JOURNAL OF COMPUTER AND SYSTEM SCIENCES 21, 218–235 (1980)

Tree-Size Bounded Alternation*

WALTER L. RUZZO

Department of Computer Science, University of Washington, Seattle, Washington 98195

Received October 15, 1979; revised June 12, 1980

The size of an accepting computation tree of an alternating Turing machine (ATM) is introduced as a complexity measure. We present a number of applications of tree-size to the study of more traditional complexity classes. Tree-size on ATMs is shown to closely correspond to time on nondeterministic TMs and on nondeterministic auxiliary pushdown automata. One application of the later is a useful new characterization of the class of languages log-space-reducible to context-free languages. Surprising relationships with parallel-time complexity are also demonstrated. ATM computations using at most space $S(n)$ and tree-size $Z(n)$ (simultaneously) can be simulated in alternating space $S(n)$ and time $S(n) \cdot \log Z(n)$ (simultaneously). Several well-known simulations, e.g., Savitch's theorem, are special cases of this result. It also leads to improved parallel complexity bounds for many problems in terms of both time and number of "processors." As one example we show that context-free language recognition in time $O(\log^2 n)$ is possible on several parallel models. Further, this bound is achievable with only a polynomial number of processors, in contrast to all previously known sub-linear time CFL recognizers.

1. INTRODUCTION

The alternating Turing machine (ATM) introduced by Chandra and Stockmeyer, and by Kozen [6, 14, 7] has proved to be a very useful computational model. Several interesting applications are known, e.g., [6, 14, 7, 10, 1, 17, 30, 15, 16]. We believe the ATM's convenience and utility may be further extended by careful study of the model's properties. In this paper we will present a simple, natural new complexity measure for ATMs, called "tree-size." Basically the "tree-size" used by an ATM on a given input is the size of its smallest accepting computation tree. Tree-size is a useful abstraction which provides a spectrum of complexity classes intermediate between nondeterminism and full alternation. This should provide a useful tool for investigating problems which do not seem to be precisely characterized by either of the later models.

While tree-size is a mathematically appealing notion, it seems to lack the intuitive economic motivation which underlies the common time and space measures. What is the cost of a larger tree-size? In fact, we will show that tree-size has close connections with more familar complexity measures. Specifically, we will show that tree-size is equal to

* This research supported by an IBM predoctoral fellowship and by NSF Grants GJ-474 and MCS77-02474.

time on two nondeterministic models, and is closely related to time on several parallel models. The later results are particularly surprising since the parallel algorithms give an exponential speed-up with only a modest increase in the required number of "processors." This result also subsumes several previously known simulations, such as Savitch's theorem [28].

Our results about context-free languages recognition illustrate the features mentioned above. CFL recognition has not been precisely characterized in terms of either space or time on Turing machines. We will give such a characterization in terms of tree-size, complementing a previously known one given in terms of auxiliary pushdown machines [35]. Applying our parallel-time simulation to this result gives an $O(\log^2 n)$ time CFL recognition algorithm for alternating Turing machines and other parallel models. Furthermore, this time bound is achievable with a polynomial number of "processors," e.g., by a combinational circuit of depth $\log^2 n$ with a polynomial number of gates. No previous solution gave less than $\log^4 n$ time, even with arbitrarily many processors, and none gave a polynomial number of processors with any sub-linear time bound. (This paper deals mainly with the relations between tree-size, space, and time on ATMs. Connections between ATMs and other parallel models are explained more fully in [26, 27].)

Sections 2 contains several definitions and examples of our models. Section 3 explores the relations between tree-size and nondeterministic time. Section 4 gives the relations with parallel time, and surveys some related literature, mainly dealing with path systems. (Most of these results first appeared in [24].)

2. Definitions

ATMs are a generalization of nondeterministic Turing machines (NTMs), described informally as follows. The states are partitioned into "existential" and "universal" states. As with an NTM, we can view a computation of an ATM as a tree of configurations. A tree is a *computation tree* of an ATM M on a string w if its nodes are labeled with configurations of M on w, such that the descendants of any non-leaf labeled by a universal (existential) configuration include all (resp. one) of the successors of that configuration. A computation tree is *accepting* if the root is labeled with the initial configuration, and all the leaves are accepting configurations. Note that for ATMs with only existential states, acceptance is essentially the same as for NTMs. We assume that ATMs have an end-marked, read-only input tape.

An ATM uses *time* $T(n)$ (*space* $S(n)$) if for all accepted inputs of length n there is an accepting computation tree of height $\leqslant T(n)$ (each of whose nodes is labeled by a configuration using space $\leqslant S(n)$). As usual, we denote the class of languages accepted by ATMs within space $O(S(n))$ by ASPACE$(S(n))$, and similarly for NSPACE, DSPACE, ATIME, NTIME, and DTIME. Likewise, ATISP$(T(n), S(n))$ denotes languages accepted by ATMs running in time $O(T(n))$ and space $O(S(n))$ simultaneously, and similarly for NTISP and DTISP.

More formal definitions of ATMs, configurations, etc. may be found in [7]. Unlike [7], we will not consider "negating" states in this paper.

DEFINITION. A language L is accepted by an alternating Turing machine M within *tree-size* bound $Z(n)$ if for every string $w \in L$ there is at least one accepting computation tree for M on w of size (number of nodes) $\leqslant Z(n)$, where n is the length of w. (Further, for $w \notin L$ there is no accepting computation tree.) Similarly, L is accepted within *simultaneous* tree-size and space bounds $Z(n)$ and $S(n)$ if there is an accepting computation tree with $\leqslant Z(n)$ nodes, each of which is labeled by a configuration using space $\leqslant S(n)$.

It is important to note that there may be many distinct accepting computation trees of varying sizes for a given machine and input. The tree-size bound in a sense reflects the nondeterministic selection of one such tree of minimal size. It may be considered to be the size of the shortest "proof" that the ATM accepts its input (in the natural proof system implicit in the ATM's definition). The nondeterministic selection means that tree-size is *not* a measure of the "hardware size" or "number of processors" used by a parallel machine which simulates the ATM. We will see instead that it is related to the *time* required by such a parallel simulation.

