, is a *break point in sequence P* if Z_i is erased on ϵ-input, but Z_{i-1} is not erased in sequence P before a subsequent move on a non-ϵ input is made.

Let $S = (K, T, I, \delta, \delta_b, q_0, Z_0, F)$ be an sa and $y = Z_1Z_2 \cdots Z_k$ be a string of stack symbols of S. We say the symbol $[Z, A_1, A_2, A_3, A_4, A_5]$ is a *transmission table* (abbreviated t.t.) *for y* if:

(1) $Z = Z_k$

(2) A_1, A_2, A_3, A_4, and A_5 are subsets of $K \times K$.

(3) (q, p) is in A_1 if and only if
$$\epsilon\colon (q, Z_0Z_1Z_2 \cdots Z_k, 0) \,\Big|\!\underline{\;(0,0)\;}\, (p, Z_0, 0).$$

(4) (q, p) is in A_2 if and only if
$$\epsilon\colon (q, Z_0Z_1Z_2 \cdots Z_k, 1) \,\Big|\!\underline{\;(k,1)\;}\, (p, Z_0Z_1Z_2 \cdots Z_k, k+1).$$

(5) (q, p) is in A_3 if and only if
$$\epsilon\colon (q, Z_0Z_1Z_2 \cdots Z_k, 1) \,\Big|\!\underline{\;(k,1)\;}\, (p, Z_0Z_1Z_2 \cdots Z_k, 0).$$

(6) (q, p) is in A_4 if and only if
$$\epsilon\colon (q, Z_0Z_1Z_2 \cdots Z_k, k) \,\Big|\!\underline{\;(k,1)\;}\, (p, Z_0Z_1Z_2 \cdots Z_k, k+1).$$

(7) (q, p) is in A_5 if and only if
$$\epsilon\colon (q, Z_0Z_1Z_2 \cdots Z_k, k) \,\Big|\!\underline{\;(k,1)\;}\, (p, Z_0Z_1Z_2 \cdots Z_k, 0).$$

Informally, the t.t. for y tells what state transitions S can make while erasing y on ϵ inputs (Condition 3). It also tells what state transitions S can make, starting at one end of y and moving off of y for the first time (Conditions 4–7). Finally, the t.t. indicates what the rightmost symbol of y is (Condition 1).

Let y be in T^* and let α be the t.t. for y. If Z is in T, then the t.t. for yZ depends only on α, Z, δ and δ_b.[5] It does not depend on y. A proof of this fact is straightforward, and we omit it.

Define B to be the set of t.t.'s for strings in T^*. If y is in T^* and u in $(B \cup T)^*$, then we say u is a *summary* of y if we can write $y = y_1y_2 \cdots y_k$ and $u = X_1X_2 \cdots X_k$, with each X_i, $1 \leqq i \leqq k$, in $B \cup T$, in such a manner that for all i, either:

(1) $y_i = X_i$ or,

(2) X_i is a t.t. for y_i.

[5] Also, note that the t.t. for yZ can be found effectively.

Note that y_i may have any length, including one, in case (2). We say that y_i and X_i *correspond* in either case.

Let w be in I^* and P be a sequence of moves of S, with input w, leading to an empty stack. If y and u are as above, we say u is a *P-minimal summary* of y if:

(1) u is a summary of y.

(2) For all $i > 1$, if X_i is in T, then y_i is useful in sequence P. (Note that $y_i = X_i$ in this case.)

(3) For all $i > 2$, if X_i and X_{i-1} are in B, then the leftmost symbol of y_i is a break point in sequence P.

Informally, a P-minimal summary combines adjacent symbols into t.t.'s unless in sequence P, one of the symbols affects a move on non-ϵ input, or a move on non-ϵ input occurs between the moves erasing the two symbols. The bottom symbol on the stack, however, is never combined into a t.t.

We shall now describe a construction of an sa S' which simulates S by nondeterministically combining strings of stack symbols of S into t.t.'s or not combining them. S', being nondeterministic, will make all possible choices as to whether or not adjacent symbols are to be combined or not combined into t.t.'s.

Formally, let $S = (K, T, I, \delta, \delta_b, q_0, Z_0, \varphi)$ be an sa, and let B be the set of t.t.'s for strings in T^*. We say $S' = (K', T', I, \delta', \delta_b', q_0, Z_0, \varphi)$ is a *summary machine* for S if:

(1) $K' = K \cup \{\bar{q} \mid q \text{ is in } K\} \cup \{[q, \beta] \mid q \text{ is in } K \text{ and } \beta \text{ in } B\}$.

(2) $T' = T \times B$.

