R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 14, n° 1, 1980, p. 67 à 82)

TWO-WAY MULTIHEAD AUTOMATA OVER A ONE-LETTER ALPHABET (*)

by Burkhard MONIEN (1)

Communicated by W. BRAUER

Abstract. — *Let $H(k)$ and $NH(k)$ denote the classes of languages over a one-letter alphabet acceptable by deterministic (respectively nondeterministic) two-way k-head finite automata. It will be shown that $H(k) \subsetneq H(k+1)$ and $NH(k) \subsetneq NH(k+1)$ holds for all $k \geq 1$. Hierarchy results are also proved for the classes of languages over a one-letter alphabet defined by k-counter automata with linear bounded counters and by k-register machines with linear bounded registers, respectively.*

Résumé. — *On désigne par $H(k)$ et $NH(k)$ les familles des langages sur un alphabet à une seule lettre qui sont reconnaissables par des automates finis bilatères à k têtes déterministes (resp. non déterministes). On montre que $H(k) \subsetneq H(k+1)$ et que $NH(k) \subsetneq NH(k+1)$ pour tout $k \geq 1$. On donne également une hiérarchie dans les familles de langages définis par des automates à k-c ompteurs linéairement bornés et par des machines à k-registres linéairement bornés respectivement.*

1. INTRODUCTION

In this paper we show that for languages over a one-letter alphabet two-way automata with $k+1$ heads are more powerful than two-way automata with k heads.

This result is related to the results of [3 and 4] concerning the refinement of complexity classes. It is wellknown that SPACE (log n), the class of languages acceptable within space bound log n, is identical with the class of languages acceptable by two-way multihead automata. Each two-way multihead automaton can be viewed as a log n-space bounded machine with restricted storage abilities.

It was shown before that over a one-letter alphabet two-way $(k+4)$-head automata are more powerful than k-head automata ([4, 5]) and that over a two-letter alphabet $k+1$ heads are more powerful than k ([2]; in [1] it was shown that

(*) Received October 1978, revised July 1979.
(1) G. H. Paderborn, Fachbereich 17, Mathematik-Informatik, Paderborn.

R.A.I.R.O. Informatique théorique/Theoretical Informatics, 0399-0540/1980/ 67 /$ 5.00

$k+2$ heads are more powerful than k heads). It is also known that $k+1$ heads are better than k for one-way automata [6].

The method used here to show that $k+1$ heads are more powerful than k for two-way automata, even for a one-letter alphabet, is similar to the method used in [1 to 5]. We define transformations which map multihead languages onto languages defined by fewer heads. These transformations allow us to use the assumption "$k+1$ heads have the same power as k heads" repeatedly to arrive at a contradiction.

We also prove hierarchy results for the classes of languages over a one-letter alphabet defined by k-counter automata and by k-register machines whose counters (or registers) are linearly bounded by the length of the input.

I am very obliged to I. H. Sudborough who drew my attention upon the subject of this paper and to J. I. Seiferas who proposed a more transparent way for the construction of the proof.

2. DEFINITIONS AND RESULTS

A *two-way k-head automaton* consists of a finite control and an input tape where k heads may move independently in both directions. The input is placed between two endmarkers ($\vdash$ and $\dashv$). The automaton starts in a distinguished starting state with its k heads on the left endmarker. It accepts the input string if it stops in an accepting state. The automaton is called deterministic if its next move function is deterministic, otherwise it is called nondeterministic. Let $H_\Sigma(k)$ $[NH_\Sigma(k)]$, be the class of languages over the alphabet Σ acceptable by deterministic [nondeterministic] two-way k-head automata.

A *two-way k-counter automaton* consists of a finite control, an input tape where one head is moving in both directions and k counters. With $C_\Sigma(k)[NC_\Sigma(k)]$ we denote the class of all languages over Σ acceptable by deterministic [nondeterministic] k-counter automata whose counters are always linearly bounded by the length of the input.

A *k-register machine* consists of a finite control and k registers. (In fact a register is just the same as a counter, namely a storage unit which can store one natural number and on which the operations $+1$, -1 and the predicate $\stackrel{?}{=} 0$ can be carried out.)

The machine starts with the input number in register 1 and the other registers storing zero. Note that such a machine can destroy its input number. It accepts an input number by reaching an accepting state. A register machine accepts a subset of $\mathbb{N} \cup \{0\}$. Throughout this paper we denote by $\mathbb{N}$ the set of natural

numbers. Let $R(k)$ $[RN(k)]$ be the class of all languages $L \subset \{0\}^*$ such that $\{n \mid O^n \in L\}$ is accepted by a deterministic [nondeterministic] k-register machine whose registers are always linear bounded by the length of the input.

