# Remarks on Multihead Pushdown Automata and Multihead Stack Automata 

Satoru Miyano<br>Department of Mathematics, Kyushu University 33, Fukuoka 812, Japan

Received August 23, 1982; revised December 7, 1982


#### Abstract

It is shown that $k+1$ heads are better than $k$ for one-way multihead pushdown (resp. stack) automata if they do not have endmarkers on the input tape and accept by final state with at least one input head at the right end of the input string. In addition, for two way multihead pushdown (resp. stack) automata, "hardest languages" are described. It is also shown that for two-way multihead pushdown (resp. stack) automata there is a language with the hardest time and space complexity which can be written as $\sqrt[+]{L}$ for some one-way multihead pushdown (resp. stack) automaton language $L$, where $\sqrt[+]{L}=\left\{w \mid w^{n}\right.$ is in $L$ for some $n \geqslant 1\}$. A representation theorem for recursively enumerable languages is also given.


## 1. Introduction

The question whether $k+1$ heads are better than $k$ has been discussed for several kinds of multihead automata; one-way multihead finite automata [11, 16, 19-20], two-way multihead finite automata [10, 12, 14-15, 18], two-way multihead pushdown automata [10], two-way multihead stack automata [9], one-way multihead stackcounter automata and one-way multihead counter automata [13]. However, it has not been proved that there is a hierarchy with respect to the number of input heads for one-way multihead pushdown automata and multihead stack automata. This is mainly because these automata are too complex to be analyzed by the counting argument, but on the other hand they are not so powerful to allow the diagonalization.

In this paper, we consider one-way multihead pushdown (resp. stack) automata $[5,8]$ without endmarkers on the input tape. We assume that input head detections are not allowed and they accept an input by final state with at least one input head at the right end of the input. This definition differs from the one in [8] only in nonexistence of the endmarkers. Under these assumptions, we will prove in Section 2 that $k+1$ heads are more powerful than $k$ for one-way multihead pushdown automata and multihead stack automata. Usually, the diagonalization argument is used for twoway automata to prove that an additional input head increases the recognition power and the counting argument is applied to one-way automata. In contrast, our result is proved by reducing the hierarchy results on two-way multihead automata $[9,10]$ to one-way automata. Therefore, in our hierarchy results, the language which separates
the class for $k+1$ input heads from that for $k$ is actually obtained by the diagonalization argument. Our proof also shows that the hierarchy results on twoway automata are useful to derive some hierarchy results on one-way automata without endmarkers.

The hardest languages [6] have received attentions since they are hardest in the sense of time and space complexity and they are useful for the representations of language families. It is known that context-free languages, nondeterministic and deterministic context-sensitive languages, and recursively enumerable languages have the hardest languages $[4,6]$. On the other hand, there are no hardest languages for deterministic context-free languages [7], linear context-free languages [2], one counter languages [1] and regular languages [4]. It was shown in [17] that the class of twoway nondeterministic pushdown automaton languages has a language $G_{0}$ which is of the form $G_{0}=\sqrt[+]{L}=\left\{w \mid w^{n}\right.$ is in $L$ for some $\left.n \geqslant 1\right\}$ for some context-free language $L$ and is hardest in the sense of time and space complexity. This result is espcially interesting because of the relationship between one-way automata and two-way automata.

In Section 3 we describe a hardest language for two-way $k$-head pushdown (resp. $k$-head stack, $(k+1)$-head finite) automata for $k \geqslant 1$. Then we show that the class of two-way $k$-head pushdown (resp. stack) automaton languages has a language with the hardest time and space complexity which can be written as $\sqrt[\downarrow]{L}$ for some one-way $k$ head pushdown (resp. stack) automaton language. We also give a representation theorem for recursively enumerable languages by means of the root closure operation $\sqrt[+]{ }$ and the two-way deterministic pushdown automaton languages.

## 2. Hierarchy Results

We assume that one-way multihead pushdown (resp. stack) automata do not have endmarkers on the input tape and they accept an input by entering a final state with at least one input head at the right end of the input. We omit the formal definitions of these devices.

