# ALTERNATING MULTIHEAD FINITE AUTOMATA*
(extended abstract)

K. N. King

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
U.S.A.

## 1. Introduction

We consider a restricted version of the alternating Turing machine model introduced in [2] and [13]. Our machine, the alternating multihead finite automaton, has no storage tape, just a read-only input tape with multiple heads. A primary reason for studying such automata is that alternation adds so much power to automata that it is necessary to examine very restricted alternating automata if one wishes to compare alternating classes with familiar classes such as the context-free languages, log space, and so forth. (Note that an alternating pushdown automaton is as powerful as a deterministic Turing machine with an exponential time bound [1].) We are also interested in determining the simplest kind of device for which alternation adds computational power.

We hope that the study of alternating multihead finite automata gives us information about the structure of deterministic polynomial time, a very important class, since (as we show later) the languages accepted by two-way alternating multihead finite automata are exactly the languages accepted by deterministic Turing machines in polynomial time.

Our results are summarized in Section 6.

## 2. Preliminaries

The symbol N denotes the set of positive integers, $\Lambda$ denotes the empty string, and if x is a string, $|x|$ denotes the length of x. All logarithms are to the base 2.

### Multihead finite automata

**Definition.** A two-way alternating finite automaton with k heads (2afa(k)) is a structure $M = (K,\Sigma,\delta,\tau,q_0,U,F)$, where K is a finite set of states; $\Sigma$ is the input alphabet (not containing $\not{\epsilon}$ and \$); $\delta$ is the transition function, mapping $K \times (\Sigma \cup \{\not{\epsilon},\$\})$ into the subsets of $K \times \{-1,0,+1\}$, with the restriction that for all $p,q \in K$, $(q,d) \in \delta(p,\not{\epsilon})$ implies that $d \geq 0$ and $(q,d) \in \delta(p,\$)$ implies that $d \leq 0$; $\tau$ is the head selector function, mapping K into $\{1,2,\ldots,k\}$; $q_0 \in K$ is the initial state; $U \subseteq K$ is the set of universal states; and $F \subseteq K$ is the set of accepting states.

If $(p,d) \in \delta(q,a)$ and $\tau(q) = h$, then M, if it is in state q with head h scanning a on the input tape, may enter state p and move head h to the right d squares.

**Definition.** Let M be a 2afa(k) as defined above. If $U = \emptyset$, then M is a two-way

*This work represents a portion of the author's doctoral dissertation completed at the University of California, Berkeley. The research was supported by NSF grants MCS 74-07636-A01 and MCS 79-15763.

nondeterministic finite automaton with k heads (2nfa(k)). If for all $q \in K$, $a \in \sum \cup \{¢,\$\}$, $\delta(q,a)$ is empty or a singleton, then M is a two-way deterministic finite automaton with k heads (2dfa(k)). M is a one-way alternating finite automaton with k heads (1afa(k)) if $\delta$ maps $K \times (\sum \cup \{¢,\$\})$ into the subsets of $K \times \{0,+1\}$. If M is one-way and nondeterministic [deterministic], then M is called a 1nfa(k) [1dfa(k)]. A configuration of M on input $x \in \sum^*$ is a $(k+1)$-tuple $(q,i_1,\ldots,i_k)$, where $q \in K$ and $1 \leq i_j \leq |x| + 2$ for $1 \leq j \leq k$.

A configuration of M describes a state of M's computation: q is the state of M's finite control and $i_1,\ldots,i_k$ represent the positions of M's k heads.

**Definition.** Let M be a 2afa(k) as defined above. The transition relation on configurations of M with input x is given by $(q,i_1,\ldots,i_k) \vdash_{M,x} (p,i_1,\ldots,i_h+d,\ldots,i_k)$ if and only if $(p,d) \in \delta(q,a)$, where $\tau(q) = h$ and a is the $i_h$th symbol of $¢x\$$. For C and D configurations of M, we say that D is a successor of C (on input x) if $C \vdash^+_{M,x} D$. If $C \vdash_{M,x} D$, then D is an immediate successor of C.

The initial configuration of M is $(q_0,1,\ldots,1)$. A configuration $C = (q,i_1,\ldots,i_k)$ is said to be universal if $q \in U$, existential if $q \in K - U$, and accepting if $q \in F$. An accepting computation tree of M on input x is a tree T whose nodes are labelled by configurations of M on input x such that (i) the root of T is labelled by the initial configuration of M, (ii) if C is a universal configuration that labels an internal node y of T, then the labels of the immediate descendants of y are exactly the immediate successors of C on input x, (iii) if C is an existential configuration that labels an internal node y of T, then y has exactly one immediate descendant, whose label is one of the immediate successors of C on input x, and (iv) the leaves of T are labelled with accepting configurations. We say that M accepts x if there is an accepting computation tree of M on input x. Let $L(M) = \{x \in \sum^* \mid M \text{ accepts } x\}$.

The family of languages $\{L(M) \mid M \text{ is a 2afa(k)}\}$ is denoted 2AFA(k). Similarly, the families accepted by 2nfa(k), 2dfa(k), 1afa(k), 1nfa(k), and 1dfa(k) are denoted 2NFA(k), 2DFA(k), 1AFA(k), 1NFA(k), and 1DFA(k), respectively.

The following lemma states that we can assume that an alternating multihead finite automaton can detect coincidence of heads, even though the model does not explicitly contain this feature.

**Lemma 2.1.** Let M be a 2afa(k) [1afa(k)] that has the (additional) ability to detect when two heads coincide. There exists a 2afa(k) [1afa(k)] M' lacking this capability that accepts the same language as M.

**Proof.** Whenever M', during its simulation of M, wants to branch, depending on whether two heads coincide, it enters an existential state to guess whether the heads coincide. If M' guesses that they do, it enters a universal state to choose whether to continue the simulation (under the assumption that the heads coincide) or to move the two heads to the right simultaneously, entering an accepting state if

and only if both heads reach the right endmarker at the same time. A guess that the heads do not coincide is handled in a similar fashion. □

**Multihead pushdown automata**

**Definition.** A two-way nondeterministic pushdown automaton with k heads (2npda(k)) is a structure $M = (K,\Sigma,\Delta,\delta,\tau,q_0,Z_0,F)$, where K, $\Sigma$, $\tau$, $q_0$, and F are the same as for a 2afa(k), $\Delta$ is a finite set of pushdown symbols, $Z_0 \in \Delta$ is the start symbol, and $\delta$ maps $K \times (\Sigma \cup \{\not{c},\$\}) \times \Delta$ into the subsets of $K \times \{-1,0,+1\} \times (\{\Lambda\} \cup \Delta \cup \Delta^2)$.