Let us furthermore denote by $\mathrm{SPACE}_\Sigma(L(n))$ $[NSPACE_\Sigma(L(n))]$ the class of languages over the alphabet Σ which are acceptable by deterministic [nondeterministic] Turing machines within space bound $L(n)$.

It is straightforward to see that for every $k \geqq 1$,

$$R(k) \subset C_{\{0\}}(k-1) \subset H_{\{0\}}(k) \subset C_{\{0\}}(k) \subset R(k+2)$$

and that

$$\bigcup_{k \in N} R(k) = \bigcup_{k \in \mathbb{N}} C_{\{0\}}(k) = \bigcup_{k \in N} H_{\{0\}}(k) = \mathrm{SPACE}_{\{0\}}(\log n).$$

The corresponding result holds in the nondeterministic case.

In the following we only consider languages $L \subset \{O^{2^n}; n \in \mathbb{N}\}$. Let $\tilde{H}(k)$ be the class of all languages L such that $L \in H_{\{0\}}(k)$ and $L \subset \{O^{2^n}; n \in \mathbb{N}\}$. In the same way we will interprete the superskript $\tilde{}$ for the other complexity classes occuring in this paper.

We use the mapping $T_k : \{2^n \mid n \in \mathbb{N}\} \to \{2^n \mid n \in \mathbb{N}\}$ defined by $T_k(2^n) = 2^{k \cdot n}$ and whenever it is appropriate we will identify $\{0\}^*$ and $\mathbb{N}$.

First we will prove the following lemmas:

LEMMA 1: *$\tilde{H}(k) \not\subseteq \widetilde{\mathrm{SPACE}}(\log n)$, $\widetilde{NH}(k) \not\subseteq \widetilde{\mathrm{NSPACE}}(\log n)$ for all $k \in \mathbb{N}$.*

LEMMA 2: *For all $L \in \widetilde{\mathrm{SPACE}}(\log n)$ $[\widetilde{\mathrm{NSPACE}}(\log n)]$ there exists a number $k \in \mathbb{N}$ such that $T_k(L) \in \tilde{R}(3)$ $[\widetilde{NR}(3)]$.*

LEMMA 3: *For all $L \in \widetilde{\mathrm{SPACE}}(\log n)$ and for $k, j \geqq 1$:*

$$T_k(L) \in \tilde{H}(j)\ [\widetilde{NH}(j)] \quad \Rightarrow \quad L \in \tilde{H}(k \cdot j)\ [\widetilde{NH}(k \cdot j)].$$

LEMMA 4: *For all $L \in \widetilde{\mathrm{SPACE}}(\log n)$ and for $k > j \geqq 2$:*

$$T_{k+1}(L) \in \tilde{H}(j)\ [\widetilde{NH}(j)] \quad \Rightarrow \quad T_k(L) \in \tilde{H}(j+1)\ [\widetilde{NH}(j+1)].$$

From these lemmas our first theorem follows immediately.

THEOREM 1: *For all $j \in \mathbb{N}$:*

$$H_{\{0\}}(j) \not\subseteq H_{\{0\}}(j+1) \quad \text{and} \quad NH_{\{0\}}(j) \not\subseteq NH_{\{0\}}(j+1).$$

Proof: We give the proof only for the deterministic case. The proof for the nondeterministic case is exactly the same.

The result is true for $j=1$ since $\{O^{2^n};\ n\in\mathbb{N}\}\in H_{\{0\}}(2)-H_{\{0\}}(1)$.

Now suppose there exists some $j\geqq 2$ such that $H_{\{0\}}(j)=H_{\{0\}}(j+1)$. This implies $\tilde{H}(j)=\tilde{H}(j+1)$ and therefore the following is true

$$
\begin{aligned}
L\in \widetilde{\mathrm{SPACE}}(\log n) &\\
&\Rightarrow \exists k,\quad T_k(L)\in\tilde{R}(3)\subset\tilde{H}(3)\quad \text{lemma 2}\\
&\quad\Rightarrow T_k(L)\in\tilde{H}(j+1)=\tilde{H}(j)\\
&\Rightarrow T_{k-1}(L)\in\tilde{H}(j+1)=\tilde{H}(j)\quad \text{lemma 4}\ \Rightarrow\dots\\
&\quad\Rightarrow T_{j+1}(L)\in\tilde{H}(j+1)=\tilde{H}(j)\quad \text{lemma 4}\\
&\Rightarrow L\in\tilde{H}((j+1)\cdot j)^{\cdot}\quad \text{lemma 3.}
\end{aligned}
$$

Therefore $H_{\{0\}}(j)=H_{\{0\}}(j+1)$ implies $\widetilde{\mathrm{NSPACE}}(\log n)\subset\tilde{H}(j\cdot(j+1))$ which is a contradiction to lemma 1. □

We will prove lemma 1, . . . , 4 in section 3. The proofs of lemma 1, 2 and 3 and (as it could be seen already) the proof of theorem 1 are not really difficult. The central point in this paper is the proof of lemma 4.