(3) δ' and δ_b' are defined as follows, for all q and p in K, a in I, Y and Z in T and α in B:

(a) If $\delta(q, a, Z)$ contains $(p, +1)$, then $\delta'(q, a, Z)$ and $\delta'(\bar{q}, a, Z)$ each contain $(\bar{p}, +1)$. If $\delta(q, a, Z)$ contains $(p, -1)$, then $\delta'(q, a, Z)$ and $\delta'(\bar{q}, a, Z)$ each contain $(p, -1)$. If $\delta(q, a, Z)$ contains $(p, 0)$ then $(p, 0)$ is in $\delta'(q, a, Z)$ and $(\bar{p}, 0)$ is in $\delta'(\bar{q}, a, Z)$. The above statements hold if a is replaced by ϵ. (If S' is within the stack and scanning a symbol of T, S' simulates S. The barred states indicate that the last move of the stack head was right, the unbarred states indicate that the last move was left.)

(b) Let $\alpha = [Z, A_1, A_2, A_3, A_4, A_5]$. If (q, p) is in A_2, then $\delta'(q, \epsilon, \alpha)$ contains $(p, -1)$. If (q, p) is in A_3, then $\delta'(q, \epsilon, \alpha)$ contains $(\bar{p}, +1)$. If (q, p) is in A_4, then $\delta'(\bar{q}, \epsilon, \alpha)$ contains $(p, -1)$. If (q, p) is in A_5, then $\delta'(\bar{q}, \epsilon, \alpha)$ contains $(\bar{p}, +1)$. (S' simulates the state transitions that S could make while moving through one of the strings represented by a t.t.)

(c) $\delta_b'(\bar{q}, \epsilon, Z) = \{(q, 0)\}$. (Upon reaching the top of the stack, S' goes to an unbarred state.)

(d) If $\delta_b(q, a, Z)$ contains $(p, -1)$, $(p, 0)$, (p, E) or (p, Y), then $\delta_b'(q, a, Z)$ contains the same element or elements. The above is also true if a is replaced by ϵ. (S' simulates moves of S at the top of stack when the rightmost nonblank is in T.)

(e) If $\delta_b(q, \epsilon, Z)$ contains (p, Y), then $\delta_b'(q, \epsilon, Z)$ contains (p, β), where β is the t.t. for the string consisting of Y alone. (If S prints a symbol on ϵ input, S' can also print a t.t. for that symbol.)

(f) Let $\alpha = [Z, A_1, A_2, A_3, A_4, A_5]$. If $\delta_b(q, \epsilon, Z)$ contains $(p, -1)$, $(p, 0)$ or (p, Y), then $\delta_b'(q, \epsilon, \alpha)$ contains the same element or elements. In addition, if $\delta_b(q, \epsilon, Z)$ contains (p, Y), then $\delta_b'(q, \epsilon, \alpha)$ contains $([p, \beta], E)$ and for all X in T', $\delta'([p, \beta], \epsilon, X) = \{(p, \beta)\}$. β is the t.t. for any yY such that α is the t.t. for y. $\delta_b'(q, \epsilon, \alpha)$ also contains (p, γ), where γ is a t.t. for Y alone. (With a t.t. as the rightmost nonblank, S' can simulate the ϵ input moves of S, with the exception of an erase, which will be considered next. If S prints a symbol, S' has the option of incorporating that symbol into the t.t. that was the rightmost nonblank. Note that it is impossible for the leftmost stack symbol to be a t.t., so so that when α is erased by this rule, there will always be a symbol below it. S' may instead print a t.t. for the single symbol printed.)

(g) Let $\alpha = [Z, A_1, A_2, A_3, A_4, A_5]$. If A_1 contains (q, p), then $\delta_b'(q, \epsilon, \alpha)$ contains (p, E). (S' can erase a t.t., making any state transition that the t.t. says is possible if S erased the string it represents.)

If S' accepts some input w by empty stack, then a sequence of moves of S leading to acceptance of w can be constructed in a straightforward manner. Thus, $\mathfrak{N}(S') \subseteq \mathfrak{N}(S)$. It is also true that $\mathfrak{N}(S') \subseteq \mathfrak{N}(S)$, and we shall prove more in the following lemma. In the next two lemmas, S, its summary machine S' and all notation are as in the above definitions. We shall hereafter assume that S has property A.

