RAIRO
INFORMATIQUE THÉORIQUE

I. H. SUDBOROUGH

Some remarks on multihead automata

RAIRO – Informatique théorique, tome 11, n° 3 (1977), p. 181-195.

<http://www.numdam.org/item?id=ITA_1977__11_3_181_0>

R.A.I.R.O. Informatique théorique/Theoretical Computer Science
(vol. 11, n° 3, 1977 p. 181 à 195).

SOME REMARKS ON MULTIHEAD AUTOMATA(*)

by I. H. SUDBOROUGH (1)

Communicated by R. V. BOOK

Abstract. — *Various relationships are described between families of languages defined by multihead finite state automata and pushdown automata. The main results are* (1) *a translational method for bounding the complexity of multihead pda languages in terms of the complexity of single head pda languages,* (2) *a simulation algorithm which shows that nondeterministic pda with k heads are as powerful as nondeterministic finite state automata with 2k heads, and* (3) *an improved hierarchical result which shows that k + 4 heads are better than k heads (for finite state automata) even on languages over a single letter alphabet.*

INTRODUCTION

It is known that the family of languages recognized by multihead twoway pushdown automata is identical to the family P-TIME of languages recognized in polynomial time by deterministic multitape Turing machines [7]. It is also known that the family of languages recognized by deterministic (nondeterministic) multihead two-way finite state automata is identical to the family DLOG (NLOG) of languages recognized by deterministic (nondeterministic) log (n)-tape bounded Turing machines [17]. It is not known whether or not P-TIME = DLOG, P-TIME = NLOG, or DLOG = NLOG. These problems have been investigated recently in the literature [4-9, 13, 17, 21, 22, 29, 30]. They can be loosely interpreted in the following forms. Does adding an unbounded auxiliary pushdown store increase the computing power of a tape bounded Turing machine? Does allowing nondeterministic transitions increase the computing power of tape bounded Turing machines?

It is known that every context-free language (i.e. nondeterministic one-way pushdown automaton language) can be recognized by (1) a deterministic multitape Turing machine in $n^{2.81}$ steps [31], (2) a unit cost random access machine (RAM) in $n^2 \log n$ steps [32] and a deterministic multitape Turing

(*) This work is supported in part by NSF Grant No. GJ-43228.

(*) Received, Aug. 1975; revised, Oct. 1976.

(1) Department of Computer Sciences, Technological Institute, Northwestern University, Evanston, Illinois.

machine within tape $(\log n)^2$ [25]. It is not known is these results are optimal. A context-free language is known whose time or space complexity is the least upper bound on the time or space complexity for the whole family of context-free languages [14]. For some special cases, better results are known ; the family of linear context-free languages and the family of one counter languages are both recognizable by deterministic Turing machines in time n^2 [15, 24]. Two-way nondeterministic one-head pushdown automata languages can be recognized in space n^2 and time n^4 by deterministic Turing machines and time $0(n^3)$ by RAM's [1]. Two-way deterministic one-head pda languages can be recognized in space n and time $n^2 \log n$ by deterministic Turing machines [1] and in linear time by unit cost RAM's [10].

The results described here extend some of these complexity bounds for the case of multihead automata. The main results are that (1) every nondeterministic one-way k-head pda language can be recognized by a deterministic Turing machine in time $n^{2.81k}$, and (2) every nondeterministic two-way $2k$-head finite state automaton language can be recognized by a nondeterministic two-way k-head pda and, hence, by a deterministic multitape Turing machine in time n^{4k} and by a RAM in time $0(n^{3k})$, (3) every nondeterministic two-way (one-way) k-head finite state automaton language can be recognized by a deterministic $2k$-head (k-head) pda and, hence, by a RAM in time $0(n^{2k})$ $(0(n^k))$, and (4) that there are languages over a single letter alphabet which are recognized by a nondeterministic (deterministic) two-way $(k + 4)$-head finite state automaton but cannot be recognized by any such automaton which k heads.

The reader is referred to [3, 11, 16-19] for formal definitions of multihead automata, Turing machines, and random access machines. We assume that in any transition of a multihead automaton only one of the heads is used to scan the input tape and only that head may move to the right or to the left during that transition. This means the set of states can be partitioned into collections of states which "control" a particular head. (The reader is referred to [16] for further details.) For any X in { D, N }, l in { 1, 2 }, and positive integer k, let lXFA(k) (lXPDA(k)) denote the family of languages recognized by (one-way, two-way) (deterministic, nondeterministic) finite state automata (pushdown automata) with k read-only heads on the input tape. For the special case when $k = 1$ we shall often omit the value 1. That is, the family 2NPDA (1) will be denoted by 2NPDA. We shall say that a deterministic Turing machine (deterministic random access machine) M *accepts in time* $T(n)$ (accepts in time $0(T(n))$ if each input w accepted by M is accepted within $T(|w|)$ steps (is accepted within $cT(|w|)$ steps, for some constant $c > 0$). A deterministic Turing machine M with a two-way read-only input tape and a separate read-write worktape *accepts within tape* $L(n)$ if each input w accepted by M is accepted by a computation in which M scans at most $L(|w|)$ distinct cells on the worktape. For X in { D, N } and any positive integer k, let XSPACE($L(n)$, k) denote the family of languages recognized by (deterministic, nondeterministic) $L(n)$-tape boun-

ded Turing machines with a k symbol tape alphabet. Thus DLOG $= \bigcup_k$ DSPACE(log n, k) and NLOG $= \bigcup_k$ NSPACE(log n, k). We shall assume throughout these remarks that the tape bounded Turing machines have a one-way infinite worktape and that they have the power to detect when the two-way worktape head is scanning the leftmost square. (The effects of various changes in the basic tape bounded Turing machine model in a restricted worktape symbol alphabet environment are discussed in [27]). It is well known that constant factors are not important for tape or time bounds on multitape Turing machines [18]. Thus we shall not be concerned in general with the base of the logarithm in discussing log (n)-tape bounded Turing machines. For symbol restricted models constant factors are relevant and, in this case, log (n) will be used as notational shorthand for the bound $[\log_2(n)]$, where $[x]$ denotes the greatest integer less than x. (We shall also use the notation $\lceil x \rceil$ to denote the least integer greater than x, where x is a real number, and $|w|$ to denote the length of a string w of symbols over some alphabet).