In section 4 we will formulate and prove the hierarchy results for the classes defined by counter automata and register machines.

3. PROOFS OF THE BASIC LEMMAS

In all the proofs there is no difference at all between the nondeterministic and the deterministic case. Therefore we will always consider only the deterministic case.

Proof of lemma 1

In this proof we use two results from [4] and we denote by $\mathrm{Space}_\Sigma(L(n), m)$ the class of all languages over Σ which are accepted by Turing machines which have one input tape, one worktape, one read-only input head, one work tape head, m work tape symbols and which operate on its worktape with the tape bound $L(n)$.

In [4] it was proved that

$$H_\Sigma(k)\subset \mathrm{Space}_\Sigma(\log_2 n, 2^k)$$

and that

$$\mathrm{Space}_{\{0\}}(n)\neq\mathrm{Space}_{\{0\}}(n, m)$$

for every $m\in\mathbb{N}$.

Now let bi : $\mathbb{N}\to\{0,1\}^*$ be the bijective mapping defined by: bi $(n)=\varphi \Leftrightarrow 1\varphi$ is the binary notation of n. Let $un:\{0,1\}^*\to\mathbb{N}$ be the inverse mapping of bi.

Then

$$L\in\mathrm{Space}_{\{0,1\}}(n)\quad\Rightarrow\quad un(L)\in\mathrm{Space}_{\{0\}}(\log_2 n)$$

and

$$L\in\mathrm{Space}_{\{0\}}(\log_2 n, m)\quad\Rightarrow\quad \mathrm{bi}(L-\{\varepsilon\})\in\mathrm{Space}_{\{0,1\}}(n, 2m+1).$$

The first of these results is obvious. In order to prove the second result we define a Turing machine whose working tape is divided into two tracks. The lower one stores during the simulation the same inscription as the Turing machine M which accepts L. The upper track stores the position of the input head of M in binary notation. We need one additional symbol in order to encode the position of the work tape head.

From the above results we get immediately the following:

$$L\in\mathrm{Space}_{\{0\}}(n)\quad\Rightarrow\quad un(L)\in\widetilde{\mathrm{Space}}(\log n)$$

and

$$L\in\widetilde{\mathrm{Space}}_{\{0\}}(\log_2 n, m)\quad\Rightarrow\quad \mathrm{bi}(L-\{\varepsilon\})\in\mathrm{Space}_{\{0\}}(n, 2m+1).$$

Now we are ready to prove the lemma. Suppose there exists some $k\in\mathbb{N}$ such that $\tilde{H}(k)=\widetilde{\mathrm{Space}}(\log n)$.

Then the following implication holds:

$$
\begin{aligned}
L\in\mathrm{Space}_{\{0\}}(n)\quad&\Rightarrow\quad un(L)\in\widetilde{\mathrm{Space}}(\log n)=\tilde{H}(k)\\
&\Rightarrow\quad un(L)\in\widetilde{\mathrm{Space}}(\log_2 n, 2^k),\\
&\Rightarrow\quad L=\mathrm{bi}(un(L))\in\mathrm{Space}_{\{0\}}(n, 2^{k+2}).
\end{aligned}
$$

Therefore $\widetilde{\mathrm{Space}}(\log n)=\tilde{H}(k)$ implies

$$\mathrm{Space}_{\{0\}}(n)=\mathrm{Space}_{\{0\}}(n, 2^{k+2})$$

which is a contradiction to the result mentioned at the beginning of this proof. □

Proof of lemma 2

Let $L \in$ Space (log n) be an arbitrary chosen language and let M be a Turing machine accepting L within space bound log n.

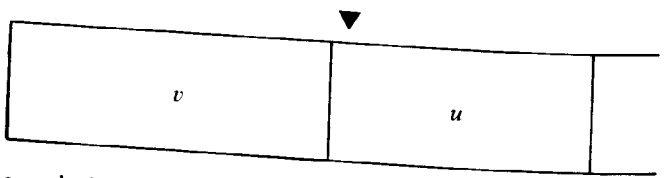
Let M' be the following modification of M:

1. M' writes bi (n), where n is the input number, on its working tape (bi is defined as in the proof of lemma 1).

2. During the rest of the computation M' never uses its input tape again. M' simulates M and during this simulation its working tape is divided into 3 tracks. On its first track M' stores bi (n), on its second track the position of the input head of M in binary notation and on its third track the inscription of the worktape of M.