If $(p,d,\gamma) \in \delta(q,a,Z)$ and $\tau(q) = h$, then M, if it is in state q with head h scanning a on the input tape and with Z on top of the pushdown, may enter state p, move head h to the right d squares, and replace Z on the pushdown by the string $\gamma$.

**Definition.** Let M be a 2npda(k) as defined above. A configuration of M on input x is a (k + 2)-tuple $C = (q,i_1,\ldots,i_k,\alpha)$, where $q \in K$, $1 \le i_j \le |x| + 2$ for $1 \le j \le k$, and $\alpha \in \Delta^*$; C is a surface configuration if $\alpha \in \Delta$.

A configuration has the same meaning as for a multihead finite automaton, except that the string $\alpha$ represents the contents of M's pushdown, with the top of the pushdown to the right. The definition of the transition relation on configurations is straightforward, as are the definitions of deterministic and one-way. A string x is in L(M) if and only if $(q_0,1,\ldots,1,Z_0) \overset{*}{\vdash}_{M,x} (q,i_1,\ldots,i_k,\alpha)$ for some $q \in F$. Let 2NPDA(k) = {L(M) | M is a 2npda(k)}. The classes 2DPDA(k), 1NPDA(k), and 1DPDA(k) are defined similarly.

We now give two lemmas that will be the basis for simulations of multihead pushdown automata in Section 3. The proofs of these lemmas are straightforward.

**Lemma 2.2.** Let M be a 2npda(k) [1npda(k)]. We can construct a 2npda(k) [1npda(k)] M' such that L(M') = L(M) and such that, whenever M' enters an accepting state, there is only one symbol on the pushdown.

**Definition.** Let M be a 2npda(k), let $P = (p,i_1,\ldots,i_k,Z)$ and $Q = (q,j_1,\ldots,j_k,Y)$ be surface configurations of M on some input x, and let n be a nonnegative integer. The pair (P,Q) is n-realizable (on input x) if $P \overset{n}{\vdash}_{M,x} Q$.

**Lemma 2.3.** If P and Q are as defined above, then the pair (P,Q) is n-realizable if and only if either

(1) $P \overset{n}{\vdash}_{M,x} Q$ via a series of moves that do not change the pushdown height, or

(2) there exist surface configurations $R = (r,\ell_1,\ldots,\ell_k,X)$ and $S = (s,m_1,\ldots,m_k,W)$ such that $(p,i_1,\ldots,i_k,Z) \vdash_{M,x} (r,\ell_1,\ldots,\ell_k,YX)$, $(s,m_1,\ldots,m_k,W) \vdash_{M,x} (q,j_1,\ldots,j_k,\Lambda)$, and (R,S) is (n-2)-realizable, or

(3) there exist a surface configuration R and an integer i, $1 \le i \le n$, such that (P,R) and (R,Q) are i- and (n-i)-realizable, respectively.

**Turing machines and auxiliary pushdown automata**

Let ASPACE(S(n)) denote the class of languages accepted by S(n) space-bounded

alternating Turing machines, and define NSPACE(S(n)) and DSPACE(S(n)) analogously for nondeterministic and deterministic Turing machines. Let P denote the class of languages accepted by polynomial-time-bounded deterministic Turing machines.

An auxiliary pushdown automaton (auxpda) [3] is just an off-line nondeterministic Turing machine with an additional pushdown store. Let AuxPDA(S(n)) denote the class of languages accepted by S(n) space-bounded auxpda's.

## 3. Relationships with nonalternating pushdown automata

The following theorem states that the 'limit' of the 2AFA(k) classes, as well as those of the more familiar 2DPDA(k) and 2NPDA(k) classes, is the family P.

**Theorem 3.1.**

$$\bigcup_{k\in N} 2DPDA(k) = \bigcup_{k\in N} 2NPDA(k) = AuxPDA(\log n) = P = ASPACE(\log n) = \bigcup_{k\in N} 2AFA(k).$$

**Proof.** Cook [3] showed that AuxPDA(log n) = P; the result P = ASPACE(log n) is due to [1]. The equalities $\bigcup_{k\in N} 2DPDA(k) = AuxPDA(\log n)$ and $\bigcup_{k\in N} 2NPDA(k) = AuxPDA(\log n)$ may be shown by a standard construction (see [8], for example) that shows how to simulate a log n space-bounded tape by multiple two-way input heads, and vice-versa. The fact that $ASPACE(\log n) = \bigcup_{k\in N} 2AFA(k)$ follows from this same construction. □

Theorem 3.1 raises the question of the number of heads required for an alternating finite automaton to simulate a deterministic (or nondeterministic) multihead pushdown automaton, and vice-versa. (All devices in the sequel have multiple input heads, so to save space we usually omit the word 'multihead.')

**Theorem 3.2.** For $k \geq 1$, $2NPDA(k) \subseteq 2AFA(3k)$.

**Proof.** Let $M = (K,\Sigma,\Delta,\delta,\tau,q_0,Z_0,F)$ be a 2npda(k). By Lemma 2.2 we can assume that $w \in L(M)$ if and only if $(q_0,1,\ldots,1,Z_0) \vdash^n_{M,w} (q,j_1,\ldots,j_k,Y)$ for some $q \in F$, $1 \leq j_1,\ldots,j_k \leq |w| + 2$, $Y \in \Delta$, and $n \geq 0$. Lemma 2.3 now enables us to construct a 2afa(3k) M' that accepts the same language as M. On input w, M' need only check that $(P_0,Q_f)$ is n-realizable, where $P_0 = (q_0,1,\ldots,1,Z_0)$ and $Q_f = (q,j_1,\ldots,j_k,Y)$. Note that M' can store a surface configuration of M using k heads, plus some finite memory, so M' can store three surface configurations of M simultaneously. Initially, M' uses 2k heads to store $P_0$ and $Q_f$ (the latter is chosen existentially).

To check a pair (P,Q) for realizability, M' chooses existentially which one of (1), (2), or (3) of Lemma 2.3 to check. If M' chooses to check (1), it attempts to transform P into Q via a series of moves (chosen existentially) that do not change the pushdown height. If a surface configuration derived from P by a series of such moves matches Q (M' can detect this, since it has the ability to check for coincidence of heads), then M' enters an accepting state. Condition (2) is checked by choosing existentially a move that takes P to some configuration R and a move that takes some configuration S to Q, changing P into R and Q into S, then checking (R,S) for realizability. If M' chooses to check (3), it chooses existentially some surface configuration R (using the extra k heads to store R) then enters a universal state to choose whether to check (P,R) or (R,Q) for realizability.

