Characterizations of Some Tape and Time Complexity Classes of Turing Machines in Terms of Multihead and Auxiliary Stack Automata

OSCAR H. IBARRA

Department of Computer, Information, and Control Sciences, Institute of Technology, University of Minnesota, Minneapolis, Minnesota 55455

Received July 17, 1970

Several classes of multihead and auxiliary stack automata are introduced and are used to characterize some tape and time complexity classes of Turing machines.

INTRODUCTION

In [1], a unified model of an automaton called a balloon automaton was introduced. A subset of the balloon automata was defined as a closed class if it obeys certain closure properties, and it was noted that a type of automaton seems to form a class if its definition involves only ways in which the infinite memory may be locally manipulated. It was shown that most well-known devices can be described by certain closed classes of balloon automata. However, as was pointed out in that paper, tape and time complexity classes of Turing machines do not form closed classes. An interesting line of inquiry posed there was: Find some model which would describe as closed classes the tape and time complexity classes of Turing machines.

The results of the present paper come in two parts. The first part is in some way related to the foregoing question. More specifically, we have attempted to characterize several tape and time complexity classes of Turing machines in terms of devices whose definitions involve only ways in which their infinite memory may be manipulated and no restrictions are imposed on the amount of memory that they use. The basic model for these devices is the multihead stack automaton, a generalization of the stack automaton introduced in [2]. The second part of the paper investigates the computational power of stack automata which are equipped with Turing machine storage tapes.

The paper is divided into three sections. Section 1 begins with an informal description of the well-known (multitape) deterministic (respectively, nondeterministic) Turing machine [3]. The notion of an auxiliary deterministic pushdown automaton studied in [4] is then briefly defined. The rest of the section is devoted to introducing several new extensions to the model of a stack automaton introduced in [2]. A $k$-head
stack automaton ($k$-SA) is defined as a stack automaton with $k$ input heads. The deterministic version of the device is called $k$-DSA. A $k$-SA (respectively, $k$-DSA) whose stack is nonerasing is called $k$-NESA (respectively, $k$-NEDSA). Of particular interest is a $k$-NESA (respectively, $k$-NEDSA) which has the property that once its stack head enters the interior of the stack, it can never write again. Such a device is called a $k$-head checking stack automaton ($k$-CSA) (respectively, $k$-CDSA). Finally, the notion of an auxiliary stack automaton (aux SA) (respectively, aux DSA, aux NESA, aux NEDSA, aux CSA, aux CDSA) is introduced. Intuitively, it is a 1-SA (respectively, 1-DSA, 1-NESA, 1-NEDSA, 1-CSA, 1-CDSA) which is equipped with Turing machine storage tapes. Tape and time bounds for the auxiliary devices are defined as in Turing machines except that, in the case of tape bounds, the stack is not counted.

In Section 2, we derive results concerning the computational power of the several classes of $k$-SA that we have introduced. In particular, we show that a set is accepted by a $k$-SA (respectively, $k$-DSA, $k$-NESA, $k$-NEDSA, $k$-CSA) if and only if it is accepted by a $2^{cn}$-time bounded deterministic Turing machine (respectively, $2^{cn \log n}$-time bounded deterministic Turing machine, $n^{2k}$-tape bounded nondeterministic Turing machine, $n^{k \log n}$-tape bounded deterministic Turing machine, $n^{k}$-tape bounded nondeterministic Turing machine) for some constant $c$. This generalizes some of the well-known results for the case when $k = 1$ (see Refs. [4, 7]).

In Section 3, we investigate the computational power of auxiliary stack automata. We prove that for any $L(n) \gg \log n$, a set is accepted by a $L(n)$-tape bounded aux DSA (respectively, $L(n)$-tape bounded aux NEDSA) if and only if it is accepted by a $2^{o(L(n))}$-time bounded deterministic Turing machine (respectively, $2^{L(n)}$-tape bounded deterministic Turing machine) for some constant $c$. Moreover, we show that aux SA and aux DSA have the same computational power and that aux NESA, aux NEDSA, and aux CSA are computationally equivalent. Finally, we show that a $L(n)$-tape bounded CDSA is equivalent to a $L(n)$-tape bounded deterministic Turing machine.

1. The Models

In this section, we shall introduce the different devices that are considered in the paper. We then conclude the section with a summary of some well-known results needed in the sequel. We start by giving an informal description of a multitape deterministic Turing machine [3].

A $k$-tape deterministic Turing machine (or simply, $k$-DTM) $U$ consists of a finite state control with a read-only input tape with end markers and $k$ semi-infinite (to the right) read-write storage tapes. Each of the tapes has a head attached to the finite state control. The tapes are divided into cells. Each cell of a storage tape is capable of holding a symbol from a finite storage alphabet. The input tape contains a string of symbols from a finite input alphabet $\Sigma$ and is of the form $\varphi x \mathcal{S}$, with $\varphi$ and $\mathcal{S}$ the
distinguished end markers and x not containing $\varnothing$ or $\mathcal{S}$. At any point in time, $U$ is in some state with each of the tape heads scanning a symbol on its tape. On a single move, depending on the state of the finite state control and the symbols scanned by the tape heads, the machine can: (1) change state, (2) print a new symbol on each of the cells scanned by the storage heads, and (3) move each of its tape heads, independently, one cell left or right or keep it stationary. The finite state control is designed to prevent the input head from leaving the input tape and each of the storage heads from moving left of its initial position (thus, the storage tapes are semi-infinite to the right). The machine is deterministic in the sense that there is only one possible next move at each step.

Some states of the finite state control are designated as final states, and one state is distinguished as the initial state. An input word $\varnothing x \mathcal{S}$ in $\Sigma^*$ is said to be accepted by $U$ if $U$, when started in its initial state with its input head on the left end marker of $\varnothing x \mathcal{S}$ and with all storage tapes blank, enters a final state after a finite number of moves. The set of all words accepted by $U$ is denoted by $T(U)$.

The multitape deterministic Turing machine can be extended to be nondeterministic. A $k$-tape nondeterministic Turing machine (written simply as $k$-TM) has a finite number of choices of next moves. Acceptance of a word is the same except that now we only require that there is some sequence of moves leading to a final state.

The next type of device that we shall consider is a $k$-tape auxiliary deterministic pushdown automaton (abbreviated, $k$-aux DPDA) introduced in [4]. A $k$-aux DPDA is a $k$-DTM with an extra tape called the pushdown tape which may be modified by adding symbols at the top or erasing symbols from the top. The reader is referred to [4] for the formal definition of the device and for motivations for its introduction.

The model of a stack automaton was first introduced in [2]. Intuitively, a stack automaton consists of an input tape with end markers, a finite state control, and a stack. The input head may be moved in either direction. The stack is similar to the pushdown tape in a pushdown automaton except that now the stack head is capable of moving into the stack in a read-only mode.

We shall introduce several extensions to the model just described. The first generalization is the multihead stack automaton. Intuitively, a multihead stack automaton is a stack automaton which may have more than one head on its input tape. The formal definition follows.

**Definition.** A $k$-head stack automaton (abbreviated, $k$-SA) is a 10-tuple $S = \langle k, K, \Gamma, \delta, s_0, q_0, Z_0, b, F \rangle$, where

1. $k \geq 1$ is the number of input heads.
2. $K$ is a finite nonempty set of states.

1 For any nonempty set $V$, $V^*$ denotes the set of all finite sequences of symbols in $V$ including the empty sequence, denoted by $\varnothing$. An element of $V^*$ is called a word or string in $V^*$. 
(3) $\Sigma$ is a finite nonempty set called the input alphabet. $\Sigma$ includes the left and right end markers $\varnothing$ and $\mathcal{S}$, respectively, which are assumed to be the first and last symbols of any input tape.

(4) $\Gamma$ is a finite nonempty set of nonblank stack symbols called the stack alphabet. $\Gamma$ includes a distinguished symbol $Z_0$ (the initial stack symbol) which acts as a left end marker of the stack. The symbol $b$ stands for a blank. Initially, the stack head is on the leftmost blank to the right of $Z_0$.

(5) $F \subseteq K$ is a set of final states.

(6) $q_0$ in $K$ is the initial state.

(7) $\delta$ is a mapping from $K \times (\Sigma)^k \times \Gamma^2$ into the subsets of $K \times (\{-1, 0, 1\})^k \times \{-1, 0, 1\}$ satisfying the following conditions: For each $(q, \sigma_1, \ldots, \sigma_k, Z)$ in $K \times (\Sigma)^k \times \Gamma$, if $(p, d_1, \ldots, d_k, e)$ is in $\delta(q, \sigma_1, \ldots, \sigma_k, Z)$, then $d_i \geq 0$ if $\sigma_i = \varnothing$, $d_i \leq 0$ if $\sigma_i = \mathcal{S}$, and $e \geq 0$ if $Z = Z_0$.

(8) $\delta_b$ is a mapping from $K \times (\Sigma)^k \times \Gamma$ into the subsets of $K \times (\{-1, 0, 1\})^k \times (\{-1, 0, 1\} \cup (\Gamma \setminus \{Z_0\}))$ satisfying the following conditions: For each $(q, \sigma_1, \ldots, \sigma_k, Z)$ in $K \times (\Sigma)^k \times \Gamma$, if $(p, d_1, \ldots, d_k, X)$ is in $\delta_b(q, \sigma_1, \ldots, \sigma_k, Z)$, then $d_i \geq 0$ if $\sigma_i = \varnothing$, $d_i \leq 0$ if $\sigma_i = \mathcal{S}$, and $X = \neq E$ if $Z = Z_0$. (It is assumed that the symbols $-1, 0,$ and $1$ are not in $\Gamma$).

$\delta$ is the next move mapping for the case when the stack head is not scanning a blank. It will always turn out that this blank is the leftmost blank (called the top of the stack). The interpretation of $(p, d_1, \ldots, d_k, e)$ in $\delta(q, \sigma_1, \ldots, \sigma_k, Z)$ is that if $S$ is in state $q$ with its $i$-th input head scanning $\sigma_i (1 \leq i \leq k)$ and its stack head scanning $Z$, $S$ may go to state $p$ and move its $i$-th input head in direction $d_i$ and its stack head in direction $e$, where $-1, 0,$ and $1$ are interpreted as left move, no move, and right move, respectively. The conditions on $\delta$ prevent any input head from leaving either end of the input tape and the stack head from going off the left end of the stack.

$\delta_b$ is the next move mapping for the case when the stack head is scanning the top of the stack (i.e., the leftmost blank). The interpretation of $(p, d_1, \ldots, d_k, X)$ in $\delta_b(q, \sigma_1, \ldots, \sigma_k, Z)$ is that if $S$ is in state $q$ with its $i$-th input head scanning $\sigma_i (1 \leq i \leq k)$ and its stack head on the top of the stack with the nonblank symbol $Z$ immediately to the left, then $S$ may enter state $p$, move its $i$-th input head in direction $d_i$, and make the following actions of its stack head:

(a) If $X = 0$, the stack head does not move.

(b) If $X = -1$, the stack head moves one cell to the left.

(c) If $X = E$, the stack head moves one cell to the left and replaces the new symbol scanned by a blank. Thus, the rightmost nonblank symbol is "erased" and becomes the new top of the stack.

* For any set $L$, $L^k$ denotes the $k$-fold cartesian product of $L$. 
(d) If \( X \) is in \((\Gamma - \{Z_0\})\), the stack head prints \( X \) over the blank and then moves right to the new top of the stack.

The conditions on \( \delta_b \) prevent any input head from leaving either end of the input tape and the stack head from erasing the initial stack symbol \( Z_0 \).

Note that the stack will always contain a word in \( Z_0(\Gamma - \{Z_0\})^* \) followed by an infinite sequence of blanks to the right.

We need the following notation before we can describe acceptance of words by a \( k \)-SA.

**Notation.** If \( S = \langle k, K, \Sigma, \Gamma, \delta, \delta_b, q_0, Z_0, b, F \rangle \) is a \( k \)-SA, we define a configuration of \( S \) with input \( w \) in \( \Sigma^* \) to be a combination of the state of the finite state control, the input tape, the nonblank portion of the stack, the positions of the \( k \) input heads, and the position of the stack head. The configuration is denoted by \((q, w, y, j_1, ..., j_k, i)\) where \( q \) is in \( K \), \( w \) is the input tape, \( y \) is the nonblank portion of the stack, \( j_1, ..., j_k \) are integers between 1 and \( |w| \) giving the positions of the \( k \) input heads from the left end of \( w \) (the first symbol of \( w \) being counted as position 1), and \( i \) is an integer between 1 and \(|y| + 1 \), indicating the position of the stack head on \( y \). Thus, if \( i = 1 \), then \( S \)'s stack head is on the first symbol of \( y \), and if \( i = |y| + 1 \), \( S \)'s stack head is on the top of the stack. We define a relation between configurations as follows. Write \((q, w, y, j_1, ..., j_k, i) \sim (q', w, y', j_1', ..., j_k', i')\) if configuration \((q, w, y, j_1, ..., j_k, i)\) can become configuration \((q', w, y', j_1', ..., j_k', i')\) by a single move of \( S \). We denote by \( \sim \) the reflexive and transitive closure of \( \sim \).