TRANSLATION FROM MULTIHEAD TO SINGLE HEAD PDA

Our first result is obtained by a translational technique basically similar to those described in [4-6, 12, 13, 19, 29]. It would seem to differ from these earlier translations mainly in that it allows relationships to be described between one-way multihead automata classes.

Let $\Delta = \Delta^{(1)}$ be an alphabet. For $i \geq 2$, let $\Delta^{(i)}$ be the set $\{ (c_1, c_2, \ldots, c_i) \mid c_1, c_2, \ldots, c_i \text{ are in } \Delta \}$. Let $b_2, b_3, b_4, \ldots$ be a countably infinite set of symbols which are not elements of $\bigcup_{i \geq 1} \Delta^{(i)}$. For any a in Δ let h_a be the length preserving homomorphism defined by :

$$h_a((c_1, c_2, \ldots, c_i)) = (c_1, c_2, \ldots, c_i, \text{a}) \quad \text{and} \quad h_a(b_j) = b_j,$$

for all positive integers i and j and symbols $c_1, c_2, \ldots, c_i$ in Δ. For any string $x = a_1 a_2 \ldots a_n$ of n symbols over Δ, any integer $i \geq 2$, and any string y over the alphabet $\Delta^{(i-1)} \bigcup \{ b_2, \ldots, b_{i-1} \}$, define the string $g(x, i, y)$ by :

$$g(x, i, y) = h_{a_1}(y) b_i h_{a_1}(y) b_i h_{a_2}(y) b_i h_{a_2}(y) b_i \ldots h_{a_n}(y) b_i h_{a_n}(y).$$

Define the strings $f(i, x)$, for $i \geq 1$, recursively by :

$$f(1, x) = x$$
$$f(i, x) = g(x, i, f(i - 1, x)), \text{ for } i \geq 2.$$

For $k \geq 1$ and integers $i_1, i_2, \ldots, i_k$, such that $1 \leq i_j \leq |x|$ for $1 \leq j \leq k$, the position $P_x^{(k)}(i_1, i_2, \ldots, i_k)$ of $f(k, x)$ is defined as follows :

$$P_x^{(1)}(i) = i$$
$$P_x^{(k)}(i_1\, i_2, \ldots, i_k) = P_x^{(k-1)}(i_1, i_2, \ldots, i_{k-1}) + (i_k - 1)(|f(k - 1, x)| + 1), \text{ for } k \geq 2.$$

It follows by induction $P_x^{(k)}(i_2, i_2, \ldots, i_k)$ is equal to

$$i_1 + \sum_{j=2}^{k} (i_j - 1)(|f(j-1, x)| + 1).$$

It is claimed that the symbol at position $P_x^{(k)}(i_1, 2i_2 - 1, \ldots, 2i_k - 1)$ of $f(k, x)$ is $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$, if $x = a_1 a_2 \ldots a_n$ where each a_j is a symbol in Δ. For $k = 1$ this is clearly the case, since $P_x^{(1)}(i) = i$ and the symbol at position i of $f(1, x) = x = a_1 a_2 \ldots a_n$ is a_i. For the inductive step, assume that the symbol at position $P_x^{(k)}(i_1, 2i_2 - 1, \ldots, 2i_k - 1)$ of $f(k, x)$ is $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$. By definition

$$\begin{aligned} f(k+1, x) &= g(x, k+1, f(k, x)) \\ &= h_{a_1}(f(k, x))b_{k+1}h_{a_1}(f(k, x))b_{k+1} \ \ldots \ h_{a_n}(f(k, x))b_{k+1}h_{a_n}(f(k, x)). \end{aligned}$$

Since

$$\begin{aligned} Q &= P_x^{(k+1)}(i_1, 2i_2, \ldots, 2i_{k+1} - 1) \\ &= P_x^{(k)}(i_1, 2i_2 - 1, \ldots, 2i_k - 1) + (2i_{k+1} - 2)(|f(k, x)| + 1) \end{aligned}$$

and $|f(k, x)| + 1$ is the length of $h_a(f(k, x))b_{k+1}$, for any symbol a, the symbol at position Q of $f(k+1, x)$ is the symbol at position

$$Q' = P_x^{(k)}(i_1, 2i_2 - 1, \ldots, 2i_k - 1)$$

of $h_{a_j}(f(k, x))$, where $j = i_{k+1}$. By the inductive hypothesis the symbol at position Q' of $f(k, x)$ is $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$. Therefore, the symbol at position Q' of $h_{a_j}(f(k, x))$, where $j = i_{k+1}$, is $(a_{i_1}, a_{i_2}, \ldots, a_{i_{k+1}})$. This establishes the claim.