Lemma 2.3 guarantees that, if $w \in L(M)$, there is an accepting computation for M' on w. As it checks a pair (P,Q), if M' chooses correctly which of (1), (2), or (3) holds, then it either halts (case (1)) or begins to check a 'smaller' pair (cases (2) and (3)). Thus, if M' always guesses correctly, it will halt and accept along each path.

On the other hand, if M' accepts a string w, then M must also accept it. This follows from a weaker form of Lemma 2.3 in which 'n-realizability' is replaced by 'realizability' and '$\vdash^n_{M,x}$' by '$\vdash^*_{M,x}$'. □

The simulation employed in the proof of Theorem 3.2 will not suffice to prove 1NPDA(k) $\subseteq$ 1AFA(3k), although the simulation that we use also relies on Lemma 2.3.

**Theorem 3.3.** For $k \geq 1$, 1NPDA(k) $\subseteq$ 1AFA(3k).

**Proof.** Let $M = (K,\Sigma,\Delta,\delta,\tau,q_0,Z_0,F)$ be a 1npda(k). By Lemma 2.2 we can assume that, on input w, a 1afa(3k) M' need only check that $(P_0,Q_f)$ is n-realizable, where $P_0 = (q_0,1,\ldots,1,Z_0)$ and $Q_f = (q,j_1,\ldots,j_k,Y)$, for some $q \in F$, $1 \leq j_1,\ldots,j_k \leq |w| + 1$, $Y \in \Delta$, and $n \geq 0$. (Note that one-way automata need no left endmarker ¢.)

M' uses three heads ([h,1], [h,2], and [h,3]) to simulate each head h of M. As it begins to check a pair (P,Q) for realizability, where $P = (p,i_1,\ldots,i_k,Z)$ and $Q = (q,j_1,\ldots,j_k,Y)$, M' has P and Q stored as follows. States p and q and pushdown symbols Z and Y are stored in the finite control. The head positions $i_1,\ldots,i_k$ and $j_1,\ldots,j_k$ are stored in the following way: the position of head [h,1] directly represents $i_h$, and the distance from head [h,1] to head [h,2] represents $|w| + 1 - j_h$. At this point, head [h,3] is not in use, and coincides with head [h,2]. The finite control of M' contains the symbol of w at position $j_h$ (the symbol at position $i_h$ can be read directly from the tape). The simulation depends on the fact that, if (P,Q) is realizable, then $i_h \leq j_h$, since M is one-way.

To check (P,Q) for realizability, M' guesses which one of (1), (2), or (3) of Lemma 2.3 to check. If it selects (1), it then attempts to transform P into Q via a series of moves (chosen existentially) that do not change the pushdown height. If a move requires head h to move right, M' moves heads [h,1] and [h,2] to the right. If M' detects that a surface configuration (containing state p' and pushdown symbol Z', say) derived from P by a series of such moves matches Q (all heads [h,2] scan the endmarker and both p' = q and Z' = Y), then M' enters an accepting state.

If M' chooses (2), then it guesses a move from P to some configuration R and a move from some configuration S to Q, changing P into R and Q into S, then checking (R,S) for realizability. Suppose that changing P to R requires moving head h one square to the right. M' moves heads [h,1] and [h,2] each one square to the right. If changing Q into S requires moving some head, say h, one square to the left (because the move that takes S into Q moves it one square to the right), M' moves head [h,2] one square to the right and stores in its finite control a guess for the symbol scanned by head h in S such that M can legally move from S to Q. M' next chooses universally either to begin checking (R,S) or to move heads [h,1] and [h,2]

simultaneously to the right until the latter head reaches the endmarker, entering an accepting state if and only if head [h,1] scans the symbol just guessed.

Now suppose that M' has chosen (3). M' guesses some surface configuration $R = (r,\ell_1,\ldots,\ell_k,X)$ such that $i_h \leq \ell_h \leq j_h$ for all h, storing r and X in its finite control. Each $\ell_h$ is guessed by moving head [h,3] to the right either (i) $j_h - \ell_h$ squares (if M' guesses that $j_h - \ell_h < \ell_h - i_h$) or (ii) $\ell_h - i_h$ squares (if M' guesses that $\ell_h - i_h \leq j_h - \ell_h$). These two cases are depicted in Figs. 1(a) and 1(b), respectively.

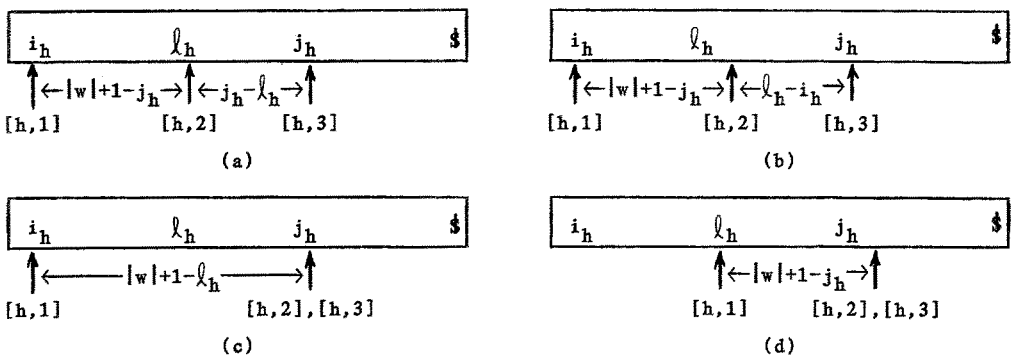



**Figure 1.**

M' now enters a universal state to choose whether to check that (P,R) is realizable or that (R,Q) is realizable.

If M' has chosen to check (P,R), it first replaces q and Y in its finite control by r and X. For each h, if case (i) applies, M' moves head [h,2] to the right until it coincides with head [h,3]. If case (ii) applies, M' moves heads [h,2] and [h,3] simultaneously to the right until it guesses that head [h,3] is now $\ell_h - i_h$ squares from the endmarker. M' next chooses universally either to verify this guess or to continue. In the former case, M' moves heads [h,2] and [h,3] to the right simultaneously (moving head [h,2] twice as fast as head [h,3]), entering an accepting state if and only if both heads reach the endmarker at the same time. In the latter case, M' moves head [h,2] to the right until it coincides with head [h,3]. In both cases (i) and (ii), each set of heads is eventually positioned as in Fig. 1(c). M' now guesses the symbols at positions $\ell_1,\ldots,\ell_k$ on the input and chooses universally either to begin checking (P,R) for realizability or to move each pair of heads [h,1], [h,2] simultaneously to the right until the latter head reaches the endmarker, entering an accepting state if and only if each head [h,1] scans the symbol guessed to be at position $\ell_h$.