Furthermore we can define M' in such a way that it has only two worktape symbols. There exists a $k \in \mathbb{N}$ such that M' uses for every computation at most $k \cdot \log_2 n$ cells on its worktape.

We will now define a 3-register machine A accepting $T_k(L)$. We apply a method which is used quite often in order to simulate Turing machines by 3-register machines. The working tape of M' is divided by the head position into two parts



and A stores during the simulation on its first two counters the numbers un (u) and un (v^R). In order to simulate one step of M' the register machine A has to divide or multiply these registers by two and to add $+1$ or -1. This can be done by using the third register.

In order to initiate this simulation A tests whether the input number is of the form $2^{k \cdot m}$, $m \in \mathbb{N}$ and sets its second counter to un (bi $(2^m)^R$) $= un$ ($00\ldots0$) $= 2^m$.

These computations can be performed by a 3-register machine.

Since M' uses at most $k \cdot \log_2 n = k \cdot m$ cells the numbers stored by the registers are bounded by $2^{k \cdot m}$ during the whole computation.

It is clear that A accepts some number $2^{k \cdot m}$ if and only if M' accepts 2^m. □

Proof of lemma 3

The result is true for $k=1$. Now suppose that $k \geqq 2$.

Let M be a j-head automaton accepting $T_k(L)$. Our $k \cdot j$-head automaton tests first whether the input is of the form O^{2^n}. It needs two heads to do this. Now suppose $\tilde{M}$ reads an input string O^{2^n}. Then it starts to simulate M. During this simulation it encodes each head position h,

$$0 \leqq h < 2^{k \cdot n}, \qquad h = l_1 + l_2 \cdot 2^n + \ldots + l_k \cdot 2^{(k-1) \cdot n}, \qquad 0 \leqq l_\nu < 2^n,$$

of M by the positions $l_1, \ldots, l_k$ of k of its heads.

Note that $h=0$ iff $l_\nu = 0$ for all $\nu = 1, \ldots, k$ and that $h = 2^{k \cdot n} + 1$ iff M moves starting from $h = 2^{k \cdot n} - 1$ ($l_\nu = 2^n - 1$, $\forall\, \nu = 1, \ldots, k$) two cells to the right. It is clear that $\tilde{M}$ can simulate the moves of the heads of M since it can test easily whether $l_\nu = 0$ or $l_\nu = 2^n - 1$. □

Proof of lemma 4

Let M be a j-head automaton accepting $T_{k+1}(L)$.

We have to construct a $(j+1)$-head automaton $\tilde{M}$ accepting $T_k(L)$.

It can be tested easily (using 2 heads) whether the input is of the form $O^{2^{k \cdot n}}$ for some $n \in \mathbb{N}$. If this is the case then $\tilde{M}$ has to test whether $O^{2^{(k+1) \cdot n}}$ is accepted by M. In order to do so $\tilde{M}$ encodes each head position h_ν, $1 \leqq \nu \leqq j$. of M:

$$0 \leqq h_\nu \leqq 2^{(k+1) \cdot n} + 1$$

by the position of its own ν-th head $\tilde{h}_\nu$:

$$0 \leqq \tilde{h}_\nu \leqq 2^{k \cdot n} + 1$$

and by an additional number σ_ν,

$$0 \leqq \sigma_\nu < 2^n$$

in such a form that always $h_\nu = \tilde{h}_\nu + \sigma_\nu \cdot 2^{k \cdot n}$.

Note that $h_\nu = 2^{(k+1) \cdot n} + 1$ if and only if $\tilde{h}_\nu = 2^{k \cdot n} + 1$ and $\sigma_\nu = 2^n - 1$.

$\tilde{M}$ uses its $(j+1)$-st head to store the j numbers $\sigma_1, \ldots, \sigma_j$ in the form

$$\tilde{h}_{j+1} = \sigma_1 + \sigma_2 \cdot 2^n + \ldots + \sigma_j 2^{(j-1) \cdot n} + 2^{(k-1) \cdot n}.$$

This is possible since $j < k$.

First $\tilde{M}$ has to encode the initial head positions of M. That means it has to set $\tilde{h}_{j+1} \leftarrow 2^{(k-1) \cdot n}$. This can be done easily (using 3 heads).

During the simulation $\tilde{M}$ always stores in its finite memory which of the σ_ν, encoded by $\tilde{h}_{j+1}$, are equal to $2^n - 1$. $\tilde{M}$ has to simulate the moves η_ν, $\eta_\nu \in \{-1, 0, +1\}$, of the j heads of M. Furthermore it has to decide which of the new σ_ν are equal to $2^n - 1$. This is simple if $0 \leqq \tilde{h}_\nu + \eta_\nu \leqq 2^{k \cdot n} + 1$. In this case $\tilde{M}$ only has to set $\tilde{h}_\nu \leftarrow \tilde{h}_\nu + \eta_\nu$. σ_ν remains unchanged.

