Alternating Pushdown Automata

(Preliminary Report)

Richard E. Ladner
University of Washington

Richard J. Lipton
Yale University

Larry J. Stockmeyer
IBM Watson Research Center

1. INTRODUCTION

1.1 Summary of Results

Alternating Turing machines were introduced by Chandra and Stockmeyer [2] and by Kozen [12] as an interesting generalization of nondeterministic Turing machines. In this paper we investigate the effect of adding alternation to the power of auxiliary pushdown automata, first investigated by Cook [3], and to auxiliary stack automata, first investigated by Ibarra [9]. We characterize the power of alternating auxiliary pushdown automata (Alt-Aux-PDA), alternating auxiliary stack automata (Alt-Aux-SA) and alternating auxiliary nonerasing stack automata (Alt-Aux-NESA) in terms of time bounded Turing machines. See Table 1 for a summary of our results in comparison to known results concerning deterministic and nondeterministic versions of these classes of automata.

There are several interesting things to note about Table 1. For each type of auxiliary machine the deterministic and nondeterministic versions have exactly the same power while the alternating version has strictly more power. Alternating auxiliary stack automata and nonerasing stack automata have exactly the same power while it is open whether or not nondeterministic auxiliary stack automata are more powerful than their nonerasing counterpart. Chandra and Stockmeyer [2] and Kozen [12] proved that alternating Turing machines with space bounded by s(n) have exactly the power of deterministic Turing machines that run in time $2^{cs(n)}$ for some $c > 0$. It is open whether or not DTIME($2^{cs(n)}$) properly includes DSPACE(s(n)). So it is not known whether or not the addition of a pushdown store alone or alternation alone increases the power of space-bounded Turing machines. However, if *both* alternation and a pushdown store are added to a space-bounded Turing machine then a more powerful device results.

	Deterministic	Nondeterministic	Alternating
AUX-PDA(s(n))	$\cup$ DTIME($2^{cs(n)}$) Cook	$\cup$ DTIME($2^{cs(n)}$) Cook	$\cup$ DTIME($2^{2^{cs(n)}}$)
AUX-SA(s(n))	$\cup$ DTIME($2^{2^{cs(n)}}$) Ibarra	$\cup$ DTIME($2^{2^{cs(n)}}$) Ibarra	$\cup$ DTIME($2^{2^{2^{cs(n)}}}$)
AUX-NESA(s(n))	$\cup$ DSPACE($2^{cs(n)}$) Ibarra	$\cup$ DSPACE($2^{cs(n)}$) Ibarra	$\cup$ DTIME($2^{2^{2^{cs(n)}}}$)

Table 1

Characterization of auxiliary pushdown and stack automata in terms of time and space bounded Turing machines. The function s(n) bounds the space used on the auxiliary storage tape of the pushdown and stack automata. Unions are over all constants $c > 0$, and s(n) is assumed to be at least log n.

	Deterministic	Nondeterministic	Alternating
PDA	———	———	$\bigcup$ DTIME(2^{cn})
SA	$\bigcup$ DTIME($2^{cn \log n}$) Cook	$\bigcup$ DTIME(2^{cn^2}) Cook	$\bigcup$ DTIME($2^{2^{cn}}$)
NESA	DSPACE(n log n) Hopcroft and Ullman	NSPACE(n^2) Hopcroft and Ullman	$\bigcup$ DTIME($2^{2^{cn}}$)

Table 2

Characterization of two-way pushdown and stack automata in terms of time and space bounded Turing machines. (Chandra and Stockmeyer previously observed that alternating PDA's have at least the power of deterministic exponential time [2].)

Two-way pushdown automata and stack automata without auxiliary storage were first investigated by Gray, Harrison and Ibarra [6] and Ginsburg, Greibach, and Harrison [5], respectively. There is no known characterization of two-way pushdown automata (either deterministic or nondeterministic) in terms of time or space bounded Turing machines. Hopcroft and Ullman have characterizations of both deterministic and nondeterministic nonerasing stack automata in terms of time or space bounded Turing machines [7] and Cook has such characterizations of the erasing versions of stack automata [3]. Table 2 summarizes our results and past results concerning pushdown automata and stack automata.

There are also several interesting things to notice about Table 2. For each type of machine the alternating version is more powerful than either the deterministic or nondeterministic version. This is true for pushdown automata because the languages accepted by nondeterministic PDA's are also accepted by deterministic Turing machines that run in polynomial time. Alternating stack automata and their nonerasing versions are equivalent while it is not known whether or not deterministic (nondeterministic) stack automata and their nonerasing versions are equivalent.

We also investigate in this paper the effect of adding bounded alternation to auxiliary pushdown automata. For instance $\exists\forall$-AUX-PDA(s(n)) is the class of languages accepted by alternating auxiliary pushdown automata with auxiliary storage bounded by s(n) where the machines are only allowed to make a sequence of existential (nondeterministic) moves followed by a sequence of universal moves. (A universal move is like a nondeterministic move except that *all* successor configurations of the configuration must lead to acceptance rather than just some successor configuration must lead to acceptance). Table 3 summarizes our results and compares them with known results concerning the deterministic and nondeterministic versions of auxiliary pushdown automata. We do not yet know the exact power of Aux-PDA's with more than three (but a bounded number of) alternating quantifiers.

AUX-PDA s(n)	Complexity Class
DET	$\bigcup$ DTIME($2^{cs(n)}$) Cook
$\exists$ (NONDET)	$\bigcup$ DTIME($2^{cs(n)}$) Cook
$\forall$	$\bigcup$ DTIME($2^{cs(n)}$)
$\forall\exists$	$\bigcup$ co-NTIME($2^{cs(n)}$)
$\exists\forall$	$\bigcup$ DSPACE($2^{cs(n)}$)
$\exists\forall\exists$,$\forall\exists\forall$	$\bigcup$ DSPACE($2^{cs(n)}$)

Table 3
Characterization of bounded-alternation auxiliary pushdown automata in terms of time and space bounded Turing machines.

It is curious to note that for a fixed s(n), $\forall\exists$-Aux-PDA's seem to accept less than the complements of languages accepted by $\exists\forall$-Aux-PDA's. The cause of this can be traced to an asymmetry between $\exists$ and $\forall$ in the way that nonterminating computations are handled in the standard definition [2,12] of acceptance by alternating automata. If we require our Aux-PDA's to always halt, even on nonaccepting branches of the computation, then the asymmetry is removed and we obtain $\exists\forall$-AUX-PDA(s(n)) $= \cup$ NTIME($2^{cs(n)}$). However, our results in Tables 1 and 2 remain the same whether or not the machines are required to halt on all computation paths, provided that s(n) is tape constructible.

1.2 Motivation.

As yet we do not know of any useful applications of our characterizations of alternating pushdown and stack automata, but each of alternation and pushdown stores alone have been useful in the past in showing exponential time lower bounds. Using alternation, Chandra and Stockmeyer have shown an exponential time lower bound on the problem of determining which player has a winning strategy in certain games [2]. Using alternation, M. Fischer and Ladner have shown the same lower bound on the validity problem in propositional dynamic logic [4]. Using auxiliary PDA's, Jazayeri, Ogden, and Rounds have shown an exponential time lower bound on the circularity problem for attribute grammars [10].

Alternation is becoming a useful tool in the classification of the complexity of certain decision problems in logic as evidenced by the work of Berman in the classification of the complexity of the theory of certain additive number systems [1] and by Kozen in the classification of the complexity of the theory of Boolean algebras [13]. Ruzzo has shown that the class of languages reducible to context free languages in log space can be characterized by a class of alternating Turing machines [14]. A thorough study of the power of alternation is needed to help us use this new tool.

In several instances our proofs for alternating machines generalize in a natural way corresponding proofs for nondeterministic machines. The concept of a computation tree for alternating machines is quite analogous to the concept of a computation sequence for nondeterministic machines. This analogy is the basis for our generalized proofs. This is further evidence that alternation is a valid generalization of nondeterminism. Alternation is truly a robust concept in complexity theory.

1.3 Outline of the Remainder of the Paper.

In Section 2 we discuss the precise meaning of an alternating computation and apply it to Turing machines. This gives us an opportunity to recall the result of Chandra and Stockmeyer [2] and Kozen [12] that alternating Turing machines that are s(n)-space bounded have exactly the same power as deterministic Turing machines that are $2^{cs(n)}$-time bounded for some $c > 0$.

In Section 3 we investigate the power of alternating auxiliary pushdown automata. We show that

$$\text{ALT-AUX-PDA}(s(n)) = \cup\ \text{DTIME}(2^{2^{cs(n)}})$$

if $s(n) \geq \log n$. The proof of the left-to-right inclusion is an interesting generalization of Cook's proof that

$$\text{AUX-PDA}(s(n)) = \cup\ \text{DTIME}(2^{cs(n)})$$

if $s(n) \geq \log n$. The right-to-left inclusion is obtained by simulating $2^{cs(n)}$-space bounded alternating Turing machines by alternating auxiliary pushdown automata with auxiliary storage s(n), and then appealing to the result of Chandra, Kozen, and Stockmeyer which implies that a $2^{2^{cs(n)}}$-time bounded deterministic Turing machine can be simulated by a $2^{cs(n)}$-space bounded alternating Turing machine. We end the section by noting that we have actually proved that

$$\text{ALT-AUX-PDA}(s(n)) = \cup\ \text{DTIME}(2^{n2^{cs(n)}})$$

if $s(n) \geq 1$.