LEMMA 3. *If P is a sequence of moves of S leading to acceptance of w by empty stack, then S' accepts w by a sequence of moves such that the stack of S' in every configuration with a state q or $\bar{q}$, q in K, is a P-minimal summary of the stack of S in some configuration which S enters during sequence of moves P.*

Proof. Let $Q_0, Q_1, \cdots, Q_m$ be the sequence of configurations that S

enters when executing sequence of moves P. Let $Q_i = (q_i, y_i, j_i)$, $0 \leqq i \leqq m$. Certainly, $Q_0 = (q_0, Z_0, 0)$ and $Q_m = (q_m, \epsilon, 0)$ for some q_m in K.

When simulating sequence of moves P, S' will not necessarily enter a configuration corresponding to every configuration of S. However, we shall show that if S' enters a configuration corresponding to Q_i, then S' can subsequently enter a configuration corresponding to some Q_k, $k > i$.

Let i be some integer between 0 and $m - 1$. Suppose that S' is in configuration (q, u, l), where:

(1) u is a P-minimal summary of y_i.

(2) If $j_i = 0$, then $l = 0$ and $q = q_i$.

(3) Let $y_i = Y_r Y_{r-1} \cdots Y_1$ and $u = U_s U_{s-1} \cdots U_1$. Then if $j_i > 0$, U_l corresponds to some substring z of y_i, which includes Y_{j_i}. Moreover, if U_l is a t.t., then in configuration Q_{i-1} the stack head of S was not scanning a symbol of z. Finally, $q = q_i$ or $\bar{q}_i$, depending on whether S last moved its stack head left or right.

We will show that there is some $k > i$ and a in $I \cup \{\epsilon\}$ such that according to sequence of moves P, $a\colon Q_i \overset{*}{\vdash} Q_k$, and for some configuration (q', u', l') of S', either:

(i) $a\colon (q, u, l) \vdash_{S'} (q', u', l')$.

(ii) $a\colon (q, u, l) \vdash_{S'} (\bar{q}', u', l')$ and $\epsilon\colon (\bar{q}', u', l') \vdash_{S'} (q', u', l')$.

(iii) $a\colon (q, u, l) \vdash_{S'} ([q', \beta], u'', l'')$ and

$\epsilon\colon ([q', \beta], u'', l'') \vdash_{S'} (q', u', l')$.

Here, β is some element of B.

In addition, we will see that Q_k and (q', u', l') are related as Q_i and and (q, u, l) are related according to statements (1), (2) and (3) above. Since the initial configurations of S and S' ($(q_0, Z_0, 0)$ in both cases) are related according to (1), (2) and (3), when we show how to find (q', u', l') from (q, u, l), we will have a complete construction for the desired sequence of moves of S'. The construction of (q', u', l') will be given by cases.

Case 1: $l > 0$ *and* U_l *is in* T.

By rule (a) of the definition of S', the next move of S is directly simulated by S'. $k = i + 1$ in this case. If $l = 1$ and S moves its stack head right, S' then removes the bar from its state (rule (c)), so condition (ii) applies. Otherwise, condition (i) applies. In either event, $q' = q_k$ or $\bar{q}_k$, whichever is correct according to (3); $u' = u$ and $l' = l + d$, where $d = j_k - j_i$.

Case 2: $l > 0$ *and* U_l *is in* B.

Since P is a sequence of moves leading to empty stack, the stack head of S must eventually leave the string of stack symbols corresponding to U_l, and since u is a P-minimal summary of y, S will make only ϵ input moves in so doing. Suppose that this happens when S enters configuration Q_k, for some $k > i$. By rule (b), S' can move its stack head in the same direction as S and enter state q_k or $\bar{q}_k$, whichever is correct according to (3). As in Case 1, if S reaches the top of stack, the bar is removed from the state of S'. Thus $q' = q_k$ or q_k, $u' = u$ and $l' = l - 1$ or $l + 1$, depending on the direction of motion of S's stack head.

Case 3: $l = 0$ *and* U_1 *is in* T.

By rule (d), S' can directly simulate the next move of S, and if S prints a symbol, rule (e) allows S' the option of printing a t.t. for that symbol. In this case, $k = i + 1$. It is trivial to find (q', u', l'), except in the event that a symbol is printed by S. If S prints a symbol, u' is either u followed by that symbol or a t.t. for that symbol, depending on which makes u' a P-minimal summary of y_k.

Case 4: $l = 0$ *and* U_1 *is in* B.

If the next move of S causes the stack head to remain fixed or move left, S' can simulate this move directly. Here, $k = i + 1$ and the value (q', u', l') is obvious.

If S next prints a symbol, S' can either (rule (f)) print that symbol, print a t.t. for that symbol or change U_1 to account for the additional symbol. Again, let $k = i + 1$. One of these choices must lead to a stack string u' which is a P-minimal summary of y_k. $l' = 0$ and $q' = q_k$. If symbol U_1 is changed, condition (iii) applies. Otherwise, condition (i) applies.