Now suppose $\tilde{h}_\nu = 2^{k.n}+1$ and $\eta_\nu = +1$. (The case $\tilde{h}_\nu = 0$ and $\eta_\nu = -1$ leads to analogous considerations.) Then $\tilde{M}$ has to set $\tilde{h}_\nu \leftarrow 2$, $\sigma_\nu \leftarrow \sigma_\nu + 1$. (Note that $\tilde{h}_\nu = 2^{k.n}+1$ and $\eta_\nu = +1$ imply $\sigma_\nu < 2^n - 1$, since $\tilde{h}_\nu + \sigma_\nu \cdot 2^{k.n} \leqq 2^{(k+1).n}+1$.)

Performing the operation on σ_ν is the difficulty in this proof. The $\sigma_1, \ldots, \sigma_j$ will be stored always by the position of the $(j+1)$-st head but we shall rotate their sequence and we shall be able to add $+1$ when σ_ν is carried from the last position to the first position.

In order to do this we need the $(j+1)$-st head, the ν-th head and one further head (we can assume that $\nu \neq 1$ and in this case we take the first head). Note that the ν-th head does not store anything and therefore it is free for intermediate computations.

We will denote the position of the first head by $\varkappa$ and the position of the $(j+1)$-st head by λ. In the case $j < k-1$ it is favourable to introduce the new numbers $\sigma_{j+1} = \sigma_{j+2} = \ldots = \sigma_{k-1} = 0$.

$$\lambda = \sum_{\mu=1}^{k-1} \sigma_\mu \cdot 2^{(\mu-1).n} + 2^{(k-1).n}.$$

Furthermore we can assume that $\varkappa < 2^{k.n}$ ($\tilde{M}$ can test whether $\varkappa < 2^{k.n}$ by going two cells to the right. If $\varkappa \geqq 2^{k.n}$ then we set $\varkappa = 2^{k.n} - 1$ and store the difference in the finite memory.) We decompose $\varkappa$ in the form

$$\varkappa = \psi_1 + \psi_2 \cdot 2^n \qquad \text{with} \quad 0 \leqq \psi_1 < 2^n, \quad 0 \leqq \psi_2 < 2^{(k-1).n}.$$

Now we are ready to describe the rotation technique.

1. $\tilde{M}$ changes the positions of head 1 and head $(j+1)$ into

$$\varkappa = \psi_2 + 2^{(k-1).n},$$
$$\lambda = R_n(\psi_1) + \sigma_1 \; 2^n + \ldots + \sigma_{k-1} \; 2^{(k-1).n},$$

where for any $x < 2^n$ $R_n(x)$ is defined in the following way:

Let $\varphi_n(x) \in \{0,1\}^*$, $|(\varphi_n(x))| = n$, be the binary notation of x lengthened by an appropriate number of leading zeros. Then $R_n(x) < 2^n$ is that number whose binary notation of length n (again allowing leading zeros) is the reversal of $\varphi_n(x)$. Note that $R_n(R_n(x)) = x$ for all $x < 2^n$.

$\tilde{M}$ reaches the above head positions by the application of the following algorithm:

$\varkappa \leftarrow \varkappa + 2^{k.n}$

While $\lambda < 2^{k.n}$ Do

Begin $\varkappa \leftarrow \left[\frac{\varkappa}{2}\right]$ and $\alpha \leftarrow \varkappa - 2\left[\frac{\varkappa}{2}\right]$

$\lambda \leftarrow \alpha + 2\lambda$

End

$\lambda \leftarrow \lambda - 2^{k.n}$

It is clear that $\tilde{M}$ can perform this computation using its ν-th head during the realization of the division. In order to see that this algorithm is correct set $\varphi_n(\psi_1) = a_{n-1} \ldots a_0 \in \{0,1\}^*$. If we further denote by $\varphi : \mathbb{N} \to \{0\} \cup \{1\} \circ \{0,1\}^*$ the binary decomposition, then after i loops, $0 \leqq i \leqq n$, the following holds:

$$\varphi(\varkappa) = 1\,0 \ldots 0\; \varphi(\psi_2)\; a_{n-1} \ldots a_i,$$
$$\langle (k-1) \cdot n \rangle$$
$$\varphi(\lambda) = 1\,\varphi_n(\sigma_{k-1}) \ldots \varphi_n(\sigma_1)\; a_0 \ldots a_{i-1}.$$