One motivation for introducing tree-size bounded computations is to provide a restriction of the alternating Turing machine which is intermediate in power between nondeterministic and (full) alternating computations. Consider an NTM and ATM, both operating in space S. Each has at most $2^{O(S)}$ distinct configurations, so their computations can be described by trees of height $2^{O(S)}$. An accepting computation by the NTM may be described by a single root-leaf path in the tree, i.e., $2^{O(S)}$ nodes. In contrast, an accepting computation by the ATM may involve essentially the entire tree, i.e., $2^{2^{O(S)}}$ nodes. If we consider ATMs using space S and tree-size Z for $2^{O(S)} \leqslant Z \leqslant 2^{2^{O(S)}}$, we get machines which are intermediate between NSPACE(S) and ASPACE(S). We expect that the classes of languages accepted by such machines are properly between NSPACE(S) and ASPACE(S). However, a proof of this fact is likely to be very difficult, since it implies that NSPACE(S) is properly contained in ASPACE(S) (which is known to equal DTIME($2^{O(S)}$)). For example, for $S = \log n$ this would show that NSPACE($\log n$) is properly contained in P, settling a long-standing open problem. Even if these questions remain unresolved, a model of intermediate power can prove very useful in classifying problems and in sharpening our intuitions about the relationships between various complexity classes.

We will use the following menumonic (if not euphonious) notation for simultaneously space- and tree-size bounded ATMs.

DEFINITION. ASPSZ($S(n)$, $Z(n)$) denotes the class of languages recognized by ATMs operating simultaneously in space $O(S(n))$, and tree-size $O(Z(n))$.

EXAMPLE 1. For every context-free language L, $L \in$ ASPSZ($\log n$, n^2). To see this, let G be a Chomsky normal form grammar for L, and $w = a_1 \cdots a_n$ a string to be recognized. Then $A \overset{*}{\Rightarrow} a_{i+1} \cdots a_j$ if and only if

(a) $i + 1 = j$ and $A \rightarrow a_j$ is a production in the grammar or

(b) $i + 1 < j$ and there is some integer k, $i < k < j$ and a production $A \rightarrow BC$ such that $B \overset{*}{\Rightarrow} a_{i+1} \cdots a_k$ and $C \overset{*}{\Rightarrow} a_{k+1} \cdots a_j$.

This procedure is easily performed by an ATM. Part (a) can be directly checked; for part (b) the machine can guess k and a rule $A \to BC$ then recursively verify both $B \overset{*}{\Rightarrow} a_{i+1} \cdots a_k$ and $C \overset{*}{\Rightarrow} a_{k+1} \cdots a_j$ by using a universal state. To test $w \in L$, test $S \overset{*}{\Rightarrow} a_1 \cdots a_n$ as above. Note that the machine is $\log n$ space bounded since it needs to store only i, j, k, A, B and C at any one time. Note that k replaces i or j and B or C replaces A before the next stage is entered, so no extra storage is needed in subsequent stages. Further, if $w \in L$, there is an accepting computation tree which parallels the parse tree for w. For each internal node in the parse tree, there will be $\log n$ nodes in the computation tree (to guess and write the integer k). For each j between 1 and n, the jth leaf of the parse tree will correspond to a sequence of j nodes in the computation tree which moves the input head to the jth input letter. Since the parse tree is of linear size, the accepting computation tree will be of size $O(n \cdot \log n + n^2)$. Thus $L \in \mathrm{ASPSZ}(\log n, n^2)$. ∎

We will be using a variant of the ATM introduced in [7] and herein called an *indexing ATM*, which allows sub-linear time bounds. In this model, the ATM does not read the input tape directly. Instead it has a special "index tape"; whenever an integer i is written on the index tape, the ith letter of the input is immediately visible to the ATM. Thus in $\log n$ steps, it can read any position of the input.

EXAMPLE 2. $\{ww^R \mid w \in \{0, 1\}^*\} \in \mathrm{ATIME}(\log n)$.
An indexing ATM can recognize palindromes in $\log n$ time by

(i) guess n and verify it by checking that the $(n + 1)$st position is the endmarker,

(ii) using universal states, choose all i between 1 and n,

(iii) for each, see that the ith and $(n - i + 1)$st positions are equal. ∎

We remark that the classes of languages accepted by ordinary and indexing ATMs in space $S(n)$ and/or time $T(n)$ are identical for $S(n) \geqslant \log n$ and $T(n) \geqslant n$. Thus we will generally not need to specify which ATM model we are using. However, the tree-size required for a problem does appear to be slightly sensitive to the model, varying by a factor of at most $O(n)$. For instance, the tree-size required by the CFL recognition algorithm in Example 1 drops from n^2 to $n \log n$ if an indexing ATM is used, since inputs can be directly examined at the leaves.

One technical result about tree-size is worth noting. Like other useful complexity measures, log-space reducibilities respect tree-size (to within a polynomial).

LEMMA 1. *If* $L' \leqslant_{\log} L$ *and* $L \in \mathrm{ASPSZ}(S(n), Z(n))$ *for* $S(n) \geqslant \log n$ *then* $L' \in \mathrm{ASPSZ}(S(n), Z'(n))$, *where* $Z'(n) = Z(p(n)) \cdot p(n)$ *for some polynomial* $p(n)$.

(Note that, e.g., Z' is a polynomial if Z is.)

Proof. The proof is essentially the same as the analogous proof for $\leqslant_{\log}$ and space complexity classes [13, 31]. We accept L' by simulating the acceptor for L, keeping track of the position of its input head. Whenever it needs to read another input letter, say the ith, we simulate the log-space transducer until it generates the ith output. The polynomial p in the lemma is the running time of the transducer. ∎

3. Relations with Nondeterministic Time

It is well known that time on a deterministic TM is reflected in the *space* required by an ATM [6, 14]. We will show that time on *non*deterministic machines is closely related to the *tree-size* required by ATMs.

Our first result concerns nondeterministic auxiliary pushdown automata (AuxPDA) [5]. Informally, an AuxPDA is a multitape TM having a two-way, end-marked, read-only input tape, a pushdown tape, and one or more two-way, read/write work tapes. "Space" on an AuxPDA means space on the work tapes only (excluding the pushdown). A more formal definition may be found in [5]. We assume without loss of generality that every move of the machine either pushes or pops the stack, and that the machine only accepts with its stack empty. We let *AuxPDA*$(S(n), T(n))$ denote the class of languages accepted by nondeterministic AuxPDAs within space $O(S(n))$ and time $O(T(n))$.

Close connections between ATMs and AuxPDAs are suggested by the fact that space S ATMs and AuxPDAs accept the same sets, namely, DTIME$(2^{O(S)})$ [5, 7]. Further, the proofs of these two results are very similar. We will show that this connection is even stronger. Namely, space S and tree-size Z ATMs are equivalent (to within a polynomial) to AuxPDAs using space S and *time* Z. This result in a sense generalizes the well-known equivalence between context-free grammars and pushdown automata. In that case having two such disparate characterizations of the same phenomena proved enormously useful. We hope our result will be similarly useful.