**Definition.** Let \( S = \langle k, K, \Sigma, \Gamma, \delta, \delta_b, q_0, Z_0, b, F \rangle \) be a \( k \)-SA and \( w \) be in \( \Sigma^* \). \( S \) accepts input \( w \) if \((q_0, w, Z_0, 1, ..., 1, 2) \sim (q, w, y, j_1, ..., j_k, i)\), for some \( q \) in \( F \), \( y \) in \( Z_0(\Gamma - \{Z_0\})^* \), and integers \( j_1, ..., j_k, i \). The set accepted by \( S \), denoted by \( T(S) \), is the set \( \{w | w \in \Sigma^* \} \) \( S \) accepts \( w \).

**Remark.** Our definition of a \( k \)-SA when \( k = 1 \) is similar to the one given in [3].

We now put some restrictions to the basic model of a \( k \)-SA to get some classes analogous to the restricted classes for the case when \( k = 1 \).

**Definition.** A \( k \)-SA \( S = \langle k, K, \Sigma, \Gamma, \delta, \delta_b, q_0, Z_0, b, F \rangle \) is called deterministic \((k-DSA, \text{for short})\) if for each \((q, \sigma_1, ..., \sigma_k, Z)\) in \( K \times (\Sigma^k) \times \Gamma, |\delta(q, \sigma_1, ..., \sigma_k, Z)| \leq 1^5 \) and \( |\delta_b(q, \sigma_1, ..., \sigma_k, Z)| \leq 1 \). A \( k \)-SA (respectively, \( k-DSA \)) is called nonerasing written simply as \( k-NESA \) (respectively, \( k-NEDSA \)) if for each \((q, \sigma_1, ..., \sigma_k, Z)\) in \( K \times (\Sigma^k) \times \Gamma, (p, d_1, ..., d_k, X) \) in \( \delta(q, \sigma_1, ..., \sigma_k, Z) \) implies \( X \neq E \).

---

\(^3\) If \( x \) and \( y \) are words, \( xy \) denotes the **concatenation** of \( x \) and \( y \). \( x^j = A \) and \( x^i = x \cdots x \) \((j \text{ times})\), \( j \geq 1 \). If \( X \) and \( Y \) are sets of words, \( XY \) denotes the set \( \{xy | x \in X, y \in Y \} \). If \( X \) is a singleton, \( \{x\} \), we write \( xY \) instead of \( \{x\}Y \).

\(^4\) For any word \( w \), \( |w| \) denotes length of \( w \), i.e., the number of symbols in \( w \).

\(^5\) For any set \( L \), \( |L| \) denotes the cardinality of \( L \).
In [5], a checking automaton was defined essentially as a one-way 1-NESA which once the stack head enters its stack, it can never write again. Here, we are interested in the class of $k$-NESA which obeys the same stack restriction. Formally, we have

**DEFINITION.** A $k$-NESA (respectively, $k$-NEDSA) $S = \langle k, K, \Sigma, \Gamma, \delta, \delta_b, q_0, Z_0, b, F \rangle$ is called checking or simply, $k$-CSA (respectively, $k$-CDSA) if the following condition is satisfied:

For all $w$ in $\#(\Sigma - \{\varepsilon, \varepsilon\})^*\varepsilon$, and configurations $(q, w, y, j_1, \ldots, j_k, i)$ and $(q', w, y', j_1', \ldots, j_k', i')$ if $(q_0, w, Z_0, 1, \ldots, 1, 2) \equiv (q, w, y, j_1, \ldots, j_k, i) \equiv (q', w, y', j_1', \ldots, j_k', i')$ and $i \leq |y|$, then $y = y'$ and $i' \leq |y'|$.

A device which is closely related to a multihead checking stack automaton is a two-tape finite-state automaton [6] which may have more than one head on its first tape. The formal definition is as follows.

**DEFINITION.** A $k$-head two-tape finite-state automaton ($k$-2FA, for short) is a 9-tuple $S = \langle k, K, \Sigma, \Gamma, \delta, q_0, Z_0, Y_0, F \rangle$, where

1. $k \geq 1$ is the number of heads on the first tape. We shall refer to this tape as the input tape and the $k$ heads as the input heads.
2. $K$, $\Sigma$, $F$, and $q_0$ have the same significance as in a $k$-SA.
3. $\Gamma$ is a finite nonempty set of **second tape symbols**. $\Gamma$ includes the **left and right end markers** $Z_0$ and $Y_0$, respectively, which are assumed to be the first and last symbols of the second tape. For uniformity and convenience in our proofs later, we shall also refer to the second tape as the stack and its head as the stack head.
4. $\delta$ is mapping from $K \times (\Sigma)^k \times \Gamma$ into the subsets of $K \times \{-1, 0, +1\}^k \times \{-1, 0, +1\}$ satisfying the following conditions:

   The interpretation of $\delta$ is similar to that when $S$ was a $k$-SA. Initially, $S$ is given a pair $(w, y)$ in $\#(\Sigma - \{\varepsilon, \varepsilon\})^*\Sigma \times Z_0(\Gamma - \{Z_0, Y_0\})^*Y_0$. A configuration of $S$ is given by $(q, w, y, j_1, \ldots, j_k, i)$, where $q$ is in $K$, $j_1, \ldots, j_k$ are the positions of the $k$ heads on the input $w$, and $i$ is the position of the second tape head on the second tape $y$. The relations $\equiv$ and $\mathcal{L}$ between configurations are defined in the obvious way. $S$ accepts a pair $(w, y)$ in $\#(\Sigma - \{\varepsilon, \varepsilon\})^*\Sigma \times Z_0(\Gamma - \{Z_0, Y_0\})^*Y_0$ if $(q_0, w, y, 1, \ldots, 1) \mathcal{L} (q, w, y, j_1, \ldots, j_k, i)$ for some $q$ in $F$, integers $j_1, \ldots, j_k$ and $i$ such that $1 \leq j_1, \ldots, j_k \leq |w|$ and $1 \leq i \leq |y|$. The set of all pairs accepted by $S$ is denoted by $T(S)$. We may think of $S$ as accepting a relation over $\#(\Sigma - \{\varepsilon, \varepsilon\})^*\Sigma \times Z_0(\Gamma - \{Z_0, Y_0\})^*Y_0$. For any relation $R$, we denote by $\text{Domain}(R)$, the set $\{x \mid$ there is a $y$ such that $(x, y)$ is in $R \}$. We shall show the connection of $k$-2FA's to $k$-CSA's in Section 2.

Finally, we introduce the following types of devices, giving only their informal descriptions, formalisms being obvious.
DEFINITION. A \( k \)-tape auxiliary stack automaton or \( k \)-aux SA (respectively, \( k \)-tape auxiliary deterministic stack automaton or \( k \)-aux DSA) is a \( k \)-TM (respectively, \( k \)-DTM) with an extra tape manipulated just like a stack in a stack automaton. If, in a \( k \)-aux SA (respectively, \( k \)-aux DSA) the stack is nonerasing, we shall call it a \( k \)-tape auxiliary nonerasing stack automaton or \( k \)-aux NESA (respectively, \( k \)-tape auxiliary nonerasing deterministic stack automaton or \( k \)-aux NEDSA). If in a \( k \)-aux NESA (respectively, \( k \)-aux NEDSA) the stack is checking, we shall call it a \( k \)-tape auxiliary checking stack automaton or \( k \)-aux CSA (respectively, \( k \)-tape auxiliary checking deterministic stack automaton or \( k \)-aux CDSA).

Convention. If, in any of the devices that we have introduced, \( k \) is equal to 1 we shall drop the prefix "\( k \)-". For example, DTM will mean 1-DTM, TM will mean 1-TM, aux DPDA will mean 1-aux DPDA, SA will mean 1-SA, etc.

Two popular measures of complexity of computation of an accepting device are bounds on the amount of tape used and the number of moves made by the device when given an input word. The main purpose of this paper is to relate the computational power of the different extended models of a stack automaton that we have introduced to some tape and time complexity classes of deterministic and nondeterministic Turing machines. Thus, the following definitions are in order.

DEFINITION. Let \( L(n) \) be a function from positive integers into positive integers. Let \( U \) be a \( k \)-TM (respectively, \( k \)-DTM, \( k \)-aux DPDA, \( k \)-aux SA, \( k \)-aux DSA, \( k \)-aux NESA, \( k \)-aux NEDSA, \( k \)-aux CSA, \( k \)-aux CDSA). \( U \) is said to be \( L(n) \)-tape bounded or of tape complexity \( L(n) \) if and only if for each input word \( w \) of length \( n \), if \( U \) accepts \( w \), there is a sequence of moves of \( U \) leading to the acceptance of \( w \) in which \( U \) uses no more than \( L(n) \) cells in any of its storage tapes (not counting the number of cells used in the pushdown tape in the case of a \( k \)-aux DPDA and the number of cells used in the stack in the cases of \( k \)-aux SA, \( k \)-aux DSA, \( k \)-aux NESA, \( k \)-aux NEDSA, \( k \)-aux CSA, and \( k \)-aux CDSA).

DEFINITION. Let \( T(n) \) be a function from positive integers into positive integers and \( U \) be as in the preceding definition. \( U \) is said to be \( T(n) \)-time bounded or of time complexity \( T(n) \) if and only if for each input word \( w \) of length \( n \), if \( U \) accepts \( w \), there is a sequence of at most \( T(n) \) moves of \( U \) leading to the acceptance of \( w \).

Convention. In the sequel, \( L(n) \) and \( T(n) \) will denote functions on positive integers. To simplify the proofs of the results of the next sections, we mention the following result whose proof is straightforward (see, for example, [3]).

THEOREM 1.1. Let \( k \geq 2 \) and \( U \) be a \( L(n) \)-tape bounded \( k \)-DTM (respectively, \( k \)-aux DPDA, \( k \)-aux DSA, \( k \)-aux NESA, \( k \)-aux CDSA, \( k \)-TM, \( k \)-aux SA, \( k \)-aux NEDSA, \( k \)-aux CSA). Then there exists a \( L(n) \)-tape bounded DTM (respectively, aux
MULTIHEAD AND AUXILIARY STACK AUTOMATA

DPDA, aux DSA, aux NEDSA, aux CDSA, TM, aux SA, aux NESA, aux CSA) U' such that T(U') = T(U).

Thus, a device with a single storage tape is no less powerful than a device with multiple storage tapes as far as tape complexity classes are concerned.

We close this section with a theorem summarizing well-known results which are needed in the sequel.

THEOREM 1.2. (a) A set is accepted by a SA (respectively, DSA) if and only if it is accepted by a n²-tape bounded aux DPDA (respectively, n log n-tape bounded aux DPDA) [4].
(b) A set is accepted by a NESA (respectively, NEDSA) if and only if it is accepted by a n²-tape bounded TM (respectively, n log n-tape bounded DTM) [7].
(c) A set accepted by a n log n-tape bounded TM is accepted by a DSA [3].
(d) A set is accepted by a n-tape bounded TM if and only if it is the domain of some relation accepted by a 2FA [6].
(e) Let L(n) ≥ log₂ n. A set is accepted by a L(n)-tape bounded aux DPDA if and only if it is accepted by a DTM of time complexity 2^{cL(n)} for some constant c [4].
(f) Let L(n) ≥ log₂ n. A set accepted by a L(n)-tape bounded TM is accepted by a (L(n))²-tape bounded DTM [8].

2. MULTIHEAD STACK AUTOMATA

In this section, we shall characterize the computational power of the different types of multihead stack automata that we have introduced in terms of TM's, DTM's, and aux DPDA's. The main result of the section is the following: A set is accepted by a machine S in row I(1 ≤ I ≤ 5) of the first column of Table I if and only if it is accepted by a machine U in row I of the second column. Several interesting corollaries then arise. For example, a set is accepted by a k-SA (respectively, k-DSA) if and only

<table>
<thead>
<tr>
<th>Machine S</th>
<th>Machine U</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. k-SA</td>
<td>n²-tape bounded aux DPDA</td>
</tr>
<tr>
<td>2. k-DSA</td>
<td>n² log n-tape bounded aux DPDA</td>
</tr>
<tr>
<td>3. k-NESA</td>
<td>n²⁴-tape bounded TM</td>
</tr>
<tr>
<td>4. k-NEDSA</td>
<td>n² log n-tape bounded DTM</td>
</tr>
<tr>
<td>5. k-CSA</td>
<td>n⁴-tape bounded TM</td>
</tr>
</tbody>
</table>
if its is accepted by $2^{cn^2}$-time bounded DTM (respectively, $2^{cn^{\log n}}$-time bounded DTM) for some constant $c$. Our method of proving the main result is not a generalization of the techniques used for the case when $k$ is equal to 1. If, in fact, we do this, we will find it necessary to provide separate and complicated arguments for each of the different classes of $k$-SA that we are considering. We present here techniques which will take care of all the classes of $k$-SA at once. These techniques have the merits of being simple and applicable to other types of automata. In particular, we shall use the same techniques in Section 3 to characterize the computational power of auxiliary stack automata.

We begin by first showing the connection between a $k$-CSA and a $k$-2FA.