In Section 4 we investigate alternating two-way finite automata. We show how to simulate such an automaton with m states by a deterministic one-way finite automaton with 2^{m2^m} states. This simulation is not only interesting in its own right but is also used in Section 5 in our characterization of alternating auxiliary stack automata.

In Section 5 we investigate the power of alternating auxiliary stack automata, both erasing and nonerasing versions. We sketch the proof that

$$\begin{aligned}\text{ALT-AUX-SA}(s(n)) &= \text{ALT-AUX-NESA}(s(n))\\ &= \cup\ \text{DTIME}(2^{2^{2^{cs(n)}}})\end{aligned}$$

if $s(n) \geq \log n$. The result follows immediately by showing

(i) $\cup$ DTIME($2^{2^{2^{cs(n)}}}$) $\subseteq$ ALT-AUX-NESA(s(n))

and

(ii) ALT-AUX-SA(s(n)) $\subseteq \cup$ DTIME($2^{2^{2^{cs(n)}}}$).

Now, (i) is proved in a way similar to the right-to-left inclusion in Section 3. To show (ii) we show that

$$\text{ALT-AUX-SA}(s(n)) \subseteq \cup\ \text{ALT-AUX-PDA}(2^{cs(n)}).$$

This argument mirrors those in [3,7,9] for nondeterministic stack automata, where reading into the stack is eliminated in a way similar to the elimination of the left moves of a two-way finite automaton. This is where our results of Section 4 concerning alternating two-way finite automata are applied. Now, by our result in Section 3,

$$\cup\ \text{ALT-AUX-PDA}(2^{cs(n)}) \subseteq \cup\ \text{DTIME}(2^{2^{2^{cs(n)}}}).$$

We end the section by noting that we have actually shown that

$$\text{ALT-AUX-SA}(s(n)) = \text{ALT-AUX-NESA}(s(n)) = \bigcup \text{DTIME}(2^{2^{n2^{cs(n)}}})$$

for $s(n) \geq 1$.

In Section 6 we investigate the power of alternating auxiliary pushdown automata with a bounded number of alternations.

2. ALTERNATING AUTOMATA

Because alternation is a fairly new concept it pays to take time to study the meaning of an alternating computation and in particular the meaning of acceptance by an alternating machine. It helps to contrast the definition of alternating automata with that of nondeterministic automata. Think of an *automaton* as a set $\mathcal{I}$ of *instantaneous descriptions* (ID's) together with

(i) an *initialization function* INIT which takes inputs to ID's,

(ii) a set $\mathcal{A} \subseteq \mathcal{I}$ of *accepting* ID's,

(iii) a *transition relation* $\vdash \subseteq \mathcal{I} \times \mathcal{I}$ with the property that for all I, $\{ J : I \vdash J \}$ is finite.

An *alternating automaton* has in addition to (i)-(iii) the added property

(iv) a set $\mathcal{U} \subseteq \mathcal{I}$ of *universal* ID's.

The members of $\mathcal{I} - \mathcal{U}$ are called *existential* ID's.

Let M be an automaton. A *computation* of M is a finite nonempty sequence $I_0, I_1, \ldots, I_n$ such that $I_i \vdash I_{i+1}$ for $0 \leq i < n$. An *accepting computation* of M on an input x is a computation $I_0, I_1, \ldots, I_n$ such that $I_0 = \text{INIT}(x)$ and $I_n \in \mathcal{A}$. We say M *accepts* x if there is an accepting computation of M on input x.

Let N be an alternating automaton. By contrast, a *computation* or *computation tree* of N is a finite, nonempty labeled tree with the properties

(a) each node π of the tree is labeled with an ID, $\ell(\pi)$,

(b) if π is an internal node (a non-leaf) of the tree, $\ell(\pi)$ is universal and $\{ J : \ell(\pi) \vdash J \} = \{J_1, \ldots, J_k\}$, then π has exactly k children $\rho_1, \ldots, \rho_k$ such that $\ell(\rho_i) = J_i$,

(c) if π is an internal node of the tree and $\ell(\pi)$ is existential then π has exactly one child ρ such that $\ell(\pi) \vdash \ell(\rho)$.

An *accepting computation (tree)* of N on an input x is a computation tree whose root is labeled with INIT(x) and whose leaves each have labels in $\mathcal{A}$. We say N *accepts* x if there is an accepting computation of N on input x. Define $L(N) = \{x : N \text{ accepts } x\}$.

Deterministic and nondeterministic automata are special cases of alternating automata. An alternating automaton is nondeterministic if $\mathcal{U} = \phi$ and is deterministic if for each I there is at most one J such that $I \vdash J$.

We extend the complexity concepts of space and time to alternating computations. With each alternating automaton N we associate a *space complexity function* SPACE which takes ID's to natural numbers. Informally, SPACE(I) is the storage "used" by the ID I. We say that N is s(n)-*space bounded* if for all n and for all x of length n, if x is accepted by N then there is an accepting computation tree of N on input x such that for each node π of the tree, $\text{SPACE}(\ell(\pi)) \leq s(n)$. We say that N is t(n)-*time-bounded* if for all n and for all x of length n, if x is accepted by N then there is an accepting computation tree of N on input x of height $\leq t(n)$.

We now apply these concepts to the Turing machine. Formally an alternating Turing machine is an object of the form

$$M = (Q, q_0, U, F, \Sigma, \Gamma, \delta)$$

where

Q is a finite set of states,

$q_0 \in Q$ is the start state,

$U \subseteq Q$ is the set of universal states,

$F \subseteq Q$ is the set of accepting states,

Σ is the input alphabet ($¢, \$ \notin \Sigma$ serve as left and right endmarkers)

Γ is the tape alphabet ($\# \in \Gamma$ is the blank symbol)

$\delta : Q \times (\Sigma \cup \{¢,\$\}) \times \Gamma \to \mathcal{P}(Q \times \{R,L\}^2 \times (\Gamma - \{\#\}))$ is the transition function

where $\mathcal{P}$ denotes the power set operation, and R (L) signifies a right (left) shift of a head. Of course, we are defining a Turing machine with a read only input tape and one storage tape.

An ID has the form (q, x, i, α, j) where $q \in Q$, $x \in \Sigma^*$, $0 \leq i \leq |x| + 1$, $\alpha \in \Gamma^*$, and $0 \leq j \leq |\alpha| + 1$. An accepting ID has its first coordinate in F while a universal ID has its first coordinate in U. INIT(x) is $(q_0, x, 0, \lambda, 1)$. The transition relation between ID's is straightforward to define (see, for example, [8]). Finally $\text{SPACE}((q, x, i, \alpha, j)) = |\alpha|$. Define

$$\text{ASPACE}(s(n)) = \{ L(M) : M \text{ is an } s(n)\text{-space bounded alternating Turing machine}\}.$$

Chandra and Stockmeyer [2] and Kozen [12] prove the following useful theorem.

Theorem 2.1 (Chandra, Kozen, Stockmeyer). If $s(n) \geq \log n$ then

$$\text{ASPACE}(s(n)) = \bigcup_c \text{DTIME}(2^{cs(n)}).$$

The class DTIME(t(n)) is the usual class of languages recognized by t(n)-time bounded deterministic Turing machines.

Remark 2.2. Theorem 2.1 continues to hold even if the input head on the alternating machine is only allowed to move one-way, as can be seen from the proofs of Theorem 2.1 in [2,12].

To see this another way let M be a normal two-way $s(n)$-space bounded alternating Turing machine with $s(n) \geq \log n$. To construct a one-way alternating machine M' we have M' simulate M step by step. In order to simulate the reading of an input symbol, M' maintains on its tape a count of the input head position of M. When M reads an input symbol, M' guesses the symbol using an existential state and then enters a universal state to choose one of two further actions: one action is to continue the simulation of M and the other is to check that the symbol guessed is actually in the input position indicated by the count. This latter action is the only time the input head of M' moves and it can do so one-way.

3. ALTERNATING AUXILIARY PUSHDOWN AUTOMATA

The definition of an alternating auxiliary pushdown automaton is similar to Cook's definition of a nondeterministic auxiliary pushdown automaton [3] except that, as described in Section 2, a subset of the states are designated as universal states. Formally an *alternating auxiliary pushdown automaton* (*Alt-Aux-PDA*) is an object of the form

$$M = (Q, q_0, U, F, \Sigma, \Gamma, \Delta, \cent, \delta)$$

where

- Q is a finite set of states,
- $q_0 \in Q$ is the start state,
- $U \subseteq Q$ is the set of universal states,
- $F \subseteq Q$ is the set of accepting states,
- Σ is the input alphabet ($\cent, \$ \notin \Sigma$),
- Γ is the auxiliary storage alphabet ($\# \in \Gamma$ is the blank symbol),
- Δ is the pushdown store alphabet
- $\cent \in \Delta$ is the bottom symbol on the pushdown store,
- δ is the transition relation where

$$\delta : Q \times (\Sigma \cup \{\cent,\$\}) \times \Gamma \times \Delta \rightarrow \mathscr{P}(Q \times \{L,R,S\}^2 \times (\Gamma-\{\#\}) \times ((\Delta-\{\cent\}) \cup \{POP,IDLE\})).$$

The input is read-only and is delimited on the left by $\cent$ and the right by $\$$. If the machine is in state q scanning σ on the input tape, γ on the auxiliary storage tape, and p on the top of the pushdown store, and if $(q', d_1, d_2, \gamma', p')$ belongs to $\delta(q,\sigma,\gamma,p)$, then the machine can enter state q', shift the input head in direction d_1 (Left, Right or Stationary), write γ' on the auxiliary storage tape and shift the head in direction d_2, and manipulate the pushdown store by either (i) *pushing* p' onto the stack if $p' \in \Delta - \{\cent\}$, (ii) *popping* the top symbol if $p' = POP$, or (iii) *idling*, that is, not changing the pushdown store, if $p' = IDLE$. We assume that δ has been constrained so that the bottom symbol $\cent$ is never popped nor pushed and that the input head is never shifted outside the area delimited by the endmarkers (inclusive). Note also that the blank symbol cannot be written on the auxiliary storage tape. Other nonessential assumptions which help smooth out our proofs are that the machine enters an accepting state only if it is reading the bottom symbol $\cent$, and the machine behaves deterministically while it is either pushing or popping --that is, if D belongs to the range of δ and if D contains more than one move, then all moves in D are of the form $(q',d_1,d_2,\gamma',IDLE)$.