Theorem 1. *For* $S(n) = \Omega(\log n)$, *and* $Z(n) = \Omega(n)$

$$\text{ASPSZ}(S(n), Z(n)^{O(1)}) = \text{AuxPDA}(S(n), Z(n)^{O(1)}).$$

More precisely, for any $S(n)$, $Z(n)$, *and* $S'(n) = \max(S(n), \log n)$, *we have*:
For non-indexing ATMs,

(1) $\text{ASPSZ}(S(n), Z(n)) \subseteq \text{AuxPDA}(S(n), Z(n))$,

(2) $\text{AuxPDA}(S(n), T(n)) \subseteq \text{ASPSZ}(S'(n), T(n) \cdot (S'(n) + n))$,

For indexing ATMs,

(1′) $\text{ASPSZ}(S(n), Z(n)) \subseteq \text{AuxPDA}(S(n), Z(n) \cdot n)$,

(2′) $\text{AuxPDA}(S(n), T(n)) \subseteq \text{ASPSZ}(S'(n), T(n) \cdot S'(n))$.

Proof. (1) We construct an AuxPDA which simulates an ATM by doing a simple "depth first search" of the ATM's computation tree, using its stack to backtrack through the universal nodes.

The AuxPDA will have the same number of work tapes as the ATM. The AuxPDA will directly encode a configiration of the ATM by having the ATM's work tape contents recorded on its own work tapes, with its work tape- and input head positions also the same as the ATM.

The AuxPDA can directly simulate moves of the ATM in unit time. The AuxPDA simulates existential moves of the ATM by using its own nondeterminism. It uses its stack to simulate universal moves by backtracking. To do this, for each simulated move it pushes onto the stack the information it needs to "undo" the move, i.e., the old state, the old contents of the tape cells under the work tape heads, and the direction of each head's motion. For each universal move it also records which alternative is the next to be explored. All this information can be pushed onto the stack in unit time. Now whenever the simulation reaches an accepting leaf of the ATM's computation tree, the AuxPDA "backtracks" by popping its stack to the last universal move for which there is an unexplored alternative. While it is popping it "undoes" the corresponding moves, using the information popped from the stack (which takes unit time per pop). Thus when the last universal move with an unexplored alternative is reached, the configuration of the AuxPDA's work- and input tapes has been restored to what it was when the ATM was about to execute the universal move. The AuxPDA now continues the simulation, following the next alternative for the universal move. If there are no universal moves with unexplored alternatives, i.e., the stack is emptied while backtracking, the AuxPDA halts and accepts.

It is easy to see that the AuxPDA accepts if and only if the ATM with the given input has an accepting computation tree. Further, the AuxPDA "traverses" each edge in an accepting computation tree exactly twice (once to simulate that move and once to undo it), and each traversal takes unit time. Thus the AuxPDA's running time is the same as the size of the smallest accepting computation tree for the ATM, i.e., $Z(n)$.

(1′) If we are simulating an indexing ATM, there is an extra cost of $O(n)$ per step to update the input head position. Otherwise the argument is the same.

(2) The ATM simulation of an AuxPDA is by a generalization of the DTM simulation of AuxPDAs given by Cook [5], which in turn generalizes the classic proof that PDAs accept only context-free languages [3, 29].

We begin by summarizing some relevant concepts from Cook's proof. Full details may be found in [5]. The simulation deals with "surface configurations" of the AuxPDA. A *surface configuration* specifies the state, the contents and head positions of the work tapes, the head position (but not the contents) of the input tape, and the topmost symbol (but no other) of the stack. Note that space $S'(n)$ is sufficient to encode a surface configuration. With a given input w, a pair (P, Q) of surface configurations is called ***realizable*** if the AuxPDA can move from P to Q ending with its stack at the same height as in P, and without popping its stack below its level in P at any point in this computation. It's easy to see that the AuxPDA accepts w if and only if there is an ***accepting pair***: a realizable pair (P_0, Q_a), where P_0 is the initial configuration, and Q_a contains a final state.Thus, the heart of the simulation is finding an accepting pair, which in turn hinges on the following simple characterization of the set of realizable pairs. See Fig. 1. A pair (P, Q) is realizable iff (a) $P = Q$, or (b) there are realizable pairs (P, U) and (V, W) with a move from U to V pushing some symbol and a move from W to Q popping the same symbol. (This is actually a slightly simplified version of Cook's characterization which is possible since we assumed that every move of the AuxPDA either pushes or pops the stack.)

We now have the necessary machinery to complete our proof. We will give an ATM which verifies the existence of an accepting pair within the required space and tree-size

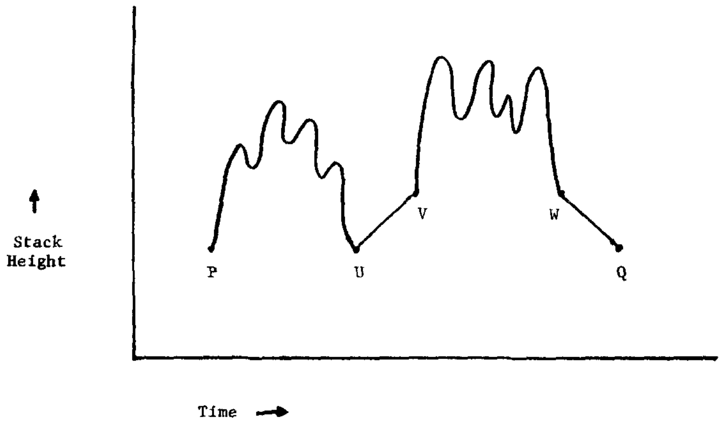

FIG. 1. Characterizing "realizable pairs."

bounds. Not surprisingly, the procedure is similar to the context free recognition procedure given in Example 1 above.

We construct a subroutine which given a pair of surface configurations (P, Q) accepts if and only if (a) $P = Q$ or (b) there exist surface configurations U, V, W with a move from U to V pushing some symbol, a move from W to Q popping the same symbol, and (P, U) and (V, W) are both realizable pairs. The later facts are verified recursively in parallel (i.e., splitting at a universal state); the other facts are easily verified directly. Finally, to test whether the input is accepted, the ATM guesses an accepting pair (P_0, Q_a), then verifies its guess using the routine above.

Correctness of the simulation is immediate from Cook's characterization and a simple induction. The space bound of $S'(n)$ is also immediate since the routine never keeps more than five surface configurations at a time (P, Q, U, V, W). The tree-size bound is derived from the AuxPDA's running time as follows. Let $R(t)$ be the number of recursive calls generated while verifying a realizable pair (P, Q) representing a t-step computation of the AuxPDA. Then

$$\begin{aligned} R(t) &= 0, && \text{if } t = 0 \\ &= 2 + R(t_1) + R(t_2), && \text{otherwise,} \end{aligned}$$

where t_1 and t_2 are the running times of the subcomputations (P, U) and (V, W) resp. Note that $t = 2 + t_1 + t_2$, hence the recurrence has the solution $R(t) = t$. The number of nodes generated by each call exclusive of recursive calls is $O(S'(n))$ (to compare P to Q or to guess U, V, W, etc.) plus $O(n)$ (to read the two necessary input letters when verifying that U can move to V and W to Q). The total tree-size is thus $O(T(n) \cdot (S'(n) + n))$ as claimed.