Since $Q = P_x^{(k)}(i_1, i_2, \ldots, i_j, \ldots, i_k)$ is equal to

$$i_1 + \sum_{j=2}^{k} (i_j - 1)(|f(j-1, x)| + 1),$$

to move from position Q *to position* $Q' = P_x^{(k)}(i_1, i_2, \ldots, i_j + 1, \ldots, i_k)$ requires moving the head $m = (|f(j-1, x)| + 1)$ squares. It can easily be observed that each substring of $f(k, x)$ whichdoes not contain occurrences of any symbol b_l, for $l > j$, has eactly m symbols between successive occurrences of the symbol b_j. This property allows one to measure the proper distances in movements of the head in the algorithm expressed in the proof of the following theorem.

LEMMA 1 : *Let $k \geqq 2$ and let M be a nondeterministic (deterministic) one-way k-head pushdown automaton. A nondeterministic (deterministic) one-way one-head pushdown automaton M' can be effectively constructed such that x is recognized by M if and only if $f(k, x\,\#)$ is recognized by M'.*

Proof : M' is constructed to simulate on input $f(k, x\,\#)$ a computation of M on input x. (What M' does on input strings not of the form $f(k, x\,\#)$ is of no

concern.) M' will represent a configuration of M where heads 1, 2, ..., k *are scanning cells* $i_2, i_2, \ldots, i_k$ on input $x\#$, where $\#$ is the right endmarker, by placing its head in position $P^{(k)}_{x\#}(i_1, 2i_2 - 1, \ldots, 2i_k - 1)$. If $x\# = a_1a_2 \ldots a_{n+1}$, where each a_i is an input symbol, for $1 \leq i \leq n$, and $a_{n+1} = \#$ (the right endmarker), then M in this configuration would be scanning the symbols $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ with heads 1, 2, ..., k, respectively. As previously observed, M' with its head at position $P^{(k)}_{x\#}(i_1, 2i_2 - 1, \ldots, 2i_k - 1)$ in $f(k, x\#)$ will scan the symbol $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$. Thus M' has all the input information on this one square that M has on the k squares its k heads scan.

It is sufficient to indicate how M' can simulate a move of one of the heads of M. If M has a transition that moves head j to the right, for some $2 \leq j \leq k$, then M' must move its head from position $P^{(k)}_{x\#}(i_1, 2i_2 - 1, \ldots, 2i_j - 1, \ldots, 2i_k - 1)$ to position $P^{(k)}_{x\#}(i_1, 2i_2 - 1, \ldots, 2(i_j + 1) - 1, \ldots, 2i_k - 1)$ on the input $f(k, x\#)$. That is, as observed, M' must move its head to the right a distance equal to twice the length of $f(j - 1, x\#)$ in order to be in the proper position. This may be done by simply (1) moving the head to the right to the first b_j symbol and adding a special symbol to the top of the pushdown store for each input symbol passed, then (2) moving the head to the right one square for each special symbol deleted from the pushdown store until all such special symbols are deleted, and the (3) repeating steps (1) and (2) one additional time. That is, let there be n symbols between each successive occurrence of the symbol b_j in each substring of $f(k, x\#)$ which does not contain an occurrence of the symbol b_l, for $l > j$. Then steps (1) and (2) take the head from cell i of one of these " blocks " of length n to cell $n - i + 1$ of the next such block to the right. Thus two executions of steps (1) and (2) move the head from cell i of one such block to cell i of the second such block to the right. Since each such block in $f(k, x\#)$ is of length $n = |f(j - 1, x\#)|$, the head has been moved by this process to the correct position.

If M has a transition that moves head one to the right one square, then M' need only move its head to the right one square. Thus, M' can with one head on $f(k, x\#)$ simulate M with its k heads on input x; M' will accept $f(k, x\#)$ if and only if M enters an accepting state. Thus, M' accepts $f(k, x\#)$ if and only if M accepts x.

We observe that $|f(k, x\#)| \leq c\,|x|^k$, for some constant c which does not depend upon x. That is, for any string x of length n, $g(x, i, y)$ is of length $2n(|y| + 1)$, for all $i \geq 1$. Let DTIME($T(n)$) denote either the family of languages recognized (1) in time $T(n)$ by deterministic Turing machines, or (2) in time $0(T(n))$ by deterministic RAM's.

THEOREM 1 : *For any* $k \geq 2$ *and any nondecreasing function* $T(n) \geq n$, *if* $1NPDA(1) \subseteq DTIME(T(n))$, *then* $1NPDA(k) \subseteq$ DTIME($T(cn^k)$), *for some constant* c.

Proof : Let M be a nondeterministic one-way k-head pushdown automaton. By Lemma 1 there is a nondeterministic one-way one-head pushdown

automaton M' such that x is accepted by M if and only if $f(k, x\#)$ is accepted by M'. By hypothesis the latter question can be answered in $0(T(|f(k, x\#)|)$ steps. Since $|f(k, x\#)| \leq c|x|^k$, for some constant c, and $T(n)$ is nondecreasing, the number of such steps is not greater than $0(T(c|x|^k))$. Thus, one can decide whether x is accepted by M by (1) generating $f(k, x\#)$ from x, and (2) using the given algorithm to decide whether $f(k, x\#)$ is accepted by M'. Step (1) requires at most $0(|f(k, x\#)|)$ steps. Thus, at most $0(T(c|x|^k))$ steps are required for the whole process.

In [31] Valiant has shown that context-free languages can be recognized in $n^{2.81}$ steps by deterministic multitape Turing machines. In [32] it has been shown that context-free languages can be recognized by random access machines (RAM's) using the unit cost measure in $0(n^2 \log n)$ steps. Thus it follows from Theorem 1, since 1NPDA(1) is the family of context-free languages, that every 1NPDA(k) language can be recognized in $n^{2.81k}$ steps by a deterministic multitape Turing machine and in $0(n^{2k} \log n)$ steps by a unit cost RAM.