Therefore $|(\varphi(\lambda))| = (k-1).n + i$ and this implies $\lambda < 2^{k.n}$ for $i < n$ and $\lambda \geqq 2^{k.n}$ for $i = n$. The loop is carried out exactly n times and this leads to the head positions $\varkappa$ and λ which we wanted:

2. $\tilde{M}$ changes the position of head 1 and head $(j+1)$ into

$$\varkappa = \sigma_{k-1} + \psi_2 \; 2^n,$$
$$\lambda = R_n(\psi_1) + 2^n \sigma_1 + \ldots + 2^{(k-2).n} \sigma_{k-2} + 2^{(k-1).n}.$$

First $\tilde{M}$ changes λ into

$$\lambda = \widetilde{R_n(\psi_1)} + 2^n . \sigma_1 + \ldots + 2^{(k-1).n} . \sigma_{k-1},$$

where

$$\widetilde{R_n(\psi_1)} = \begin{cases} R_n(\psi_1), & \text{if } R_n(\psi_1) \equiv 1 \bmod 2, \\ R_n(\psi_1) + 1, & \text{otherwise.} \end{cases}$$

Note that only the lowest bit of λ is changed in such a way that λ becomes an odd number. (It is stored in the finite memory whether $R_n(\psi_1)$ is odd or even.)

Afterwards $\tilde{M}$ performs the following algorithm:

While $\varkappa < 2^{k.n}$ Do

Begin $\alpha \leftarrow 2^{k.n}$-th bit of λ

If $\alpha = 1$ then $\lambda \leftarrow 2\lambda - 2^{k.n}$

If $\alpha = 0$ then $\lambda \leftarrow 2\lambda$

$\varkappa \leftarrow \alpha + 2\varkappa$

End

In order to see what is done by this algorithm we consider the decomposition

$$\lambda = \tilde{\lambda} + \alpha\, 2^{k.n-1}, \qquad \tilde{\lambda} < 2^{k.n-1}.$$

Then $\alpha = 1$ iff $2\lambda \geqq 2^{k.n}$. Furthermore the second and third statement in the block generate $\lambda \leftarrow 2\tilde{\lambda}$. As in 1 it can be seen that the while loop is carried out exactly n times. During this algorithm the number σ_{k-1} is carried over from λ to $\varkappa$ bit by bit. Therefore $\varkappa$ and λ are changed by this algorithm into

$$\varkappa = \sigma_{k-1} + 2^n \psi_2 + 2^{k.n},$$
$$\lambda = 2^n \widetilde{R_n(\psi_1)} + 2n\sigma_1 + \ldots + 2^{(k-1).n}\sigma_{k-2}.$$

We obtain the head positions which we wanted by:

(*a*) subtracting $2^{k.n}$ from $\varkappa$;

(*b*) adding $2^{k.n}$ to λ;

(*c*) dividing λ by 2 as long as the remainder is 0;

(d) changing $\widetilde{R_n(\psi_1)}$ into $R_n(\psi_1)$.

(Note that $2^{k.n}$ is given by the position of the right endmarker.)

In the following let us use a simple abbreviation.

Instead of

$$\varkappa = \alpha_0 + \psi_2\, 2^n, \qquad \lambda = \sum_{\mu=1}^{k-1} \alpha_\mu\, 2^{(\mu-1).n} + 2^{(k-1).n},$$
$$0 \leqq \alpha_\mu < 2^n, \qquad \forall \mu = 0, \ldots, k-1,$$

we write

$$(\varkappa; \lambda) = (\alpha_0; \alpha_1, \ldots, \alpha_{k-1}).$$

The application of the algorithm described in 1 and 2 induces the transition

$$(\alpha_0; \alpha_1, \ldots, \alpha_{k-1}) \to (\alpha_{k-1}; R_n(\alpha_0), \alpha_1, \ldots, \alpha_{k-2}).$$

Therefore we get by $k-\nu$ applications

$$(\psi_1; \sigma_1, \ldots, \sigma_{k-1}) \to (\sigma_{k-1}; R_n(\psi_1), \sigma_1, \ldots, \sigma_{k-2}) \to \ldots$$
$$\to (\sigma_\nu; R_n(\sigma_{\nu+1}), \ldots, R_n(\sigma_{k-1}), R_n(\psi_1), \sigma_1, \ldots, \sigma_{\nu-1}).$$

Now 1 is applied again. Since during this computation σ_ν is carried over from $\varkappa$ to λ bit by bit $\tilde{M}$ is able to add $+1$ (performing the binary addition of $+1$) and to

test whether the new σ_ν is equal to $2^n - 1$ (this is true iff all bits which are carried over are equal to one). Afterwards we apply 2 and get the head positions

$$(\sigma_{\nu-1}; R_n(\sigma_\nu + 1), R_n(\sigma_{\nu+1}), \ldots, R_n(\sigma_{k-1}), R_n(\psi_1), \sigma_1, \ldots, \sigma_{\nu-2}).$$