(2′) For indexing ATMs, the $O(n)$ cost for reading inputs is eliminated, giving tree-size $T(n) \cdot S'(n)$. ▮

Two slight improvements on the theorem are possible. First, note that the factor of n in the upper bound of (1′) could be eliminated if we define "indexing" AuxPDAs, analogous to indexing ATMs. Second, we observe that the tree-size upper bounds in both (2) and (2′) can be improved by a factor of $S'(n)$ as follows. The AuxPDA can be altered so that it pushes and pops blocks of exactly $S'(n)$ characters, with at least $S'(n)$ non-stack moves between these bursts of pushes or pops. This is done by keeping the top 1 to $2 \cdot S'(n) - 1$ characters of the simulated stack on a work tape, and popping or pushing a block of $1 \cdot S'(n)$ characters from/to the real stack only when the worktape contains 0 or $2 \cdot S'(n)$ characters, resp. The ATM simulation of the modified AuxPDA is similar to that given in the proof above, except that each of the $T(n)/S'(n)$ blocks of $S'(n)$ stack moves is simulated as a unit using time and tree-size $S'(n)$, and each of the $T(n)/S'(n)$ blocks of $S'(n)$ non-stack moves is simulated in unit time per move, plus n per block for the non-indexing case. The details are left to the reader. Note that these observations give tree-size and AuxPDA time equal within a *constant* factor for indexing machines, or for non-indexing machines with $S(n) = \Omega(n)$.

One easy corollary of Theorem 1 is the previously cited result that without a concurrent tree-size limit, ATM space is equivalent of AuxPDA space. More interestingly, without a concurrent space limit, tree-size is equivalent to nondeterministic TM time.

COROLLARY 1.

$$\mathrm{NTIME}(T(n)) = \mathrm{ASPSZ}(T(n), T(n)) \qquad \text{(for non-indexing ATMs).}$$

Proof.

$$\mathrm{NTIME}(T) \subseteq \mathrm{ASPSZ}(T, T) \subseteq \mathrm{AuxPDA}(T, T) \subseteq \mathrm{NTIME}(T). \quad \blacksquare$$

It is interesting to note that $\mathrm{DTIME}(T(n)) = \mathrm{ASPSZ}(\log T(n), 2^{O(T(n))})$, thus giving a space/tree-size trade-off view of the *P vs NP* question.

Another interesting corollary of theorem 1 provides a new characterization of LOGCFL, the class of languages log-space reducible to context-free languages. Sudborough has shown $\mathrm{LOGCFL} = \mathrm{AuxPDA}(\log n, n^{O(1)})$ [35]. Thus by Theorem 1:

COROLLARY 2.

$$\mathrm{LOGCFL} = \mathrm{ASPSZ}(\log n, n^{O(1)}).$$

That $\mathrm{NSPACE}(\log n) \subseteq \mathrm{LOGCFL}$ follows from [32]; the characterization in [35] suggests that the containment is proper, which would imply that $\mathrm{CFL} \not\subseteq \mathrm{NSPACE}(\log n)$. Our Corollary 2 provides further intuitive support for this conjecture. In view of Corollary 2 (or [35]) $\mathrm{CFL} \subseteq \mathrm{NSPACE}(\log n)$ only if no power is gained by extending nondeterministic log-space bounded TMs by adding alternation (or a pushdown), but retaining the polynomial tree-size (or time) bound inherent in log-space NTMs.

One reason for interest in LOGCFL is that we have fairly efficient recognition algorithms for the class, e.g., polynomial time, deterministic space $\log^2 n$ [18], and $\log^2 n$ parallel time (cf. Section 4). Corollary 2 also gives a new tool for showing membership and completeness in LOGCFL. Several non-context-free languages are known to be in the class [8, 33, 34]. Use of log-space-and-polynomial tree-size ATMs seems to simplify some of these constructions, and will probably facilitate future ones also.

4. Relations with Parallel Time

Our next result relates space-and-tree-size bounds to parallel space-and-time bounds. It seems to be at least as general as, and more efficient than similar previously known simulations. The proof of the theorem provides a uniform translation of tree-size bounded ATMs into time bounded ATMs, which can give a nearly exponential reduction in the time used in the computation. Surprisingly, this dramatic speedup costs very little in other resources. The space required is increased by only a constant factor, and the tree-size by only a polynomial.

ATMs may seem to be an unreasonably abstract model to have much relevance to parallel complexity in practice. However, recent results [26, 27] have demonstrated close connections between ATM complexity and complexity on two more concrete parallel models, namely, vector machines [22, 23] and combinational circuits [2]. In particular, we have shown that ATM time T is contained in vector machine time T. For circuits, the result is even sharper—simultaneous ATM time T and space S is equal to uniform circuit depth T and size $2^{O(S)}$ (cf. [26] for definitions). In view of the very fundamental nature of the combinational circuit model with respect to all computational devices, we expect similar results will apply to *any* reasonable model of parallel computer. Our result below gives, e.g., CFL recognition in time (depth) $\log^2 n$ on these models, versus $\log^4 n$ in [22, 23, 2]. viewed another way, this extends the class of languages known to be recognizable in time (depth) $\log^2 n$ on these models from NSPACE($\log n$) to ASPSZ($\log n$, $n^{O(1)}$). Surprisingly, this extension is not at the expense of increasing the "number of processors" (i.e., the vector length or circuit size) from a polynomial, as in the earlier simulations of NSPACE($\log n$), to a superpolynomial, as in [22, 23, 2]. This follows from the correspondence between circuit size and ATM space [26], and the fact that the simulation below does not increase space by more than a constant factor. We know of no other CFL recognition algorithm which achieves sub-linear time without also using a superpolynomial number of processors. More generally, Theorem 2 below and [26] give a characterization of NC, the class of problems solvable in parallel using a polynomial number of processors in time $\log^{O(1)} n$. Namely, NC is equal to ASPSZ($\log n$, $2^{\log^{O(1)} n}$). This class is of significant practical interest since it probably encompasses all those problems for which dramatic speedups are possible on feasible parallel computers.