It is straightforward to modify Lemma 1 and Theorem 1 so that "one-way" is replaced everywhere by "two-way". In fact, in this case many other suitable transformations have previously been described [13, 19]. Thus from the fact established in [1] that each 2NPDA(1) language is recognized by a random access machine in $0(n^3)$ steps and by a deterministic multitape Turing machine in $0(n^4)$ steps we have an easy proof of the fact that each 2NPDA(k) language is recognized by a random access machine in $0(n^{3k})$ steps and by a deterministic multitape Turing machine in $0(n^{4k})$ steps. Furthermore, the known fact that each 2DPDA(k) language is recognized by a unit cost RAM in $0(n^k)$ steps follows from the proof of the case when $k = 1$ [10]. The proof for the case $k = 1$ is perhaps easier to describe. This has been done in [3].

Lemma 1 and Theorem 1 may easily be restated for other classes of multihead automata and other complexity measures. For example, it is easily modified for the multihead stack automata of [20]. It may also be observed that given x one may produce $f(k, x\#)$ on an output tape without using more than $\log(|x|)$ worktape cells with the multitape transducer of [21, 26]. Thus the reduction from x to $f(k, x\#)$ is a log-tape reduction [21, 26].

For the multihead finite automata of [4, 12, 17, 19, 29] and the multihead one counter automata of [15] the results need to be somewhat modified. That is, the pushdown store is no longer available to measure the distance needed to obtain a new position (as described in the proof of Lemma 1). Thus, for multihead finite automata the statement of Lemma 1 would need to be revised so that the automaton M' is (1) a one-head finite state automaton with an additional counter (which is linear bounded), or (2) a two-head finite state automaton. Similar revisions would need to be made for multihead one counter automata, since the "special symbol", described in the proof of Lemma 1, cannot be used in this case.

MULTIHEAD FINITE AUTOMATA VERSUS MULTIHEAD PDA

The next result indicates that adding an auxiliairy pushdown store does add some additional computing power to a k-head finite state automaton. We show that $2\text{NFA}(2k) \subseteq 2\text{NPDA}(k)$. It is known that $2\text{DFA}(k) \subsetneq 2\text{DFA}(k+2)$ and $2\text{DPDA}(k) \subsetneq 2\text{DPDA}(k+1)$ for $k \geq 1$ [19]. The same hierarchical results have been established for the nondeterministic families [19, 27]. Combining these facts with our result we may conclude that if for any k there is an m_k such that $2\text{NPDA}(k) \subseteq 2\text{NFA}(m_k)$, then $m_k \geq 2k$.

Our result is similar to an earlier result in the literature which states that $\text{NSPACE}(\log_{2^t} n, t) \subseteq 2\text{NPDA}$ [2]. However, there is no apparent method to obtain our result from this earlier one. Nor does there seem to be a method to obtain the earlier result from ours. The best relationships known between the number of symbols in the worktape alphabet and the number of heads is given by (1) $2\text{DFA}(k) \subseteq DSPACE(\log_2 n, 2^k)$ [17, 27] and (2) $\text{DSPACE}(\log_2 n, 2^k) \subseteq 2\text{DFA}(k+3)$. (The second relationship is established in this paper; both relationships are also valid for nondeterministic families.) Our result, for example, gives $2\text{NFA}(2) \subseteq 2\text{NPDA}$. To obtain this from the above relationships would seem to require showing $\text{NSPACE}(\log_2 n, 4) \subseteq 2\text{NPDA}$. This would be a nontrivial extension of the results in [2].

THEOREM 2 : *For* $k \geq 1$, $2NFA(2k) \subseteq 2\text{NPDA}(k)$.

Proof : Let M be a nondeterministic two-way $2k$-head finite state automaton. Let the set of states of M be partitioned into two sets S_1 and S_2 such that S_1 is the set of all states which control one of the heads numbered one through k and S_2 is the set of all states which control one of the heads numbered $k+1$ through $2k$. (S_2 is the set of all states not in S_1.) We construct a nondeterministic two-way k-head pushdown automaton P_M which indirectly simulates M. We shall consider the steps in a computation by M as being divided into two sets : those steps involving states in S_1 and, secondly, those steps involving states in S_2. Clearly, in general, a computation will alternate between steps involving states in S_1 and steps involving states in S_2. We shall assume, without any loss of generality, that the initial state and each final state is in S_1.

A sequence of states $P_0, P_1, \ldots, P_{2m}, P_{2m+1}$ is called a *transition sequence* if (1) for $1 \leq i \leq m$, P_{2i-1} is in S_2, (2) P_{2m+1} and, for $0 \leq i \leq m$, P_{2i} are in S_1, (3) P_0 is the initial state of M, and (4) P_{2m+1} is a final state of M. A transition sequence $P_0, P_1, \ldots, P_{2m}, P_{2m+1}$ is S_1-*consistent* on the input string x if there exists a sequence $I_0, I_1, \ldots, I_m$, where I_0 is the initial position for heads numbered 1 through k, such that, for $0 \leq j \leq m$, if M starts in state P_{2j} with heads 1 through k in position I_j then there exists a sequence of steps such that M enters state P_{2j+1} (which is in S_2) with heads 1 through k in position I_{j+1} and M will not have entered any state in S_2 at any earlier step in the sequence. Likewise,

a transition sequence $P_0, P_1, \ldots, P_{2m}, P_{2m+1}$ is S_2-*consistent* on the input string x if there exists a sequence $I_0, I_1, \ldots, I_{m-1}$, where I_0 is the initial position for heads $k+1$ through $2k$, such that, for $0 \leq j < m$, if M starts in state P_{2j+1} with heads $k+1$ through $2k$ in position I_j, then there is a sequence of steps such that M enters state P_{2j+2} (which is in S_1) with heads $k+1$ through $2k$ in position I_{j+1} and M will not have entered any state in S_1 at any earlier step in the sequence. It follows from the definitions that there is a transition sequence which is both S_1-consistent and S_2-consistent on the input string x if, and only if, x is accepted by M.