An ID has the form (q,x,i,α,j,β) where

- $q \in Q$ indicates the current state,
- $x \in \Sigma^*$ is the input,
- i where $0 \leq i \leq |x|+1$ indicates the input head position,
- $\alpha \in (\Gamma-\{\#\})^*$ indicates the contents of the nonblank portion of the auxiliary storage tape,
- j where $0 \leq j \leq |\alpha|$ indicates the auxiliary storage head position,
- $\beta \in \cent\Delta^*$ indicates the contents of the pushdown store.

The function INIT is defined by

$$INIT(x) = (q_0, x, 0, \lambda, 0, \cent).$$

The set of accepting ID's are those of the form $(q,x,i,\alpha,j,\cent)$ where $q \in F$. The set of universal ID's are those of the form (q,x,i,α,j,β) where $q \in U$. The transition relation is straightforward but tedious to define formally. The space of an ID is measured only on the auxiliary storage tape; that is

$$SPACE((q, x, i, \alpha, j, \beta)) = |\alpha|.$$

The definition of $L(M)$ (the set of words in Σ^* that M accepts) and the definition of M being $s(n)$-space bounded now follow from the general definitions given in Section 2. Define

$$\text{ALT-AUX-PDA}(s(n)) = \{ L(M) : M \text{ is an Alt-Aux-PDA which is } s(n)\text{-space bounded} \}.$$

Our principal result concerning alternating auxiliary PDA's is the following characterization.

Theorem 3.1. If $s(n) \geq \log n$ then

$$\text{ALT-AUX-PDA}(s(n)) = \bigcup_{c>0} \text{DTIME}(2^{2^{cs(n)}}).$$

Proof. (1) We first show that for any $c > 0$

$$\text{DTIME}(2^{2^{cs(n)}}) \subseteq \text{ALT-AUX-PDA}(s(n)).$$

We do this indirectly by showing that

$$\mathrm{ASPACE}(2^{cs(n)}) \subseteq \text{ALT-AUX-PDA}(s(n))$$

and appealing to Theorem 2.1.

Let M be an alternating Turing machine which is $2^{cs(n)}$-space bounded. Let Q be the states and Γ be the tape alphabet of M. We can of course assume that M has only one tape which is one-way infinite to the right. Therefore, the ID's of M (for a fixed input x of length n) can be viewed as words of length $2^{cs(n)}$ in $\Gamma^* \cdot (q \times \Gamma) \cdot \Gamma^*$; the meaning of $\mu(q,\gamma)\nu$ where $\mu,\nu \in \Gamma^*$, $q \in Q$, $\gamma \in \Gamma$, is that $\mu\gamma\nu$ is written on the first $2^{cs(n)}$ squares of the tape, and M is in state q scanning γ. For example

$$\mathrm{INIT}(x) = (q_0,x_1)x_2x_3 \ldots x_n \#^d$$

where $x = x_1x_2 \ldots x_n$ and $d = 2^{cs(n)} - n$.

We can assume without loss of generality that each ID of M has at most two successors. This is enforced by assuming that the transition function of M is a partial function of the form

$$\delta : Q \times \Gamma \to (Q \times \{L,R,S\} \times (\Gamma - \{\#\}))^2.$$

For certain $(q,\gamma) \in Q \times \Gamma$, the two components of $\delta(q,\gamma)$ can be identical (indicating a deterministic move). $\delta(q,\gamma)$ is undefined iff q is an accepting state of M. For ID's α and β we write $\alpha \vdash_1 \beta$ (resp., $\alpha \vdash_2 \beta$) iff $|\alpha| = |\beta|$ and α can reach β in one move according to the first (second) component of δ. (The condition $|\alpha| = |\beta|$ ensures that ID's remain $2^{cs(n)}$-space bounded. For example, if $\alpha = \mu(q,\gamma)$ and the first component of $\delta(q,\gamma)$ moves the head right, then there is no β such that $\alpha \vdash_1 \beta$.)

Assume for the moment that s(n) is tape constructible [8]. Fix an input x of length n. The Alt-Aux-PDA M' which simulates M first lays off a block of s(n) tape squares on its auxiliary storage tape. In general, the pushdown store of M' will contain a string of the form

$$\alpha_0 m_1 \alpha_1 m_2 \alpha_2 m_3 \alpha_3 \ldots$$

where $\alpha_0, \alpha_1, \ldots$ are ID's of M, $m_i \in \{1,2\}$ for $i \geq 1$ (we assume $1,2 \notin Q \cup \Gamma$), $\alpha_0 = \mathrm{INIT}(x)$, and $\alpha_{i-1} \vdash_{m_i} \alpha_i$ for $i \geq 1$. The symbol m_i is chosen by a universal (existential) branch of M' if α_{i-1} is a universal (existential) ID of M. The words α_i for $i \geq 1$ are chosen by existential branching (i.e., the usual nondeterministic guessing). After each extention of α_i by a new guessed symbol, say γ, M' enters a universal state to choose one of two further actions: one action is to continue choosing α_i; the other action is to check that γ is consistent with a legal move of M according to $\vdash_{m_i}$ and to accept or reject accordingly. To perform this check, we use the fact that if $\alpha_{i-1} \vdash_m \alpha_i$ then the jth symbol of α_i is uniquely determined by m and the (j-1)th, jth, and (j+1)th symbols of α_{i-1}. The auxiliary storage tape of M' is used as a counter to measure the distance (roughly) $2^{cs(n)}$ between the jth symbol of α_i and the jth symbol of α_{i-1}. Of course, the stack must be popped to perform the check.

Notice that the alternation of M' is being used in two ways. First in choosing the m_i it is used directly to simulate the alternation of M. Second, alternation is used to (existentially) guess and (universally) check the ID's of M. Universal branching permits one branch of the computation to read into the stack, popping and destroying information, while another branch retains the information for future use.

It is useful to make more precise the fact that $\alpha \vdash_m \beta$ can be checked by performing local checks within α and β. For a word α, let $\alpha(j)$ denote the jth symbol of α for $1 \leq j \leq |\alpha|$. For $m=1,2$ there is a function

$$f_{M,m} : (\Gamma \cup Q \times \Gamma \cup \{1,2\})^3 \to \Gamma \cup Q \times \Gamma$$

such that if α and β are ID's of M with $|\alpha| = |\beta| = \ell$, then $\alpha \vdash_m \beta$ iff

$$\begin{aligned}
\beta(j) &= f_{M,m}(\alpha(j-1),\alpha(j),\alpha(j+1)) \text{ for } 1<j<\ell, \\
\beta(1) &= f_{M,m}(1,\alpha(1),\alpha(2)) = f_{M,m}(2,\alpha(1),\alpha(2)), \\
\beta(\ell) &= f_{M,m}(\alpha(\ell-1),\alpha(\ell),1) = f_{M,m}(\alpha(\ell-1),\alpha(\ell),2).
\end{aligned}$$

We now describe the procedures of M' more carefully. The *and*'s ($\wedge$) and *or*'s ($\vee$) in these procedures are implemented by using alternation; that is, $A \wedge B$ (resp., $A \vee B$) means to enter a universal (resp., existential) state to choose which one of A or B to perform.

```
INIT:   Push INIT(x) onto the pushdown store. (The s(n)
        auxiliary storage is used as a counter to ensure that
        INIT(x) is the correct length $2^{cs(n)}$.)

TOP:    If the top ID is accepting then accept; else
        If the top ID is universal then
            ((push 1 $\wedge$ push 2); call NEW); else
        If the top ID is existential then
            ((push 1 $\vee$ push 2); call NEW).

NEW:    $\ell \leftarrow 0$;
    C:  $\ell \leftarrow \ell+1$;
        If $\ell > 2^{cs(n)}$ then call TOP;
        $\bigvee_{\gamma \in \Gamma \cup Q \times \Gamma}$ push $\gamma$;
        (call CHECK $\wedge$ go to C).
```

CHECK: Remember the top pushdown symbol, say γ, in the finite control;
Pop $2^{cs(n)}$ symbols off the pushdown store -- at the point when a symbol $m \in \{1,2\}$ is popped, remember m in the finite control;
Pop and remember three more symbols, say γ_3, γ_2, and γ_1;
If $\gamma = f_{M,m}(\gamma_1,\gamma_2,\gamma_3)$ then accept ; else reject.

M' executes INIT;TOP. A proof that M accepts x iff M' accepts x can be based on the obvious correspondence between computations of M and computations of M'. Figure 1 shows the correspondence for a universal configuration α of M with two successors β_1 and β_2. Although the existential branch in the procedure NEW can push any symbol γ onto the store, only the correct choice will survive the subsequent call to CHECK. Further details of the proof that M' correctly simulates M are left to the reader.