One other aspect of our theorem is noteworthy. When combined with Theorem 1, it connects a purely sequential time complexity measure (AuxPDA time) and a parallel time complexity measure (ATM time) with an exponential speedup in the parallel case. Several results translating sequential space to parallel time are known (see e.g., Corollary 3a below or [12]). We know of only a few similar sequential time to parallel time translations, e.g., Corollary 3b below, Pippenger's interesting work on reversal bounded DTMs [21], and its precursors on finite automata [20, 9]. There is some hope that results of this kind may be of significant practical utility in simplifying the process of discovering and implementing efficient parallel algorithms.

THEOREM 2. *For* $S(n) \geqslant \log n$,

$$\mathrm{ASPSZ}(S(n), Z(n)) \subseteq \mathrm{ATISP}(S(n) \cdot \log Z(n), S(n))$$

Further, the tree-size required by the simulation is

$$O(Z(n) \cdot S(n)) \qquad \textit{for indexing } \text{ATMs}, \textit{ or}$$
$$O(Z(n) \cdot (S(n) + n)) \qquad \textit{for non-indexing } \text{ATMs}.$$

Before giving the proof, we will examine a few of its consequences, some of which are quite unexpected. These results should serve to highlight the strength of the theorem and put it in perspective with related results in the literature.

Intuitively, the theorem says that tall but sparse computation trees may be transformed into short, bushy ones. For example, for $S(n) = \log n$ a tree of polynomial height and polynomial size (verses a potential size of $2^{n^{O(1)}}$) can be transformed to one with height $\log^2 n$ (and still of polynomial size and log space). Thus, contrary to the usual situation where more time gives more power, within the class ASPSZ($\log n$, $n^{O(1)}$) any increase in time beyond $\log^2 n$ is useless. Similarly, (alternating) time $o(n)$ suffices for computations using $S(n) = \log n$ and $Z(n) = 2^{O(n/\log n)}$.

Theorem 2 immediately gives alternating time upper bounds on several well-known problems.

COROLLARY 3.

(a)	NSPACE($S(n)$)	$\subseteq$ ATIME($S^2(n)$),	[6, 14],
(b)	NTISP($T(n)$, $S(n)$)	$\subseteq$ ATIME($S(n) \cdot \log T(n)$),	[6, 14],
(c)	LOGCFL	$\subseteq$ ATIME($\log^2 n$),	
(d)	AuxPDA($S(n)$, $T(n)$)	$\subseteq$ ATIME($S(n) \cdot \log T(n)$).	

Further, since ATIME($T(n)$) is contained in DSPACE($T(n)$), this also gives deterministic space upper bounds. The following well-known simulations can be obtained as corollaries in this way.

COROLLARY 4.

(a)	NSPACE($S(n)$)	$\subseteq$ DSPACE($S(n)^2$),	[28],
(b)	NTISP($T(n)$, $S(n)$)	$\subseteq$ DSPACE($S(n) \cdot \log T(n)$),	[28],
(c)	LOGCFL	$\subseteq$ DSPACE($\log^2 n$),	[18],
(d)	AuxPDA($\log n$, $n^{O(1)}$)	$\subseteq$ DSPACE($\log^2 n$),	[4],
(e)	AuxPDA($S(n)$, $T(n)$)	$\subseteq$ DSPACE($S(n) \cdot \log T(n)$),	[19].

Although we cannot prove smaller deterministic space upper bounds for these problems, our theorem does provide easy proofs of some interesting results which refine (in a technical sense) these bounds. We believe these results, coupled with those mentioned following Corollary 2, give strong evidence that, e.g., LOGCFL is *properly* between NSPACE($\log n$) and DSPACE($\log^2 n$). For simplicity we will confine our discussion to LOGCFL, although similar remarks also apply to the other problems above. Figure 2 summarizes the relationships discussed below.

Gurari and Ibarra [11] have observed that the simulation of [18] makes only a restricted use of the space available to it. Specifically it can be implemented on a $\log n$ space deterministic auxiliary stack automaton whose stack height is bounded by $O(\log^2 n)$. This result also follows easily from our Corollary 3 and the observation that the deterministic space simulation of a time-bounded ATM given in [7] can also be done on a stack automaton. Harju [37] has recently shown that languages in LOGCFL admit an even more restricted use of space. Namely, it suffices to use a $\log^2$ height pushdown, rather than a

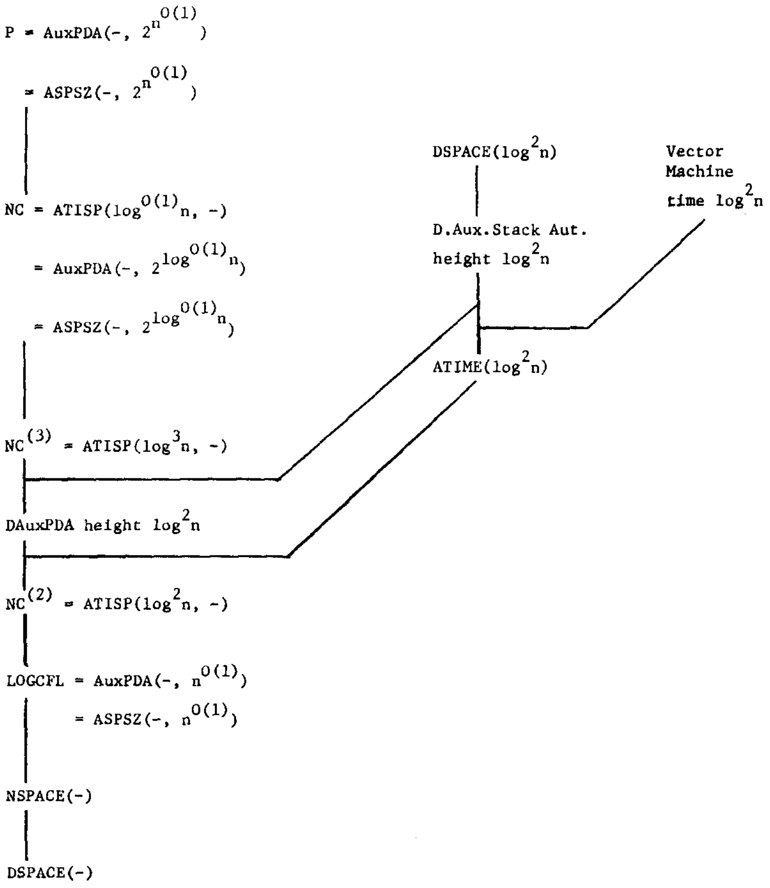
P = AuxPDA(-, $2^{n^{O(1)}}$)
= ASPSZ(-, $2^{n^{O(1)}}$)
NC = ATISP($\log^{O(1)} n$, -)
= AuxPDA(-, $2^{\log^{O(1)} n}$)
= ASPSZ(-, $2^{\log^{O(1)} n}$)
$NC^{(3)}$ = ATISP($\log^3 n$, -)
DAuxPDA height $\log^2 n$
$NC^{(2)}$ = ATISP($\log^2 n$, -)
LOGCFL = AuxPDA(-, $n^{O(1)}$)
= ASPSZ(-, $n^{O(1)}$)
NSPACE(-)
DSPACE(-)
DSPACE($\log^2 n$)
D.Aux.Stack Aut. height $\log^2 n$
ATIME($\log^2 n$)
Vector Machine time $\log^2 n$

FIG. 2. Some complexity relationships pertinent to CFL recognition. ("—" means log n space.)

stack. Harju's proof is a rather complicated direct construction, but the result follows easily from Theorem 2:

COROLLARY 5. LOGCFL $\subseteq$ *deterministic* AuxPDA *space* $\log n$ *and pushdown height* $\log^2 n$ [37].