If M' has chosen to check (R,Q), it first replaces p and Z in its finite control by r and X. Whenever case (i) applies, then M' performs the operations described above under case (ii), except that [h,1] is moved one square to the right whenever [h,2] is moved. Whenever case (ii) applies, M' performs the operations described above under case (i), except that [h,1] is moved in conjunction with

[h,2]. The final position of the heads is as shown in Fig. 1(d). □

Having given efficient simulations of one- and two-way pushdown automata by alternating finite automata, we now consider the simulation of one- and two-way alternating finite automata by pushdown automata.

**Lemma 3.4.** Let M be a 2afa(k) with s states. If $w \in L(M)$, then there is an accepting computation tree for M on w of height at most $sn^k$, where $n = |w| + 2$.

**Proof.** Let M be a 2afa(k) with s states. Suppose that $w \in L(M)$, and let T be the acceptance tree for M on w with the fewest nodes. For $n = |w| + 2$, there are $sn^k$ distinct configurations of M on w. Thus, if the height of T is greater than $sn^k$, there must be two nodes x and y labelled with the same configuration such that x is a proper ancestor of y. By removing the subtree rooted at x and replacing it by the subtree rooted at y, we obtain an acceptance tree with fewer nodes, which is a contradiction. □

**Theorem 3.5.** For $k \geq 1$,

(a) $2AFA(k) \subseteq 2NPDA(k)$,

(b) $2AFA(k) \subseteq 2DPDA(2k)$,

(c) $1AFA(k) \subseteq 2DPDA(k)$.

**Proof.** (a) Let M be a 2afa(k). A 2npda(k) M′ can simulate M on input w by performing a nondeterministic depth-first search of an accepting computation tree T for M on w (assuming that one exists). M′ uses its k heads exactly as M would, and uses its pushdown to keep track of the search by pushing a symbol representing the old state and the move from that state each time a move of M is simulated.

When M′ encounters a universal configuration C of M, it verifies that all immediate successors of C lead to accepting computations by exploring each subtree below C. When M′ encounters an existential configuration of M, it guesses which move of M to simulate and verifies that this choice leads to an accepting subtree.

(b) The simulation is the same as in (a), except that here M′ (now a dpda(2k)), when it reaches an existential configuration, must try each immediate successor in turn, since it cannot guess which one labels the root of an accepting subtree. That is, M′ does not reject the input when a nonaccepting configuration of M is reached which has no successors; rather, it backs up to its lowest existential ancestor and tries the next choice.

By Lemma 3.4, if $w \in L(M)$, then there is an accepting computation tree for M on w of height at most $sn^k$, where $n = |w| + 2$ and s is the number of states in M, so M′ maintains a count indicating the current level of the search. Whenever the count reaches $sn^k$, the search backs up. M′ uses its k additional heads to count up to $n^k - 1$. By using a finite amount of additional memory, M′ can count up to $sn^k$.

(c) The simulation is the same as in (b), except that the k heads used for the counter are not needed. We can assume that, if $w \in L(M)$, then there is an accepting computation tree for M on w such that M never makes more than $s - 1$ consecutive

moves along any path of the tree without moving a head. (This is shown as in the proof of Lemma 3.4.) This means that M' does not need the counter as in part (b): if M' simply backs up whenever M has made s consecutive moves without moving any head, then the simulation must terminate, since M can make at most k(n+1) moves that shift a head. The count of moves made without a head shift is placed on the pushdown as part of the symbol encoding the old state and move from that state. □

Parts (b) and (c) of Theorem 3.5 strengthen Sudborough's results that 2NFA(k) $\subseteq$ 2DPDA(2k) and 1NFA(k) $\subseteq$ 2DPDA(k) [22,24]. Unfortunately, part (a) is not a corresponding strengthening of Sudborough's result that 2NFA(2k) $\subseteq$ 2NPDA(k).

Theorem 3.5 yields fairly efficient simulations of one- and two-way nondeterministic pushdown automata by two-way deterministic pushdown automata. The next corollary follows from Theorem 3.2; the two after it follow from Theorem 3.3.

**Corollary 3.6.** For $k \geq 1$, 2NPDA(k) $\subseteq$ 2DPDA(6k).

This result is better than the one obtained by Kameda [12], who showed that 2NPDA(k) $\subseteq$ 2DPDA(12k+1), but not as good as the one achieved by Sudborough [23], who showed that 2NPDA(k) $\subseteq$ 2DPDA(4k). (Monien [18] claims that, by using the techniques of [15], the result 2NPDA(k) $\subseteq$ 2DPDA(3k+1) is possible.) However, it is interesting to note that Sudborough's argument depends on an intricate simulation using the ideas of Seiferas [19,20], whereas our result is a direct simulation, using an alternating finite automaton as an intermediate step. This suggests that alternation may be useful in the simulation of one type of nonalternating device by another.

The fact that Corollary 3.6 is close to Sudborough's result suggests that the results of Theorems 3.2 and 3.5 cannot be improved much.

**Corollary 3.7** [23]. Every context-free language is in 2DPDA(3).

Note that if all context-free languages were in 1AFA(2), then by Theorem 3.5, all would be in 2DPDA(2). Using Cook's simulation [4], this would imply a context-free language recognition algorithm that runs in time $O(n^2)$, which would be surprising. Thus, it seems unlikely that Theorem 3.3 can be improved, at least in the case that k = 1.

The third and final corollary gives an efficient simulation of a one-way nondeterministic pushdown automaton by a two-way deterministic pushdown automaton. This appears to be an original result, and is therefore evidence that alternating devices are useful for proving results about nonalternating devices.

**Corollary 3.8.** For $k \geq 1$, 1NPDA(k) $\subseteq$ 2DPDA(3k).

## 4. One-way alternating finite automata

The first theorem states that a one-way alternating finite automaton with a single head is no more powerful than a one-way nondeterministic or deterministic finite automaton with a single head.

**Theorem 4.1** [1]. 1AFA(1) coincides with the class of regular sets.

With two heads, however, a one-way alternating finite automaton is capable of recognizing not only nonregular sets, but also nonregular sets over a one-letter alphabet. This result yields two important corollaries concerning the relationships between the 1AFA(k) classes and the 1NFA(k) classes.