**Theorem 2.1.** A set is accepted by a $k$-CSA if and only if it is the domain of some relation accepted by a $k$-2FA.

**Proof.** Let $S = \langle k, K, \Sigma, \Gamma, \delta, \delta_0, q_0, Z_0, b, F \rangle$ be a $k$-CSA. Without loss of generality, we may assume that $S$ halts (i.e., $S$ has no next move) when it enters a final state. Thus, the stack contents for every input accepted by $S$ is finite. We construct a $k$-2FA $S'$ which checks that the only symbols appearing on its second tape are those that were written by $S$. When $S$ first enters the stack, $S'$ checks that the next symbol on its second tape is the right end marker. From this point on, $S'$ simulates the actions of $S$, going into a final state if and only if $S$ goes to a final state. We now define $S'$ formally. For each $q$ in $K$, $Z$ in $\Gamma$, and $i$ in $\{0, 1\}$, let $(q, i)$ and $(q, Z)$ be abstract symbols. Let $K' = K \cup \{(q, i) \mid q \in K, i = 0, 1\} \cup \{(q, Z) \mid q \in K, Z \in \Gamma\}$ and $F' = F \cup \{(q, i) \mid q \in F, i = 0, 1\} \cup \{(q, Z) \mid q \in F, Z \in \Gamma\}$. Let $Y_0$ be a symbol not in $\Gamma$. Let $S' = \langle k, K', \Sigma, \Gamma \cup \{Y_0\}, \delta', (q_0, 0), Z_0, Y_0, F' \rangle$ be a $k$-2FA, where $\delta'$ is defined as follows: Let $q, p$ be in $K$, $Z, Z'$ in $\Gamma$, and $(\sigma_1, \ldots, \sigma_k)$ in $(\Sigma)^k$.

1. If $(p, d_1, \ldots, d_k, 0)$ is in $\delta_0(q, \sigma_1, \ldots, \sigma_k, Z)$, then let $((p, 0), d_1, \ldots, d_k, 0)$ be in $\delta'(q, 0, \sigma_1, \ldots, \sigma_k, Z)$.

2. If $(p, d_1, \ldots, d_k, 0)$ is in $\delta_0(q, \sigma_1, \ldots, \sigma_k, Z)$, then let $((p, 0), d_1, \ldots, d_k, +1)$ be in $\delta'(q, 0, \sigma_1, \ldots, \sigma_k, Z)$ and let $((p, 0), 0, \ldots, 0)$ be in $\delta'(q, 0, \sigma_1, \ldots, \sigma_k, Z)$.

3. If $(p, d_1, \ldots, d_k, -1)$ is in $\delta_0(q, \sigma_1, \ldots, \sigma_k, Z)$, then let $((p, 1), d_1, \ldots, d_k, +1)$ be in $\delta'(q, 0, \sigma_1, \ldots, \sigma_k, Z)$ and let $((p, 0, \ldots, 0, -1)$ be in $\delta'(q, 0, \sigma_1, \ldots, \sigma_k, Y_0)$.

4. If $(p, d_1, \ldots, d_k, e)$ is in $\delta(q, \sigma_1, \ldots, \sigma_k, Z)$ with $e$ in $\{-1, 0, +1\}$, then let $(p, d_1, \ldots, d_k, e)$ be in $\delta'(q, \sigma_1, \ldots, \sigma_k, Z)$.

It is easily verified that $\text{Domain}(T(S')) = T(S)$.

Now let $S = \langle k, K, \Sigma, \Gamma, \delta, q_0, Z_0, Y_0, F \rangle$ be a $k$-2FA. A $k$-CSA $S'$ accepting $\text{Domain}(T(S))$ can be constructed to operate as follows: Given an input $w$ in $\Sigma^* \cdot \{\#, \delta\}^* \Sigma$, $S'$ nondeterministically writes to the right of $Z_0$ on its stack a word in $(\Gamma^* - \{Z_0, Y_0\})^* Y_0$. At the end of the process, $S'$ will have on its stack a word $y$ in $Z_0(\Gamma^* - \{Z_0, Y_0\})^* Y_0$ with its stack head on the top of the stack. $S'$ then moves
its stack head left to \( Z_0 \) and starts simulating the actions of \( S \) on the pair \((w, y)\). \( S' \) accepts the input if and only if \( S \) goes to a final state during the simulation. It is clear that \( S' \) operating as described is a \( k \)-CSA and that \( T(S') = \text{Domain}(T(S)) \). We omit the formal construction of \( S' \).

The following result follows from the above result and Theorem 1.2 (d).

**Corollary 2.1.** A set is accepted by a CSA if and only if it is accepted by a \( n \)-tape bounded TM.

We now proceed to show the first half of our characterization result. For convenience, we shall refer to Table II in the expositions of Lemma 2.1 and Theorem 2.2.

**TABLE II**

<table>
<thead>
<tr>
<th>( U )</th>
<th>( U' )</th>
<th>( S )</th>
<th>( S' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( n^k )-tape bounded aux DPDA</td>
<td>( n^2 )-tape bounded aux DPDA</td>
<td>SA</td>
<td>k-SA</td>
</tr>
<tr>
<td>2. ( n^k \log n )-tape bounded aux DPDA</td>
<td>( n \log n )-tape bounded aux DPDA</td>
<td>DSA</td>
<td>k-DSA</td>
</tr>
<tr>
<td>3. ( n^k )-tape bounded TM</td>
<td>( n^2 )-tape bounded TM</td>
<td>NESA</td>
<td>k-NESA</td>
</tr>
<tr>
<td>4. ( n^k \log n )-tape bounded DTM</td>
<td>( n \log n )-tape bounded DTM</td>
<td>NEDSA</td>
<td>k-NEDSA</td>
</tr>
<tr>
<td>5. ( n^k )-tape bounded TM</td>
<td>( n )-tape bounded TM</td>
<td>CSA</td>
<td>k-CSA</td>
</tr>
<tr>
<td>6. ( n^k \log n )-tape bounded TM</td>
<td>( n \log n )-tape bounded TM</td>
<td>DSA</td>
<td>k-DSA</td>
</tr>
</tbody>
</table>

**Lemma 2.1.** Let \( k \geq 2 \) and \( U \) be a machine in row \( I (1 \leq I \leq 6) \) of the first column of Table II. Let \( \Sigma \) be the input alphabet of \( U \) and \( \xi \) and \( \delta \) be new symbols not in \( \Sigma \). Then there exists a machine \( U' \) in row \( I \) of the second column such that \( T(U') = \{ \xi(\xi \delta \delta)^j \delta \delta^{k-1} \delta \mid \xi \delta \delta \in T(U) \} \).

**Proof.** By Theorem 1.1, it suffices to construct \( U' \) having three storage tapes. We describe briefly the operation of \( U' \). The formal construction is left to the reader. The input alphabet of \( U' \) is \( \Sigma \cup \{ \xi, \delta \} \), where \( \xi \) and \( \delta \) are new symbols not in \( \Sigma \) which will serve as end markers of inputs to \( U' \). Given an input \( \xi y \delta \) to \( U' \), \( y \) in \( \Sigma^* \), \( U' \) carries out the following steps:

1. \( U' \), using only its first storage tape, checks that the input \( \xi y \delta \) is of the form \( \xi(\xi x \delta)^j \delta \delta \) for some \( \xi x \delta \in \xi (\Sigma - \{ \xi, \delta \})^* \delta \) and \( j \geq 1 \). This step is easily done by \( U' \) deterministically using no more than \( |\xi y \delta| \) cells in the first storage tape. If successful, \( U' \) goes to the next step; otherwise, \( U' \) halts in a nonfinal state.

2. \( U' \), using only its second storage tape, checks that \( j = |\xi x \delta|^{k-1} \). Again, \( U' \) can carry out this step deterministically using no more than \( |\xi y \delta| \) cells in the second storage tape. If successful, \( U' \) goes to the next step; otherwise, \( U' \) halts in a nonfinal state.

\* For any \( j \geq 1 \), \( (\xi x \delta)^j \) is the word \( \xi x \delta \) concatenated \( j \) times. The pair of parentheses is only used to avoid ambiguity.
(3) $U'$ simulates $U$ on the first $\xi_xS$ block of the input $\xi(\xi_xS)^{|\xi_xS|^k-1} S$ using its third storage tape to correspond to the storage tape of $U$, and accepting the input if and only if $U$ goes to a final state during the simulation.

Clearly, $T(U') = \{\xi(\xi_xS)^{|\xi_xS|^k-1} S \mid \xi_xS \in T(U)\}$ and $U'$ has the desired tape bound.

We now use the preceding lemma to prove the first important result of this section.

Theorem 2.2. Let $k \geq 1$ and $U$ be a machine in row $I(1 \leq I \leq 6)$ of the first column of Table II. Then $T(U) = T(S')$ for some machine $S'$ in row $I$ of the fourth column.

Proof. By Theorem 1.2 and Corollary 2.1, we need only take care of the case when $k \geq 2$. Given a machine $U$ in row $I(1 \leq I \leq 6)$ of the first column of Table II, there is a machine $U'$ in row $I$ of the second column such that $T(U') = \{\xi(\xi_xS)^{|\xi_xS|^k-1} S \mid \xi_xS \in T(U)\}$ by Lemma 2.1. Then by Theorem 1.2 and Corollary 2.1, $T(U') = T(S)$ for some machine $S$ in row $I$ of the third column. The desired $S'$ will be constructed from $S$. Informally, $S'$ will operate as follows: Given $\xi_xS$ in $\xi(\Sigma - \{\xi, S\})^* S$, the first input head of $S'$ and its stack head will simulate the actions of the input head and the stack head of $S$ when $S$ is operating on $\xi(\xi_xS)^{|\xi_xS|^k-1} S$. In order for $S'$ to completely simulate the actions of $S$ on $\xi(\xi_xS)^{|\xi_xS|^k-1} S$, $S'$ must keep track of which block of the $|\xi_xS|^{k-1}$ blocks of $\xi_xS$ the input head of $S$ is on at any point in time. To do this, $S'$ uses its last $(k - 1)$ input heads to act as a counter which would indicate at any point in time, the block in which the input head of $S$ is on. Clearly, $(k - 1)$ heads can indicate a count from 1 to $|\xi_xS|^{k-1}$. Note that to simulate the transition of the input head of $S$ from a given block to the next "higher" (respectively, "lower") block, $S'$ need only move its first input head to the left end marker $\xi$ (respectively, right end marker $S$) and increments (respectively, decrements) the counter by 1. We now define $S'$ formally. Let $S = \langle I, K, (\Sigma \cup \{\xi, S\}), I', \delta, \delta_0, q_0, Z_0, b, F'\rangle$. For each $q$ in $K$, $i = 1, 2, \ldots, k$, let $(q, \xi), (q, S), (q, 1, i), (q, 2, i)$ be abstract symbols. Let $K' = K \cup \{(q, \xi), (q, S) \mid q \in K\} \cup \{(q, 1, i), (q, 2, i) \mid q \in K, i = 1, 2, \ldots, k\}$ and $F' = F \cup \{(q, \xi), (q, S) \mid q \in F\} \cup \{(q, 1, i), (q, 2, i) \mid q \in F, i = 1, 2, \ldots, k\}$. Let $S' = \langle k, K', \Sigma, I', \delta', \delta_0', (q_0, \xi), Z_0, b, F'\rangle$, where $\delta'$ is defined as follows: Let $(q, \sigma, Z)$ be in $K \times \Sigma \times I'$.}

(1) Simulation of $S$ when the input head of $S$ is scanning $\xi$ and the next move of $S$ requires its input head to remain stationary on $\xi$. If $(p, 0, e)$ is in $\delta(q, \xi, Z)$, then let $((p, \xi), 0, \ldots, 0, e)$ be in $\delta'((q, \xi), \xi, \ldots, \xi, Z)$.

(2) Simulation of $S$ when the input head of $S$ is scanning $\xi$ and the next move of $S$ requires its input head to move right on $\xi$. (Note that a left move on $\xi$ cannot occur by our restriction on $\delta$. See Section 1).

If $(p, +1, e)$ is in $\delta(q, \xi, Z)$, then let $(p, 0, \ldots, 0, e)$ be in $\delta'((q, \xi), \xi, \ldots, \xi, \xi, Z)$.

(3) Simulation of $S$ when the input head of $S$ is scanning $S$ and the next move of $S$ requires its input head to remain stationary on $S$.

If $(p, 0, e)$ is in $\delta(q, S, Z)$, then let $((p, S), 0, \ldots, 0, e)$ be in $\delta'((q, S), S, \ldots, S, Z)$. 


(4) Simulation of $S$ when the input head of $S$ is scanning $\tilde{S}$ and the next move of $S$ requires its input head to move left on $\tilde{S}$. (Note that a right move on $\tilde{S}$ cannot occur by our restriction on $\delta$. See Section 1).

If $(p, -1, e)$ is in $\delta(q, \tilde{S}, Z)$, then let $(p, 0, ..., 0, e)$ be in $\delta'((q, \tilde{S}), \tilde{S}, ..., \tilde{S}, Z)$.

(5) Simulation of $S$ when the input head of $S$ is scanning a symbol on an input block $\phi x \tilde{S}$ and its next move requires its input head to remain on the same block.

If $(p, d, e)$ is in $\delta(q, \sigma, Z)$ and either $\sigma \neq \phi$ and $\sigma \neq \tilde{S}$, $\sigma = \phi$ and $d \geq 0$, or $\sigma = \tilde{S}$ and $d \leq 0$, then let $(p, d, 0, ..., 0, e)$ be in $\delta'((q, \sigma, \sigma_2, ..., \sigma_k), Z)$ for all $\sigma_2, ..., \sigma_k$ in $\Sigma$.

(6) Simulation of $S$ when the input head of $S$ is scanning $\tilde{S}$ and the next move of $S$ requires its input head to move right on $\tilde{S}$.