Proof. We simply observe that the simulation of an ATM by an AuxPDA given in the proof of Theorem 1 part (1) uses pushdown height equal to the running time of the ATM. Further, the AuxPDA can be made deterministic at the cost of increasing its running time (to explore both subtrees of existential as well as universal nodes). The result follows since Theorems 1 and 2 imply Corollary 6 below. ▮

COROLLARY 6. LOGCFL $\subseteq$ ATISP($\log^2 n$, $\log n$).

We remark that Corollary 5 is apparently not as strong as Corollary 6, since the best ATM time and space upper bound we can give for such deterministic AuxPDAs is ATISP($\log^3 n$, $\log n$)—a result which itself depends on Theorems 1 and 2.

The proof of Corollary 5 leads us to one final observation about AuxPDAs, which we believe has not appeared in the literature.

COROLLARY 7. *L is accepted by a nondeterministic* AuxPDA *in* $\log n$ *space and polynomial time, if and only L is accepted by such a machine which, furthermore, uses at most* $\log^2 n$ *pushdown height.*

In view of Corollary 5, such machines may be made deterministic without loss of space or pushdown height, but at the cost of increasing their running time to a superpolynomial ($2^{O(\log^2 n)}$).

Finally we will give the proof of Theorem 2. It generalizes constructions in [18, 4, 19] (cf. Corollary 4c, d, e above). Careful application of those techniques is sufficient to establish Corollary 3 above [24, 25], but not sufficient to establish the theorem, since the resultant ATM uses as much space as time. Thus the principal novelty in the proof below is in reducing the space required. Harju [37] seems to have overcome a similar barrier in the construction of [18], although as noted above his result is not as strong as ours.

Proof of Theorem 2. The proof is similar in outline to the proof of Theorem 1, part 2. We will define a set of "fragments" which *may* be part of the simulated ATM's computation; those fragments which *are* part of that computation are called "realizable"; we characterize the realizable fragments in a way which allows an efficient recognition procedure.

Let M be an $S(n)$-space- and $Z(n)$-tree-size bounded ATM, and let w be an input to M of length n. WLOG assume each of M's configurations has at most two successors. A computation *fragment* of M on w is an ordered pair (r, L), where r is a configuration of M using space $\leqslant S(n)$, and L is a set of such configurations. A fragment is *realizable* if there is a computation tree of M with root r whose leaves are either accepting or in L. The *size* of a realizable fragment is the size of the smallest associated tree. Note that the fragment

$(r_0, \varnothing)$, where r_0 is the initial configuration of M on w is realizable (and of size $Z(n)$) if and only if M accepts w (within tree-size $Z(n)$).

LEMMA 2. *(r, L) is realizable if and only if either:*

(a) *there is a computation tree of size $\leqslant 3$ which demonstrates that (r, L) is realizable, or*

(b) *there are realizable fragments $(r, L' \cup \{s\})$ and (s, L'') of strictly smaller size than (r, L) with $L = L' \cup L''$. (See Fig. 3.)*

Proof. "if": computation trees demonstrating the realizability of $(r, L' \cup \{s\})$ and (s, L'') can be grafted together to get a tree for (r, L) by replacing all "s" leaves in the tree for $(r, L' \cup \{s\})$ by the tree for (s, L'').

"only if": Pick a computation tree demonstrating the realizability of (r, L). If it has $\leqslant 3$ nodes, we are done by case (a). Otherwise it must have some node s which is neither the root nor a leaf. Split the tree at s. Let L' be the subset of L which labels leaves *not*

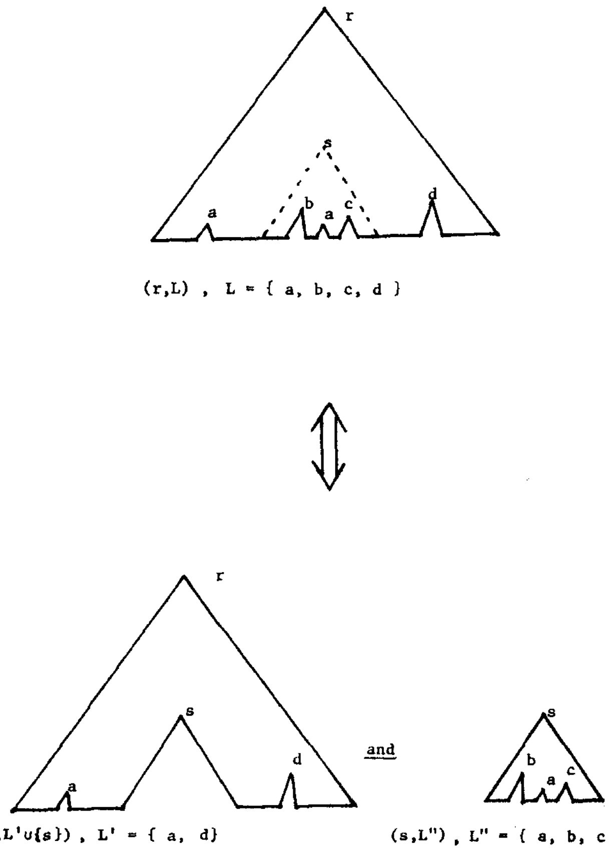
r
s
a
b
a
c
d
(r,L) , L = { a, b, c, d }
r
s
a
d
and
s
b
a
c
(r,L'∪{s}) , L' = { a, d}
(s,L") , L" = { a, b, c }

FIG. 3. Characterizing "realizable fragments."

having that node s as an ancestor, and L'' the subset of L having s as an ancestor. Then $(r, L' \cup \{s\})$ and (s, L'') are realizable, and smaller than (r, L). ▌

Lemma 2 immediately gives a correct ATM algorithm for simulating M. The major component is a procedure for verifying that a given fragment (r, L) is realizable. This is done by guessing if the size of (r, L) is $\leqslant 3$. If so, the desired computation tree is constructed by directly simulating M starting in configuration r. If the size is > 3, we guess s, L', L'' with $L = L' \cup L''$, then recursively verify both $(r, L' \cup \{s\})$ and (s, L'') in parallel (splitting at a universal state). We decide whether M accepts w by testing whether $(r_0, \varnothing)$ is realizable, where r_0 is the initial configuration.