We denote the class of languages recognized by one-way nondeterministic (resp. deterministic) $k$-head pushdown automata by $\operatorname{INPDA}(k)$ (resp. IDPDA $(k)$ ). The class of languages recognized by one-way nondeterministic (resp. deterministic) $k$ head stack automata is denoted by 1NSA $(k)$ (resp. IDSA $(k)$ ). Two-way multihead pushdown (resp. stack, finite) automata have the right and left endmarkers on the input tape and they accept by final state with all input heads on the right endmarker. We denote by 2NSA $(k)$ (resp. 2DSA $(k)$ ) the class of two-way nondeterministic (resp. deterministic) $k$-head stack automaton languages. Similarly, 2NPDA(k) (resp. 2DPDA $(k)$ ) and 2NFA $(k)$ (resp. 2DFA $(k)$ ) describe the class of two-way nondeterministic (resp. deterministic) $k$-head pushdown automaton languages and the class of two-way nondeterministic (resp. deterministic) $k$-head finite automaton languages, respectively. We abbreviate 2NSA(1), 2DSA(1), 2NPDA(1), 2DPDA(1) as 2NSA, 2DSA, 2NPDA, 2DPDA, respectively.

The following hierarchy results will be proved by using the hierarchy results on the corresponding two-way input head automata.

```
Theorem 2.1. (1) \(1 \operatorname{NSA}(k+1) \neq 1 \operatorname{NSA}(k)\) for \(k \geqslant 1\).
    (2) \(\operatorname{IDSA}(k+1) \neq 1 \mathrm{DSA}(k)\) for \(k \geqslant 1\).
    (3) \(\operatorname{INSA}(k) \neq 1 \mathrm{DSA}(k)\) for \(k \geqslant 1\).
    (4) \(1 \mathrm{NPDA}(k+1) \neq 1 \mathrm{NPDA}(k)\) for \(k \geqslant 1\).
    (5) \(\operatorname{IDPDA}(k+1) \neq 1 \mathrm{DPDA}(k)\) for \(k \geqslant 1\).
```

Lemma 2.2. For each two-way $k$-head stack (resp. pushdown) automaton $M$, there is a two-way $k$-head stack (resp. pushdown) automaton $M^{\prime}$ which reverses its input heads on the right and left endmarkers and recognizes the same language as $M$. Furthermore, if $M$ is deterministic, then $M^{\prime}$ can be chosen to be deterministic.

Proof. Lemma is easily shown for two-way $k$-head pushdown automata. For twoway $k$-head stack automata, a technical idea is required. Let $M$ be a two-way $k$-head stack automaton and let $\Gamma$ be the set of stack symbols of $M$. We use two new distinct symbols $a$ and $b$ not in $\Gamma$. For each stack symbol $A$ in $\Gamma, M^{\prime}$ on input $w=a_{1} a_{2} \cdots a_{n}$ of length $n$ uses the string $a^{i} b^{n-i} A$ for some $i . M^{\prime}$ simulates $M$ on $w$ in the following way:
(1) Pushdown operation: $M$ is pushing down a symbol $A$ on the top of the stack with the first input head on $a_{i}$. If the first input head of $M^{\prime}$ is sweeping from left to right (resp. right to left), then $M^{\prime}$ performs the following steps:

1. $M^{\prime}$ moves the first input head to the right (resp. left) endmarker and puts the string $a^{n-i}$ (resp. $a^{i}$ ) on the stack.
2. $M^{\prime}$ enters the stack search mode. By using the string $a^{n-i}$ (resp. $a^{i}$ ) stored in the stack, $M^{\prime}$ moves the first input head to the position of $a_{i} . M^{\prime}$ moves the stack pointer to the top of the stack.
3. $M^{\prime}$ moves the first input head to the left (resp. right) endmarker and puts the string $b^{i} A$ (resp. $b^{n-i} A$ ) on the stack.
4. $M^{\prime}$ enters the stack search mode, $M^{\prime}$ moves the first input head to the position of $a_{i}$ by using the string $b^{i}$ (resp. $b^{n-i}$ ) and returns the stack pointer to the top of the stack.
(2) Input head move: If $M$ moves the $j$ th input head left (resp. right) while the $j$ th input head of $M^{\prime}$ is sweeping from left to right (resp. right to left), then two cases arise according to the mode of the stack. Assume that the $j$ th input head of $M$ is on $a_{i}$. If $M$ is in the stack search mode, $M^{\prime}$ performs the following steps:
5. $M^{\prime}$ moves the $j$ th input head to the right (resp. left) endmarker and simultaneously moves down the stack pointer to the position $x_{i}$ (resp. $x_{n-i}$ ) of $x_{1} \cdots x_{n} A$, where $x_{1} \cdots x_{n} A$ is the string stored instead of the stack symbol $A$.
6. $M^{\prime}$ moves the stack pointer to the position of $x_{1}$ and puts the $j$ th input head on $a_{i-1}\left(\right.$ resp. $\left.a_{i+1}\right)$.
7. $M^{\prime}$ returns the stack pointer to the position of $A$.