If $(p, +1, e)$ is in $\delta(q, \tilde{S}, Z)$, then

(a) $((p, \tilde{S}), 0, ..., 0, e)$ is in $\delta'((q, \tilde{S}), \tilde{S}, ..., \tilde{S}, Z)$.

(b) for $1 < i \leq k$, $\sigma_i$ in $(\Sigma - \{\tilde{S}\})$, $\sigma_{i+1}, ..., \sigma_k$ in $\Sigma$, let

$$((p, 1, i), 0, ..., 0, +1, 0, ..., 0, e)$$

be in $\delta'((q, \sigma_i, ..., \sigma_k), Z)$.

(c) for $1 < i \leq k$, $\sigma'$ in $(\Sigma - \{\tilde{S}\})$, $\sigma_i, ..., \sigma_k$ in $\Sigma$, $Z'$ in $\Gamma$, let

$$((p, 1, i), 0, ..., 0, -1, 0, ..., 0)$$

be in $\delta'((q, 1, i), \sigma', ..., \sigma_k, Z')$.

(d) for $1 < i \leq k$, $\sigma_i, ..., \sigma_k$ in $\Sigma$, $Z'$ in $\Gamma$, let $(p, 0, ..., 0)$ be in $\delta'((q, 1, i), \sigma_i, ..., \sigma_k, Z')$.

(7) Simulation of $S$ when the input head of $S$ is scanning $\phi$ and the next move of $S$ requires its input head to move left on $\phi$.

If $(p, -1, e)$ is in $\delta(q, \phi, Z)$, then

(a) $((p, \phi), 0, ..., 0, e)$ is in $\delta'((q, \phi), \phi, ..., \phi, Z)$.

(b) for $1 < i \leq k$, $\sigma_i$ in $(\Sigma - \{\phi\})$, $\sigma_{i+1}, ..., \sigma_k$ in $\Sigma$, let

$$((p, 2, i), 0, ..., 0, -1, 0, ..., 0, e)$$

be in $\delta'((q, \phi, ..., \phi, \sigma_i, ..., \sigma_k), Z)$.

(c) for $1 < i \leq k$, $\sigma'$ in $(\Sigma - \{\phi\})$, $\sigma_i, ..., \sigma_k$ in $\Sigma$, $Z'$ in $\Gamma$, let

$$((p, 2, i), 0, ..., 0, -1, 0, ..., 0)$$

be in $\delta'((q, 2, i), \sigma', ..., \sigma_k, Z')$.

(d) for $1 < i \leq k$, $\sigma_i, ..., \sigma_k$ in $\Sigma$, $Z'$ in $\Gamma$, let $(p, 0, ..., 0)$ be in $\delta'((p, 2, i), \phi, ..., \phi, \sigma_i, ..., \sigma_k, Z')$.
δ_δ' is defined by replacing the occurrences of δ by δ_δ, δ' by δ_δ' and e by X (in \{-1, 0, E\} \cup (\mathcal{T} - \{Z_0\}) in rules (1)-(7).

It is a straightforward matter to verify that \( T(S') = \{ εxS \mid εxS \text{ in } ε(\Sigma - \{ε, S\})^*S, \}
\hat{ε}(εxS)|x|\delta^k \delta^{-1} \delta \text{ in } T(S) \} = T(U).

We shall prove the converse of Theorem 2.2 (for 1 ≤ I ≤ 5), thus showing the equivalences in Table I. First, we introduce the following notation:

**Notation.** Let \( \Sigma \) be a finite nonempty set which includes the distinguished symbols \( \epsilon \) and \( S \). Let \( k \geq 2 \) and \( \Sigma_k \) be the set of abstract symbols \( \{(j_1, \ldots, j_{k-1}, \epsilon) \mid \epsilon \in \Sigma, j_i \in \{0, 1\} \text{ for } 1 \leq i \leq k - 1\} \). If \( \sigma \) is in \( \Sigma \), \( \sigma \) will denote any element in the set \( \{(j_1, \ldots, j_{k-1}, \sigma) \mid j_i \in \{0, 1\} \text{ for } 1 \leq i \leq k - 1\} \). Let \( h_k \) be the homomorphism from \( \Sigma_k^* \) into \( \Sigma^* \) defined by \( h_k((j_1, \ldots, j_{k-1}, \epsilon)) = \sigma \) for each \( (j_1, \ldots, j_{k-1}, \epsilon) \) in \( \Sigma_k \). Let \( g_k \) be the homomorphism of \( \Sigma_k^* \) into the \( (k - 1) \) tuples of natural numbers defined by \( g_k((j_1, \ldots, j_{k-1}, \epsilon)) = (j_1, \ldots, j_{k-1}) \) for each \( (j_1, \ldots, j_{k-1}, \epsilon) \) in \( \Sigma_k \), and \( g_k(xy) = g_k(x) + g_k(y) \) for \( x, y \in \Sigma_k^* \).

Let \( \phi xS \) be in \( \phi(\Sigma - \{\epsilon, S\})^*S, |\phi xS| = n \). Let \( j_1, \ldots, j_{k-1} \) be integers between 1 and \( n \). \( R(\phi xS, j_1, \ldots, j_{k-1}) \) will denote the word \( x \) in \( \Sigma_k^* \) with the following properties:

1. \( x = \delta_1 \cdots \delta_n \) for some \( \delta_1, \ldots, \delta_n \) in \( \Sigma_k \).
2. \( h_k(x) = \phi xS \).
3. \( g_k(x) = (1, \ldots, 1) \).
4. \( h_k(\delta_i) \) contains a 1 in the \( i \)-th coordinate, \( 1 \leq i \leq k - 1 \).

There are \( m = n^{k-1} \) distinct words of \( \Sigma_k^* \) of length \( n \) in the set
\[
(R(\phi xS, j_1, \ldots, j_{k-1}) \mid 1 \leq j_1, \ldots, j_{k-1} \leq n).
\]
Let \( \alpha_1, \ldots, \alpha_m \) be the words in the set, ordered lexicographically. We denote by \( \phi_k(\phi xS) \) the word \( \alpha_1 \cdots \alpha_m \). Note that \( |\phi_k(\phi xS)| = n^k = |\phi xS|^k \).

**Example.** If \( k = 4 \), \( \phi xS = \phi abcdeS, j_1 = j_2 = j_3 = 1, j_4 = 4 \), then \( R(\phi abcdeS, 1, 1, 4) = (1, 1, 0, \epsilon)(0, 0, 0, a)(0, 0, 0, b)(0, 0, 1, c)(0, 0, 0, d)(0, 0, 0, e)(0, 0, 0, S) \).

The significance of the above notation is the following: Let \( k \geq 2 \) and \( S \) be a \( k \)-head automaton (of any type) and \( \phi xS \) be an input to \( S \). Then any possible configuration of the first \( k - 1 \) input heads of \( S \) on \( \phi xS \) which \( S \) can enter in the course of processing \( \phi xS \) can be represented by a word in \( \Sigma_k^* \) as follows: If the first \( k - 1 \) input heads are on positions \( j_1, \ldots, j_{k-1} \) of the input \( \phi xS \), then this configuration is represented by the word \( R(\phi xS, j_1, \ldots, j_{k-1}) \) in \( \Sigma_k^* \).

**Definition.** A word \( y \) in \( \Sigma_k^* \) is well-formed if and only if the following conditions are met:
(1) There exists some $m \geq 1$ such that $y = \tilde{a}_1 y_1 \tilde{S}_1 \tilde{a}_2 y_2 \tilde{S}_2 \cdots \tilde{a}_m y_m \tilde{S}_m$ with $h_k(\tilde{a}_i) = \tilde{a}$, $h_k(\tilde{S}_i) = \tilde{S}$, and $h_k(y_i)$ in $(\Sigma - \{\tilde{a}, \tilde{S}\})*$ for $1 \leq i \leq m$.

(2) $\tilde{g}_k(\tilde{a}_i y_i \tilde{S}_i) = (1, \ldots, 1)$ for $1 \leq i \leq m$.

(3) There exists $1 \leq i \leq m$ such that $\tilde{g}_k(\tilde{a}_i) = (1, \ldots, 1)$.

Note that $\tilde{\phi}_k(\tilde{a}x\tilde{S})$ is well formed.

For ease in stating and discussing the next four results, we introduce the following table:

<table>
<thead>
<tr>
<th>$S$</th>
<th>$S'$</th>
<th>$U$</th>
<th>$U'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $k$-SA</td>
<td>SA</td>
<td>$n^2$-tape bounded aux DPDA</td>
<td>$n^{2k}$-tape bounded aux DPDA</td>
</tr>
<tr>
<td>2. $k$-DSA</td>
<td>DSA</td>
<td>$n \log n$-tape bounded aux DPDA</td>
<td>$n^k \log n$-tape bounded aux DPDA</td>
</tr>
<tr>
<td>3. $k$-NESA</td>
<td>NESA</td>
<td>$n^2$-tape bounded TM</td>
<td>$n^k$-tape bounded TM</td>
</tr>
<tr>
<td>4. $k$-NEDSA</td>
<td>NEDSA</td>
<td>$n \log n$-tape bounded DTM</td>
<td>$n^k \log n$-tape bounded DTM</td>
</tr>
<tr>
<td>5. $k$-CSA</td>
<td>CSA</td>
<td>$n$-tape bounded TM</td>
<td>$n^k$-tape bounded TM</td>
</tr>
</tbody>
</table>

The following lemma is the key result to proving the main theorem.

**Lemma 2.2.** Let $k \geq 2$ and $S = \langle k, K, \Sigma, \Gamma, \delta, \delta_0, q_0, Z_0, b, F \rangle$ be a machine in row $I(1 \leq I \leq 4)$ of the first column of Table III. Then there exists a machine $S'$ in row $I$ of the second column whose input alphabet is $\tilde{\Sigma}_k \cup \{\tilde{a}, \tilde{S}\}$ ($\tilde{a}$ and $\tilde{S}$ are new symbols not in $\tilde{\Sigma}_k$ which serve as end markers of inputs to $S'$) with the following property: For each $\tilde{a}x\tilde{S}$ in $\tilde{\epsilon}(\Sigma - \{\tilde{a}, \tilde{S}\})*\tilde{S}$, $\tilde{\phi}_k(\tilde{a}x\tilde{S})\tilde{S}$ is in $T(S')$ if and only if $\tilde{a}x\tilde{S}$ is in $T(S)$.

**Proof.** Let $S' = \langle k', K', \tilde{\Sigma}_k \cup \{\tilde{a}, \tilde{S}\}, \Gamma', \delta', \delta_0', q_0', Z_0', b', F' \rangle$, where $\Gamma' = \Gamma \cup \tilde{\Sigma}_k \cup \{Z_0', *, A\}$, and we assume that $\Sigma, \tilde{\Sigma}_k$, and $\Gamma'$ are pairwise disjoint and $Z_0', \ast$, and $A$ are new symbols not in $\Gamma \cup \Sigma \cup \tilde{\Sigma}_k$. We shall describe the operation of $S'$ informally. It will be seen from our discussion that $K'$ is finite and the $\delta'$ and $\delta_0'$ can be defined formally. We describe the operation of $S'$ on an input $\tilde{a}y\tilde{S}$ in $\tilde{\epsilon}\Sigma_*\tilde{S}$. If $y$ is in fact $\tilde{\phi}_k(\tilde{a}x\tilde{S})$ for some $\tilde{a}x\tilde{S}$ in $\tilde{\epsilon}(\Sigma - \{\tilde{a}, \tilde{S}\})*\tilde{S}$, then $S'$ would accept $\tilde{a}y\tilde{S}$ if and only if $\tilde{a}x\tilde{S}$ is in $T(S)$.

**Phase 1** [Check whether $y$ is well formed]. $S'$ can easily do this phase deterministically using the finite control and without using the stack. If successful, $S'$ goes to Phase 2; otherwise $S'$ halts in a nonfinal state.

**Phase 2** [Check whether there exists an $x$ in $(\Sigma - \{\tilde{a}, \tilde{S}\})*$ such that $h_k(y) = (\tilde{a}x\tilde{S})^m$ for some $m \geq 1$]. Since $y$ is well-formed, it can be written as $\tilde{a}_1 y_1 \tilde{S}_1 \tilde{a}_2 y_2 \tilde{S}_2 \cdots \tilde{a}_m y_m \tilde{S}_m$ for some $m \geq 1$. We shall often refer to a $\tilde{a}_i y_i \tilde{S}_i$ as an input block and use the symbol $A_i$ to denote an input block. If we are interested in the $i$-th block, we write $A_i$. $S'$ first
copies $h_k(A_1)$ on the stack to the right of $Z_0'$. Let $h_k(A_1) = \epsilon x \delta$. Then it compares for each $1 \leq i \leq m$, the contents of the stack (which is $\epsilon x \delta$) with $h_k(A_i)$. It is clear that $S'$ can do this phase deterministically without having to erase on the stack. If successful, $S'$ goes to the next phase; otherwise, $S'$ halts in a nonfinal state.

**Phase 3 [Initialization of the stack].** $S'$ writes on the stack the symbol $*$ followed by $n (= | \epsilon x \delta |)$ copies of $y Z_0$. This is easily done by $S'$ deterministically. Thus, at the end of this phase, the stack will contain $Z_0' \epsilon x \delta y Z_0 y Z_0 \cdots y Z_0 b b \cdots$, where the number of $Z_0$'s to the right of $*$ is $n$. $S'$ then goes to Phase 4.