**Theorem 4.2.** 1AFA(2) contains a nonregular language over a single-letter alphabet.

**Proof.** Let $L = \{0^{2^n} \mid n \geq 1\}$. We construct a 1afa(2) M that recognizes L. Initially both heads are at the left end of the input. M operates as follows:

(1) Guess where the middle of the remaining input is and place head 1 there. (If the two heads were on the last square of the input, then enter an accepting state).

(2) Universally choose to perform either (i) or (ii).

(i) Check whether the guess in (1) was correct by moving head 2 to the right twice as fast as head 1. Enter an accepting state if and only if both heads reach the endmarker at the same time.

(ii) Move head 2 right until it coincides with head 1, then go to step (1). □

**Corollary 4.3.** $1AFA(2) \not\subseteq \bigcup_{k\in N} 1NPDA(k)$.

**Proof.** When restricted to a single-letter alphabet, the family $\bigcup_{k\in N} 1NPDA(k)$ contains only regular sets [7]. □

Note that Corollary 4.3 allows Theorem 3.3 and Corollary 3.8 to be strengthened to $1NPDA(k) \subsetneq 1AFA(3k)$ and $1NPDA(k) \subsetneq 2DPDA(3k)$, respectively.

**Corollary 4.4.** $1AFA(2) \not\subseteq \bigcup_{k\in N} 1NFA(k)$.

**Corollary 4.5.** For $k \geq 2$, $1NFA(k) \subsetneq 1AFA(k)$.

An important question concerning the one-way alternating finite automaton classes is whether $1AFA(k) \subsetneq 1AFA(k+1)$ for $k \geq 1$. Yao and Rivest [25] proved the result for one-way deterministic and nondeterministic finite automata, and in fact showed that $1DFA(k+1) - 1NFA(k) \neq \emptyset$ for all $k \geq 1$. From Theorems 4.1 and 4.2, it is clear that $1AFA(1) \subsetneq 1AFA(2)$. However, we have so far been unable to show that $1AFA(k) \subsetneq 1AFA(k+1)$ for $k \geq 2$.

We now show that there is a language in 1AFA(2) that is log-space complete for the family P (see [10] for a definition of log-space completeness and the $\leq_{log}$ relation). This will enable us to show that, if every 1afa(2) could be simulated by a 2dfa(k) (or 2nfa(k)), then a long-standing open problem in complexity theory would be solved.

A <u>path system</u> is a structure $S = (X,R,S,G)$, where X is a finite set, $R \subseteq X \times X \times X$, $S \subseteq X$, and $G \subseteq X$. Cook [5] (see [11] as well) has shown that the following problem PATH is log-space complete for P: Given a path system $S = (X,R,S,G)$, determine whether any element of S is admissible. (We say that x is <u>admissible</u> if $x \in G$ or there exist y and z such that $(x,y,z) \in R$ and both y and z are admissible.)

Cook encodes PATH as a language SP over the alphabet {0,1,*}. A string in SP consists of binary codes for the members of X, followed by the triples of R and the

members of S and G. We shall require a slightly different encoding.

**Definition.** Let $\Sigma = \{0,1\}$. Let $L_C$ be the set of all strings in the regular set $(\Sigma^+\#)^+([\Sigma^+\#\Sigma^+\#\Sigma^+])^*(\Sigma^+\#)^+$ that satisfy property C. A string of the form

$$v_1\# \cdots v_p\#[x_1\#y_1\#z_1] \cdots [x_q\#y_q\#z_q]w_1\# \cdots w_r\# \tag{4.1}$$

satisfies property C if there exists a binary tree T whose nodes are labelled with positive integers such that

(1) if i is the label of the root of T, then there exists j such that $v_j = x_i$ (unless T consists of a single node, in which case $v_j = w_i$);

(2) each internal node of T, labelled with, say, i, has exactly two immediate descendants, labelled with, say, j and k (from left to right), where $y_i = x_j$ ($z_i = x_k$) if the left (right) descendant is internal and $y_i = w_j$ ($z_i = w_k$) if it is a leaf;

(3) along each root-leaf path of T (labelled, say, $i_1,\ldots,i_s$), we have $i_k > i_{k-1}$, for each k, $2 \leq k \leq s - 1$.

Thus, a string satisfies property C if it encodes an instance of PATH for which there is a tree T certifying the admissibility of some element in S such that, along any path of T, the triples occur in the same order as in the string.

**Lemma 4.6.** $SP \leq_{\log} L_C$.

**Proof.** We construct a log n space-bounded Turing transducer M as follows. On input w, where w is the encoding of a path system S = (X,R,S,G), M first outputs the elements of S, encoded in binary and separated by #'s. M next outputs the triples of R, each separated internally by #'s and enclosed within a [] pair. This step is repeated a total of n times, where n is the cardinality of X. Finally, M outputs the elements of G, encoded in binary and separated by #'s. Clearly the original encoding of PATH permits M to operate in log |w| space.

Note that, if $x \in S$ is an admissible node, then there is a binary tree certifying admissibility of height at most n. Using this observation, it is easy to show that $w \in SP$ if and only if $f(w) \in L_C$, where f is the function computed by M. □

**Lemma 4.7.** $L_C \in$ 1AFA(2).

**Proof.** The construction of a 1afa(2) M to recognize $L_C$ is straightforward. Whenever M has existentially selected a string $x_i$, it enters a universal state to choose whether to place one head on $y_i$ or $z_i$ (say the former). The other head then moves right (past the first) until M guesses that an $x_j$ matching $y_i$ has been found. □

We now reach the main result of this section, which states that every 1afa(2) can be simulated by a 2dfa(k) [2nfa(k)], for some k, if and only if P = DSPACE(log n) [P = NSPACE(log n)]. The problem of whether deterministic polynomial time is equal to deterministic (or nondeterministic) log space is a classical open problem in complexity theory, to which a negative answer seems likely.

**Theorem 4.8.**

(a) $1AFA(2) \subseteq \bigcup_{k\in N} 2DFA(k)$ if and only if P = DSPACE(log n).

(b) $1AFA(2) \subseteq \bigcup_{k\in N} 2NFA(k)$ if and only if P = NSPACE(log n).

**Proof.** Since $\bigcup_{k\in N}$ 1AFA(k) $\subseteq$ P, $\bigcup_{k\in N}$ 2DFA(k) = DSPACE(log n), and $\bigcup_{k\in N}$ 2NFA(k) = NSPACE(log n), the reverse direction of each equivalence is clear.