If $M$ is in the push-pop mode, $M^{\prime}$ performs the following steps:

1. $M^{\prime}$ moves the $j$ th input head to the right (resp. left) endmarker and puts the string $a^{n-i+1}$ (resp. $a^{i+1}$ ) on the stack.
2. $M^{\prime}$ pops up the string $a^{n-i+1}$ (resp. $a^{i+1}$ ) and simultaneously moves the $j$ th input head to the position of $a_{i-1}$ (resp. $a_{i+1}$ ).

The remaining operations can be directly simulated. By definition, it is obvious that if $M$ is deterministic, so is $M^{\prime}$.

Proof of Theorem 2.1. (1) Since 2NSA $(k+1) \neq 2 \mathrm{NSA}(k)$ [9], there exists a language $L \subseteq \Sigma^{*}$ in 2NSA $(k+1)-2$ NSA $(k)$. By Lemma $2.2, L$ is recognized by a two-way nondeterministic $(k+1)$-head stack automaton $M_{0}$ that reverses its input heads only on the endmarkers. It is easy to construct from $M_{0}$ a one-way nondeterministic ( $k+1$ )-head stack automaton $M_{1}$ which satisfies (i) and (ii).
(i) $M_{0}$ accepts $w$ if and only if $M_{1}$ accepts $\left(w \$ w^{R} \$\right)^{n} w \$$ for some $n \geqslant 0$, where $\$$ is a symbol not occurring in the input alphabet $\Sigma$ of $M_{0}$.
(ii) Every string accepted by $M_{1}$ has the form $w_{1} \$ \cdots w_{2 n} \$ w_{2 n+1} \$$ for some $n \geqslant 0$, where $w_{i}$ is in $\Sigma^{*}$ for $i=1, \ldots, 2 n+1$.

Suppose that $1 \mathrm{NSA}(k+1)=1 \mathrm{NSA}(k)$. Then let $M_{2}$ be a one-way nondeterministic $k$-head stack automaton which recognizes the same language as $M_{1}$. Consider a two-way nondeterministic $k$-head stack automaton $M_{3}$ which moves as follows: Given an input $w, M_{3}$ on $w$ simulates $M_{2}$ on ( $\left.w \$ w^{R} \$\right)^{n} w \$$ for some $n \geqslant 0$ by reversing its input heads on the endmarkers. If $M_{2}$ enters a final state, then $M_{3}$ moves all input heads to the right endmarker and accept $w$ by entering the final state of $M_{3}$. By (ii), if $M_{2}$ enters a final state, then it accepts ( $\left.w \$ w^{R} \$\right)^{n} w \$$ for some $n \geqslant 0$. Therefore by (i), $w$ is in $L$. Conversely, if $w$ is in $L$, then by (i) $M_{2}$ accepts $\left(w \$ w^{k} \mathbb{\$}\right)^{n} w \mathbb{\$}$ for some $n \geqslant 0$. Therefore $M_{3}$ accepts $w$. Hence $L$ is in $2 \mathrm{NSA}(k)$, a contradiction. Thus 1NSA $(k+1) \neq 1$ NSA $(k)$.

Conditions (2)-(5) can be proved in the same way by using the hierarchy results $2 \operatorname{DSA}(k+1) \neq 2 \mathrm{DSA}(k) \quad[9], \quad 2 \mathrm{NSA}(k) \neq 2 \operatorname{DSA}(k) \quad[3,9], \quad 2 \mathrm{NPDA}(k+1)+$ $2 \operatorname{NPDA}(k)[10], 2 \mathrm{DPDA}(k+1) \neq 2 \mathrm{DPDA}(k)$ [10], respectively.