Finally, if S next erases a symbol, it must be done on ϵ input. Also, before making a move on non-ϵ input in sequence P, S must erase its entire stack string corresponding to U_1. (Note that S may print symbols during this sequence of moves. However, by property A, S's stack head does not leave the top of stack.) Let Q_k be the configuration reached immediately after S erases the stack string corresponding to U_1. Since S's stack head does not leave the top of stack between configurations Q_i and Q_k, it must be (rule (g)) that $\delta_b'(q_i, \epsilon, U_1)$ contains (q_k, E). Thus $q' = q_k$, $u' = U_s U_{s-1} \cdots U_2$ and $l' = 0$.

We have now completed construction of the desired sequence of moves of S', and the lemma is proven.

LEMMA 4. *If w of length n is in $\mathfrak{N}(S)$, then S' accepts w by a sequence of moves in which the stack never grows longer than $3n + 2$ symbols.*

Proof. The sequence of moves of S' constructed in Lemma 3 is the desired sequence. Let this sequence be P'. Observe that in sequence P', every configuration of S' reached either has a stack which is a P-minimal summary of a stack of S or has a state which is of the form $[q, \beta]$, q in K, β in B. The latter configurations have a stack which is shorter than the stack of the next configuration reached by S' in sequence of moves P'. Thus, it suffices to show that a P-minimal summary of a stack of S cannot be longer than $3n + 2$ symbols.

Let u in T'^* be a P-minimal summary of y in T^*. The symbols of u can be grouped into four nondisjoint categories.

(1) The leftmost symbol of u.

(2) A symbol in T which corresponds to a useful symbol of y.

(3) A symbol in B which corresponds to a substring z of y such that the leftmost symbol of z is a break point.

(4) A symbol in B which is immediately to the right of a symbol in T.

Every symbol of u falls into at least one of the above categories. The justification for this claim is that by the definition of P-minimal summary, every symbol in T is either the leftmost on the stack or corresponds to a useful symbol and a symbol of B to the right of another symbol of B corresponds to a break point. Clearly, one symbol is in category (1). There cannot be more than n useful symbols, nor more than n break points. Finally, no more than $n + 1$ symbols are in category (4), since the only symbols in T are in categories (1) or (2). Thus, $|u| \leq 3n + 2$.[5]

IV. ENUMERATING SEQUENCES OF MOVES

We have now shown that every sa is equivalent to one which, if it accepts w, accepts it by a sequence of moves in which the stack never grows longer than $3|w| + 2$. At this point, we could directly simulate such an sa by a nondeterministic linear bounded automaton. However, our goal is to show how to simulate an sa with a deterministic lba. We will find it necessary to use an encoding for sequences of moves of an sa. This encoding will have the desirable feature that the sequence of moves of S' constructed in Lemma 3 will have an encoding whose length is proportional to the length of the input to S'.

It will not do to simply encode the elementary moves of S'. With input of length n, S' might make more than n^2 elementary moves. The encoding we choose will use symbols representing the effect of a sequence of moves of S'. We will later see that an lba can determine if S' can make a sequence of moves having the effect indicated by any symbol (or in one case, a string of symbols). Moreover, the sequence of moves P' constructed in Lemma 3 can be represented by a sequence of symbols whose length is proportional to the length of the input.

To begin, we will define five types of sequences of moves. Each sequence of one of these types will be called a *composite move.*

(1) A *true input move* is a move made when the stack head is at the top of stack, the sa either erases, prints or leaves its stack head fixed, and a non-ϵ input is used. In any such situation, an sa has a finite number of choices of moves, and these choices can be numbered $1, 2, \cdots$. A true input move will be symbolized by an integer from 1 to r, for some particular r appropriate to the sa being described.

(2) A *stack scan* has been previously defined and is a sequence of moves beginning and ending at the top of stack such that the stack head is not at the top of stack in any intermediate configuration. A stack scan will be symbolized by the string $(q, p)\ 0^i$,[6] $i \geqq 1$, where q and p are the states at the beginning and end of the stack scan and i the number of moves made on a non-ϵ input.

(3) An *ϵ-erase move* is the erasure of a single symbol from the top of stack on ϵ input. It is symbolized by $E(q, p)$, where q and p are the states before and after the erasure.

(4) An *ϵ-print move* is the printing of a single symbol on ϵ-input.

[6] 0^i is 0 written i times.

It is symbolized by $G(q, p, Z)$, where q and p are the states before and after printing, and Z is the symbol printed.