For the forward direction of (a), suppose that L $\in$ P. Then L $\leq_{log}$ SP. Applying Lemma 4.6 and the transitivity of $\leq_{log}$ [10], we have that L $\leq_{log}$ $L_C$. Assuming that 1AFA(2) $\subseteq$ $\bigcup_{k\in N}$ 2DFA(k), by Lemma 4.7 we have $L_C$ $\in$ $\bigcup_{k\in N}$ 2DFA(k) = DSPACE(log n). Since L is log-space reducible to a language in DSPACE(log n), L itself must be in DSPACE(log n) [10]. The forward direction of (b) is proved in a similar fashion. □

Theorem 4.8 is analogous to Sudborough's result [21] that 1NFA(2) $\subseteq$ $\bigcup_{k\in N}$ 2DFA(k) if and only if NSPACE(log n) = DSPACE(log n). Galil [6] has also proved a similar theorem, using 2DPDA(1) instead of 1AFA(2).

Part (a) of Theorem 4.8 may be stated as: 1AFA(2) $\subseteq$ DSPACE(log n) if and only if P = DSPACE(log n). Therefore, $\bigcup_{k\in N}$ 1AFA(k) $\subseteq$ DSPACE(log n) if and only if P = DSPACE(log n). The best upper bound that we know of on the deterministic space complexity of the family $\bigcup_{k\in N}$ 1AFA(k) is given in the following theorem.

**Theorem 4.9.** $\bigcup_{k\in N}$ 1AFA(k) $\subseteq$ DSPACE(n).

**Proof.** The proof of Theorem 3.5(c) gives a simulation of a 1afa(k) M by a 2dpda(k) M'. An examination of the simulation reveals that the height of the latter machine's pushdown, on an input of length n, is bounded by sk(n+1), where s is the number of states in M. From M' we can construct a Turing machine M'' that simulates M', using its storage tape to hold the positions of the latter's k heads and the contents of its pushdown. □

We know that $\bigcup_{k\in N}$ 1DFA(k) $\subsetneq$ $\bigcup_{k\in N}$ 2DFA(k) and $\bigcup_{k\in N}$ 1NFA(k) $\subsetneq$ $\bigcup_{k\in N}$ 2NFA(k) (since $\bigcup_{k\in N}$ 1DFA(k) and $\bigcup_{k\in N}$ 1NFA(k), when restricted to a one-letter alphabet, contain only the regular sets [7]). The corresponding question for alternating finite automata remains open. However, Theorems 3.1 and 4.9 yield the following.

**Corollary 4.10.** $\bigcup_{k\in N}$ 1AFA(k) = $\bigcup_{k\in N}$ 2AFA(k) implies P $\subseteq$ DSPACE(n).

## 5. Two-way alternating finite automata

As in the one-way case, single-head two-way alternating finite automata are no more powerful than their nondeterministic and deterministic counterparts.

**Theorem 5.1** [14]. 2AFA(1) coincides with the class of regular sets.

We next consider the question of whether 2AFA(k) $\subsetneq$ 2AFA(k+1). In the case of two-way deterministic and nondeterministic finite automata, the answer is yes. Ibarra [9] showed that 2DFA(k) $\subsetneq$ 2DFA(k+2). Monien [16] improved this result to 2DFA(k) $\subsetneq$ 2DFA(k+1). The nondeterministic case proved to be more difficult, and remained open until Seiferas [20] proved that 2NFA(k) $\subsetneq$ 2NFA(k+2). Monien then noted in his 1977 corrigendum that the techniques of [16], when combined with Seiferas's result, yield 2NFA(k) $\subsetneq$ 2NFA(k+1). In a recent paper, Monien [17] shows that 2DFA(k) $\subsetneq$ 2DFA(k+1) and 2NFA(k) $\subsetneq$ 2NFA(k+1) hold even when the languages are over a single-letter alphabet.

We now show that 2AFA(k) $\subsetneq$ 2AFA(k+1), using the techniques of [9] and [16]. We first give a crude result showing that 3k+3 heads are better than k heads.

**Lemma 5.2.** For $k \geq 1$, 2AFA(k) $\subsetneq$ 2AFA(3k+3).

**Proof.** Let $k \geq 1$. By Theorem 3.5(a), we know that 2AFA(k) $\subseteq$ 2NPDA(k). Ibarra [9] has shown that 2NPDA(k) $\subsetneq$ 2NPDA(k+1). Also, by Theorem 3.2, 2NPDA(k+1) $\subseteq$ 2AFA(3k+3). Combining these inclusions, we have 2AFA(k) $\subsetneq$ 2AFA(3k+3). □

**Definition.** Let $\Sigma$ be an alphabet, let $\dashv$ and $\vdash$ be symbols not in $\Sigma$, and let $k \geq 1$. We define a function $f_{\Sigma,k} : \Sigma^* \rightarrow ((\Sigma \cup \{\dashv,\vdash\})^k)^*$ as follows. Let $a_1,\ldots,a_m$ be symbols in $\Sigma$ and define $a_0 = \dashv$, $a_{n-1} = \vdash$, where $n = m + 2$. Then $f_{\Sigma,k}(a_1 \ldots a_m) = \alpha_0\alpha_1\ldots\alpha_{n^k-1}$, where $\alpha_j = (a_{i_1},\ldots,a_{i_k})$ for $j = i_1+i_2n+\ldots+i_kn^{k-1}$ with $0 \leq i_p \leq n-1$ for all p, $1 \leq p \leq k$.

The next three lemmas (analogous to those in [16]) use the $f_{\Sigma,k}$ function to develop translational results. In Theorem 5.6, these are combined with Lemma 5.2 to show that, for two-way alternating finite automata, k + 1 heads are better than k.

**Lemma 5.3.** For $k \geq 1$, $j \geq 2$, $L \subseteq \Sigma^*$, $L \in$ 2AFA(kj+1) implies $f_{\Sigma,k}(L) \in$ 2AFA(j+1).

**Lemma 5.4.** For $k \geq 1$, $j \geq 2$, $L \subseteq \Sigma^*$, $f_{\Sigma,j+1}(L) \in$ 2AFA(j) implies $f_{\Sigma,j}(L) \in$ 2AFA(j+1).