The k-head pushdown automaton P_M will operate in two phases. Phase I consists of writing an S_1-consistent transition sequence on the pushdown store. Phase II is to determine whether the pushdown store contains an S_2-consistent transition sequence. Some of the details of the manner in which P_M executes each phase is described below :

Phase I : In this phase P_M constructs on its pushdown store an S_1-consistent transition sequence. P_M begins by nondeterministically selecting a position I for its k heads and a final state s in S_1 of M. (The position I is selected by moving the k heads into that position.) Since it is desired that the transition sequence appear in order on the pushdown store, the elements of the sequence are obtained in reverse order. That is, a final state is at the bottom of the pushdown store and the initial state is on top. P_M writes the state s on the pushdown store and executes the following steps, for $s_0 = 0$ and $I_0 = I$:

(1) Among the set of all states s in S_1 and head positions I such that M may in one step move from state s with heads 1 through k in position I to state s_0 with heads 1 through k in position I_0, P_M chooses one such state s' and position I'. (If the set is empty, then P_M stops without accepting the input. Since each of the possible head positions investigated must be obtained by moving a single head either right of left from position I_0, P_M has no difficulty keeping track of its original head position I_0 during this investigation.) P_M then moves its k heads to position I', sets I_0 to I', and sets s_0 to s'. If the new s_0 is the initial state of M and I_0 is the initial position for the heads 1 through k of M, then P_M may choose next to enter Phase II. Otherwise, P_M nondeterministically chooses whether to execute step (1) or step (2) of Phase I.

(2) P_M writes the state s_0 on its pushdown store, chooses nondeterministically a state s in S_2, writes s on the pushdown store, and executes step (1) of Phase I with $s_0 = s$.

Phase II. In this phase P_M determines whether the transition sequence which appears in the pushdown store is S_2-consistent. At the beginning of this phase s_0 is set to the initial state of M and I_0 is set to the initial position for heads $k+1$ through $2k$ of M. P_M places its k heads in position I_0 and executes the following steps :

(1) P_M deletes the top two state symbols, say s_1 and s_2, from the pushdown store. If $s_1 \neq s_0$, then P_M halts without accepting the input string. Otherwise, if

$s_1 = s_0$, then P_M sets s_0 to s_2. If the pushdown store is empty, then P_M halts and accepts the input. Otherwise, P_M executes next step (2) of Phase II.

(2) P_M chooses nondeterministically a state s and position I such that M may in one step move from state s_0 with heads $k + 1$ through $2k$ in position I_0 to state s and heads $k + 1$ through $2k$ in position I. P_M sets s_0 to s and I_0 to I, moves its k heads to position I, and, if s_0 is in S_1, then P_M executes next step (1) of Phase II. Otherwise, if s_0 is in S_2, then P_M re-executes step (2) of Phase II.

It follows that P_M recognizes those, and only those, input strings x for which there is an S_1-consistent and S_2-consistent transition sequence. As indicated, this is equivalent to the statement that P_M recognizes x if, and only if, M accepts x.

It is not known whether 2DFA($2k$) $\subseteq$ 2DPDA(k) is true for any value of $k \geq 1$. We conjecture that 2DFA(2) $\subseteq$ 2DPDA(1) is false. A "hardest" language L for the family 2DFA(2) is described later in these remarks which is in 2DPDA(1) if and only if 2DFA(2) $\subseteq$ 2DPDA(1).

It follows from Theorem 2 and our observations on the time bounds for recognizing multihead pda languages that 2NFA($2k$) is contained in the class of languages recognized in time n^{4k} by deterministic multitape Turing machines and time $0(n^{3k})$ by random access machines. That is, each language in 2NFA(k) can be recognized in time $n^{4\lceil k/2 \rceil}$ by a Turing machine and time $0(n^{3\lceil k/2 \rceil})$ by a RAM. These are improvements on earlier stated time bounds [15].

The next result gives an upper bound on the number of heads needed by a deterministic two-way pda to simulate a nondeterministic two-way k-head finite state automaton. It is known that for each k there is an m_k such that 2NPDA(k) $\subseteq$ 2DPDA(m_k) [7]. It is also known that m_k need not be larger than $12k + 1$ [23]. The following result yields a smaller bound for the class 2NFA(k).