(5) An *ϵ-stationary sequence* is a sequence of moves on ϵ input beginning in some configuration $(q, y, 0)$ and ending in configuration $(p, y, 0)$, such that the stack in any intermediate configuration is of the form yy', for some y', and the stack head does not leave the top of stack. This sequence of moves will be symbolized by $N(q, p)$. Informally, an ϵ-stationary sequence is a sequence of printing, erasing and stationary moves, beginning and ending with the same stack, such that no symbol of the initial stack is erased.

Observe that given a configuration of an sa and a symbol for a stack scan or ϵ-stationary sequence, it is possible that more than one valid sequence of moves of the sa is symbolized. However, the existence of an ϵ-stationary sequence symbolized by $N(q, p)$ depends only upon q, p and the rightmost nonblank of the stack. Also, we shall see that it is possible for an lba to decide, for a particular $(q, p)0^i$ whether there is a stack scan using i particular non-ϵ inputs that is symbolized by $(q, p)0^i$.

Observe that from a given configuration of the sa, the application of different sequences of moves symbolized by the same symbol or string of symbols results in the same configuration.

Let us fix our attention on the stack automata S and S' of Lemmas 3 and 4. Let $S' = (K', T', I, \delta', \delta_b', q_0, Z_0, \varphi)$, as before. Let $D_1 = \{(q, p) \mid q \text{ and } p \text{ in } K'\}$, $D_2 = \{E(q, p), G(q, p, Z), N(q, p) \mid q \text{ and } p \text{ in } K', Z \text{ in } T'\}$, and $D_3 = D_2 \cup \{1, 2, \cdots, r\}$, where r is the maximum number of choices of moves of S' in any situation. Let $D = (D_1 00^* \cup D_3)^*$.

If $P_1, P_2, \cdots, P_m$ are sequences of moves of S' symbolized, respectively, by $\alpha_1, \alpha_2, \cdots, \alpha_m$, each α_i in D_3 or $D_1 00^*$, then the sequence of moves composed of $P_1, P_2, \cdots, P_m$ in order is said to be symbolized by the string $\alpha_1 \alpha_2 \cdots \alpha_m$ in D.

LEMMA 5. *Let P' be the sequence of moves of S' constructed in Lemma 3. Then P' is symbolized by a string α in D such that:*

(1) *α does not contain two consecutive symbols representing ϵ-stationary sequences.*

(2) *If α can be written as $\alpha_1 G(q_1, p_1) \alpha_2 E(q_2, p_2) \alpha_3$, then α_2 contains either a symbol representing a true input move or a string of symbols representing a stack scan.*

Proof. (1) is trivial. To show (2), observe that the sequence of moves can be broken into subsequences of moves consisting of either a single true input move, a stack scan or a sequence of ϵ-input moves at the top of stack. It is sufficient to show that any sequence of ϵ-input moves at the top of stack is symbolized by a string in D_2^* such that all ϵ-erase symbols precede all ϵ-print symbols.

To that effect, let $(p_1, y_1, 0)$, $(p_2, y_2, 0)$, $\cdots$, $(p_s, y_s, 0)$ be the sequence of configurations of S' resulting from such a sequence of moves. Let y_j be as short as any y_i, $1 \leqq i \leqq s$, and shorter than any y_i, $1 \leqq i < j$. Then the first $j - 1$ moves in this sequence are surely symbolized by a string of ϵ-erase and ϵ-stationary symbols. Also, it should be clear that the remaining moves can be symbolized by a sequence of ϵ-print and ϵ-stationary symbols.

LEMMA 6. *The sequence of moves P' constructed from sequence of moves P in Lemma 3 is symbolized by a string in D whose length is no longer than $21n + 3$ if the input to S' is of length n.*

Proof. The string α constructed in Lemma 5 has the desired property. To see this, let $\#_0$ be the number of 0's in α, $\#_I$ the number of symbols $1, 2, \cdots, r$, $\#_S$ be the number of symbols in D_1 and $\#_E$, $\#_P$ and $\#_N$ be the number of ϵ-erase, ϵ-print and ϵ-stationary symbols in α, respectively.

Surely $\#_I + \#_S \leqq n$, since each true input move and each stack scan uses at least one input symbol. Similarly, $\#_0 \leqq n$. By (1) of Lemma 5, $\#_N \leqq 1 + \#_I + \#_S + \#_E + \#_P$. Also, since a symbol can be erased only if it is printed or is on the stack initially, $\#_E \leqq \#_P + \#_I + 1$. We will show that $\#_P \leqq 4n$. Thus, $\#_E \leqq 5n + 1$ and $\#_N \leqq 10n + 2$. Hence, $|\alpha| \leqq 21n + 3$.