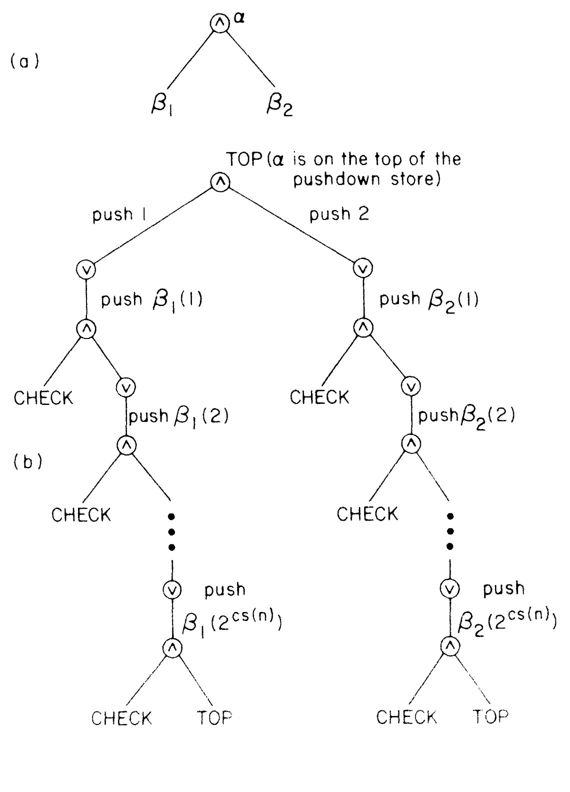

Figure 1. (a) A universal branch of the Turing machine M, and (b) the corresponding portion of an accepting computation of the Alt-Aux-PDA M'.

If s(n) is not constructible then M' guesses the value of s(n) using existential branching. If M accepts x, then M' accepts x after guessing the correct value s(n); so M' is s(n)-space bounded. If M does not accept x, then for no guessed value of s(n) does M' accept x.

(2) To prove that
$\text{ALT-AUX-PDA}(s(n)) \subseteq \cup\, \text{DTIME}(2^{2^{cs(n)}})$

we generalize the proof of Cook [3] that a s(n)-space bounded nondeterministic auxiliary PDA can be simulated by a $2^{cs(n)}$-time bounded deterministic Turing machine.

Let M be an s(n)-space bounded Alt-Aux-PDA and let x be an input of length n. A *surface ID* is an object of the form (q,x,i,α,j,B) where q indicates the current state, x is the input, i indicates the input head position, α indicates the contents of the auxiliary storage tape, j indicates the auxiliary storage head position, and B indicates the top symbol on the pushdown store. Let $top((q,x,i,\alpha,j,B)) = B$. Recall that we made the restriction that the PDA behaves deterministically when pushing or popping. Hence only when the pushdown store is idle can there be true branching in the automaton. With this in mind we can think of the machine in three possible *modes*: PUSH, POP, and IDLE. IDLE surface ID's are partitioned into U-IDLE and E-IDLE ID's depending on whether they are universal or existential. We can view the transition relation $\vdash$ of M as a relation on surface ID's provided that the first component of the relation is not in POP mode.

A *surface computation* is a finite rooted tree whose nodes are labeled with surface ID's and which has the following properties:

(a) On each path from the root to a leaf, the sequence of modes traversed (including the mode of the root but not the mode of the leaf) can be generated by the following grammar SURFACE:

$<S> \rightarrow \lambda$
$<S> \rightarrow$ IDLE
$<S> \rightarrow <S><S>$
$<S> \rightarrow$ PUSH$<S>$POP;

that is, the PUSH's and POP's are matched analogously to left and right parentheses.

(b) A PUSH node has exactly one child labeled with the surface ID obtained by simulating the PDA for one move. A POP node has exactly one child whose label is obtained by simulating the PDA for one move and looking back along the path to the root to the matching PUSH node to obtain the top symbol of the pushdown store. (This is possible because of (a)).

(c) If a node is labeled with a U-IDLE surface ID r, then for each w such that $r \vdash w$ the node has a child with label w. If a node is labeled with an E-IDLE surface ID r, then the node has exactly one child and for some w such that $r \vdash w$ the child has label w.

The surface ID (q,x,i,α,j,B) is s(n)-*bounded* iff $|\alpha| \leq s(|x|)$. A surface computation is s(n)-*bounded* iff all surface ID's in the computation are s(n)-bounded. If r is an s(n)-bounded surface ID and W is a set of s(n)-bounded surface ID's, then we write

$$r \Rrightarrow W$$

iff there is an s(n)-bounded surface computation T whose root is labeled r and whose leaf labels are contained in W, and we say that T *witnesses* $r \Rrightarrow W$ (see Figure 2). Because M accepts only when it reads the bottom symbol ¢, then x is accepted by M if and only if $I(x) \Rrightarrow A$ for some set A of accepting surface ID's, where

$$I(x) = (q_0,\ x,\ 0,\ \lambda,\ 0,\ ¢)$$

is the initial surface ID.

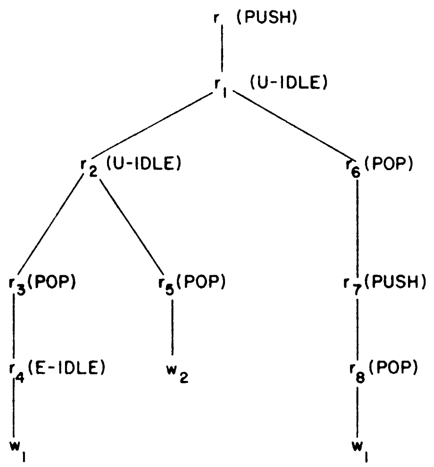

Figure 2. A surface computation which witnesses $r \Rrightarrow \{w_1,w_2,w_3\}$.

We now define a kind of proof system. In this proof system we will "prove" *terms* of the form $r \to \{w_1,\ldots,w_k\}$ where $r,w_1,\ldots,w_k$ are s(n)-bounded surface ID's. The system has the following *proof rules*:

1. $$\frac{}{r \to \{r\}}$$

2. (a) $$\frac{r \to W,\ W \subseteq V}{r \to V}$$

(b) $$\frac{r \to W \cup \{w\},\ w \to V}{r \to W \cup V}$$

3. (a) $$\frac{\begin{array}{c} r\ (\text{E-IDLE}) \\ | \\ w \end{array}}{r \to \{w\}}$$

(b) $$\frac{\begin{array}{c} r\ (\text{U-IDLE}) \\ \swarrow \quad \downarrow \quad \searrow \\ w_1 \quad w_2 \ \cdots \ w_k \end{array}}{r \to \{w_1,\cdots,w_k\}}$$

4. $$\frac{\begin{array}{c} r\ (\text{PUSH}) \\ | \\ w \end{array},\ w \to \{v_1,\ \cdots\ ,v_k\},\quad \begin{array}{c} v_1\ (\text{POP}) \\ | \\ z_1 \end{array},\ \cdots\ ,\ \begin{array}{c} v_k\ (\text{POP}) \\ | \\ z_k \end{array}}{r \to \{z_1,\cdots,z_k\}}$$

In each proof rule, the term below the horizontal bar can be concluded from the information above the bar. Lower case letters stand for s(n)-bounded surface ID's and upper case letters stand for sets of s(n)-bounded surface ID's. In 3a, r is an existential surface ID and $r \vdash w$. In 3b, r is a universal surface ID and $\{w_1,\cdots,w_k\} = \{\ w : r \vdash w\ \}$; in order to apply this rule, *all* successors of r must be s(n)-bounded. In 4, w is the successor of the surface ID r which is in PUSH mode. Furthermore, z_i is the successor of v_i which is in POP mode and $\text{top}(z_i) = \text{top}(r)$.

For x fixed, there are at most

$$bn \cdot s(n) \cdot 2^{b's(n)} \leq 2^{b''s(n)}$$

s(n)-bounded surface ID's for some constants $b, b', b'' > 0$ (recall that $s(n) \geq \log n$), so there are at most $2^{2^{ds(n)}}$ terms for some $d > 0$. This is the key to our $2^{2^{cs(n)}}$-time bounded simulation of M. In order to prove that the simulation is correct, we first show that the proof rules are sound and complete.

Lemma 3.2. Let r be a surface ID and let G be a set of surface ID's. Then

$$r \twoheadrightarrow G \text{ iff } r \Rrightarrow G.$$

Proof. (if) Let T be an s(n)-bounded surface computation whose root is labeled r and whose leaf labels are contained in G. The proof is by induction on the *size* (i.e., the number of nodes) of T. If the size of T is 1, then $r \in G$, so $r \twoheadrightarrow G$ is proved by rules 1 and 2a. If the size of T is greater than 1, we consider three cases.

(i) r is an E-IDLE surface ID. Let ω be the (unique) child of the root of T, and let w be the label of ω. Let T' be the maximal subtree of T with root ω. Note that T' witnesses $w \Rrightarrow G$. Since the size of T' is less than the size of T, it follows that $w \twoheadrightarrow G$. But $r \twoheadrightarrow \{w\}$ by rule 3a, so $r \twoheadrightarrow G$ by rule 2b.

(ii) r is a U-IDLE surface ID. Let $\omega_1,\cdots,\omega_k$ be the children of the root of T, and let w_i be the label of ω_i for $1 \le i \le k$. Let T_i be the maximal subtree of T with root ω_i, and note that T_i witnesses $w_i \Rrightarrow G$. Now $r \twoheadrightarrow \{w_1,\cdots,w_k\}$ by rule 3b and $w_i \twoheadrightarrow G$ by induction, so $r \twoheadrightarrow G$ by rule 2b.