**Lemma 5.5.** For $k \geq 2$, $L \subseteq \Sigma^*$, $f_{\Sigma,k}(L) \in$ 2AFA(j) implies $L \in$ 2AFA(kj).

**Theorem 5.6.** For $k \geq 1$, 2AFA(k) $\subsetneq$ 2AFA(k+1).

**Proof.** First, assume that 2AFA(k+2) $\subseteq$ 2AFA(k). Let $L \in$ 2AFA(j(k+1)+1). By Lemma 5.3, $f_{\Sigma,j}(L) \in$ 2AFA(k+2), so by assumption, $f_{\Sigma,j}(L) \in$ 2AFA(k). By Lemma 5.5, $L \in$ 2AFA(jk), so

$$2AFA(j(k+1)+1) \subseteq 2AFA(jk) \text{ for all } j \geq 2. \tag{5.1}$$

Note that

$$2AFA(jk) \subseteq 2AFA((j-1)(k+1)+1) \text{ if } j \geq k. \tag{5.2}$$

By alternately applying (5.1) and (5.2), we get

$2AFA(3k(k+1)+1) \subseteq 2AFA(3k^2) \subseteq 2AFA((3k-1)(k+1)+1) \subseteq 2AFA(3k^2-k) \subseteq \ldots \subseteq 2AFA(k^2)$.

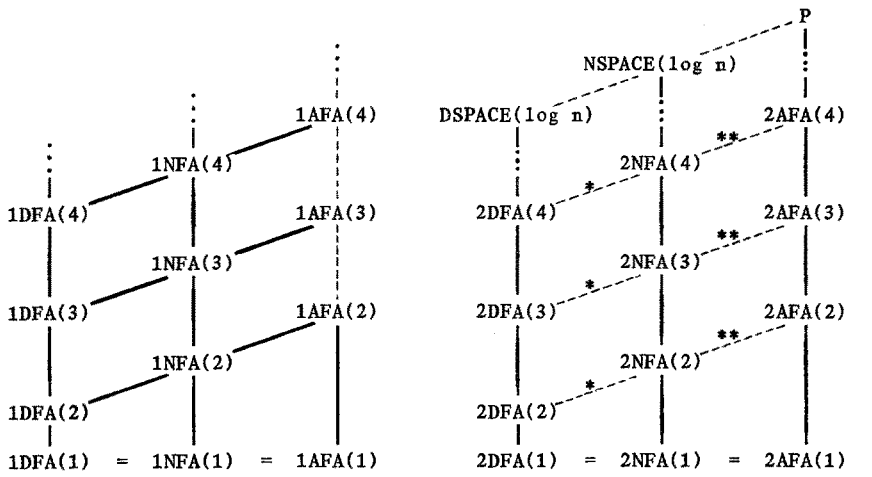
(Each application of (5.2) followed by an application of (5.1) causes the number of heads to decrease by k. Thus, since we started with $3k^2$ (after the first application of (5.1)), we eventually reach $k^2$.) For $k \geq 2$, this yields $2AFA(3k^2+3) \subseteq 2AFA(k^2)$, contradicting Lemma 5.2. Thus, 2AFA(k) $\subsetneq$ 2AFA(k+2) for all $k \geq 2$.

Since 2AFA(1) is the family of regular sets, clearly the theorem is true for k = 1. Choose some $k \geq 2$ and assume for the sake of contradiction that 2AFA(k+1) $\subseteq$ 2AFA(k). Let $L \in$ 2AFA((k+1)k+1). By Lemma 5.3, $f_{\Sigma,k+1}(L) \in$ 2AFA(k+1) = 2AFA(k). By Lemma 5.4, $f_{\Sigma,k}(L) \in$ 2AFA(k+1) = 2AFA(k) and, by Lemma 5.5, $L \in$ 2AFA($k^2$). Since $k \geq 2$, we have $2AFA(k^2+2) \subseteq 2AFA(k^2)$, contradicting our previous result. □

## 6. Conclusions

Figure 2(a) summarizes the relationships between the one-way multihead finite automata classes; Fig. 2(b) does the same for the two-way classes. If two classes

are connected by a line, then the lower class is contained in the upper one. A solid line indicates proper containment; a dashed line indicates that proper containment is not known.




Additional relevant facts:

$1NFA(2) \notin \bigcup_{k\in N} 1DFA(k)$ [25]

$1AFA(2) \notin \bigcup_{k\in N} 1NFA(k)$ [Corollary 4.4]

$1DFA(k+1) \notin 1NFA(k)$ [25]

(a)

*If any inclusion is not strict, then DSPACE(log n) = NSPACE(log n) [21].

**If any inclusion is not strict, then NSPACE(log n) = P [Theorem 4.8(b)].

(b)

**Figure 2.**

Table 1 gives the best simulation results currently known for multihead finite automata and pushdown automata. The entry 2k in the row labelled by 2nfa and column labelled by 2dpda indicates that the best simulation of a 2nfa(k) by a multihead 2dpda currently known requires 2k heads. The superscript following an entry indicates where the result appears. A superscript * indicates that the result is in this paper. If no superscript appears, then the result is obvious from the definitions of the devices. In some cases (marked with an X), it is known that no simulation is possible. In other cases, if any simulation is possible (regardless of how many heads the simulating machine has), then certain open problems are solved (indicated using the code D = DSPACE(log n), N = NSPACE(log n)).

Our study of alternating finite automata reveals that these devices are quite similar to pushdown automata. The resemblance is particularly striking in the case of two-way alternating finite automata. We have proved that $2AFA(k) \subsetneq 2AFA(k+1)$, just as $2NPDA(k) \subsetneq 2NPDA(k+1)$ and $2DPDA(k) \subsetneq 2DPDA(k+1)$ [9]. Furthermore, the limit of the 2AFA(k) classes is P, as is the limit of the 2NPDA(k) and 2DPDA(k) classes. Most importantly, we showed that $2NPDA(k) \subseteq 2AFA(3k) \subseteq 2NPDA(3k)$.