To show that $\#_P \leqq 4n$, we observe that by (2) of Lemma 5, we can write α as $\gamma_1\alpha_1\beta_1\gamma_2\alpha_2\beta_2 \cdots \gamma_n\alpha_n\beta_n$, where the α's are strings of true input moves and stack scan symbols, the β's are strings of ϵ-erase and ϵ-stationary symbols and the γ's are strings of ϵ-print and ϵ-stationary symbols. Any of these strings may be ϵ.

Now, for all i between 1 and n, let j_i and k_i be the number of symbols in T and B, respectively, printed on the ϵ-print moves symbolized by γ_i. Since every stack of S' during sequence of moves P' is a P-minimal summary of a stack of S, or a prefix thereof, no more than n symbols in T can ever be printed in sequence P'. Therefore, we must have $\sum_{i=1}^{n} j_i \leqq n$.

Every time a symbol of B is printed when another symbol of B is the rightmost stack symbol, the symbol printed represents a break point in

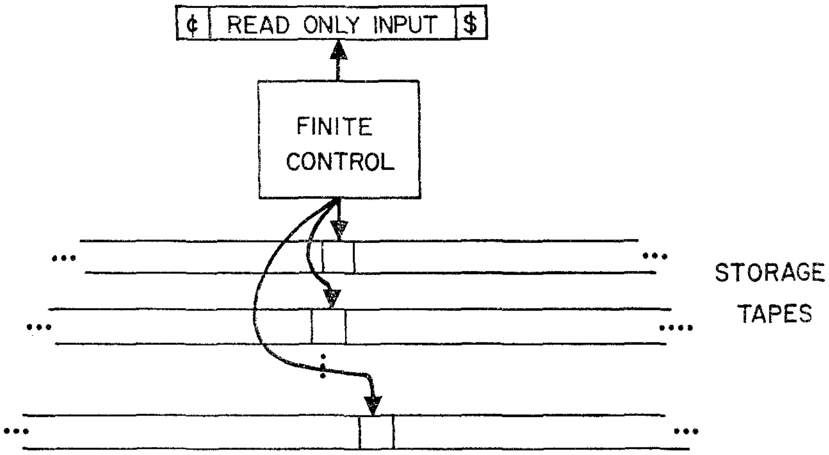

FIG. 2.

sequence P. Thus, among all the γ_i's, there can be at most n instances of two ϵ-print symbols adjacent or separated by a single ϵ-stationary symbol. The number of such instances in γ_i is at least $k_i - j_i - 1$. Thus, $\sum_{i=1}^{n} k_i - j_i - 1 \leqq n$. Since $\sum_{i=1}^{n} j_i \leqq n$, we have $\sum_{i=1}^{n} (k_i - 1) \leqq 2n$. Thus, $\sum_{i=1}^{n} k_i \leqq 3n$ and $\#_P \leqq 4n$.

V. TURING MACHINES OF TAPE COMPLEXITY N

We have so far shown that every sa language L is accepted by an sa S' which accepts by empty stack all words in L by a sequence of moves such that the stack length is proportional to the input length. Also, we have developed an encoding for sequences of moves such that this sequence of moves has an encoding whose length is proportional to the input length.

Now, we shall exhibit a device which generates all possible encodings for sequences of moves that are of length appropriate to the input concerned and tests each one to see if it symbolizes a sequence of moves that S' could make. The device is a variety of multitape off-line Turing machine.

The multitape off-line Turing machine is the device shown in Fig. 2. It consists of a read only input tape with endmarkers, a finite control and k storage tapes for some $k \geqq 1$. Scanning a cell of each tape is a tape head. Based on the state of the finite control and the symbol scanned by each of its tape heads, the device may:

(1) Change the state of the finite control,

(2) Change the symbols on the cells scanned by each of the storage tape heads, and

(3) Move each of its heads (including the input head), independently, one cell left or right or leave it fixed.

Initially, the finite control is in the designated "start state," with the input on the input tape surrounded by endmarkers and all cells of all storage tapes blank. The Turing machine accepts its input if and only if its computation leads it to one of the designated "final states."[7]

A multitape off-line Turing machine M is of *tape complexity* $L(n)$ [*7*, *8*] if for no input of length n does M scan more than $L(n)$ cells on any of its storage tapes. Given a Turing machine of tape complexity $L(n)$ with k storage tapes, it is easy to construct a one storage tape Turing machine accepting the same language and of tape complexity $L(n)$. (Just treat the single storage tape as though it had k tracks, one for each tape of the original device.)