(iii) r is a PUSH surface ID. As in the previous two cases, we describe how to break T into pieces such that there is a proof for each piece by induction and such that the pieces can be put together using the proof rules to give a proof of $r \twoheadrightarrow G$. Let ρ be the root and ω be the child of the root of T. Let w be the label of ω. If τ and ξ are nodes of T, we write surf(τ,ξ) to indicate that either $\tau = \xi$, or ξ is a proper descendant of τ and the sequence of modes traversed on the path from τ to ξ (including the mode of τ but not that of ξ) can be generated by the grammar SURFACE defined above. Let T' be the (unique) subtree of T such that the root of T' is ρ and, for all nodes ξ of T' with $\xi \neq \rho$, surf(ρ,ξ) iff ξ is a leaf of T'. (In other words, we obtain T' by pruning each branch of T at the node where the pushdown store first returns to the same level as r.) Let $\zeta_1,\cdots,\zeta_m$ be the leaves of T'. For $1 \le i \le m$, let ν_i be the parent of ζ_i and let $\{v_1,\cdots,v_k\}$ be the set of labels of the nodes $\nu_1,\cdots,\nu_m$. Possibly $k < m$ since some of these nodes might be labeled the same. Since surf(ρ,ζ_i) but not surf(ρ,ν_i), it follows that surf(ω,ν_i) and v_i must be in POP mode for all i. Therefore, the set of labels of $\zeta_1,\cdots,\zeta_m$ must be $\{z_1,\cdots,z_k\}$ where v_i pops to yield z_i and top(z_i) = top(r) for all i.

By deleting the nodes $\rho, \zeta_1,\cdots,\zeta_m$ from T' we obtain a surface computation which witnesses $w \Rrightarrow \{v_1,\cdots,v_k\}$, so $w \twoheadrightarrow \{v_1,\cdots,v_k\}$ by induction. By applying rule 4

(4.1) $$r \twoheadrightarrow \{z_1,\cdots,z_k\}.$$

Now for $1 \le i \le m$, let T_i be the maximal subtree of T with root ζ_i; note that T_i witnesses $z_i \Rrightarrow G$, so $z_i \twoheadrightarrow G$. Combining this with (4.1) using rule 2b gives $r \twoheadrightarrow G$.

Obviously the root of T cannot be in POP mode, so the proof by induction is complete.

(only if) This is proved by induction on the length of a proof that $r \twoheadrightarrow G$. If the last step of the proof (the step from which $r \twoheadrightarrow G$ is concluded) is an application of rule 1, 3a, or 3b, then it is immediate by the definition of surface computation that $r \Rrightarrow G$.

If the last step is an application of rule 2a, then the surface computation which witnesses the antecedant term also witnesses the conclusion term. If the last step uses rule 2b with $G = W \cup V$, then by induction there are surface computations T and T' which witness $r \Rrightarrow W \cup \{w\}$ and $w \Rrightarrow V$, respectively. If each leaf of T which is labeled w is replaced by the root of T' (which is also labeled w), the resulting tree witnesses $r \Rrightarrow W \cup V$.

In the case that the last step uses rule 4, the induction step follows by a similar pasting of surface computations, and we leave this case to the reader ▮

Assuming that s(n) is tape constructible, the deterministic Turing machine M' which simulates M generates all provable terms by simply applying the proof rules until no new terms can be proved. Initially the set $\mathcal{J}$ of provable terms is empty. At a given stage, M' attempts to generate a new term by applying the proof rules to $\mathcal{J}$ in all possible ways. If t is the number of terms, then the time for each stage is polynomial in t. Since there are at most t stages, the total time is polynomial in t, that is, $2^{2^{cs(n)}}$ for some $c > 0$. M' accepts x iff there is a term $I(x) \twoheadrightarrow A$ where A is a set of accepting surface ID's. So L(M) = L(M') by Lemma 3.2.

If s(n) is not constructible, then M' iterates the above procedure for $s(n) = 1,2,3,\cdots$ until it accepts. If M does accept x, then M' will discover this fact when s(n) reaches its correct value. The total time is a supergeometric series which is dominated by the last term $2^{2^{cs(n)}}$.

Theorem 3.1 can be easily generalized to cases where s(n) grows slower than log n. (We assume for convenience that $s(n) \ge 1$.) In part (1) of the proof, the s(n) storage is used only as a counter with capacity $2^{cs(n)}$. For general s(n), the input head can be utilized in conjunction with the s(n) storage to provide a counter with capacity $n2^{cs(n)}$. In part (2) of the proof, the key quantity is the number of s(n)-bounded surface ID's (with x fixed); in general, $n2^{cs(n)}$ is an upper bound on their number. Therefore, we have actually proved the following.

Theorem 3.3 If $s(n) \ge 1$ then

$$\text{ALT-AUX-PDA}(s(n)) = \bigcup_{c>0} \text{DTIME}(2^{n2^{cs(n)}}).$$

4. ALTERNATING TWO-WAY FINITE AUTOMATA

In this section we show how to simulate an alternating two-way finite automaton with m states by a deterministic one-way finite automaton with 2^{m2^m} states. Our construction combines ideas in Chandra and Stockmeyer's construction of a deterministic one-way finite automaton which simulates an alternating one-way finite automaton [2] and Shepherdson's construction of a deterministic one-way finite automaton which simulates a deterministic two-way finite automaton [15]. Our simulation will be used in Section 5 where we show that an alternating auxiliary stack machine with $s(n)$ auxiliary storage can be simulated by an alternating auxiliary PDA with $2^{cs(n)}$ auxiliary storage for some $c > 0$.

Formally an *alternating two-way finite automaton* (Alt-2-FA) is an object of the form:

$$M = (Q, q_0, U, F, \Sigma, \delta)$$

where:

- Q is a finite set of states,
- $q_0 \in Q$ is the start state,
- $U \subseteq Q$ is the set of universal states,
- $F \subseteq Q$ is the set of accepting states,
- Σ is the input alphabet ($¢,\$ \notin \Sigma$ serve as left and right endmarkers)
- δ is the transition relation where

$$\delta : Q \times (\Sigma \cup \{¢, \$\}) \rightarrow \mathscr{P}(Q \times \{L,R,S\})$$

The symbol $\mathscr{P}$ denotes the power set operation. The input is delimited on the left by $¢$ and on the right by $\$$. For convenience we assume that the machine starts in the start state while reading the right endmarker. The machine accepts only if it has just read the right endmarker and moved right into an accepting state. Further, the machine moves right or left deterministically, that is, if $(p,d) \in \delta(q,a)$ and $d \in \{R,L\}$ then $\delta(q,a) = \{(p,d)\}$. Hence the automaton can only do universal or existential branching while the read head is stationary. It is useful to give a precise definition of acceptance by alternating two-way finite automata using the terminology of Section 2. An ID has the form (q,z,i) where $q \in Q$, $z \in (\Sigma \cup \{¢,\$\})^*$ and $1 \leq i \leq |z|+1$. A universal ID has the form (q,z,i) where $q \in U$. An accepting ID has the form $(q, z, |z|+1)$ where $q \in F$. The initialization function INIT mapping Σ^* into the set of ID's is defined by

$$\text{INIT}(x) = (q_0, ¢x\$, |x|+2).$$

Let $z = a_1a_2...a_n$ where $a_i \in \Sigma \cup \{¢, \$\}$. The transition relation is defined by

$(q,z,i) \vdash (p,z,i)$ if $(p,S) \in \delta(q,a_i)$ and $1 \leq i \leq n$,

$(q,z,i) \vdash (p,z,i+1)$ if $(p,R) \in \delta(q,a_i)$ and $1 \leq i \leq n$.

$(q,z,i) \vdash (p,z,i-1)$ if $(p,L) \in \delta(q,a_i)$ and $1 < i \leq n$.

Let $M = (Q, q_0, U, F, \Sigma, \delta)$ be an Alt-2-FA. We now begin the description of a deterministic one-way FA M' which simulates M. For each $q \in Q$ define a new symbol $\bar{q}$ and let $\bar{Q} = \{ \bar{q} : q \in Q\}$. If $A \subseteq Q$. then $\bar{A} = \{ \bar{r} : r \in A \}$. Define a *term* to be an object of the form $q \rightarrow A$ where $q \in Q$ and $A \subseteq Q \cup \bar{Q}$. A term $q \rightarrow A$ is *closed* if $A \subseteq \bar{Q}$. A *partial response* is a set of terms while a *response* is a set of closed terms. The states of M' are exactly the responses.

At this point we give a nonconstructive definition of the rest of the components of M'. Later in this section we show how to construct it. For each $z \in (\Sigma \cup \{¢,\$\})^*$ we define a response $\mathscr{R}(z)$. A closed term $q \rightarrow \bar{A}$ is in $\mathscr{R}(z)$ if $q \in Q$, $A \subseteq Q$ and there is a computation tree of M whose root is labeled with $(q, z, |z|)$ and each leaf is labeled with $(p, z, |z|+1)$ for some $p \in A$. In other words there is a computation tree of M starting in state q and reading the rightmost symbol of z such that each branch ends in a state of A while moving off the right end of z.

Lemma 4.1. If $\mathscr{R}(w) = \mathscr{R}(z)$ then $\mathscr{R}(wa) = \mathscr{R}(za)$ for $a \in \Sigma \cup \{¢,\$\}$.