**Notation.** It will be seen later that the stack contents at any time is of the form

$$Z_0' \epsilon x \delta y Z_1 y Z_1^* \Delta y Z_2^* \Delta y Z_2 \cdots y Z_l y Z_l \cdots y Z_l y Z_l^* \Delta y Z_l$$

where $l \geq 1$, $Z_1 = Z_0$, the number of $Z_i$'s in $B_i = y Z_i \cdots y Z_i$ is equal to $n (= | \epsilon x \delta |)$, and $a_j, b_j$ are positive integers. A string of $a_j$'s followed by the symbol $\Delta$ and then followed by a string of $b_j$'s arises during the writing operation as we shall see later. The $B_i$'s will be called stack blocks. An arbitrary stack block will be denoted by $B$.

**Phase 4 [Representation of the initial configuration of $S$].** This phase has three steps. It prepares $S'$ for the simulation of $S$ on input $\epsilon x \delta$.

**Step 1.** $S'$ positions its input head on the first symbol of the first input block $A_i = \epsilon_i y_i \delta_i$ for which $g_k(\epsilon_i) = (1, \ldots, 1)$. This represents the fact that the first $k - 1$ input heads of $S$ are on the first symbol of the input $\epsilon x \delta$.

**Step 2.** $S'$ moves its stack head to the first (counting from the left) $Z_1$ in the stack block $B_1$. Note that since $Z_1 = Z_0$, the stack head of $S'$ is on the initial stack symbol, $Z_0$, of $S$. The stack head being on the first $Z_1$ of block $B_1$ indicates that the $k$-th input head of $S$ is scanning the first symbol of input $\epsilon x \delta$.

**Step 3.** The sequence of states that $S$ goes through during the computation is recorded in the finite control of $S'$. In addition, the finite control of $S'$ must remember whether or not $S$ should be scanning a blank. If $S$ should be scanning a blank in some state $q$, $S'$ records the tuple $(q, 0)$ in its finite control; otherwise, $S'$ records the tuple $(q, 1)$. $S'$ records the tuple $(q_0, 0)$ indicating the fact that $S$ is in its initial state and that the symbol that $S$ should be scanning is a blank. $S'$ then goes to Phase 5.

**Phase 5 [Simulation of $S$].** To simplify our discussion, we introduce the following notation: If $B$ is one of the stack blocks of the form $y Z y Z \cdots y Z$ and the stack head is on $B$ scanning one of the $Z$'s, $\text{Stack}(B, Z)$ will denote the number of $Z$'s from the left end of $B$ up to and including the one being scanned by the stack head. Thus,
if the stack head is on the first \(Z\) of \(B\), \(\text{Stack}(B, Z) = 1\). If \(A\) is one of the input blocks of the form \(\hat{\gamma}_1 y_1 \hat{\delta} (1 \leq i \leq m)\), and the input head is on \(A\) scanning one of the symbols of \(A\), then \(\text{Input}(A)\) will denote the number of symbols from the left end of \(A\) up to and including the symbol being scanned by the input head. Finally, if the input head is on an input block \(A\), \(\text{Block}(\hat{\gamma} y \hat{\delta}, A)\) will denote the number of input blocks from the left end of \(\hat{\gamma} y \hat{\delta}\) up to and including the block \(A\) on which the input head is on. Thus, if \(A = A_1\), then \(\text{Block}(\hat{\gamma} y \hat{\delta}, A) = 1\).

Before we describe the simulation process, we mention two tasks which \(S'\) will frequently do during the simulation of \(S\).

**Task 1.** If the input head is on block \(A\) and the stack head is on block \(B\), scanning one of the \(Z\)'s with \(\text{Stack}(B, Z) = r\), then \(S'\) can alter the positions of the input and stack heads so that at the end of the process, \(\text{Input}(A) = r\) and the stack head is on the first symbol of \(B\).

To perform the above task, \(S'\) moves its input head to the symbol directly to the left of \(A\). Then \(S'\) moves its input head one cell right for each \(Z\) encountered by the stack head while moving left on \(B\) (including the initial \(Z\) scanned by the stack head) until the stack head encounters an \(\ast\). Then the stack head is moved one cell right to get it back to the first symbol of block \(B\).

**Task 2.** If the input head is on block \(A\) with \(\text{Input}(A) = r\) and the stack head is on block \(B\), then \(S'\) can alter the positions of the input and stack heads so that at the end of the process, \(\text{Stack}(B, Z) = r\) and the input head is on the first symbol of \(A\).

To do the above task, \(S'\) first moves its stack head left until it encounters an \(\ast\). Then \(S'\) moves its stack head right to the next \(Z\) for every symbol encountered by the input head while moving left on \(A\) (including the symbol initially scanned by the input head) until the input head steps off the left end of \(A\). \(S'\) then moves the input head one cell right to get it back to the first symbol of block \(A\).

We now describe the simulation process. We may assume that \(S'\) has just simulated a move of \(S\) and has recorded in its finite state control the state of \(S\) in the form of a tuple \((q, j), j \in \{0, 1\}\). We may also assume that the stack head is on some stack block, say, \(B\) scanning the \(r\)-th \(Z\) of that block (thus, \(\text{Stack}(B, Z) = r\)). This represents the fact that the \(k\)-th head of \(S\) is on the \(r\)-th symbol of \(\hat{\gamma} x \hat{\delta}\). The finite control of \(S'\) records the \(Z\). Assume further that the input head of \(S'\) is on the first symbol of some input block, say, \(A\) (\(A\) represents the configuration of the first \((k - 1)\) heads of \(S\)). The following steps are taken.

**Step 1.** \(S'\) examines the tuple \((q, j)\). If \(q\) is a final state of \(S\), then \(S'\) halts in a final state. Otherwise, Step 2 is executed.

**Step 2.** \(S'\) determines the \(r\)-th symbol of \(\hat{\gamma} x \hat{\delta}\) which is supposed to be the symbol scanned by the \(k\)-th head of \(S\). This is done by performing Task 1. At the end of Task 1, the input head of \(S'\) is on the \(r\)-th symbol of \(A\). Let this symbol be \(\hat{\gamma}_k\) and
let \( h_k(\delta_k) = \sigma_k \). The finite state control records \( \sigma_k \). \( S' \) then performs Task 2, so that at the end of the process, the input head is on the first symbol of \( A \) and the stack head is on block \( B \) with Stack(\( B, Z \)) = \( r \).

**Step 3.** \( S' \) determines the symbols that are supposed to be scanned by the first \((k - 1)\) heads of \( S \). This is easily done by \( S' \) by having its input head go through block \( A \) and applying the homomorphisms \( h_k \) and \( g_k \) to each symbol of block \( A \) and picking up the appropriate symbols \( \sigma_1, \ldots, \sigma_{k-1} \) that the first \((k - 1)\) input heads of \( S \) should be scanning. The finite control records these symbols and the input head is positioned back to the first symbol of \( A \). Note that to perform Step 3, the stack head is never moved.

**Step 4.** \( S' \) examines in its finite control the previous state of \( S \) which it has recorded in the form of \((q, j)\). We consider two cases.

**Case 1.** \( j = 0 \) [This means that stack head of \( S \) is supposed to be scanning a blank]. \( S' \) determines the next move of \( S \) when \( S \) is in state \( q \), its \( k \) input heads scanning \( \sigma_1, \ldots, \sigma_k \), respectively, and its stack head is scanning a blank with \( Z \) directly to its left. Thus, suppose \( (p, d_1, \ldots, d_k, X) \) is in \( \delta_b(q, \sigma_1, \ldots, \sigma_k, Z) \). \( S' \) records in its finite control the following information: \( p, d_1, \ldots, d_k, \) and \( X \). If \( p \) is a final state of \( S \), then \( S' \) halts in a final state. Otherwise, \( S' \) performs the following actions depending on the case:

(a) If \( X = -1 \) or \( 0 \), then \( S' \) records in its finite control the next state of \( S \) in the form \((p, 1)\) or \((p, 0)\), respectively. Then \( S' \) executes Step 5.

(b) If \( X = E \), then \( S' \) records in its finite control the next state of \( S \) in the form \((p, 0)\). Then \( S' \) performs Task 1. Thus, at the end of Task 1, Input(\( A \)) = \( r \) and Stack(\( B, Z \)) = \( 1 \). Let \( B' = yZ_1yZ_2 \cdots yZ_n \) be the stack block directly to the left of \( B \). (Note that \( B' \) always exists since block \( B_1 = yZ_0yZ_0 \cdots yZ_0 \) is never erased.) \( S' \) without moving its input head, erases all symbols to the right of \( B' \) (thus erasing block \( B \)) and positions its stack head on the first symbol of \( B' \). Then \( S' \) performs Task 2 so that at the end of Task 2, Stack(\( B', Z' \)) = \( r \). \( S' \) then executes Step 5.

(c) If \( X = Z' \), then \( S' \) records in its finite control the state of \( S \) in the form \((p, 0)\). Then \( S' \) performs Task 1 so that at the end of Task 1, Input(\( A \)) = \( r \). Using the input head, \( S' \) writes \( * \) to the stack to the right of block \( B \) and then positions the input head on the first symbol of \( A \). Suppose Block(\( \delta_bS, A \)) is equal to \( s \). With the aid of the input head \( S' \) now writes \( * \) to the right of \( A \). Then \( S' \) writes the new block \( B' = yZ_1yZ_2 \cdots yZ_n \) on the stack to the right of the last \( * \) (the number of \( Z' \) in \( B' \) is equal to \( n = | \delta_bS | \)). The writing of \( B' \) is easily done by \( S' \) deterministically using the input head. Having written \( B' \), \( S' \) now uses the \( * \)'s to the right of \( A \) to locate block \( A \) and positions the input head on the symbol directly to the left of this block. Then \( S' \) uses the \( \times \)'s to the left of \( A \) to move the input head \( r \) cells to the right. Finally, \( S' \) moves its stack head to the first symbol of the new block \( B' \) and performs Task 2, thus, making Stack(\( B', Z' \)) = \( r \). \( S' \) then executes Step 5.
Case 2. \( j = 1 \) [This means that the stack head of \( S \) is not supposed to be scanning a blank]. \( S' \) determines the next move of \( S \) when \( S \) is in state \( q \), its \( k \) input heads scanning \( \sigma_1 \),..., \( \sigma_k \), respectively, and its stack head is scanning \( Z \). Suppose \( (p, d_1 \ldots, d_k, e) \) is in \( \delta(q, \sigma_1 \ldots, \sigma_k, Z) \). \( S' \) records in its finite control the following information: \( p, d_1 \ldots, d_k, \) and \( e \). If \( p \) is a final state of \( S \), then \( S' \) halts in a final state.

Otherwise, \( S' \) performs the following actions depending on the case:

(a) If \( e = 0 \), then \( S' \) records in its finite control the next state of \( S \) in the form \( (p, 1) \). Then \( S' \) executes Step 5.

(b) If \( e = -1 \) (respectively, +1), \( S' \) records in its finite control the next state of \( S \) in the form \( (p, 1) \). Then \( S' \) performs Task 1. Thus at the end of Task 1, Input \( (A) = r \) and Stack\((B, Z) = 1 \). Let \( B' = yZ' \cdots yZ' \) be the stack block directly to the left (respectively, to the right) of \( B \). \( S' \), without moving its input head moves its stack head to the first symbol of \( B' \). (Note that there is always a block to the left of \( B \) since the stack head of \( S \) can never move left off \( Z_0 \). On the other hand, it is possible that there is no block to the right of \( B \), i.e., \( B \) is the top stack block. In this case, \( S' \) just moves its stack head back to the first symbol of \( B \) and changes the state \( (p, 1) \) it has recorded in its finite control to the form \( (p, 0) \).)

Step 5. In this step, \( S' \) carries out the moves \( d_1 \ldots, d_k \) of the \( k \) input heads of \( S \). It is always the case that when this step is entered, the input head of \( S' \) is on input block \( A \) (representing the old configuration of the first \( k-1 \) input heads of \( S \) on \( \epsilon x S ') \) and the stack head is on some stack block, say, \( B = yZ \cdots yZ' \) scanning one of the \( Z \)'s (\( B \) is the block after the actions of the stack head of \( S \) has been simulated with Stack\((B, Z) \) representing the old position of the \( k \)-th input head of \( S \) on \( \epsilon x S ') \). \( S' \) performs the following actions:

\( S' \) moves its stack head left to the next \( Z \) if \( d_k = -1, \) right to the next \( Z \) if \( d_k = +1, \) and does not move if \( d_k = 0. \)

\( S' \) then moves its stack head left and finds the leftmost symbol of \( y \) (\( y \) being directly to the left of the \( Z \) chosen). Note that \( y = A_1 A_2 \cdots A_m \). With the aid of the input head on input block \( A \), the stack head of \( S' \) searches for the first \( A_i \) which would represent the configuration of the first \( (k-1) \) heads of \( S \) when the configuration represented by \( A \) is modified by the moves \( d_1 \ldots, d_{k-1} \). (Note that this process can be done deterministically.) If no such \( A_i \) exists, then \( S' \) halts in a nonfinal state. If \( A_i \) is found \( (A_i \) always exists if \( y \) is equal to \( \phi_s(\epsilon x S ') \)), \( S' \) then moves its input head to the leftmost symbol of the input \( \epsilon y S' = \epsilon A_1 \cdots A_m S'. \) With the aid of the stack head which is on \( A_i \), the input head searches for the first \( A_j \) in \( \epsilon A_1 \cdots A_m S' \) equalling \( A_i \) on the stack. Again, this can be done deterministically. When \( A_j \) is found, \( S' \) positions its input head on the first symbol of \( A_j \) and then moves its stack head right to the first \( Z \) it encounters. Phase 5 is then repeated.