THEOREM 3 : *For $k \geq 1$, $2NFA(k) \subseteq 2DPDA(2k)$.*

Proof. Let M be a nondeterministic two-way k-head finite automaton. We construct a deterministic two-way $2k$-head pushdown automaton P_M to recognize those, and only those, strings recognized by M. P_M operates basically by trying all possible computations of M on the given input. (The algorithm is similar to the depth first search algorithm for finding paths in a graph given on pages 16-18 of [13]). The auxiliary store of P_M is used to record the most recent attempted computation of M. P_M uses k of its heads to simulate a computation of M and the other k heads to count the number of steps of M s computation simulated. Since a computation by M may not terminate, it would appear that the extra heads used for a counter are necessary in general. The fact that k heads are sufficient to detect nondetermination follows from the observation that there are only cn^k distinct configurations of M on an input of length n, for some $c > 0$. Thus if M recognizes a word x then there must be a computation which indicates acceptance of x and does not repeat a configuration. Finally, it is straightforward to implement with k heads on an input x a counter to represent numbers as large

as $c\,|x|^k$, for any $c \geq 0$. In the following algorithm we shall refer to the number represented by the position of heads $k + 1$ through 2k as "the counter." Let the variable s be set initially to the initial state of M and the variable j be set initially to one. Heads 1 through k of P_M are placed initially on the left endmarker and the counter is set to zero. Let h be the head selector function of M. For each state t of M and input symbol a we shall order the possible transitions of M in state t scanning a with head $h(t)$. Therefore, we may refer to choice i, for $1 \leq i \leq m$, where m is the number of such choices. (If m is zero, then there are no transitions in state t scanning the symbol a.) A choice will be a pair (t', λ) consisting of a next state t' and an indication that head $h(t)$ is moved right λ squares $(\lambda \in \{-1, 0, 1\})$.

(1) If M in state s with head $h(s)$ scanning the symbol scanned presently by head $h(s)$ of P_M does not have a *j-th* choice, then P_M next executes step (2). Otherwise, let (t, λ) be choice j. P_M writes (s, λ, j) on the pushdown store, sets s to t, sets j to one, moves head $h(s)$ to the right λ cells, and adds one to the counter. If the new state s is a final state, then P_M stops and accepts the input. If the state s is not final but the counter is presently at its maximum value, then P_M next executes step (2). Otherwise, P_M executes step (1) again.

(2) If the auxiliary store of P_M is empty, then P_M stops and rejects the input. Otherwise, P_M takes the top triple (s', λ', j') off the store, sets s to s', moves head $h(s')$ to the left λ' cells, sets j to $j' + 1$, decreases the counter by one, and executes step (1). □

For nondeterministic one-way k-head finite state automata we may obtain a better result. That is, the previous algorithm required k additional heads, used as a counter, to detect nondeterminating computations by the nondeterministic k-head automaton. In the special case of one-way k-head finite state automata one can eliminate transitions that leave each head fixed in place ; therefore, one can eliminate nonterminating computations. That is, if M is a one-way k-head automaton with s states and M has more than s consecutive transitions which leave every head in the same position, then M has a shorter equivalent computation. Therefore, any such automaton M can be replaced by an equivalent automaton M' with the same number of heads that does not possess transitions which leave every head in the same position. By eliminating the k heads used for a counter in the previous algorithm one obtains the following result.

COROLLARY 2 : *For* $k \geq 1$, $1NFA(k) \subseteq 2DPDA(k)$.

It follows from our earlier observations on time bounds for multihead pda languages that any language in 1NFA(k) can be recognized in time $0(n^k)$ by a RAM. It is unknown whether or not 1NFA($k + 1$) $\subseteq$ 2DPDA(k) is valid for any value of $k \geq 1$.

AN IMPROVED HIERARCHICAL RESULTS FOR MULTIHEAD FINITE AUTOMATA

The next result shows that NSPACE($\log_2 n$, 2^k) $\subseteq$ 2NFA($k + 3$) and DSPACE ($\log_2 n$, 2^k) $\subseteq$ 2DFA($k + 3$). This improves on earlier results [17, 27]. First we establish a special case (which settles an open question posed by Seiferas (27]) :

LEMMA 2 : $DSPACE(log_2 n, 2) \subseteq 2DFA(4)$
$NSPACE(log_2 n, 2) \subseteq 2NFA(4)$

Proof : Let T be a $\lfloor \log_2 n \rfloor$ tape bounded Turing machine with two work-tape symbols. We construct a four-head finite state automaton M_T which indirectly simulates T as follows. One of the heads of M_T is used to simulate the movements of the input head of T. Let the other three heads be called heads A, B, and C. Then M_T with heads A and B shall represent the contents of $T's$ worktape as two integers in 2-adic notation. That is, let T have the worktape symbols 1 and 2. If T has the string $a_0 a_1 \ldots a_m a_{m+1} \ldots a_{m+p}$ on its worktape, where a_i is either the symbol 1 or 2 ($1 \leqslant i \leqslant m + p$) and a_{m+1} is the current symbol scanned by T, then M will represent this worktape configuration by placing head A on square $a_0 2^m + a_1 2^{m-1} + \ldots + a_m$ and head B on square $a_{m+1} + a_{m+22} + \ldots + a_{m+p} 2^{p-1}$. Head C is used in the process of halving or multiplying by two these head positions in order to represent a new worktape configuration of T and in detecting whether the symbol scanned is 1 or 2. That is, as either head A or head B moves toward the left endmarker head C may, starting at the left endmarker, move two squares to the right for each square head A or head B passes. Thus when head A or head B reaches the left endmarker head C is at a position twice as far to the right as where head A or B started. A similar process may also be used for halving the head positions.

M_T may detect which symbol is scanned by T by halving the position of head B and in the process detecting whether the position was an odd or even number of squares to the right. The position is an odd number of squares to the right if and only if the symbol scanned is 1. M_T may then return head B to its original position by reversing this process.

M_T may represent a transition of T which prints the symbol d_1 over the symbol d_2(d_1, d_2 in $\{ 1, 2 \}$) and moves the worktape head to the right by : (1) multiplying by two the position of head A and then moving d_1 more squares to the right and (2) moving head B to the left d_2 squares and then halving the new position of head B. In a similar manner M_T may represent transitions of T which move the worktape head left.