Before proving Lemma 4.1 we complete the definition of M'. The transition function δ' is defined by

$$\delta'(\mathscr{R},a) = \begin{cases} \mathscr{R}(za) & \text{if } \mathscr{R} = \mathscr{R}(z), \\ \phi & \text{if } \mathscr{R} \neq \mathscr{R}(z) \text{ for any } z. \end{cases}$$

δ' is well defined by Lemma 4.1. The start state of M' is $\mathscr{R}(¢)$, and $\mathscr{R}$ is an accepting state of M' if $q_0 \rightarrow \bar{F} \in \delta'(\mathscr{R},\$)$. Now x is accepted by M if and only if $q_0 \rightarrow \bar{F} \in \mathscr{R}(¢x\$)$ if and only if $\delta'(\mathscr{R}(¢),x)$ is an accepting state of M'. Hence M and M' accept the same language.

Proof of Lemma 4.1. In order to prove this it is useful to associate with each $z \in (\Sigma \cup \{¢,\$\})^*$ a partial response $\mathscr{PR}(z)$. A term $q \rightarrow A$ is in $\mathscr{PR}(z)$ if $q \in Q$, $A \subseteq Q \cup \bar{Q}$ and there is a computation tree T of M whose root is labeled with $(q, z, |z|)$ and each leaf is labeled with either $(p, z, |z|)$ for some $p \in A$ or $(p, z, |z|+1)$ for some $\bar{p} \in A$. We say that T *witnesses* $q \rightarrow A \in \mathscr{PR}(z)$. In words, $q \rightarrow A \in \mathscr{PR}(z)$ if there is a computation tree of M starting in state q and reading the rightmost symbol of z such that each branch ends in a state $p \in A$ while reading the rightmost symbol of z again or ends in a state p where $\bar{p} \in A$ while moving off the right end of z. The response $\mathscr{R}(z)$ is exactly the set of closed terms in $\mathscr{PR}(z)$. Since $\mathscr{PR}(w) = \mathscr{PR}(z)$ implies $\mathscr{R}(w) = \mathscr{R}(z)$ then we can prove the lemma by showing that $\mathscr{R}(w) = \mathscr{R}(z)$ implies $\mathscr{PR}(wa) = \mathscr{PR}(za)$.

We now introduce a constructive method of producing $\mathscr{PR}(wa)$ from $\mathscr{R}(w)$ and a. We consider a kind of proof system where we "prove" terms. Let $\mathscr{R}$ be a response and let $a \in \Sigma \cup \{¢,\$\}$.

Proof System for $\mathscr{R}$,*a*

1. $$\frac{}{q \to \{q\}}$$

2. (*a*) $$\frac{q \to A,\ A \subseteq B}{q \to B}$$

(*b*) $$\frac{q \to A \cup \{p\},\ p \to B}{q \to A \cup B}$$

3. $$\frac{\delta(q,a) = \{(p_1,S),\ldots,(p_k,S)\},\ q \text{ universal}}{q \to \{p_1,\ldots,p_k\}.}$$

4. $$\frac{(p,S) \in \delta(q,a),\ q \text{ existential}}{q \to \{p\}}$$

5. $$\frac{\delta(q,a) = \{(p,R)\}}{q \to \{\bar{p}\}}$$

6. $$\frac{\delta(q,a) = \{(p,L)\},\ p \to \bar{A} \in \mathscr{R},\ A \subseteq Q}{q \to A}$$

Let $TH(\mathscr{R},a)$ be the set of terms "provable" using the proof system for $\mathscr{R}$,a.

Claim $\mathscr{PR}(wa) = TH(\mathscr{R}(w),a)$.

We prove that $\mathscr{PR}(wa) \subseteq TH(\mathscr{R}(w),a)$ inductively on the size of witnesses for terms in $\mathscr{PR}(wa)$. Suppose $q \to A \in \mathscr{PR}(wa)$ is witnessed by a single node tree with the label $(q,wa,|wa|)$. Clearly $q \in A$ so that $q \to A$ is provable using rules 1 and 2a. Suppose now that $q \to A \in \mathscr{PR}(wa)$ is witnessed by a tree T with more than one node. There are four cases to consider.

(i) $\delta(q,a) = \{(p_1,S),\ldots,(p_k,S)\}$ and q is universal. In this case the root of T is labeled with $(q, wa, |wa|)$ with k immediate children $\pi_1,\ldots,\pi_k$ labeled respectively with $(p_1,wa,|wa|),\ldots,(p_k,wa,|wa|)$. The complete subtree rooted at π_i witnesses $p_i \to A \in \mathscr{PR}(wa)$. By the induction hypothesis $p_i \to A \in TH(\mathscr{R}(w),a)$. By 3 and 2b we obtain $q \to A \in TH(\mathscr{R}(w),a)$.

(ii) $\delta(q,a) = \{(p_1,S),\ldots,(p_k,S)\}$ and q is existential. This case is just like (i) except we use 4 and 2b.

(iii) $\delta(q,a) = \{(p,R)\}$. We must have $\bar{p} \in A$ for T to witness $q \to A \in \mathscr{PR}(wa)$. Hence by rules 5 and 2a we have $q \to A \in TH(\mathscr{R}(w),a)$.

(iv) $\delta(q,a) = \{(p,L)\}$. The child π of the root of T has the label $(p,wa,|wa|-1)$. Every path from π to a leaf of T must pass through a node with label of the form $(r,wa,|wa|)$. That is, the computation must return to reading the rightmost symbol of wa again. Let $P = \{\rho_1,\ldots,\rho_k\}$ be the descendants of π with the properties (a) ρ_i is labeled $(r_i,wa,|wa|)$, (b) no node between π and ρ_i has a label with third coordinate $|wa|$ and (c) every path from π to a leaf passes through a node of P. Let T' be the unique subtree of T, whose root is π and whose set of leaves is P. If we change the second coordinate of every node label of T' from wa to w then T' witnesses $p \to \{\bar{r}_1,\ldots,\bar{r}_k\} \in \mathscr{PR}(w)$. Since this is a closed term then $p \to \{\bar{r}_1,\ldots,\bar{r}_k\} \in \mathscr{R}(w)$. By 6, $q \to \{r_1,\ldots,r_k\} \in TH(\mathscr{R}(w),a)$. The complete subtree of T rooted at ρ_i witnesses $r_i \to A \in \mathscr{PR}(wa)$. So by the induction hypothesis $r_i \to A$ is provable for all i. Hence by 2b, $q \to A \in TH(\mathscr{R}(w),a)$.

We prove that $TH(\mathscr{R}(w),a) \subseteq \mathscr{PR}(wa)$ by induction on proof length. Suppose $q \to A \in TH(\mathscr{R}(w),a)$ has a proof of length ℓ. If the last step of the proof (the step from which $q \to A$ is concluded) is an application of rule 1, 3, 4, or 5, then it is immediate that there is a computation tree which witnesses $q \to A \in \mathscr{PR}(wa)$. If the last step is an application of rule 2a then the same computation tree that witnesses the antecedent term also witnesses the conclusion term. If the last step is an application of rule 2b then suppose that $q \to B \cup \{p\}$ and $p \to C$ are the antecedents from which $q \to B \cup C$ is concluded $(A = B \cup C)$. By the induction hypothesis there are computation trees T and T' which witness $q \to B \cup \{p\}$ and $p \to C$ respectively. If each leaf of T labeled $(p,wa,|wa|)$ is replaced with the tree T' (which has root labeled $(p,wa,|wa|)$ then the resulting tree witnesses $q \to B \cup C \in \mathscr{PR}(wa)$.

If the last step is an application of rule 6 then suppose $q \to A$ is concluded from $\delta(q,a) = \{(p,L)\}$, $p \to \bar{A} \in \mathscr{R}(w)$ and $A \subseteq Q$. Since $p \to \bar{A} \in \mathscr{R}(w)$, then there is a computation tree T' with root labeled $(p,w,|w|)$ and each leaf labeled $(r,w,|w|+1)$ for some $r \in A$. First modify T' so that the second coordinate of each node label is wa instead of w. The tree with root labeled $(q,wa,|wa|)$ and root with one child which is the root of the modified T' witnesses $q \to A$.

This concludes the proof of the claim. We are now ready to complete the proof of Lemma 4.1. Assume $\mathscr{R}(w) = \mathscr{R}(z)$. By the claim, $TH(\mathscr{R}(w),a) = \mathscr{PR}(wa)$ and $TH(\mathscr{R}(z),a) = \mathscr{PR}(za)$. Thus $\mathscr{PR}(wa) = \mathscr{PR}(za)$. ∎

We now know how to construct the transition function δ' of M'. If $\mathscr{R}$ is a response and $a \in \Sigma \cup \{¢,\$\}$ then first using the proof system for $\mathscr{R}$ and a construct the set $TH(\mathscr{R},a)$. Now $\delta'(\mathscr{R},a)$ is the set of closed terms in $TH(\mathscr{R},a)$.

In the next section we use the fact that, if m is the number of states of M, then a response can be written in space 2^{cm} and $\delta'(\mathscr{R},a)$ can be computed in deterministic time 2^{dm} for some $c,d > 0$. The method of computing $\delta'(\mathscr{R},a)$ is to generate the terms in $TH(\mathscr{R},a)$ using the proof system for $\mathscr{R}$ and a.