It is clear that \( S' \) satisfies the property stated in the lemma.
We now show that Lemma 2.2 holds for the case of k-CSA (i.e., row 5 of Table III). However, we find it more convenient to prove the lemma when \( S \) is a k-2FA.

**Lemma 2.3.** Let \( k \geq 2 \) and \( S = \langle k, K, \Sigma, \Gamma, \delta, q_0, Z_0, Y_0, F \rangle \) be a k-2FA. Then there exists a CSA \( S' \) with input alphabet \( \Sigma_k \cup \{ \bar{e}, \bar{s} \} \) (\( \bar{e} \) and \( \bar{s} \) are new symbols not in \( \Sigma_k \) which serve as end markers of inputs to \( S' \)) and \( S' \) has the following property: For each \( \xi \in \Sigma \) in \( \xi(\Sigma - \{ \bar{e}, \bar{s} \})^*S \), \( \phi_0(\xi S)S \) is in \( T(S') \) if and only if \( (\xi S, Z_0^0, Y_0) \) is in \( T(S) \) for some \( Z_0^0, Y_0 \) in \( Z_0(\Sigma - \{ Z_0, Y_0 \})^*Y_0 \).

**Proof.** Let \( S' = \langle 1, K', \Sigma_k \cup \{ \bar{e}, \bar{s} \}, \Gamma', \delta', q_0', Z_0', b', F' \rangle \), where \( \Gamma' = \Gamma \cup \Sigma \cup \Sigma_k \cup \{ Z_0', \bar{e}, \bar{s} \} \) and we assume that \( \Sigma, \Sigma_k \), and \( \Gamma \) are pairwise disjoint and \( Z_0 \) and \( \bar{e}, \bar{s} \) are new symbols not in \( \Gamma \cup \Sigma \cup \Sigma_k \). We describe informally the operation of \( S' \) on input \( \xi \) in five phases:

**Phase 1** [Check whether \( y \) is well-formed]. This phase is similar to Phase 1 of Lemma 2.2.}

**Phase 2** [Initialization of the stack]. \( S' \) carries out the following steps:

**Step 1.** \( S' \) nondeterministically generates to the right of \( Z_0' \) on the stack (with the aid of the input head) a word of the form \( \xi S \ast \bar{e} i \bar{e} j \bar{e} k \) for some \( i \geq 2, j \geq 1, k \geq 1 \). If successful, \( S' \) goes to the next step; otherwise, \( S' \) halts in a nonfinal state.

**Step 2.** \( S' \) checks that \( Z_1 \ast Z_2 \ast \cdots \ast Z_l \ast Z_0 = Z_0 \) for some \( l \geq 2 \), \( t_i \geq 1 \). If successful, \( S' \) goes to the next step; otherwise, \( S' \) halts in a nonfinal state.

**Step 3.** [Since \( y \) is well-formed, it can be written as \( e_1 y_1 e_2 y_2 e_3 y_3 \cdots e_m y_m e_m \) for some \( m \geq 1 \).] \( S' \) checks that \( h_y(A_1) = \xi S \). If successful, \( S' \) goes to the next step; otherwise, \( S' \) halts in a nonfinal state.

**Step 4.** \( S' \) checks that \( t_1 = t_2 = \cdots = t_l = | \xi S | \). This is easily done with the aid of the input head. If successful, \( S' \) goes to the next phase; otherwise, \( S' \) halts in a nonfinal state.

**Phase 3.** [At the end of Phase 2, the stack will contain a word of the form]

\[
Z_0' \xi S \ast \bar{e} i \bar{e} j \bar{e} k \ast \bar{e} i \bar{e} j \bar{e} k \ast \bar{e} i \bar{e} j \bar{e} k \ast \cdots \ast \bar{e} i \bar{e} j \bar{e} k \ast \bar{e} i \bar{e} j \bar{e} k, \]

where \( l \geq 2, Z_1 = Z_0, Z_1 = Y_0, Z_i \) in \( (\Gamma - \{ Z_0, Y_0 \}) \) for \( 1 < i < l \), and the number of \( Z_i \)'s in \( B_i = yZ_i \cdots yZ_i \) is equal to \( | \xi S | \). \( S' \) checks that the input blocks \( A_1, A_2, \ldots, A_m \) are such that \( h_y(A_1) = h_y(A_2) = \cdots = h_y(A_m) = \xi S \). If successful, \( S' \) goes to Phase 4; otherwise, \( S' \) halts in a nonfinal state.

**Phase 4** [Representation of the initial configuration of \( S \)]. It is the same as Phase 4.
of Lemma 2.2 except that now the sequence of states that \( S \) goes through during the computation is recorded in the finite state control of \( S' \) in the form of \((q, 1)\) since \( S \) can never scan a blank on its stack (note that by our convention, we refer to the second tape of a \( k \)-2FA as a stack). Thus, the initial state of \( S \) recorded in the finite control of \( S' \) is \((q_0, 1)\).

**Phase 5** [Simulation of \( S \)]. It is the same as Phase 5 of Lemma 2.2 except that now the states of \( S \) as recorded in the finite control of \( S' \) will be of the form \((q, 1)\). Thus, Step 4, Case 1 of that phase is not applicable.

It is easy to verify that \( S' \) as defined above is a CSA and that \( S' \) has the property stated in the lemma.

**Corollary 2.2.** Let \( k \geq 2 \) and \( S = \langle k, K, \Sigma, \Gamma, \delta, \delta_0, q_0, Z_0, b, F \rangle \) be a \( k \)-CSA. Then there exists a CSA \( S' \) with input alphabet \( \Sigma \cup \{ \xi, \xi' \} \), and \( S' \) has the following property: For each \( \xi x S \) in \( \epsilon(\Sigma - \{ q, \xi' \})^* S, \xi \phi_b(\xi x S) S \) is in \( T(S') \) if and only if \( \xi x S \) is in \( T(S) \).

**Proof.** This follows from Lemma 2.3 and Theorem 2.1.

We are now ready to prove the main result of the section.

**Theorem 2.3.** Let \( k \geq 1 \). A set is accepted by a machine \( S \) in row \( I \) \((1 \leq I \leq 5)\) of the first column of Table III if and only if it is accepted by a machine \( U' \) in row \( I \) of the fourth column.

**Proof.** By Theorem 2.2, we need only prove the "only if" part and by Theorem 1.2 and Corollary 2.1, we need only take care of the case when \( k \geq 2 \). Given a machine \( S = \langle k, K, \Sigma, \Gamma, \delta, \delta_0, q_0, Z_0, b, F \rangle \) in row \( I \) \((1 \leq I \leq 5)\) of the first column of Table III, there exists a machine \( S' \) in row \( I \) of the second column whose input alphabet is \( \Sigma_k \cup \{ \xi, \xi' \} \) (\( \xi \) and \( \xi' \) are new symbols not in \( \Sigma \) which serve as end markers of inputs to \( S' \)) with the following property: For each \( \xi x S \) in \( \epsilon(\Sigma - \{ q, \xi' \})^* S, \xi \phi_b(\xi x S) S \) is in \( T(S') \) if and only if \( \xi x S \) is in \( T(S) \) (by Lemma 2.2 and Corollary 2.2). Then by Theorem 1.2 and Corollary 2.1, there exists a machine \( U \) in row \( I \) of the third column such that \( T(U) = T(S') \). The desired machine \( U' \) in row \( I \) of the fourth column is constructed as follows. \( U' \) has two storage tapes and operates as follows: Given an input \( \xi x S \) in \( \epsilon(\Sigma - \{ q, \xi' \})^* S, U' \) first generates \( \xi \phi_b(\xi x S) S \) on its first storage tape without using its second storage tape. It is easy to verify that \( U' \) can do this step deterministically and using exactly \( | \xi \phi_b(\xi x S) S | \) cells in its first storage tape. Then \( U' \) simulates \( U \) on the word \( \xi \phi_b(\xi x S) S \) it has generated on its first storage tape by using its second storage tape to correspond to the single storage tape of \( U \). \( U' \) goes to a final state if and only if \( U \) goes to a final state during the simulation. Clearly, \( T(U') = T(S) \) and by Theorem 1.1 the two storage tapes of \( U' \) can be reduced to one. This completes the proof of the theorem.

**Remark.** The "only if" part of Theorem 2.3 could have been proven directly for
the cases of $k$-SA, $k$-DSA, $k$-NESA, $k$-NEDSA using arguments similar to the ones in [4, 7]. Our method of proof was chosen to present new techniques which seem to be applicable to similar types of problems concerning other types of automata. For example, the same techniques are used to characterize the computational power of auxiliary stack automata in Section 3.

The next result follows from Theorems 2.1 and 2.3.

**Corollary 2.3.** Let $k \geq 1$. Then the following statements are equivalent for any set $L$:

1. $L$ is accepted by a $n^{2k}$-tape bounded TM.
2. $L$ is accepted by $k$-NESA.
3. $L$ is accepted by a $2k$-CSA.
4. $L$ is the domain of some relation accepted by a $2k$-2FA.

**Corollary 2.4.** Let $k \geq 1$. A set is accepted by a $k$-SA (respectively, $k$-DSA) if and only if it is accepted by a DTM of time complexity $2^{cn^a}$ (respectively, $2^{cn^{logn}}$) for some constant $c$.

**Proof.** Follows from Theorem 2.3 and Theorem 1.2(e).

Several other corollaries follow from the tape and time hierarchy results in [9, 10] and Theorem 1.2(f). We mention a few.

**Corollary 2.5.** Let $k \geq 1$. Then the class of sets accepted by $(k + 1)$-SA's (respectively, $(k + 1)$-DSA's, $(k + 1)$-NEDSA's) properly contains the class of sets accepted by $k$-SA's (respectively, $k$-DSA's, $k$-NEDSA's).

**Corollary 2.6.** Let $k \geq 1$. Then the class of sets accepted by $(2k + 1)$-NESA's (respectively, $(2k + 1)$-CSA's) properly contains the class of sets accepted by $k$-NESA's (respectively, $k$-CSA's).

**Corollary 2.7.** Let $k \geq 1$. Then the class of sets accepted by $(k + 1)$-SA's (respectively, $(k + 1)$-NESA's) properly contains the class of sets accepted by $2k$-DSA's (respectively, $2k$-NEDSA's).

We conclude this section with the following open problem: Is there a class of tape bounded (or time bounded) Turing machines which will characterize the class of $k$-CDSA's?

### 3. Auxiliary Stack Automata

In this section, we shall use the techniques of Section 2 to characterize some tape and time complexity classes of Turing machines in terms of the different classes of auxiliary stack automata that we have introduced.
We begin with the following lemma.

**Lemma 3.1.** Let \( c \) be a nonnegative integer and \( U \) be a \( 2^{cL(n)} \)-tape bounded aux DPDA (respectively, \( 2^{cL(n)} \)-tape bounded DTM) with input alphabet \( \Sigma \). Then there exists a \( n \)-tape bounded aux DPDA (respectively, \( n \)-tape bounded DTM) \( U' \) with input alphabet \( \Sigma \cup \{ \xi, \zeta \} \) (\( \xi \) and \( \zeta \) are new symbols not in \( \Sigma \) which serve as end markers of inputs to \( U' \)) such that \( T(U') = \{ \xi(\xi S)\gamma^m \xi S \mid m \geq 1, \xi S \text{ in } \xi(\Sigma - \{ \xi, \zeta \})^*S \} \), \( U \) with input \( \xi S \) has a sequence of moves leading to the acceptance of \( \xi S \) and in which \( U \) uses at most \( |\xi S| \cdot 2^{cm} \) cells in its storage tape. Thus, for each word \( \xi S \) accepted by \( U \), there are infinitely many \( m \) for which \( \xi(\xi S)\gamma^m \xi S \) is accepted by \( U' \).

**Proof.** The proof is similar to the proof of Lemma 2.1. By Theorem 1.1, it suffices to construct \( U' \) with three storage tapes. We describe briefly the operation of \( U' \) when given an input \( \xi S \) in \( \xi \Sigma^*S \).

**Step 1.** \( U' \), operating deterministically and using only its first storage tape, checks that the input is of the form \( \xi(\xi S)'\gamma^m \xi S \) for some \( \xi S \) in \( \xi(\Sigma - \{ \xi, \zeta \})^*S \) and \( j \geq 1 \). \( U' \) can carry out this step using no more than \( |\xi(\xi S)'\gamma^m \xi S| \) cells in the first storage tape. If successful, \( U' \) goes to the next step; otherwise, \( U' \) halts in a nonfinal state.

**Step 2.** \( U' \) now uses its second storage tape to check that \( j = 2^{cm} \) for some \( m \geq 1 \). This step is easily done deterministically using no more than \( |\xi(\xi S)'\gamma^m \xi S| \) cells in the second storage tape. (Essentially, \( U' \) uses \( 2^c \) symbols in a space of \( m \) cells, starting with \( m = 1 \), to count the number of \( \xi S \) blocks in the input). If successful, \( U' \) goes to the next step; otherwise, \( U' \) halts in a nonfinal state.