Since $R_n(R_n(\alpha)) = \alpha$ for all $\alpha < 2^n$ a further application of this rotation technique leads to

$$(\psi_1; \sigma_1, \ldots, \sigma_{\nu-1}, \sigma_\nu + 1, \sigma_{\nu+1}, \ldots, \sigma_{k-1}).$$

By this whole computation we get the old position of the first head again, we have changed the position of the $(j+1)$ st head in such a way that σ_ν is replaced by $\sigma_\nu + 1$ and we have tested whether the new σ_ν is equal to $2^n - 1$. This shows that $\tilde{M}$ is able to simulate M step by step. □

4. HIERARCHY RESULTS FOR COUNTER AUTOMATA AND REGISTER MACHINES

Theorem 1 leads immediately to hierarchy results for the complexity classes defined by counter automata and register machines. Since

$$R(k) \subset C_{\{0\}}(k-1) \subset H_{\{0\}}(k) \nsubseteq H_{\{0\}}(k+1) \subset C_{\{0\}}(k+1) \subset R(k+3),$$

we get immediately

$$C_{\{0\}}(k) \nsubseteq C_{\{0\}}(k+2), \quad \forall k \geqq 0$$

and

$$R(k) \nsubseteq R(k+3), \quad \forall k \geqq 1,$$

and the same results hold in the nondeterministic case.

In the following we will improve three of these four results. Let us consider first register machines. We show that for languages containing only elements of the form 2^m, $m \in \mathbb{N}$, only $k+1$ registers are necessary in order to simulate a k-head automaton.

LEMMA 5 :

$$\tilde{H}(k) \subset \tilde{R}(k+1), \qquad \widetilde{NH(k)} \subset \widetilde{NR}(k+1) \qquad \textit{for} \quad k \geqq 2.$$

Proof: The proof is the same for deterministic and for nondeterministic automata. We consider here the deterministic case. Let $L \subset \{0^{2^n};\ n \in N\}$ be some language such that there exists a k-head automaton M accepting L. We will define a $(k+1)$-register machine $\tilde{M}$.

$\tilde{M}$ tests first whether its input number is of the form 2^n by dividing the input number successively by two and storing the number of divisions by its third

register. If the input number is of the form 2^n then afterwards n is stored by register 3. M checks whether n is an even or an odd number (we will see later why this is done) and computes 2^n again afterwards.

Now suppose the input number is 2^n. Then $\tilde{M}$ simulates M. In order to do so it stores the head positions $h_1, \ldots, h_k$ of M by its first k registers in the form

$$r_\nu = \tilde{h}_\nu + 2^n, \qquad \forall \nu = 1, \ldots, k,$$

where

$$\tilde{h}_\nu = \begin{cases} h_\nu, & \text{if } 0 \leqq h_\nu \leqq 2^n, \\ 2^n, & \text{if } h_\nu = 2^n + 1. \end{cases}$$

(If $h_\nu = 2^n + 1$ then additionally a corresponding bit is stored in the finite memory of $\tilde{M}$.) Note that $0 \leqq r_\nu \leqq 2 \cdot 2^n$ holds for all $\nu = 1, \ldots, k$.

First M has to initiate this encoding by setting $r_1 = 2^{n+1}$ and $r_2 = \ldots = r_k = 2^n$. In order to simulate one step of M the register machine $\tilde{M}$ has to decide for all $\nu = 1, \ldots, k$ whether $\tilde{h}_\nu = 0$ or $\tilde{h}_\nu = 2^n$. Note that this is the case if and only if r_ν is a power of two. Therefore $\tilde{M}$ proceeds in the following way:

(*a*) It checks whether there exists $\mu \in \{1, \ldots, k\}$ such that $r_\mu \neq r_\nu$. (This can be done by means of register $k+1$.)

(*b*) Let such a μ exist and suppose that $r_\mu < r_\nu$. Then $\tilde{M}$ divides the ν-th register successively by two (using register $k+1$) as long as the remainder is zero. The number of divisions is not stored. In this way $\tilde{M}$ decides whether r_ν is a power of two. If this is the case then $r_\nu = 2^{n+1}$, since $2^n \leqq r_\nu, r_\mu \leqq 2^{n+1}$ and $r_\mu < r_\nu$. Afterwards $\tilde{M}$ multiplies the ν-th register by two as long as the content of register ν is smaller then r_μ (content of register μ). It has computed r_ν again when register ν stores for the first time a number greater than r_μ. (It is clear that in the same way $\tilde{M}$ decides whether $r_\nu = 2^n$ if $r_\mu > r_\nu$.)