Remark. It should be noted that the technique used in this section is also applicable to other hierarchy results. For instance, from the fact $2 \mathrm{NSA}(k) \neq$ 2NPDA $(k)$ [ 9 ] we can derive $1 \mathrm{NSA}(k) \neq 1 \mathrm{NPDA}(k)$. This is already known since the one-way multihead pushdown automaton languages have the semilinear property [8], but on the other hand there is a nonsemilinear one-way stack automaton language. We can also observe an implication that if $2 \mathrm{NPDA}(k) \neq 2 \mathrm{DPDA}(k)$, then $1 \mathrm{NPDA}(k) \neq 1 \mathrm{DPDA}(k)$. However, it seems more difficult to prove 2NPDA $(k) \neq$ 2DPDA( $k$ ).

## 3. Hardest Languages

Definition [6]. A language $L_{0} \subseteq \Sigma_{0}^{*}$ is said to be hardest for a class $C$ of languages if $L_{0}$ is in $C$ and for every language $L \subseteq \Sigma^{*}$ in $C$ there is a homomorphism $h_{L}: \Sigma^{*} \rightarrow \Sigma_{0}^{*}$ such that for each nonempty string $w$ in $\Sigma^{+} w$ is in $L$ if and only if $h_{L}(w)$ is in $L_{0}$.

For a language $L \subseteq \Sigma^{*}$, the root closure of $L$ is defined by $\sqrt[+]{L}=\left\{w \mid w^{n}\right.$ is in $L$ for some $n \geqslant 1\}$. It was proved in [17] that 2NPDA has a language with the lardest time and space complexity which is written as the root closure of some context-free language. The proof depends on the existence of hardest context-free language and therefore it does not work for 2DPDA since there is no hardest deterministic contextfree language.

Theorem 3.1 shows that hardest languages exist for various two-way multihead automaton language classes and its corollary extends the result in [17] for multihead nondeterministic and deterministic pushdown and stack automata.

Theorem 3.1. A hardest language exists for each of the classes 2NSA( $k$ ), 2DSA $(k), 2 \mathrm{NPDA}(k), 2 \mathrm{DPDA}(k), 2 \mathrm{NFA}(k+1)$ and $2 \mathrm{DFA}(k+1)$ for $k \geqslant 1$.

Proof. First, we describe hardest languages $L[2 N S A]$ and $L[2 D S A]$ for 2NSA and 2DSA, respectively. Let $M$ be a two-way stack automaton with the stack alphabet $\{A, B\}$. A move of $M$ is a quintuple $(p, x, d, \varphi, q)$, where $p$ and $q$ are states of $M, x$ is an input symbol, $d$ is an input head direction and $\varphi$ is an operation for the stack. It means that if $M$ in state $p$ reads the input symbol $x$ and the operation $\varphi$ is executable, then $M$ moves its input head according to $d$, operates $\varphi$ on the stack and changes its state to $q$. For an input symbol $x$, we denote by $M(x)$ the conventional tuple by tuple encoding of the collection of the moves of $M$ when $M$ reads the symbol $x$. We assume that the first state $p_{1}$ is the initial state and the $i$ th state $p_{i}$ is encoded as $0^{i-1} 10^{m-i} 1$ if $p_{i}$ is a final state else $0^{i-1} 10^{m-i+1}$, where $m$ is the number of states in $M$. The language $L[2 N S A]$ is defined as the collection of the strings satisfying the following conditions: For each $w$ in $L[2 N S A]$, there is a two-way stack automaton $M$ with the stack alphabet $\{A, B\}$ and an input $a_{1} \cdots a_{n}$ of $M$ such that $M$ accepts $a_{1} \cdots a_{n}$ and $w$ is of the form $w=\left[M(\$) M\left(a_{1}\right) M(\$)\right] \cdots\left[M(\$) M\left(a_{n}\right) M(\$)\right]$, where $\$$ and $\phi$ are the left and right endmarkers, respectively. The language $L[2 D S A]$ is defined by restricting $M$ to be deterministic in the above definition. For each $L \subseteq \Sigma^{*}$ in 2NSA (resp. 2DSA), we can take a two-way nondeterministic (resp. deterministic) stack automaton $M$ with the stack alphabet $\{A, B\}$ which recognizes $L$. Then we define a homomorphism $h_{L}: \Sigma^{*} \rightarrow \Sigma_{0}^{*}$ by $h_{L}(x)=[M(\mathbb{\$}) M(x) M(\$)]$ for $x$ in $\Sigma$, where $\Sigma_{0}$ is the alphabet of $L[2 N S A]$ (resp. $L[2 D S A]$ ). Notice that for each nonempty string $w$ in $\Sigma^{+}, w$ is in $L$ if and only if $h_{L}(w)$ is in $L[2 N S A]$ (resp. $L[2 D S A]$ ).