All that remains is to analyze the time and space requirements of this algorithm. First we claim that the number of levels of recursion needs to be no more than $O(\log Z(n))$. This follows from the simple lemma below which shows that "s" in case (b) of Lemma 2 can be chosen so that both subproblems $(r, L' \cup \{s\})$ and (s, L'') have size bounded by a fixed fraction of the size of (r, L). Since the two subproblems formed by splitting are of roughly equal size, after about $\log Z$ splits the subproblems will all be of size $\leqslant 3$ which can be directly checked (case (a)).

LEMMA 3. *Let T be a tree having $z > 3$ nodes, with each node having at most two descendants. Then there is a node s in T such that $z/12 < z_1, z_2 \leqslant 11z/12$, where z_1 is the number of descendants of s (including s), and z_2 is the number of nodes which are not proper descendants of s (again including s, so $z_1 + z_2 = z + 1$).*

Proof sketch. Consider a path from the root of T which follows the edge into the larger subtree at each step. The number of descendants of a node on this path differs from the number for the next node on the path by at most a factor of two. Thus the z_1, z_2 values cannot jump over the 1/12–11/12 interval. In particular, it suffices to chose s to be the first node on the path with no more than $11z/12$ descendants. The algebraic details are left to the reader. ▌

Next we claim our procedure need never deal with fragments (r, L) whose corresponding computation trees have more than three non-accepting leaves. (In particular $|L| \leqslant 3$.) In any tree with three non-accepting leaves, there is a node s which is an ancestor of exactly two of them. If our procedure splits at this node s, then both subproblems will have exactly two non-accepting leaves. This node s may *not* give a 1/12–11/12 split as discussed above, but since neither subproblem has three non-accepting leaves, we are free to chose such a split in the next level of the recursion. Thus we get a balanced split at least every other time, so the maximum depth of recursion is still $O(\log Z(n))$.

Total space required is $O(S(n))$ to hold r, s, L, L', L''. Total time is $O(S(n) \cdot \log Z(n))$: $\log Z$ levels of recursion, each of which takes time S to manipulate r, s, L, L', L'', etc.

We omit the proof that the simulation above uses tree-size $\leqslant S(n) \cdot Z(n)$. The analysis is very similar to that for Theorem 1, part 2.

This completes the proof of Theorem 2. ▌

Results similar to our Theorem 1 and its proof have been independently discovered by researchers investigating *path systems*, introduced by Cook in [4]. We will close with a brief survey of these related results. (Some familiarity with path systems is assumed.)

Path systems and ATMs are closely related. One can easily "solve" path systems with an ATM, or conversely map an ATM and its input into a path system which is "solvable" iff the ATM accepts its input. Furthermore, these constructions preserve space, time, tree-size, and other parameters in a straightforward way. The main difference between the models is that path systems are not uniform. That is, to "solve" a problem with an ATM, we give *one* ATM which works for all problem instances, whereas to "solve" it via path systems we give a *different* path system for each problem instance. For applications of path systems to computational complexity, uniformity is usually introduced by giving an easy-to-compute transformation from problem instances to path systems, e.g., a log-space reduction. With ATMs, of course, the uniformity is built in.

Path system work related to tree-size stems from Cook's original paper on the subject [4] which introduced the notion of a tree-like path system, and showed that CFLs and AuxPDA($\log n$, $n^{O(1)}$) were reducible to solution of tree-like path systems. A path system is tree-like if each node in some solution tree is distinctly labeled. It is "easy" to solve such systems, but seemingly much harder to decide whether a system *is* tree-like. Sudborough [33] alleviated this problem by dropping the distinctness requirement from Cook's definition, introducing "$Z(n)$-tree-solvable" path systems. $Z(n)$ is a bound on the size of the solution trees, closely analogous to the notion of tree-size presented in this paper. Tree-like systems are a subset of the n-tree-solvable systems. Sudborough [33] sharpened Cook's result, showing the set of encodings of $n^{O(1)}$-tree-solvable systems is a $\leqslant_{\log}$-complete language in LOGCFL or, equivalently in AuxPDA($\log n$, $n^{O(1)}$). This is the analog of our Corollary 2. Sudborough [36] and Gurari and Ibarra [11] extended this result to show $Z(n)$-tree-solvable systems are $\leqslant_{\log}$-hard (but not -complete) for AuxPDA($\log n$, $Z(n)$), which corresponds to our Theorem 1, part (2). (The results in [11] are derived for "parameterized" path systems, a generalization which adds a measure of uniformity to path systems, giving a computational model somewhat more general (and somewhat more complex) than an alternating Turing machine.) Sudborough [36] also gives a simulation of path systems by AuxPDAs which corresponds to our Theorem 1, part (1). (For technical reasons related to the non-uniformity of path systems this gives $\leqslant_{\log}$-completeness only for $Z(n) = n^{O(1)}$; cf. [36].)

Cook's original paper and much of the subsequent work cited above seems to have been motivated by the clever DSPACE($\log^2 n$) CFL recognition algorithm of [18]. Most of these papers also give analogous DSPACE upper bounds as in our Corollary 4; [11] also gives the stack automaton bound discussed earlier.

While this work is technically very similar to ours, we believe the use of the ATM, a natural, uniform computational model, has provided us with a much more intuitively appealing formulation of the concepts, techniques, and results. As one small example, our Theorem 1 states the equivalence of two classes of machines, where the path system results only give a (superficially) weaker $\leqslant_{\log}$-hardness (not even -completeness) result. In fact, the path system result is technically as strong as ours, but difficulties arising from the non-uniformity of path systems have obscured this clean correspondence. Our Theorem 2 is perhaps a more significant although more subjective example of the advantages we perceive for our approach. So far as we know, this theorem is not implied by any other in the literature, nor is it easily provable using techniques given elsewhere. Further, the theorem and some of its consequences were quite unexpacted. In spite of this, we feel its proof is simpler than that of [18] or any of its other ancestors. We believe the simplicity of our model contributed directly to the simplicity of the proof, and even more importantly, to its discovery. We hope others will find this model equally useful, but we have no doubt that the continued study of path systems, AuxPDAs, and other models will also yield useful insights.