Thus M_T may simulate T with only four heads. M_T will be nondeterministic if and only if T is nondeterministic. □

We can extend Lemma 2 to the general case in many ways. The technique described here is due to Seiferas [28]. It is known that the family of languages

recognized by $\lfloor \log_2 (n) \rfloor$-tape bounded Turing machines with 2^k worktape symbols is identical to the family of languages recognized [by k L $\log_2 (n)$]-tape bounded Turing machines with two worktape symbols. Using this observation we see that k "pages" each of length $\lfloor \log_2 (n) \rfloor$ can be represented by $k + 1$ heads, where two of these heads, as in the proof of Lemma 2, are used to represent the currently scanned page. With the bookkeeping head C and the head used for simulating the input head, as described in the proof of Lemma 2, we have a total of $k + 3$ heads. Therefore, we have established the following result :

THEOREM 4 : $DSPACE(log_2 n, 2^k) \subseteq 2DFA(k + 3)$
$NSPACE(log_2 n, 2^k) \subseteq 2NFA(k + 3)$

COROLLARY : *For* $k \geq 1$, *there is an* $L \subseteq \{ 1 \}^*$ *which is in* $2NFA(k + 4) - 2NFA(k)$ $(2DFA(k + 4)\text{-}2DFA(k))$.

Proof : Seiferas has shown the existence of an $L \subseteq \{ 1 \}^*$ which is in NSPACE($\log_2 n$, 2^{k+1})-NSPACE($\log_2 n$, 2^k) (DSPACE($\log_2 n$, 2^{k+1})-DSPACE($\log_2 n$, 2^k)) [27]. The result then follows from the relationships NSPACE($\log_2 n$, 2^{k+1}) $\subseteq$ 2NFA($k + 4$), 2NFA(k) $\subseteq$ NSPACE($\log_2 n$, 2^k), and the similar relationships for deterministic classes. □

This hierarchical result improves on earlier results in [4, 12, 27]. It is unknown whether 2NFA($k + 3$) − 2NFA(k) or 2DFA($k + 3$) − 2DFA(k) contains a language over a single letter alphabet.

We note in conclusion that each of the families 2DFA(k), 2NFA(k), 2DPDA(l), and 2NPDA(l), for $k \geq 2$ and $l \geq 1$, possess a language L_0 with the property that for any language L in the family there is a homomorphism h such that $L - \{ e \} = h^{-1}(L_0)$. That is, there is a hardest language in each of these families. (A hardest context-free language is described in [14].) That each of these families is closed under the operation of inverse homomorphism is easily verified. Thus if $\mathscr{L}_1$ and $\mathscr{L}_2$ are two such families then $\mathscr{L}_1 \subseteq \mathscr{L}_2$ if and only if the language L_0 for the family $\mathscr{L}_1$ is in $\mathscr{L}_2$. Furthermore the time and tape complexity of the hardest language L_0 for a class $\mathscr{L}$ is the least upper bound for the complexity of the whole class. (This is true if the complexity bounding function f for the hardest language is *semihomogeneous*. That is, for every $c > 0$ there is a $d > 0$ such $f(cn) \geq df(n)$, for all n.)

Corresponding to each k-head automaton M with input alphabet $\{ 0, 1 \}$ we may associate a unique encoding string e_M over an alphabet not containing the symbols 0 or 1. This encoding function should satisfy the property that (1) the set of all encodings is a regular set and (2) there is an automaton V in the class encoded such that if V is provided with a string representing a state of an automaton M, also in the class, (this string being provided either by the position of one of the heads on a substring of the encoding of M, in the case of a class of multihead finite state automata, or by the initial portion of the contents of the

pushdown store, in the case of a class of multihead pda) and if V is provided with an input symbol and, in the case of a class of multihead pda, the top pushdown store symbol, then V is able to determine from an input string e_M the set of possible next transitions of M. Most row by row encodings of transition tables for automata satisfy these conditions. For any given class, let $L_0 = \{ a_1 x a_2 x \ldots a_n x \mid n \geqslant 1$ and x is the encoding of an automaton M in the given class such that M recognizes $a_1 a_2 \ldots a_n \}$. Let L be a language in one of the stated classes. If M is a k-head automaton that recognizes L, then let h be the homomorphism defined by $h(0) = 0e_M$ and $h(1) = 1e_M$. It follows that for any nonempty string x over the alphabet $\{ 0, 1 \}$ that $h(x)$ is in L_0 if and only if x is in L. Furthermore, by the conditions imposed on the encodings, L_0 is recognized by an automaton in the given class. It is clear that for languages L in the given class which are not over the alphabet $\{ 0, 1 \}$ there are languages $L' \subseteq \{ 0, 1 \}^*$ in the class such that $L = g^{-1}(L')$. Therefore, the claim has been established. As an illustration of the information obtained from this observation we note that 2DPDA(k) $\neq$ DLOG for any k. A different technique has been used to show that 2DPDA(1) $\neq$ DLOG in [13]. Our observation follows from the known facts that DLOG $= \bigcup_k$ 2DFA(k) and 2DFA(k) $\subsetneq$ 2DFA($k + 2$) for all k [17, 19]. That is, DLOG cannot possess a hardest language (in the sense indicated) and, therefore, cannot be identical to any class which does. Whether or not DLOG $\subseteq$ 2DPDA(k) for some k or 2DPDA $\subseteq$ DLOG is unknown. Galil has shown that 2DPDA $\subseteq$ DLOG if and only if P-TIME = DLOG [13].