We now see that $L[2 N S A]$ is in 2NSA. We describe the moves of a two-way stack automaton $M_{0}$ which recognizes $L[2 N S A]$. Given an input $w, M_{0}$ first checks whether $w$ is in the right form. Assume that $w=\left[M(\$) M\left(a_{1}\right) M(\$)\right] \cdots$
$\left[M(\$) M\left(a_{n}\right) M(\$)\right]$ for some $M$ and $a_{1} \cdots a_{n} . M_{0}$ simulates $M$ on $a_{1} \cdots a_{n}$ in the following way: If $M$ is in state $p_{i}$ with the input head on $a_{k}$ and performing a move ( $p_{i}, a_{k}, d, \varphi, p_{j}$ ), then $M_{0}$ puts its input head on the move ( $p_{l}, a_{k}, d, \varphi, p_{j}$ ) in the $k$ th block $\left[M(\mathbb{\$}) M\left(a_{k}\right) M(\$)\right]$. The moves of $M$ on the endmarkers are simulated only when the input head of $M_{0}$ is in $\left[M(\$) M\left(a_{1}\right) M(\$)\right]$ and $\left[M(\$) M\left(a_{n}\right) M(\$)\right] . M_{0}$ executes the operation $\varphi$ for the stack. Then according to the input head direction $d$, $M_{0}$ moves its input head to the right or left block or remains in the same block while $M_{0}$ have to keep the next state $p_{j}$ in some way. If $M$ is in the push-pop mode, then $M_{0}$ can keep the state $p_{j}$ on the top of the stack. Coping with the case that $M$ is in the stack search mode, $M_{0}$ uses the string $a^{m} X$ instead of a stack symbol $X$, where $a$ is a symbol not in $\{A, B\}$ and $m$ is the number of states in $M$. Then by the stack pointer position in the string $a^{m} X, M_{0}$ can keep the state $p_{j}$. Using the state kept in the stack, $M_{0}$ can search the next move in the appropriate block. In this way $M_{0}$ simulates $M$ 's state transitions. Thus $L[2 N S A]$ is in 2NSA. Since a two-way deterministic stack automaton can check whether an encoding of a two-way stack automaton is deterministic or not, we also see that $L[2 D S A]$ is in 2DSA.

For two-way pushdown automata, we can define in a similar manner the languages $L[2 N P D A]$ and $L[2 D P D A]$. These languages can be shown to be hardest in the same way as in the case of two-way stack automata except that we need not use the string $a^{m} X$ representing a pushdown symbol $X$. For $k \geqslant 2$, we can define the hardest languages $L[2 N S A(k)], L[2 D S A(k)], L[2 N P D A(k)], L[2 D P D A(k)], L[2 N F A(k)]$, and $L[2 D F A(k)]$ in the same way. In these cases, no difficulty arises in the simulation of state transitions since other input head can be used to keep the next state $p_{j}$.

Corollary 3.2. There exists a language $L_{1} \subseteq \Sigma_{1}^{*}$ in 2NSA( $k$ ) (resp. 2DSA( $k$ ), 2NPDA( $k$ ), 2DPDA( $k$ )) with the following properties:
(1) $L_{1}=\sqrt[+]{L_{2}}$ for some $L_{2}$ in $1 \mathrm{NSA}(k)$ (resp. IDSA(k), $\operatorname{INPDA}(k)$, 1DPDA(k)).
(2) For every language $L \subseteq \Sigma^{*}$ in 2NSA( $k$ ) (resp. 2DSA( $k$ ), 2NPDA( $k$ ), 2DPDA $(k)$ ), there is a homomorphism $h_{L}: \Sigma^{*} \rightarrow \Sigma_{1}^{*}$ such that for each nonempty string $w, w$ is in $L$ if and only if $h_{L}(w) \$$ is in $L_{1}$, where $\$$ is a symbol in $\Sigma_{1}$.