A fundamental result of [*7*] is that given a single tape off-line Turing machine of tape complexity $L(n)$, one can find a device of the same type, accepting the same language, of tape complexity $cL(n)$, for any constant $c > 0$. Moreover, the single tape off-line Turing machine is equivalent [*8*] to the deterministic linear bounded automaton of [*4*], [*5*], [*6*].[8]

From the above, in order to show that every sa language is accepted by a deterministic lba, it suffices to show that the language is accepted by a multitape off-line Turing machine of tape complexity $L(n) = kn$, for some constant k. We shall thus proceed to the main result. The sa S and S' and all notation is the same as what has been developed in the previous sections.

VI. THE MAIN RESULT

THEOREM. *If* $L = \mathfrak{N}(S)$, *then* L *is recognized by a deterministic linear bounded automaton.*

Proof. By Lemma 2, we assume S has property A. Let $S' = (K', T',$

[7] In [*7*], the off-line Turing machine is required to halt whether or not it accepts the input, while we do not require this. It is trivial to show [*1*] that for the off-line Turing machine of tape complexity n, which we shall define, the halting requirement does not change the recognizing power of the devices. The particular device which we will construct will, in fact, always halt.

[8] The lba is a Turing machine with a single tape upon which the input is placed. The lba may rewrite on its input, but its head cannot leave the region upon which the input is placed.

I, δ', δ_b', q_0, Z_0, φ) be the summary machine for S. We will construct a five storage tape off-line Turing machine M recognizing L.

Suppose w of length n is input to M. On storage tape 1, M will systematically generate strings in D (strings symbolizing sequences of moves of S'). Each string generated will be tested to see if it symbolizes a valid sequence of moves of S' which does not cause the length of the stack of S' to exceed $3n + 2$ symbols. Tape 2 of M will hold the stack of S' during the test. The remaining three tapes find use when M must determine if a given stack scan is possible from a given configuration.

Initially, M marks off blocks of length $3n + 2$ on tapes 2 and 3, a block of length $3n + 3$ on tape 4, a block of length $21n + 3$ on tape 1 and a block of length $2s(3n + 2)$ on tape 5, where s is the number of states of S'.

To describe the operation of M, let us assume that a particular string α in D is on tape 1, with that tape head at the left end of α and $|\alpha| \leqq 21n + 3$. Also, the input head of M is at the leftmost input symbol. M will test whether α symbolizes a valid sequence of moves of S' by computing, in turn, the configuration of S' after each composite move symbolized by α. The stack of S' is simulated by M on tape 2 and the state of S' is stored in M's finite control. Since every composite move begins and ends at the top of stack, M knows where the stack head of S' is.

Initially, the stack string on tape 2 is Z_0, and M records that q_0 is the state of S'. Let us suppose that the simulation of S' by M has proceeded to some configuration (possibly the initial configuration). Stack string y is on tape 2, and M has recorded q as the state of S'. The input head of M is scanning some input symbol a (which has not yet influenced a move of S') and the head of tape 1 is scanning a symbol of α other than a 0. M must find, if it exists, the unique configuration that can result from the move or moves of S' symbolized by the symbol of α (and following 0's in the case of a stack scan) currently under the head on tape 1. If no such configuration exists, M must generate a new string in D and test that. We shall consider the construction of the new configuration of S' by cases.

Case 1. The symbol of α scanned is an ϵ-print or an ϵ-erase symbol. M "knows" from δ_b' whether the move symbolized is allowed to S'. If so, M simulates this move. If not, M generates a new string in D for testing. However, it is possible that an ϵ-print move will cause the stack string on tape 2 to exceed length $3n + 2$. In that case, M generates a new string in D instead of simulating the move.

Case 2. The symbol of α scanned is a true input move symbol, say i. M determines from the input symbol scanned, the state of S' and the rightmost symbol of y whether S' has at least i choices of true input move and, if so, what the ith choice is. M simulates this move directly if it exists and moves its input head right. If no such move exists, or if the move would cause the stack of S' to become larger than $3n + 2$ symbols, a new string in D is generated.

Case 3. The symbol of α scanned is an ϵ-stationary sequence. M can determine from the rightmost symbol of y whether there exists a sequence of moves of S' resulting in the desired state transition.[9] If so, M changes the state of S', and M generates a new string in D otherwise.

Case 4. The symbol of α scanned begins the representation of a stack scan. Let (q_1, q_2) be the symbol now scanned on tape 1, and let this symbol be followed by i 0's. M checks that q_1 is the current state of S and sets up a test on tapes 3, 4 and 5 to determine if a stack scan using the i input symbols under and to the right of M's input head is possible.