**Step 3.** \( U' \) "measures" \( |\xi(\xi S)'\gamma^m \xi S| \) cells on its third storage tape using some left and right end markers, say \( \alpha \) and \( \beta \), respectively. (Thus at the end of the process, the third storage tape will contain \( \alpha b \cdots b \beta \beta \), where the number of blanks \( b \) between \( \alpha \) and \( \beta \) is \( |(\xi S)'\gamma^m| \)). \( U' \) then simulates the actions of \( U \) on the first \( \xi S \) block of the input, using the cells bounded by the markers \( \alpha \) and \( \beta \) to correspond to the storage cells of \( U \). If, during the simulation, \( U \) enters a final state, then \( U' \) halts in a final state. If, during the process of simulation, the storage head of \( U' \) encounters the right end marker \( \beta \), then \( U' \) halts in a nonfinal state. (Note that since the storage head of \( U \) cannot move left of its initial position, the storage head of \( U' \) can never encounter \( \alpha \) during the simulation of \( U \)).

It is clear that \( U' \), operating as described, accepts the desired set and that \( U' \) has the desired tape bound.

**Theorem 3.1.** Let \( c \) be a nonnegative integer and \( U \) be a \( 2^{cL(n)} \)-tape bounded aux DPDA (respectively, \( 2^{cL(n)} \)-tape bounded DTM). Then there exists a \( L(n) \)-tape bounded aux DSA (respectively, \( L(n) \)-tape bounded aux NEDSA) \( S' \) such that \( T(S') = T(U) \).
Proof. The proof is similar to that of Theorem 2.2. By Theorem 1.1, it suffices to construct $S'$ with two storage tapes. Given a $2^{cL(n)}$-tape bounded aux DPDA (respectively, $2^{cL(n)}$-tape bounded DTM) $U$ with input alphabet $\Sigma$, let $U'$ be as defined in Lemma 3.1. Since $U'$ is $n$-tape bounded, there exists a DSA (respectively, NEDSA) $S$ such that $T(S) = T(U')$ by Theorem 1.2. Thus, $T(S) = \{\varepsilon(xS)^{2^m} | m \geq 1, \varepsilon xS \in (\Sigma - \{\varepsilon, S\})^* S, U \text{ with input } \varepsilon xS \text{ has a sequence of moves leading to the acceptance of } \varepsilon xS \text{ and in which } U \text{ uses at most } |\varepsilon xS| \cdot 2^m \text{ cells in its storage tape}\}$. We may assume without loss of generality that $S$ halts for all inputs (i.e., every sequence of moves leads to a halting configuration) [11]. We shall construct the desired $S'$ from $S$. We describe informally the operation of $S'$ when given an input $\varepsilon xS$ in $\varepsilon(\Sigma - \{\varepsilon, S\})^* S$. The input head of $S'$ and its stack head will simulate the actions of the input head and the stack head of $S$ when $S$ is operating on an input of the form $\varepsilon(\varepsilon xS)^{2^m} S$. For a fixed value of $m$, $S'$ has to simulate the actions of $S$ on input $\varepsilon(xS)^{2^m} S$. In order for $S'$ to completely simulate $S$, it must be able to tell at any point in time which of the $2^m$ blocks of $\varepsilon xS$ the input head of $S$ is on. However, it is clear that for a fixed value of $m$, $2^c$ symbols in a space of $m$ cells in the first storage tape can indicate a count of from 1 to $2^m$. Thus, by using the first storage tape as a counter, $S'$ is able to tell at any point in time the block on which the input head of $S$ is on. Hence, for a fixed $m$, simulation of the actions of $S$ when $S$ is operating on input $\varepsilon(\varepsilon xS)^{2^m} S$ can be done by $S'$ using no more than $m$ cells in its first storage tape. We now describe the overall operation of $S'$. The second storage tape of $S'$ is used to hold the value of $m$ in unary notation. $S'$ starts out with a value of $m$ equal to 1 and simulates $S$ as described above. Thus, initially, $S'$ will simulate the actions of $S$ when $S$ is operating on input $\varepsilon(\varepsilon xS)^{2^m} S$. If, during the simulation, $S$ enters a final state, then $S'$ also enters a final state. When $S$ halts (by assumption, $S$ halts for all inputs), $S'$ examines the state of $S$. If it is a final state, $S'$ halts in a final state. If the state is nonfinal, $S'$ increments the value of $m$ in the second storage tape by 1 and $S'$ does the whole process of simulating the actions of $S$ on $\varepsilon(\varepsilon xS)^{2^m} S$ but this time, for the new value of $m$. (Of course, the old stack contents of $S'$ generated in the simulation of $S$ for the old value of $m$, which is now "garbage", is not erased and the new "developing" stack contents will be on top of the old one. Moreover, the starting position of the $m$ cells in the first storage tape used to indicate a count of from 1 to $2^m$ is always the same for any $m$.) It is clear that $S'$ operating as described accepts exactly the set accepted by $U$. Since for a given $m$, $S'$ uses no more than $m$ cells in any of its two storage tapes to simulate the actions of $S$ when $S$ is operating on input $\varepsilon(\varepsilon xS)^{2^m} S$, it follows that if $\varepsilon xS$ is accepted by $U$, $S'$ would accept $\varepsilon xS$ by a sequence of moves using no more than $L(\varepsilon xS)$ cells in any of its two storage tapes. Thus, $S'$ in $L(n)$-tape bounded.

Corollary 3.1. Let $c$ be a nonnegative integer and $U$ be a $2^{cL(n)}$-tape bounded DTM. Then there exists a $L(n)$-tape bounded aux CSA $S'$ such that $T(S') = T(U)$.
Proof. Given U with input alphabet \( \Sigma \), let \( U' \) be as defined in Lemma 3.1. Since \( U' \) is \( n \)-tape bounded, there exists a 2FA \( S = \langle 1, K, \Sigma \cup \{\xi, S\}, \Gamma, \delta, Z_0, Y_0, F \rangle \) such that \( \text{Domain}(T(S)) = T(U') \) by Theorem 1.2(d). Moreover, we may assume without loss of generality that \( S \) halts for all pairs of words in \( \xi \Sigma^* S \times Z_0(\Gamma - \{Z_0, Y_0\})^* Y_0 \). (This follows from the constructions in [6] and the fact that a \( n \)-tape bounded \( TM \) can be made halting.) The desired \( L(n) \)-tape bounded aux CSA \( S' \) will be constructed from \( S' \). Again, it suffices to construct \( S' \) with two storage tapes. The technique for constructing \( S' \) is similar to the construction of \( S' \) in Theorem 3.1. Let \( Z_0' \) be the initial stack symbol of \( S' \) where \( Z_0' \) is not in \( \Gamma \). \( S' \) first generates a word of the form \( Z_0w_1Y_0Z_0w_2Y_0 \cdots Z_0w_iY_0 \Delta \) to the right of \( Z_0' \) in the stack, where \( i \geq 1 \), \( w_i \) in \( (\Gamma - \{Z_0, Y_0\})^* \), and \( \Delta \) is a new symbol not in \( \Gamma \). Then \( S' \) moves its stack head back to \( Z_0' \). The simulation of \( S \) is then carried out as in the proof of Theorem 3.1 except that now the simulation for any value of \( m \) must start with the stack head of \( S' \) on the leftmost symbol of \( Z_0w_mY_0 \). \( S' \) is also designed to halt in a nonfinal state should its stack head encounter \( \Delta \).

We shall prove the converse of Theorem 3.1 but, first, we introduce the following notation:

**Notation.** Let \( \Sigma \) be a finite nonempty set which includes the symbols \( \xi \) and \( S \) and \( W \) be a finite nonempty set disjoint from \( \Sigma \). For each \( a \) in \( (\Sigma \cup W) \), let \((0, a)\) and \((1, a)\) be abstract symbols. Let \( \hat{\Sigma} = \{(i, a) \mid a \text{ in } \Sigma, \ i = 0, 1\} \) and \( \hat{W} = \{(i, a) \mid a \text{ in } W, \ i = 0, 1\} \). Any element of the form \((i, a)\) in \( \hat{\Sigma} \) will be denoted by \( \hat{a} \). Let \( h \) be a homomorphism of \( (\hat{\Sigma} \cup \hat{W})^* \) into \( (\Sigma \cup W)^* \) defined by \( h((i, a)) = a \) for each \((i, a)\) in \( (\hat{\Sigma} \cup \hat{W})^* \). Let \( g \) be a homomorphism of \( (\hat{\Sigma} \cup \hat{W})^* \) into the natural numbers defined by \( g(\hat{a}) = 0 \), \( g((i, a)) = i \) for each \((i, a)\) in \( \hat{\Sigma} \), and \( g(xy) = g(x) + g(y) \) for \( x, y \) in \( (\hat{\Sigma} \cup \hat{W})^* \). Let \( \phi \) be in \( (\Sigma - \{\xi, S\})^* S \), \( \phi x \) in \( W^* W \), \( \omega \) in \( W^e \). Let \( i \) and \( j \) be positive integers such that \( 1 \leq i \leq N \), \( 1 \leq j \leq n \). \( \phi \) will denote the word \( \alpha \) in \( (\hat{\Sigma} \cup \hat{W})^* \) with the following properties:

\[
(1) \quad \alpha = \hat{\alpha}_1 \hat{\alpha}_2 \cdots \hat{\alpha}_n \hat{Z}_1 \cdots \hat{Z}_N \text{ for some } \hat{\alpha}_1, \ldots, \hat{\alpha}_n \in \hat{\Sigma} \text{ and } \hat{Z}_1, \ldots, \hat{Z}_N \in \hat{W}.
\]

\[
(2) \quad h(\omega) = \phi x \omega.
\]

\[
(3) \quad g(\hat{\alpha}_1 \cdots \hat{\alpha}_n) = 1, \quad g(\hat{Z}_1 \cdots \hat{Z}_N) = 1.
\]

\[
(4) \quad g(\hat{\alpha}_j) = 1, \quad g(\hat{Z}_i) = 1.
\]

Clearly, there are \( m = n \cdot N \cdot \mid W^e \mid^N \) distinct words of \( (\hat{\Sigma} \cup \hat{W})^* \) of length \( n + N \) in the set \( \{R_n(\phi x \omega, j, w, i) \mid w \text{ in } W^* W, \mid w \mid = N, \ 1 \leq j \leq n, \ 1 \leq i \leq N\} \). Let \( \alpha_1, \alpha_2, \ldots, \alpha_m \) be the words in the set, ordered lexicographically. We denote by \( \phi_n(\phi x \omega) \) the word \( \alpha_1 \alpha_2 \cdots \alpha_m \). Note that \( \mid \phi_n(\phi x \omega) \mid = n \cdot N \cdot \mid W^e \mid^N \cdot (n + N) \) where \( n = \mid \phi x \omega \mid \).

The significance of the above notation is the following: Given an aux SA (respectively, aux NESA) \( S \) with input alphabet \( \Sigma \) and storage alphabet \( W \) and an input word \( \phi x \omega \) in \( (\Sigma - \{\xi, S\})^* S \), we can represent the input tape configuration (that is, the word
\(\phi x B\) and the input head position on \(\phi x B\) and the storage tape configuration (that is, the contents of the storage tape and the storage head position on the storage tape) for a storage space of \(N\) cells by a word in \((\Sigma \cup \tilde{W})^*\) as follows: If \(\phi x B = \sigma_1 \cdots \sigma_n (n \geq 2)\) with the input head on \(\sigma_1\) and the storage contents in \(N\) cells is \(Z_1 Z_2 \cdots Z_N\) with the storage head on \(Z_1\), then we represent this input tape-storage tape configuration by the word 

\[(0, \sigma_1)(0, \sigma_{n-1})(1, \sigma_j)(0, \sigma_{n+1}) \cdots (0, \sigma_n)(0, Z_1)(0, Z_{n-1}) \cdots (0, Z_N) = R_N(\phi x B, j, Z_1 \cdots Z_N, i).\]

Thus, any input tape-storage tape configuration that \(S\) can enter in the course of processing \(\phi x B\) using only a storage space of \(N\) cells is represented in the word \(\phi_N(\phi x B)\) in \((\Sigma \cup \tilde{W})^*\).

The following lemma is analogous to Lemma 2.2.

**Lemma 3.2.** Let \(S\) be a \(L(n)\)-tape bounded aux \(SA\) (respectively, \(L(n)\)-tape bounded aux \(NESA\)) with input alphabet \(\Sigma\) and storage alphabet \(W\) with \(\Sigma \cap W = \emptyset\). (Note that one of the symbols of \(W\) is a blank.) Then there exists a SA (respectively, NESA) \(S'\) with input alphabet \((\Sigma \cup \tilde{W}) \cup \{\tilde{\ell}, \tilde{\mathcal{S}}\}\) (\(\tilde{\ell}\) and \(\tilde{\mathcal{S}}\) are new symbols not in \(\Sigma \cup \tilde{W}\) which serve as end markers of inputs to \(S'\)) and \(S'\) has the following property: For each \(\phi x B\) in \(\phi(\Sigma - \{\tilde{\ell}, \tilde{\mathcal{S}}\})^*\), \(\phi_N(\phi x B)\) is in \(T(S')\) if and only if \(S\) has a sequence of moves leading to the acceptance of \(\phi x B\) using no more than \(N\) cells in its storage tape. Thus, \(\phi_N(\phi x B)\) is in \(T(S')\) for some \(N \leq L(\phi x B)\) if and only if \(\phi x B\) is in \(T(S)\).