Proof. It is not hard to modify Lemma 2.2 so that $M^{\prime}$ in Lemma 2.2 reverses its input head on the endmarkers and in each right to left sweep it does not change its state and no stack (resp. pushdown) operation is applied to the stack (resp. pushdown store). We consider the case of $2 \mathrm{NSA}(k)$. Other cases can be shown in the same way. Let $L_{0} \subseteq \Sigma_{0}^{*}$ be a hardest language for 2NSA $(k)$. Let $\$$ be a symbol not in $\Sigma_{0}$. Then we can define a language $L_{2} \subseteq\left(\Sigma_{0} \cup\{\$\}\right)^{*}$ in $1 \mathrm{NSA}(k)$ with the following properties:
(i) $w$ is in $L_{0}$ if and only if $(w \$)^{n}$ is in $L_{2}$ for some $n \geqslant 1$.
(ii) Each string in $L_{2}$ is of the form $w_{1} \$ w_{2} \$ \cdots w_{n} \$$ for some $n \geqslant 1$, where $w_{i}$ is in $\Sigma_{0}^{*}$ for $i=1, \ldots, n$.

Let $L_{1}=\sqrt{L_{2}}$. It is easy to see that $L_{1}$ is in $2 \operatorname{NSA}(k)$. Since $L_{0}$ is hardest for 2NSA( $k$ ), for each language $L \subseteq \Sigma^{*}$ in $2 \mathrm{NSA}(k)$ there is a homomorphism $h_{L}$ : $\Sigma^{*} \rightarrow \Sigma_{0}^{*}$ such that $w$ is in $L$ if and only if $h_{L}(w)$ is in $L_{0}$ for each $w$ in $\Sigma^{+}$. By (i), we see that $w$ is in $L$ if and only if $h_{L}(w) \$$ is in $L_{1}$.

We remark that since $h_{L}(w)$ is computable from $w$ by a deterministic Turing machine in linear time without using any worktape, the language described in Corollary 3.2 has the hardest time and space complexity.

The question whether 2DPDA is closed under root closure is posed in [17]. The following theorem shows that if 2DPDA is closed under root closure, then all recursively enumerable languages are in 2DPDA. Hence 2DPDA is not closed under root closure.

ThEOREM 3.3. For each recursively enumerable language $L \subseteq \Sigma^{*}$, there is a language $L^{\prime} \subseteq(\Sigma \cup\{\$\})^{*}$ in 2DPDA (resp. 2DFA(2)) such that $L \mathbb{S}=\sqrt[+]{L^{\prime}} \cap \Sigma^{*} \$$, where $\$$ is a symbol not in $\Sigma$.

Proof. Since $L$ is recursively enumerable, there exists a one-way deterministic two-counter machine $M$ which recognizes $L$. We firstly describe the moves of a twoway deterministic pushdown automaton $M^{\prime}$ by means of $M$. The strings accepted by $M^{\prime}$ have the form $(w \$)^{n}$ for some $n \geqslant 1$, where $w$ is in $\Sigma^{*}$. Assume that $M^{\prime}$ is given an input $(w \$)^{n}$. The string $w \$$ is called a block. $M^{\prime}$ simulates $M$ on $w$. The first counter of $M$ is simulated by the pushdown store of $M^{\prime}$. The second counter is simulated by the block position where the input head of $M^{\prime}$ stays. The input head position of $M$ is simulated by the input head position of $M^{\prime}$ in the block. When $M$ increases or decreases the content of the second counter by one, $M^{\prime}$ moves its input head to the right or left block while $M^{\prime}$ keeps the input head position of $M$ on the top of the pushdown store. If the second counter becomes greater than the number of blocks in the input, then $M^{\prime}$ stops and rejects the input. If $M$ accepts $w$, then $M^{\prime}$ accepts the given input. Notice that if $w$ is accepted by $M$, then $M^{\prime}$ accepts ( $\left.w \$\right)^{n}$ for sufficiently large $n$ and vice versa. Similarly, a two-way two-head deterministic finite automaton on the input ( $w \$)^{n}$ for sufficiently large $n$ can simulates $M$ on $w$ by using the block positions of the input heads as the counters of $M$.