One-way alternating finite automata, on the other hand, are more powerful than one-way pushdown automata. While one-way alternating finite automata can simulate

| | 1dfa | 1nfa | 1afa | 2dfa | 2nfa | 2afa | 1dpda | 1npda | 2dpda | 2npda |
|---|---|---|---|---|---|---|---|---|---|---|
| 1dfa | k | k | k | k | k | k | k | k | k | $\lceil k/2 \rceil$ [24] |
| 1nfa | N=D [21] | k | k | N=D [21] | k | k | ? | k | $k^{[24]}$ | $\lceil k/2 \rceil$ [24] |
| 1afa | X [7] | X [7] | k | P=D* | P=N* | k | X [7] | X [7] | k* | k* |
| 2dfa | X [7] | X [7] | ? | k | k | k | X [7] | X [7] | k | $\lceil k/2 \rceil$ [24] |
| 2nfa | X [7] | X [7] | ? | N=D [21] | k | k | X [7] | X [7] | $2k^{[24]}$ | $\lceil k/2 \rceil$ [24] |
| 2afa | X [7] | X [7] | ? | P=D* | P=N* | k | X [7] | X [7] | 2k* | k* |
| 1dpda | ? | ? | 3k* | ? | ? | 3k* | k | k | k | k |
| 1npda | N=D [21] | ? | 3k* | N=D [21] | ? | 3k* | ? | k | 3k* | k |
| 2dpda | X [7] | X [7] | ? | P=D [6] | P=N [6] | 3k* | X [7] | X [7] | k | k |
| 2npda | X [7] | X [7] | ? | P=D [6] | P=N [6] | 3k* | X [7] | X [7] | $4k^{[23]}$ | k |

[Warning: The entries in the first six rows are invalid if the device to be simulated has only one head, since then the device accepts a regular set and thus can be simulated by any device in the table using only a single head.]

**Table 1.**

one-way pushdown automata (we showed that 1NPDA(k) $\subsetneq$ 1AFA(3k)), even the smallest nonregular class, 1AFA(2), contains a nonregular language over a single-letter alphabet and a language that is log-space complete for P. (One may argue that alternating finite automata are the simplest 'natural' alternating devices, so it is interesting that alternation adds so much power, even at this low level of complexity.) The last two results imply that 1AFA(2) contains a language that is not recognizable by any one-way nondeterministic multihead pushdown automaton and that, if every 1afa(2) can be simulated by a two-way deterministic [nondeterministic] finite automaton, then P = DSPACE(log n) [P = NSPACE(log n)]. Since 1AFA(k) $\subseteq$ 2DPDA(k), we see that one-way alternating finite automata are intermediate in power between one-way nondeterministic and two-way deterministic pushdown automata.

Our study of alternating finite automata even yielded two results about nonalternating pushdown automata (2NPDA(k) $\subseteq$ 2DPDA(6k) and 1NPDA(k) $\subseteq$ 2DPDA(3k)), thus giving evidence that alternation is useful for proving properties of even fairly simple nonalternating devices.

The most interesting open problem concerning alternating finite automata is the question of whether 1AFA(k) $\subsetneq$ 1AFA(k+1) for all $k \geq 1$. This would seem to be the case, but finding a proof appears to be very difficult. Two related problems ask whether allowing two-way motion on the input tape adds power to alternating finite automata: First, is $\bigcup_{k \in N}$ 1AFA(k) $\subsetneq$ $\bigcup_{k \in N}$ 2AFA(k)? Second, for $k \geq 2$, is 1AFA(k) $\subsetneq$ 2AFA(k)? Corollary 4.10 suggests that the first inclusion is indeed proper. There is also the problem of improving the results in the simulation table. When a simulation is known, it may be possible to reduce the number of heads required for the simulating device. When an implication for an open problem is given, it may be possible to strengthen the implication or prove that no simulation is possible. And, if the entry is a ?, then any result, negative or positive, would be an improvement.

## References


1. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, Alternation, JACM 28 (1981), 114-133.
2. A. K. Chandra and L. J. Stockmeyer, Alternation, Conf. Rec. IEEE 17th Ann. Symp. on Found. of Comp. Sci. (1976), 98-108.
3. S. A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, JACM 18 (1971), 4-18.
4. S. A. Cook, Linear time simulation of deterministic two-way pushdown automata, in Information Processing 71, North-Holland, Amsterdam (1972).
5. S. A. Cook, An observation on time-space trade-off, J. Comput. Sys. Sci. 9 (1974), 308-316.
6. Z. Galil, Some open problems in the theory of computation as questions about two-way deterministic pushdown automaton languages, Math. Systems Theory 10 (1977), 211-228.
7. M. A. Harrison and O. H. Ibarra, Multi-tape and multi-head pushdown automata, Inform. Control 13 (1968), 433-470.
8. J. Hartmanis, On non-determinancy in simple computing devices, Acta Inform. 1 (1972), 336-344.
9. O. H. Ibarra, On two-way multihead automata, J. Comput. Sys. Sci. 7 (1973), 28-36.
10. N. D. Jones, Space-bounded reducibility among combinatorial problems, J. Comput. Sys. Sci. 11 (1975), 68-85.
11. N. D. Jones and W. T. Laaser, Complete problems for deterministic polynomial time, Theoretical Computer Science 3 (1977), 105-117.
12. T. Kameda, Pushdown automata with counters, J. Comput. Sys. Sci. 6 (1972), 138-150.
13. D. Kozen, On parallelism in Turing machines, Conf. Rec. IEEE 17th Ann. Symp. on Found. of Comp. Sci. (1976), 89-97.
14. R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer, Alternating pushdown automata, Conf. Rec. IEEE 19th Ann. Symp. on Found. of Comp. Sci. (1978), 92-106.
15. B. Monien, Characterizations of time-bounded computations by limited primitive recursion, Second Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 14, Springer-Verlag (1974), 280-293.
16. B. Monien, Transformational methods and their application to complexity problems, Acta Inform. 6 (1976), 95-108. Corrigendum, Acta Inform. 8 (1977), 383-384.
17. B. Monien, Two-way multihead automata over a one-letter alphabet, R.A.I.R.O. Informatique théorique 14 (1980), 67-82.
18. B. Monien and I. H. Sudborough, The interface between language theory and complexity theory, in 'Formal Language Theory: Perspectives and Open Problems,' R. V. Book, ed., Academic Press, New York, 1980, pp. 287-323.
19. J. I. Seiferas, Techniques for separating space complexity classes, J. Comput. Sys. Sci. 14 (1977), 73-99.
20. J. I. Seiferas, Relating refined space complexity classes, J. Comput. Sys. Sci. 14 (1977), 100-129.
21. I. H. Sudborough, On tape-bounded complexity classes and multihead finite automata, J. Comput. Sys. Sci. 10 (1975), 62-76.
22. I. H. Sudborough, On deterministic context-free languages, multihead automata, and the power of an auxiliary pushdown store, Proc. 8th Ann. Symp. on Theory of Computing (1976), 141-148.
23. I. H. Sudborough, Separating tape bounded auxiliary pushdown automata classes, Proc. 9th Ann. Symp. on Theory of Computing (1977), 208-217.
24. I. H. Sudborough, Some remarks on multihead automata, R.A.I.R.O. Informatique théorique 11 (1977), 181-195.
25. A. C. Yao and R. L. Rivest, k + 1 heads are better than k, JACM 25 (1978), 337-340.