Note added in proof : The author has recently shown that 2NPDA(k) $\subseteq$ 2DPDA($4k + 1$) which improves on results cited in the paragraph before theorem 3. This improved result and other results relevant to the topics described in this paper are contained in *Separating Tape Bounded Auxiliary Pushdown Automata Classes,* Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, held in Boulder, Colorado (U.S.A.), May 2-4, 1977.

REFERENCES

1. A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, *Time and Tape Complexity of Pushdown Automata*, Information and Control, 13, 3, 1968, 186-206.
2. A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, *On the Computational Power of Pushdown Automata*, J. Computer and System Sci., 4, 2, 1970, 129-136.
3. A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.
4. C. BEERI, *Reductions in the Number of Heads for the Nondeterminancy Problem in Multihead Automata*, Technical Report, Institute of Mathematics, The Hebrew Institute of Jerusalem, Israel.
5. R. BOOK, *Translational Lemmas, Polynomial Time, and (Log (n))^j-Space*, Theoretical Computer Science, 1, 1976, 215-226.

6. R. BOOK, *Comparing Complexity Classes*, J. Computer and System Sci., 9, 1974, 213-229.
7. S. A. COOK, *Characterizations of Pushdown Machines in Terms of Time-Bounded Computers*, J. ACM, 18, 1971, 4-18.
8. S. A. COOK, *An Observation on Time-Storage Trade Off*, J. Computer and System Sci., 9, 1974, 308-316.
9. S. A. COOK and R. SETHI, *Storage Requirements for Deterministic Polynomial Time Recognizable Languages*, Sixth Annual ACM Symposium on Theory of Computing, 1974, 40-46.
10. S. A. COOK, *Linear Time Simulation of Deterministic Two-Way Pushdown Automata*, Proceedings of IFIP Congress, 1971, North Holland Publishers.
11. S. A. COOK and R. A. RECKHOW, *Time Bounded Random Access Machines*, J. Computer and System Sci., 7, 1973, 354-375.
12. P. FLAJOLET and J. M. STEYAERT, *Decision Problems for Multihead Finite Automata*, Proceedings of Symposium and Summer School on the Mathematical Foundations of Computer Science, High Tatras, Czechoslavakia, 1973, 225-230.
13. Z. GALIL, *Two-Way Deterministic Pushdown Automaton Languages and Some Open Problems in the Theory of Computation*, Proceedings of Fifteenth Annual IEEE Symposium on Switching and Automata Theory, 1974, 170-177.
14. S. A. GREIBACH, *The Hardest Context-Free Language*, SIAM J. on Computing, 2, 1973, 304-310.
15. S. A. GREIBACH, *A Note on the Recognition of One Counter Language*, Revue Française d'Automatique, Informatique et Recherche Opérationnelle, 1975, 5-12.
16. M. A. HARRISON and O. H. IBARRA, *Multitape and Multihead Pushdown Automata*, Information and Control, 13, 1968, 433-470.
17. J. HARTMANIS, *On Nondeterminancy in Simple Computing Devices*, Acta Informatica, 1, 1972, 336-344.
18. J. E. HOPCROFT and J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.
19. O. H. IBARRA, *On Two-Way Multihead Automata*, J. Computer and System Sci., 7. 1973, 28-36.
20. O. H. IBARRA, *Characterizations of Some Tape and Time Complexity Classes of Turing Machines in Terms of Multihead and Auxiliary Stack Automata*, J. Computer and System Sci., 5, 1971, 88-117.
21. N. D. JONES, *Space-Bounded Reducibility among Combinational Problems*, J. Computer and System Sci., 11, 1975, 68-85.
22. N. D. JONES and W. T. LAASER, *Complete Problems for Deterministic Polynomial Time*, Proceeding of Sixth Annual ACM Symposium on Theory of Computing, 1974, 33-39.
23. T. KAMEDA, *Pushdown Automata with Counters*, J. Computer and System Sci., 6, 1972.
24. T. KASAMI, *A Note on Computing Time for Recognition of Languages Generated by Linear Grammars*, Information and Control, 10, 1967, 209-214.
25. P. M. LEWIS, R. E. STEARNS and J. HARTMANIS, *Memory Bounds for the Recognition of Context-Free and Context-Sensitive Languages*, Proceedings of Sixth Annual IEEE Conference on Switching Circuit Theory and Logical Design, 1965, 199-212.
26. A. R. MEYER and L. J. STOCKMEYER, *Word Problems Requiring Exponential Time*, Proceedings of Fifth Annual ACM Symposium on Theory of Computing, 1973, 1-9.

27. J. SEIFERAS, *Nondeterministic Time and Space Complexity Classes*, Ph. D. dissertation, MIT, 1974. Project Mac Report TR-137, MIT, Cambridge, Mass.

28. J. SEIFERAS, personal communication.

29. I. H. SUDBOROUGH, *On Tape-Bounded Complexity Classes and Multihead Finite Automata*, J. Computer and System Sci., 10, 1975.

30. I. H. SUDBOROUGH, *On Deterministic Context-Free Languages*, Multihead Automata , *and the Power of an Auxiliary Pushdown Store*, Proceedings of Eighth Annual ACM Symposium on Theory of Computing, 1976, 141-148.

31. L. VALIANT, *General Context-Free Recognition in Less than Cubic Time*, J. Computer and System Sci., 10, 1975, 308-315.

32. R. WEICKER, *General Context Free Language Recognition by a RAM with Uniform Cost Criterion in Time $n^2 \log n$*, Technical Report No. 182, February, 1976, The Pennsylvania State University, University Park, Pa.