## 4. Conclusion

By using the results on two-way input head automata, we showed several hierarchy results on one-way input head automata without endmarkers. Unfortunately, the idea presented in this paper does not seem to work for one-way multihead automata with endmarkers except the case of single input head nondeterministic automata. For oneway multihead alternating finite automata with endmarkers, the question whether an additional input head increases the power of automata is posed as an open question in [12]. For this question, if the input tape does not have endmarkers, we can give a partial solution that $1 \mathrm{AFA}(k+2) \neq 1 \mathrm{AFA}(k)$ for $k \geqslant 1$, where $1 \mathrm{AFA}(k)$ denotes the
class of languages recognized by one-way $k$-head alternating finite automata without endmarkers. For the proof of this fact, firstly notice that for every two-way $k$-head alternating finite automaton, there exists a two-way ( $k+1$ )-head alternating finite automaton which reverses its input heads on the endmarkers and recognizes the same language. It should be also noticed that the argument in the proof of Theorem 2.1 works for alternating multihead finite automata. By combining this observation with the result that two-way $(k+1)$-head alternating finite automata are more powerful than those with $k$ heads [12], we can prove that $1 \mathrm{AFA}(k+2)+1 \mathrm{AFA}(k)$ for $k \geqslant 1$.

## Acknowledgments

The author is grateful to Setsuo Arikawa and Takeshi Hayashi for many inspiring conversations about hardest languages and to an anonymous referee for his suggestions on the proof of Theorem 2.1.

## References

1. J.-M. Autebert, Non-principalité du cylindre des languages a compteur, Math. Systems Theory 11 (1977), 157-167.
2. L. Boasson and M. Nivat, Le cylindre des languages linéaires, Math. Systems Theory 11 (1977), 147-155.
3. S. A. Cook, Characterizations of pushdown machines in terms of time bounded computers, J. Assoc. Comput. Mach. 18 (1971), 4-18.
4. K. Culik II and H. A. Maurer, On simple representations of language families, RAIRO Inform. Theor. 13 (1979), 241-250.
5. S. Ginsburg, S. A. Greibach and M. A. Harrison, One-way stack automata, J. Assoc. Comput. Mach. 14 (1967), 389-418.
6. S. A. Greibach, The hardest context-free language, SIAM J. Comput. 2 (1973), 304-310.
7. S. A. Greibach, Jump PDAs and hierarchies of deterministic context-free languages, SIAM J. Comput. 3 (1974), 111-127.
8. M. A. Harrison and O. H. Ibarra, Multi-tape and multi-head pushdown automata, Inform. Control 13 (1968), 433-470.
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. O. H. Ibarra, On two-way multihead automata, J. Comput. System Sci. 7 (1973), 28-37.
11. O. H. Ibarra and C. E. Kim, On 3-head versus 2-head finite automata, Acta Inform. 4 (1975), 193-200.
12. K. N. King, Alternating multihead finite automata, in "Automata, Languages, and Programming" (S. Even and O. Kariv, Eds.), Lecture Notes in Computer Science, No. 115, pp. 506-520, SpringerVerlag, Berlin, 1981.
13. S. Miyano, A hierarchy theorem for multihead stack-counter automata, Acia Inform. 17 (1982), 63-67.
14. B. Monien, Transformational methods and their application to complexity problems, Acta Inform. 6 (1976), 95-108.
15. B. Monien, Two-way multihead automata over a one-letter alphabet, RAIRO Inform. Theor. 14 (1980), 67-80.
16. A. L. Rosenberg, On multihead finite automata, IBM I. Res. Develop. 10 (1966), 388-394.
17. W. Rytter, A hardest language recognized by two-way nondeterministic pushdown automata, Inform. Process. Lett. 13 (1981), 145-146.
18. J. I. Seiferas, Relating refined space complexity classes, J. Comput. System Sci. 14 (1977), 100-129.
19. I. H. Sudborough, One-way multihead writing finite automata, Inform. Control 25 (1976), 1-20.
20. A. C. Yao and R. L. Rivest, $k+1$ heads are better than $k$, J. Assoc. Comput. Mach. 25 (1980), 337-340.