5. Conclusion

In summary, we feel that tree-size-bounded alternating Turing machines have proven to be a useful new tool in the study of computational complexity. They have helped clarify the relationships between different automata classes (ATMs and AuxPDAs), given a new characterization of an important class of languages (LOGCFL), and facilitated proofs of membership in that class. A result which seems to be of even more general interest was a powerful new simulation theorem which unified and improved several well-known simulation results, and gave improved upper bounds on the parallel-time required for several classes of problems, while simultaneously reducing the number of processors required. We hope that this new tool will be equally useful in the future.

Acknowledgment

Thanks to A. Borodin, M. Harrison, L. Snyder, M. Tompa and the referees for useful discussions and comments.

References

1. L. Berman, Precise bounds for Presberger arithmetic and the reals with addition, *in* "Proceedings, IEEE 18th Symposium on the Foundations of Computer Science, 1977" pp. 95–99.
2. A. Borodin, On relating time and space to size and depth, *SIAM J. Comput.* **6**, 4 (1977), 733–743.
3. N. Chomsky, "Context Free Grammars and Pushdown Storage," MIT Research Lab. of Electronics, Quarterly Progress Report 65, 1962.
4. S. A. Cook, Path systems and language recognition, *in* "Proceedings, 2nd Annual ACM Symposium on Theory of Computing, 1970," pp. 70–72.
5. S. A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18**, 1 (Jan 1971), 4–18.
6. A. K. Chandra and L. J. Stockmeyer, Alternation, *in* "Proceedings, IEEE 17th Annual Symposium on Foundations of Computer Science, 1976," pp. 98–108.
7. A. K. Chandra, D. Kozen, and L. J. Stockmeyer, "Alternation," IBM research division, Yorktown Heights, N.Y., Report RC 7489, 1978.
8. W. Erni, "Some Further Languages Log-Tape Reducible to Context-Free Languages," Institut fur Angewandte Informatik und Formale Beschreibungverfahren, Universitat Karlsruhe, W. Germany.

9. M. J. Fischer and R. E. Ladner, "Parallel Prefix Computations," Univ. of Washington TR 77-03-02, March 1977.
10. M. J. Fischer and R. E. Ladner, Propositional modal logic of programs, *in* "Proceedings, 9th Annual ACM Symposium on Theory of Computing, 1977," pp. 286–294.
11. E. M. Gurari and O. H. Ibarra, Path systems: constructions, solutions, and applications, *SIAM J. Comput.* **9**, 2 (1980), 348–374.
12. L. M. Goldschlager, A unified approach to models of synchronous parallel machines, *in* "Proceedings, 9th ACM Symposium on Theory of Computing 1978," pp. 89–94.
13. N. D. Jones, Space bounded reducibility among combinatorial problems, *J. Comput. System Sci.* 11 (Aug 1975), 68–85.
14. D. Kozen, On parallelism in Turing machines, *in* "Proceedings, IEEE 17th Annual Symposium on Foundations of Computer Science, 1976," pp. 89–97.
15. D. Kozen, "Complexity of Boolean Algebras," IBM research division, Yorktown Heights, N.Y., Report RC 7488, 1979.
16. H. R. Lewis, "Complexity Results for Classes of Quantificational Formulas," Harvard Univ. Aiken Computation Laboratory, TR-02-79, 1979.
17. R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer, Alternating pushdown automata, *in* "Proceedings, 19th IEEE Symposium on Foundations of Computer Science, 1978," pp. 92–106.
18. P. M. Lewis, R. E. Stearns, and J. Hartmanis, Memory bounds for recognition of context-free and context sensitive languages, *in* "Proceedings, IEEE 6th Annual Symposium on Switching Circuit Theory and Logic Design, 1965," pp. 191–202.
19. B. Monien, Relationships between pushdown automata and tape-bounded Turing machines, *in* "Automata, Languages, and Programming" (M. Vivat, Ed.), pp. 575–583, American Elsevier, New York, 1972.
20. Yu. P. Ofman, On the algorithmic complexity of discrete functions, *Sov. Phys. Dokl.* **7** (1963), 589–591.
21. N. Pippenger, On simultaneous resource bounds, *in* "Proceedings, IEEE 20th Symposium on Foundations of Computer Science, 1979," pp. 307–311.
22. V. R. Pratt, M. O. Rabin, and L. J. Stockmeyer, A characterization of the power of vector machines, *in* "Proceedings, 6th Annual ACM Symposium on Theory of Computing, 1974," pp. 122–134.
23. V. R. Pratt and L. J. Stockmeyer, A characterization of the power of vector machines, *J. Comput. Systems Sci.* **12**, 2 (1976), 198–221.
24. W. L. Ruzzo, "General Context-Free Language Recognition," Ph.D. Dissertation, University of California, Berkeley, 1978.
25. W. L. Ruzzo, Tree-size bounded alternation, *in* "Proceedings, 11th ACM Symposium on Theory of Computing, 1979," pp. 352–359.
26. W. L. Ruzzo, On uniform circuit complexity, *in* "Proceedings, IEEE 20th Symposium on Foundations of Computer Science, 1979," pp. 312–318.
27. W. L. Ruzzo, "An Improved Characterization of the Power of Vector Machines," University of Washington Computer Science Department Technical Report 78-10-01, in press.
28. W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4**, 2 (1970), 177–192.
29. M. P. Schützenberger, On context-free languages and pushdown automata, *Inform. Contr.* **6** (1963), 246–264.
30. L. J. Stockmeyer and A. K. Chandra, Provably difficult combinatorial games, *SIAM J. Comput.* **8** (May 1979), 151–174.
31. L. J. Stockmeyer and A. R. Meyer, Word problems requiring exponential time, *in* "Proceedings, 5th ACM Symposium on Theory of Computing, 1973," pp. 1–9.
32. I. H. Sudborough, A note on tape-bounded complexity classes and linear context-free languages, *J. Assoc. Comput. Mach.* **22**, 4 (Oct 1975), 499–500.

33. I. H. SUDBOROUGH, Time and tape bounded auxiliary pushdown automata, *in* "Mathematical Foundations of Computer Science," Lecture Notes in Computer Science, No. 53, pp. 493–503, Springer-Verlag, Berlin/New York, 1977.
34. I. H. SUDBOROUGH, The complexity of the membership problem for some extensions of context-free languages, *Internat. J. Comput. Math. Sect. A* **6** (1977), 191–215.
35. I. H. SUDBOROUGH, On the tape complexity of deterministic context-free languages, *J. Assoc. Comput. Mach.* **25**, 3 (1978), 405–414.
36. I. H. SUDBOROUGH, Relating open problems on the tape complexity of context-free languages and path system problems, *in* "Proceedings, 12th Conference on Information Sciences and Systems," pp. 329–335, Johns Hopkins Press, Baltimore, 1978.
37. T. HARJU, A simulation result for the auxiliary pushdown automata, *J. Comput. System Sci.* **19** (1979), 119–132.