5. ALTERNATING STACK AUTOMATA

An (alternating) stack automaton is just like an (alternating) pushdown automaton except that the interior contents of the pushdown store may be read, but not changed except by normal pushing or popping. A stack can also be viewed as a Turing machine tape, one-way infinite to the right, with the restriction that symbols can be changed only on the right end of the non-blank tape contents. In the special case of a nonerasing stack automaton, the stack cannot be popped, or, equivalently, stack symbols cannot be erased. As in [9], we consider stack automata with a space-bounded auxiliary storage tape. Formally, an *alternating auxiliary stack automaton (Alt-Aux-SA)* is of the form

$$M = (Q, S, q_0, U, F, \Sigma, \Gamma, \Delta, ¢, \delta)$$

where Q (the states), q_0 (the start state), U (the universal states), F (the accepting states), Σ (the input alphabet), Γ (the auxiliary storage alphabet), Δ (the stack alphabet), and $¢$ (the bottom, or leftmost, stack symbol) are as in the definition of Alt-Aux-PDA's. $S \subseteq Q$ and the states in S are called *scan states*. We also define $P = Q - S$ and refer to states in P as *pushdown states*. The blank symbol # belongs to Δ as well as to Γ. The transition function is of the form

$$\delta : Q \times (\Sigma \cup \{¢,\$\}) \times \Gamma \times \Delta \rightarrow \mathscr{P}(Q \times \{L,R,S\}^2 \times (\Gamma - \{\#\}) \times ((\Delta - \{\#,¢\}) \cup \{POP, IDLE, L, R\})),$$

where R (L) in the last component signifies a right (left) shift of the stack head.

When the state of the machine is in S (P) the machine is said to be in *scan mode (pushdown mode)*. Initially, the machine is in pushdown mode (i.e., $q_0 \in P$), the stack contains $¢\#\#\#\cdots$ and the stack head is scanning $¢$. Generally, when in pushdown mode the stack contains $¢\beta\#\#\#\cdots$ for some $\beta \in (\Delta - \{\#,¢\})^*$ and the stack head is scanning the rightmost symbol of β; the machine behaves just as an Alt-Aux-PDA, manipulating the stack by pushing, popping, or idling. At some point, the machine can enter scan mode without moving the stack head. While in scan mode the machine can read stack symbols and move the stack head left and right (and idle), but it cannot push or pop. When in scan mode the machine behaves much like an alternating Turing machine with an auxiliary storage tape, and with the stack providing a read-only "input" (in addition to the original read-only input in Σ^*). The machine must remain in scan mode until it first reads a blank (which must be the blank just to the right of β). Then it must shift the stack head left and enter pushdown mode. Furthermore, we require that $F \subseteq P$, but we now allow M to accept when reading stack symbols other than $¢$. We assume that the machine behaves deterministically when either pushing, popping, moving the stack head left or right, or changing from pushdown mode to scan mode or vice versa. Some of these conventions for stack automata differ from those in [7,9], but they are convenient for our proofs.

An *alternating auxiliary nonerasing stack automaton (Alt-Aux-NESA)* is an Alt-Aux-SA which cannot pop the stack.

An ID of an Alt-Aux-SA has the form (q,x,i,α,j,β,k) where q, x, i, α, j, and β have the same meaning as for ID's of Alt-Aux-PDA's, and k, $1 \leq k \leq |\beta|+1$, indicates the position of the stack head. The stack positions are numbered from left to right so, for example, $k=1$ if M is reading the leftmost symbol $¢$, or $k = |\beta|$ if M is reading the top nonblank stack symbol. Now

$$\mathrm{INIT}(x) = (q_0, x, 0, \lambda, 0, ¢, 1),$$

the accepting ID's have their first coordinate in F, and the universal ID's have their first coordinate in U. The definition of the transition relation should be clear from the discussion above. Let

$$\mathrm{SPACE}((q, x, i, \alpha, j, \beta, k)) = |\alpha|.$$

Define

$$\text{ALT-AUX-SA}(s(n)) = \{L(M) : M \text{ is an Alt-Aux-SA which is } s(n)\text{-space bounded}\},$$

$$\text{ALT-AUX-NESA}(s(n)) = \{L(M) : M \text{ is an Alt-Aux-NESA which is } s(n)\text{-space bounded}\}.$$

One consequence of our characterization of space bounded alternating auxiliary stack automata is that the ability to erase the stack is inessential.

Theorem 5.1. Let $s(n) \geq \log n$.

$$\text{ALT-AUX-SA}(s(n)) = \text{ALT-AUX-NESA}(s(n)) = \bigcup_{c>0} \mathrm{DTIME}(2^{2^{2^{cs(n)}}}).$$

Since obviously ALT-AUX-NESA$(s(n)) \subseteq$ ALT-AUX-SA$(s(n))$, Theorem 5.1 follows immediately from Lemmas 5.2 and 5.3.

Lemma 5.2. If $s(n) \geq \log n$, then

$$\bigcup \mathrm{DTIME}(2^{2^{2^{cs(n)}}}) \subseteq \text{ALT-AUX-NESA}(s(n)).$$

Proof sketch. By Theorem 2.1, it is sufficient to prove that

$$\bigcup \mathrm{ASPACE}(2^{2^{cs(n)}}) \subseteq \text{ALT-AUX-NESA}(s(n)).$$

The proof is very similar to the first part of the proof of Theorem 3.1. The only difference is that now the ID's of the alternating Turing machine are words of length $2^{2^{cs(n)}}$. The extra exponential is handled by preceding each symbol of an ID by a binary address; for each ID, the addresses run consecutively from 0 to $2^{2^{cs(n)}}-1$. Since the length of an address is only $2^{cs(n)}$, s(n) storage is sufficient to record a pointer to a particular bit-position within an address. Therefore an s(n)-space bounded Alt-Aux-NESA can check that two physically consecutive addresses are numerically consecutive (in fact, this can be done deterministically), and it can check that the address of a symbol deep in the stack which is being scanned in scan mode matches the address of the symbol on the top of the stack (universal branching is used here). This ability to match addresses is used to implement a procedure similar to CHECK in the proof of Theorem 3.1. ∎

Lemma 5.3. If $s(n) \geq \log n$, then

$$\text{ALT-AUX-SA}(s(n)) \subseteq \bigcup \text{DTIME}(2^{2^{2^{cs(n)}}}).$$

Proof. By Theorem 3.1, it is sufficient to prove that

$$\text{ALT-AUX-SA}(s(n)) \subseteq \bigcup \text{ALT-AUX-PDA}(2^{cs(n)}).$$

Let

$$M = (Q, S, q_0, U, F, \Sigma, \Gamma, \Delta, ¢, \delta)$$

be an Alt-Aux-SA which is s(n)-space bounded. A $2^{cs(n)}$-space bounded Alt-Aux-PDA M' will perform a step-by-step simulation of M when M is in pushdown mode. During this simulation, M' will maintain on its pushdown store the response of the stack contents of M (see §4); this will allow M' to simulate M when M is in scan mode.

Fix an input x of length n. When in scan mode, M can be viewed as an alternating two-way finite automaton $\mathcal{A}_M$ with about $2^{es(n)}$ states for some constant $e > 0$. The states of $\mathcal{A}_M$ are of the form (q, i, α, j) where $q \in S$, i where $0 \leq i \leq n+1$ indicates M's input head position, $\alpha \in (\Gamma-\{\#\})^*$ with $|\alpha| \leq s(n)$ indicates the contents of M's auxiliary tape, and j indicates the auxiliary storage head position. The universal states of $\mathcal{A}_M$ are those of the form (q, i, α, j) where $q \in U$. The input alphabet of $\mathcal{A}_M$ is Δ, the stack alphabet of M. For our purposes here, we need not specify an initial state or accepting states for $\mathcal{A}_M$. It should be obvious how the transition function of $\mathcal{A}_M$ is obtained from that of M; note that the ID

$$((q, i, \alpha, j), \beta, k) \text{ of } \mathcal{A}_M$$

corresponds to the ID

$$(q, x, i, \alpha, j, \beta, k) \text{ of } M.$$

To describe the simulation, suppose that M is in some ID

(5.1) $\qquad (q, x, i, \alpha, j, \beta, |\beta|).$

M' will maintain q, i, α, and j on its auxiliary storage tape. Furthermore, the pushdown store of M' will contain

$$\beta_1\mathcal{R}(\rho_1)\beta_1\beta_2\mathcal{R}(\rho_2)\beta_2 \cdots \beta_\ell\mathcal{R}(\rho_\ell)\beta_\ell$$

where $\beta = \beta_1\beta_2 \cdots \beta_\ell$, $\beta_i \in \Delta$ for all i, ρ_i is the length i prefix of β (in particular, $\rho_\ell = \beta$), and the response $\mathcal{R}$ is with respect to $\mathcal{A}_M$. The concept of a *response* is defined and discussed in Section 4. To recapitulate briefly in the context of this proof, if $q \in S$ and $(q, i, \alpha, j) \twoheadrightarrow A$ is a term in $\mathcal{R}(\beta)$ where A is a set of states of $\mathcal{A}_M$, then there is a computation tree of M with root (5.1) and such that if $(q',x,i',\alpha',j',\beta',k)$ is a leaf of the computation, then $(q',i',\alpha',j') \in A$, $\beta' = \beta$, and $k = |\beta|+1$; in particular, at each leaf the stack scan has just finished and M must reenter pushdown mode.

We now describe how M' simulates M in an ID (5.1) in various cases. If $q \in F$ then M' accepts. If (5.1) has more than one successor, then M' simulates this directly by using its own alternation (recall that the stack head cannot move in this case). If M pops the stack, then M' pops $\beta_\ell\mathcal{R}(\beta)\beta_\ell$ off the pushdown store (while updating q, i, α, j as necessary).