Let $y = Y_m Y_{m-1} \cdots Y_1$. The cells of tape 3 will hold an array V_m, $V_{m-1}, \cdots, V_1$, and the cells of tape 4 will hold an array W_m, W_{m-1}, $\cdots, W_0$. Each of the V's and W's can take any subset of K' as a value. Initially, $V_1 = \{q \mid \delta_b'(q_1, \epsilon, Y_1) \text{ contains } (q, -1)\}$, and $W_1 = \{q \mid \delta_b'(q_1, a, Y_1) \text{ contains } (q, -1)\}$, where a is the symbol under M's input head. All other V's and W's are empty. M performs the following routine i times, shifting its head on tape 1 through the 0's to count to i:

(1) Using the block of length $2s(3n + 2)$ on tape five as a counter the following is done $2s(3n + 2)$ times: For each j, $1 \leqq j \leqq m$, if $\delta'(q, \epsilon, Y_j)$ contains (p, d) and q is in V_j or W_j, then p is adjoined to V_{j-d} or W_{j-d}, respectively. If $\delta'(q, a, Y_j)$ contains (p, d), a is the symbol currently under M's input head and q is in V_j, then p is adjoined to W_{j-d}.

(2) M moves its input head right.

(3) Except on the last (ith) iteration, M replaces V_j by W_j, $1 \leqq j \leqq m$, and makes W_j empty, $0 \leqq j \leqq m$.

Observe that if after performing step (1), q is in W_j, $0 \leqq j \leqq m$, then it must be that for some integer l and state p, p was initially in V_l and $a\colon (p, y, l) \overset{(m,1)}{\underset{S'}{\vdash}} (q, y, j)$. Also, if $a\colon (p, y, l) \overset{(m,1)}{\underset{S'}{\vdash}} (q, y, j)$, then there is a sequence of moves of S', beginning in configuration (p, y, l) and ending in configuration (q, y, j), such that no pair of state

[9] This question is also solvable, but we do not offer a proof of this fact.

and stack head position repeats more than twice. Thus, if p was initially in V_l, after $2s(3n + 2)$ repetitions of the central operation of step (1), q will be in W_j.

Thus, after i iterations of the above process, q_2 will be in W_0 if and only if u: $(q_1, y, 0) \left|\frac{ss}{S'}\right. (q_2, y, 0)$, where u is the string consisting of the i input symbols at or to the right of the position of M's input head when the test for a possible stack scan was started.

We have now completed description of how M tests a string α in D, $|\alpha| \leqq 21n + 3$, to see whether it symbolizes a valid sequence of moves of S' which does not cause the stack of S' to grow longer than $3n + 2$. Certainly, M can test whether the configuration after completion of the moves symbolized by α causes S' to have an empty stack. If so, M accepts its input, and if not, M generates another sequence in D whose length does not exceed $21n + 3$.

If the input w to M is in $\mathfrak{N}(S')$, then by Lemmas 4 and 6, M will eventually generate on tape 1 a string α which symbolizes a sequence of moves, leading to acceptance of w, for which S's stack does not exceed length $3n + 2$. M will then successfully test α and accept w. Also, if M accepts its input after testing some α in D, then there is a sequence of moves of S', leading to acceptance, which is symbolized by α. We conclude that M accepts L.

RECEIVED: March 1967; REVISED July 1, 1968.

REFERENCES

GINSBURG, S., GREIBACH, S. A., AND HARRISON, M. A. (1967), Stack automata and compiling. *JACM* **14**, No. 1, 172–201.

GINSBURG, S., GREIBACH, S. A., AND HARRISON, M. A. (1967), One way stack automata. *JACM* **14**, No. 2, 389–418.

GINSBURG, S. (1966), The Mathematical Theory of Context Free Languages. McGraw-Hill, New York.

KURODA, S. Y. (1964), Classes of languages and linear bounded automata. *Inf. Control* **7**, No. 2, 207–223.

LANDWEBER, P. S. (1963), Three theorems on phase structure grammars of Type 1. *Inf. Control* **6**, No. 2, pp. 131–136.

MYHILL, J. (1960), Linear bounded automata. *WADD Tech. Note*, No. 60–165, Wright Patterson AFB, Ohio.

HARTMANIS, J., LEWIS II, P. M., AND STEARNS, R. E. (1965), Hierarchies of memory limited computation. IEEE Conf. Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, 179–190.

LEWIS II, P. M., STEARNS, R. E., AND HARTMANIS, J. (1965), Memory bounds for the recognition of context free and cotntext sensitive languages. IEEE Conf. Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, 191–202.