**Proof.** The argument is similar to that of Lemma 2.2, but simpler. We describe the operation of \(S'\) on an input \(\phi y B\) in \(\phi(\Sigma \cup \tilde{W})\), omitting the details.

**Phase 1** [Check the input for proper format]. \(S'\) checks that for some \(m \geq 1, N \geq 1\) \(\phi x B\) in \(\phi(\Sigma - \{\tilde{\ell}, \tilde{\mathcal{S}}\})^*\), \(y, y_1, \ldots, y_m, z_1, \ldots, z_m\) in \(\tilde{W}\) the following are true:

1. \(y = y_1 z_1 y_2 z_2 \cdots y_m z_m\); 2. for \(1 \leq i \leq m, h(y_i) = \phi x B, h(z_i) = N, g(y_i) = 1, g(z_i) = 1\); 3. if \(\phi x B = \sigma_1 \cdots \sigma_n\), then for some \(1 \leq i \leq m, y_i z_i = (1, \sigma_1)(0, \sigma_2) \cdots (0, \sigma_n)(1, b)(0, b) \cdots (0, b),\) where \(b\) stands for the blank symbol, and \(y_i z_i\) represents the initial input tape-storage tape configuration of \(S\).

It is easily verified that \(S'\) can do the above task deterministically by possibly writing some word \(w\) in the stack but without erasing.

In what follows, \(y_1 z_1, \ldots, y_m z_m\) will be called input blocks.

**Phase 2** [Representation of the initial configuration of \(S\)]. \(S'\) writes \(y Z_0\) to the right of whatever it has written previously in the stack and positions its stack head on the top of the stack. \((Z_0\) is the initial stack symbol of \(S\).) Thus, after this process, the stack will contain \(wy Z_0 b b \cdots\). \(S'\) then positions its input head on the first symbol of the first input block (counting from the left) which represents the initial input tape-storage tape configuration of \(S\).

**Phase 3** [Simulation of \(S\)]. It will be seen that the stack content at any time will
be of the form \(wyZ_1\Delta^r_1yZ_2\Delta^r_2 \cdots \Delta^r_k-yZ_kbb\ldots\), where \(\Delta\) is a new symbol and \(Z_1 = Z_0, Z_2, \ldots, Z_k\) are symbols in the stack alphabet of \(S\). We now describe the simulation process. We may assume that the input head of \(S'\) is on the first symbol of some input block which represents the current input tape-storage tape configuration of \(S\) and that the stack head of \(S'\) is either on the top of the stack or on some \(Z_i(1 \leq i \leq k)\). (This represents the current stack configuration of \(S\)) The following steps are taken:

**Step 1.** \(S'\) determines the input symbol and storage tape symbol currently scanned by the input and storage heads of \(S\), respectively.

**Step 2.** \(S'\) determines the move of \(S\). We consider the following cases:

- **Case 1.** \(S\) does not move on the stack. In this case, \(S'\) proceeds to Step 3.
- **Case 2.** \(S\) moves left (right) on the stack. In this case, \(S'\) moves left (right) to \(Z_{i-1} (Z_{i+1})\) and proceeds to Step 3.
- **Case 3.** \(S\) erases on the stack. (This implies that \(S'\) is on the top of the stack). In this case, \(S'\) erases \(\Delta^r_k-yZ_k\) and proceeds to Step 3.
- **Case 4.** \(S\) writes a stack symbol \(Z_{k+1}\) in the stack. (Again, this implies that \(S'\) is on the top of the stack.) In this case, \(S'\) must write \(yZ_{k+1}\) to the right of \(wyZ_1\Delta^r_1yZ_2\Delta^r_2 \cdots \Delta^r_{k-1}yZ_k\). So as not to lose the input head position in the writing process, \(S'\) must first write \(\Delta^r_k\) in the stack indicating that the input head is on the \(r_k\)-th input block. Then \(S'\) writes \(yZ_{k+1}\) on the stack with the aid of the input head, after which the input head is restored to its original position using \(\Delta^r_k\).

**Step 3.** \(S'\) moves the input head to the first input block which represents the next input tape-storage tape configuration of \(S\). This is easily done using a copy of \(y\) preceding each \(Z_i\) in the stack. \(S'\) then executes Phase 3 again, accepting the input if and only if in the process of simulation, \(S\) enters a final state.

We now use Lemma 3.2 to prove the following result:

**Theorem 3.2.** Let \(L(n) \geq \log_2 n\). Let \(S\) be a \(L(n)\)-tape bounded aux SA (respectively, \(L(n)\)-tape bounded aux NESA) with input alphabet \(\Sigma\) and storage alphabet \(W(\Sigma \cap W = \emptyset)\). Then there exists a \(2^cL(n)\)-tape bounded aux DPDA (respectively, \(2^cL(n)\)-tape bounded TM) \(U'\) such that \(T(U') = T(S)\) for some integer \(c\).

**Proof.** By Theorem 1.1, it suffices to construct \(U'\) with three storage tapes. Given a \(L(n)\)-tape bounded aux SA (respectively, \(L(n)\)-tape bounded aux NESA) \(S\), let \(S'\) be a SA (respectively, NESA) as defined in Lemma 3.2. Then by Theorem 1.2 there exists a \(n^2\)-tape bounded aux DPDA (respectively, \(n^2\)-tape bounded TM) \(U\) such that \(T(U) = T(S')\). We may assume without loss of generality that every sequence of moves of \(U\) on any input leads to a halting configuration [4]. We describe the operation of \(U'\) on input \(\phi x \mathcal{S}\) in \(\phi(\Sigma - \{\phi, \mathcal{S}\})^*\mathcal{S}\) informally. Let \(N = 1\). The first storage tape
of $U'$ is used to hold the value of $N$ in unary notation. $U'$ first generates the word $	ilde{\phi}_N(\vartriangleleft S)S$ on its second storage tape. The reader can easily convince himself that $U'$ can carry out this process using only the second storage tape and the value of $N$ and using exactly $|\tilde{\phi}_N(\vartriangleleft S)S|$ cells in the second storage tape. $U'$ then simulates the actions of $U$ on the word $	ilde{\phi}_N(\vartriangleleft S)S$ using its third storage tape to correspond to the storage tape of $U$. If, during the simulation, $U$ enters a final state, then $U'$ also enters a final state. When $U$ halts (by assumption, $U$ halts on any input and any sequence of moves), $U'$ examines the state of $U$. If the state is a final state, $U'$ halts in a final state. If the state is nonfinal, $U'$ increments the value of $N$ in the first storage tape by 1 and replaces the word in the second storage tape by the new word $\tilde{\phi}_N(\vartriangleleft S)S$. (Of course, the first symbol of this word must be on the same cell occupied by the first symbol of the old word.) Then $U'$ again simulates the actions of $U$ on the new word. (It is assumed that the first cell used in the third storage tape in this simulation is the same as the first cell used in the previous simulation.)

Now, $\vartriangleleft S$ is in $T(S)$ if and only if $\tilde{\phi}_N(\vartriangleleft S)S$ is in $T(U)$ for some $N \leq L(|\vartriangleleft S|)$. It follows that $T(U') = T(S)$. If $\vartriangleleft S$ is in $T(S)$, then since $U$ is $n^k$-tape bounded, $U'$ has a sequence of moves leading to the acceptance of $\vartriangleleft S$ using exactly $N$ cells in its first storage tape, $|\tilde{\phi}_N(\vartriangleleft S)S|$ cells in its second storage tape, and no more than $|\tilde{\phi}_N(\vartriangleleft S)S|^k$ cells in its third storage tape for some $N \leq L(|\vartriangleleft S|)$. Since $|\phi_N(\vartriangleleft S)| = n \cdot N \cdot |W|^N \cdot (n + N)$, where $n = |\vartriangleleft S|$, it follows that the bound on the third storage tape of $U'$ is $(n \cdot N \cdot |W|^N \cdot (n + N) + 2)^k$. Since $N \leq L(n)$ and $L(n) \geq \log_2 n$, the bound is less than $2^{cL(n)}$ for some integer $c$. Thus, $U'$ is $2^{cL(n)}$-tape bounded.

We are now ready to state the two main results of this section.

**Theorem 3.3.** Let $L(n) \geq \log_2 n$. Then the following statements are equivalent for any set $L$:

1. $L$ is accepted by a $L(n)$-tape bounded aux DSA.
2. $L$ is accepted by a $L(n)$-tape bounded aux SA.
3. $L$ is accepted by a $2^{cL(n)}$-tape bounded aux DPDA for some constant $c$.
4. $L$ is accepted by a $2^{cL(n)}$-time bounded DTM for some constant $c$.

**Proof.** This follows from Theorems 3.1, 3.2, and 1.2(e).

**Theorem 3.4.** Let $L(n) \geq \log_2 n$. Then the following statements are equivalent for any set $L$:

1. $L$ is accepted by a $L(n)$-tape bounded aux NEDSA.
2. $L$ is accepted by a $L(n)$-tape bounded aux CSA.
3. $L$ is accepted by a $L(n)$-tape bounded aux NESA.
4. $L$ is accepted by a $2^{cL(n)}$-tape bounded DTM for some constant $c$. 
Proof. Follows from Theorems 3.1 and 3.2, Corollary 3.1 and the fact that a set accepted by a $2^{cL(n)}$-tape bounded TM is accepted by a $(2^{cL(n)})^2$-tape bounded DTM (Theorem 1.2(f)).

From Theorems 2.3, 3.3, and 3.4, we get the following corollaries:

**Corollary 3.2. The following statements are equivalent for any set $L$:**

1. $L$ is accepted by a $k$-SA for some $k \geq 1$.
2. $L$ is accepted by a $\log n$-tape bounded aux DSA.
3. $L$ is accepted by a $n^c$-tape bounded aux DPDA for some constant $c$.
4. $L$ is accepted by a $2^{n^c}$-time bounded DTM for some constant $c$.

**Corollary 3.3** The following statements are equivalent for any set $L$:

1. $L$ is accepted by a $k$-NESA for some $k \geq 1$.
2. $L$ is accepted by a $\log n$-tape bounded aux NEDSA.
3. $L$ is accepted by a $n^c$-tape bounded DTM for some constant $c$.

We may define a writing stack automaton (respectively, writing nonerasing stack automaton) as a SA (respectively, NESA) which is capable of rewriting a symbol scanned under its input head by another symbol. We omit the formal definition. It is clear that this device is equivalent to a $n$-tape bounded aux SA (respectively, $n$-tape bounded aux NESA). Thus, we get the following corollary which was first proved in [12] by direct arguments.

**Corollary 3.4.** A set is accepted by a writing stack automaton (respectively, writing nonerasing stack automaton) if and only if it is accepted by a $2^{2^c}$-time bounded DTM (respectively, $2^{2^c}$-tape bounded DTM) for some constant $c$. Moreover, the deterministic and nondeterministic versions of the device have the same computational power.

We now relate the computational power of aux CDSA’s to DTM’s.

**Theorem 3.5.** Let $L(n) \geq \log n$. Then a set is accepted by a $L(n)$-tape bounded aux CDSA if and only if it is accepted by a $L(n)$-tape bounded DTM.

**Proof.** It suffices to show the “only if” part. Suppose $S$ is a $L(n)$-tape bounded aux CDSA. Let $U$ be a DTM with four storage tapes. The first storage tape of $U$ will correspond to the storage tape of $S$. $U$ simulates $S$ without recording the stack but recording the position $i$ of the stack head of $S$ in its second storage tape in binary notation until $S$ leaves the top of the stack. Since $U$ has not recorded the stack, it must resimulate $S$ everytime $S$ moves its stack. To find the symbol written on cell $i$ of the

---

7 This proof is due to Professor Stephen Cook.
stack, \( U \) simulates \( S \) from scratch until \( S \) has written on cell \( i \). The two remaining storage tapes of \( U \) are used for this purpose. It remains to show that \( U \) is \( cL(n) \)-tape bounded for some constant \( c \). We need only check that the second storage tape satisfies this bound. If \( S \) has \( s \) states, \( p \) stack symbols, and \( w \) storage tape symbols, then \( S \) would accept an input of length \( n \) if and only if it accepts it in a sequence of moves in which the stack never grows beyond \( s \cdot p \cdot n \cdot w^{c(n)} \cdot L(n) \) cells. It follows that at most \( cL(n) \) tape cells are used in the second storage tape of \( U \). By standard techniques, \( c \) can be reduced to unity.

**Corollary 3.5.** The following statements are equivalent for any set \( L \):

1. \( L \) is accepted by a two-way multihead deterministic finite-state machine.
2. \( L \) is accepted by a \( k \)-CDSA for some \( k \geq 1 \).
3. \( L \) is accepted by a log \( n \)-tape bounded aux CDSA.
4. \( L \) is accepted by a log \( n \)-tape bounded DTM.

**Proof.** (1) certainly implies (2). It is also clear that any \( k \)-CDSA can be simulated by a log \( n \)-tape bounded aux CDSA. That (3) implies (4) follows from Theorem 3.5. The equivalence of (1) and (4) is a well-known unpublished result of Alan Cobham and others.

**References**

10. F. C. Hennie and R. E. Stearns, Two tape simulation of multitape Turing machines, 
12. J. A. Giuliano, "Writing Stack Acceptors," IEEE Conference Record of the Eleventh 
   Annual Symposium on Switching and Automata Theory, pp. 181–193, Santa Monica, 
   California, 1970.