If M pushes a symbol $a \in \Delta$ onto the stack, then M' pushes a, then existentially pushes some string of symbols in the alphabet used to encode responses, and then pushes a again. M' then enters a universal state to choose one of two further actions. One action is to continue the simulation as though $\mathcal{R}(\beta a)$ was guessed correctly. The other action is to verify that the guess really was correct. We must argue that this can be done using $2^{cs(n)}$ space. As discussed in Section 4 following the proof of Lemma 4.1, the set

$$\text{UPDATE} = \{(\mathcal{R}, a, \mathcal{R}') : \mathcal{R} = \mathcal{R}(z) \text{ and } \mathcal{R}' = \mathcal{R}(za) \text{ for some } z \in \Delta^* \}$$

can be accepted by a deterministic Turing machine within time 2^{dm} where $m \leq 2^{es(n)}$ is the number of states of $\mathcal{A}_M$; this is done by applying the proof system for $\mathcal{R}$,a. So by Theorem 2.1 and Remark 2.2, UPDATE is accepted by an alternating Turing machine M'' which is $2^{cs(n)}$-space bounded; moreover M'' has a one-way input head which we can assume starts on the right end of the input and moves left. (We are using here the fact that the space $2^{cs(n)}$ is at least logarithmic in the length of the "input" $(\mathcal{R},a,\mathcal{R}')$. This is true because, as remarked in §4, a response can be written in space $2^{c'm}$ for some constant c'.) So M' can simulate M'' where the pushdown head of M' plays the role of the left-moving input head of M''.

If M' has entered scan mode (i.e., if $q \in S$ in (5.1)), then M' existentially guesses a term

$$(q,i,\alpha,j) \to A$$

where A is a set of states of $\mathcal{A}_M$ and writes it on the auxiliary storage tape. M' then enters a universal state to choose one of two further actions. One action is to check that $(q,i,\alpha,j) \to A \in \mathcal{R}(\beta)$; this is done in the obvious way by popping the stack. The other action is to universally choose some state $(q',i',\alpha',j') \in A$ and continue the simulation as though M were in the ID

$$(q',x,i',\alpha',j',\beta,|\beta|+1).$$

By convention, from this ID M must move the stack head left and reenter pushdown mode, so M is again in an ID of the form (5.1). ∎

As discussed at the end of Section 3, we have actually proved the following result for general s(n).

Theorem 5.4. Let $s(n) \geq 1$.

$$\text{ALT-AUX-SA}(s(n)) = \text{ALT-AUX-NESA}(s(n)) = \bigcup_{c>0} \text{DTIME}(2^{2^{n2^{cs(n)}}})$$

6. BOUNDED ALTERNATION

Bounded alternation pushdown automata are just like the machines of Section 3 except that the number of alternations allowed during accepting computations is now bounded. Just as in the study of quantifier hierarchies it is convenient to allow the automata to start either in an existential or universal state. For integer $k \geq 1$, M is a Σ_k-Aux-PDA (resp., Π_k-Aux-PDA) if M is an Alt-Aux-PDA, the initial state of M is existential (resp., universal), and for every input x accepted by M there is an accepting computation tree of M on x such that along every root-to-leaf path of the tree there are at most $k-1$ places where M changes from an existential state to a universal state or vice versa. For example, a Σ_1-AUX-PDA is nothing more than a nondeterministic auxiliary PDA. As before let Σ_k-AUX-PDA(s(n)) and Π_k-AUX-PDA(s(n)) be the classes of languages accepted by these automata when they are s(n)-space bounded.

We first note that:

Theorem 6.1. Let $s(n) \geq \log n$.

(1) Σ_1-AUX-PDA $= \cup\ \text{DTIME}(2^{cs(n)})$

(2) Π_1-AUX-PDA $= \cup\ \text{DTIME}(2^{cs(n)})$

Of course, (1) is just Cook's theorem [3]. We omit the proof of (2) but remark that it follows in essentially the same way as (1).

Based on this theorem, one might conjecture that Σ_k and Π_k agree for all k. However there is a basic asymmetry between existential and universal states that is demonstrated by our main theorem.

Theorem 6.2. Let s(n) be tape constructible and $s(n) \geq \log n$.

(1) Π_2-AUX-PDA $= \cup\ \text{co-NTIME}(2^{cs(n)})$

(2) Σ_2-AUX-PDA $= \cup\ \text{DSPACE}(2^{cs(n)})$

(3) Π_3-AUX-PDA $= \cup\ \text{DSPACE}(2^{cs(n)})$

(4) Σ_3-AUX-PDA $= \cup\ \text{DSPACE}(2^{cs(n)})$

Outline of proof. By now the reader should be familiar with a number of our basic techniques. Therefore we only sketch the main ideas.

First, we show how to use Π_2-Aux-PDA's to simulate co-NTIME($2^{cs(n)}$). Let M be a nondeterministic Turing machine which is $2^{cs(n)}$-time bounded. While in universal states, the Π_2-Aux-PDA M' chooses a sequence of ID's of M. M' is careful neither to choose too long an ID nor too many -- it achieves this by using its auxiliary tape as a counter. Then M' switches to existential states to guess where the ID sequence is incorrect, and M' accepts if it is incorrect. So M' accepts iff M does not accept.

Now we explain how to use Σ_2-Aux-PDA's to simulate DSPACE($2^{cs(n)}$). Let M be a deterministic Turing machine which is $2^{cs(n)}$-space bounded. The Σ_2-Aux-PDA M' existentially guesses an ID sequence of M; M' does not attempt to count the number of ID's in the sequence, but rather M' existentially guesses when the sequence should end. Then M' switches to universal states to check that the ID sequence is correct everywhere.

It is interesting to note the root of the asymmetry here. In the first case we could not allow the universal states to "get in a loop" since then the Π_2-Aux-PDA would never accept, but in the second case we could allow the existential states of the Σ_2-Aux-PDA to loop.

The left-to-right inclusions are obtained in part by further analyzing Cook's methods [3]. For example, a Σ_2-Aux-PDA can be thought of as two finite automata, each with about $2^{cs(n)}$ states. The states of the first FA are the existential surface ID's; this FA nondeterministically outputs a word which is the contents of the pushdown store at the end of the sequence of existential moves. The second FA is a universal FA which takes pushdown-store-contents as input; its states are the universal surface ID's.

The Σ_2-Aux-PDA accepts iff the first FA can generate a word accepted by the second FA. This can be decided in space polynomial in the number of states of the two FA. (3) and (4) are proved by a more complicated elaboration of this idea. The left-to-right inclusion in (1) also uses this idea together with the above observation that the universal surface ID's of a Π_2-Aux-PDA cannot get into a loop; this implies that the pushdown store contains a word of length less than $2^{cs(n)}$ at the point when the alternation from universal to existential occurs. ∎

We close this section by asking whether or not

$$\Sigma_k\text{-AUX-PDA} = \cup \, \text{DSPACE}(2^{cs(n)})$$

for all $k \geq 2$?

REFERENCES

1. L. Berman, Precise bounds for Presburger arithmetic and the reals with addition, Proc. 18th Symp. on Foundations of Computer Science (1977), 95-99.
2. A. K. Chandra and L. J. Stockmeyer, Alternation, Proc. 17th Symp. on Foundations of Computer Science (1976), 98-108.
3. S. A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, *J.ACM 18* (1971), 4-18.
4. M. J. Fischer and R. E. Ladner, Propositional modal logic of programs, Proc. 9th ACM Symp. on Theory of Computing (1977), 286-294.
5. S. Ginsburg, S. A. Greibach, and M. A. Harrison, Stack automata and compiling, *J.ACM 14* (1967), 172-201.
6. J. Gray, M. A. Harrison, and O. H. Ibarra, Two-way pushdown automata, *Information and Control 11* (1967), 30-70.
7. J. E. Hopcroft and J. D. Ullman, Nonerasing stack automata, *J. Comput. System Sci. 1* (1967), 166-186.
8. J. E. Hopcroft and J. D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.
9. O. H. Ibarra, Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata, *J. Comput. System Sci. 5* (1971), 88-117.
10. M. Jazayeri, W. F. Ogden, W. C. Rounds, The intrinsically exponential complexity of the circularity problem for attribute grammars, *C.ACM 18* (1975), 697-706.
11. D. Kleitman and G. Markowski, On Dedekind's problem: the number of isotone Boolean functions II, *Trans. AMS 213* (1975), 373-390.
12. D. Kozen, On parallelism in Turing machines, Proc. 17th Symp. on Foundations of Computer Science (1976), 89-97.
13. D. Kozen, Complexity of Boolean algebras, unpublished report, Computer Science Division, University of California, Berkeley, CA, 1978.
14. W. L. Ruzzo, General context free language recognition, Ph.D. thesis, Computer Science Division, University of California, Berkeley, CA, 1978.
15. J. C. Shepherdson, The reduction of two-way automata to one-way automata, *IBM J. Res. 3* (1959), 198-200.

Addresses and Acknowledgments

Richard E. Ladner, Department of Computer Science, University of Washington, Seattle, WA 98195; the work of this author was supported in part by NSF Grant No. MCS 77-02474 and was performed in part while visiting Yale University and the IBM Watson Research Center.

Richard J. Lipton, Department of Computer Science, Yale University, New Haven, CT 06520; currently on leave at the Department of Computer Science, University of California, Berkeley, CA 94720; work supported in part by NSF Grant No. MCS 78-81486.

Larry J. Stockmeyer, Mathematical Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.