(*c*) $r_1 = r_\mu$, $\forall \mu = 2, \ldots, k$.

In this case $\tilde{M}$ uses 3 of its register to divide r_1 successively by two (as long as the remainder is zero) and to store the number of divisions in its register 3. (Therefore it can recompute r_1 again.) If r_1 is a power of two, then $r_1 = 2^n$ or $r_1 = 2^{n+1}$. $\tilde{M}$ checks whether the number stored by register 3 (this is n or $n+1$) is an even or an odd number. Knowing whether n is an even or an odd number $\tilde{M}$ can decide whether $r_1 = 2^n$ or $r_1 = 2^{n+1}$. □

From lemma 5 and theorem 1 we get immediately:

THEOREM 2: *For all* $j \in \mathbb{N}$:

$$R(j) \nsubseteq R(j+2) \quad \textit{and} \quad NR(j) \nsubseteq NR(j+2).$$

Let us consider now deterministic counter automata. We will prove an result analogous to lemma 4.

LEMMA 6: *For all* $L \in$ Space (log n) *and for* $k > j \geqq 2$:

$$T_{k+1}(L) \in \tilde{C}(j-1) \quad \Rightarrow \quad T_k(L) \in \tilde{C}(j).$$

Proof: The proof is very much the same as the proof of lemma 4. Let M be a deterministic $(j-1)$ counter automaton accepting $T_{k+1}(L)$ whose counters are always bounded by the input number. We have to construct a deterministic j-counter automaton $\tilde{M}$ accepting $T_k(L)$. There will exist a constant $d \in \mathbb{N}$ such that its counters are bounded by dm if m is the input number.

Suppose the input is of the form $0^{2^{k \cdot n}}$ with some $n \in \mathbb{N}$ (this can be checked easly by M). In this case $\tilde{M}$ simulates M. It encodes the head position h and the contents c_ν, $1 \leqq \nu \leqq j-1$, of the counters of M by its own head position $\tilde{h}$ and the contents $\tilde{c}_\nu$, $1 \leqq \nu \leqq j$, of its own counters in such a way that

$$0 \leqq \tilde{h}, \ \tilde{c}_j \leqq 2^{k \cdot n},$$

$$0 \leqq \tilde{c}_\nu \leqq d \cdot 2^{k \cdot n}, \qquad \forall \nu = 1, \ldots, j-1$$

with some constant $d \in N$ (which will be determined later) and

$$h = \tilde{h} + \sigma_0 \, 2^{k \cdot n},$$

$$c_\nu = \tilde{c}_\nu + \sigma_\nu \, 2^{k \cdot n}, \qquad \forall \nu = 1, \ldots, j-1,$$

with some σ_ν,

$$0 \leqq \sigma_\nu < 2^n, \qquad \forall \nu = 0, \ldots, j-1$$

and

$$\tilde{c}_j = \sigma_0 + \sigma_1 \, 2^n + \ldots + \sigma_{j-1} \, 2^{(j-1) \cdot n} + 2^{(k-1) \cdot n}$$

hold.

This encoding already shows that this proof is slightly more difficult than the proof of lemma 4. We allow the $\tilde{c}_\nu$ to grow up to d times the length of the input. This is necessary because (in contrast to the situation of lemma 4) we can't test in each step whether the number stored by a counter is equal to $2^{k \cdot n}$. This is possible only if one of the counters is set to zero (or if the head reaches an endmarker). Therefore we must allow the counters to grow until one of them (or the head) is free. We show in the following that during such a computation the counters grow at most up to $d \cdot 2^{k \cdot n}$ for some d.

During the simulation of one step of M the automaton M has to distinguish two cases:

(*a*) The head of $\tilde{M}$ does not scan one of the endmarkers and $\tilde{c}_\nu > 0$ for all $\nu = 1, \ldots, j-1$. In this case also the head of M does not scan one of the endmarkers and also $c_\nu > 0$ for all $\nu = 1 \ldots j-1$. $\tilde{M}$ simply changes its head position and its counters in the same way as M would change its head position and its counters.

(*b*) $\tilde{h} = 0$ or $\tilde{h} = 2^{k.n}+1$ or $\tilde{c}_\nu = 0$ for some $\nu \in \{1, \ldots, j-1\}$. In this situation $\tilde{M}$ has one counter free for intermediate computations. It checks for all $\nu = 1, \ldots, j$ whether $\tilde{c}_\nu \geqq 2^{k.n}$ and if this is the case it performs $\tilde{c}_\nu \leftarrow \tilde{c}_\nu - 2^{k.n}$ and $\sigma_\nu \leftarrow \sigma_\nu + 1$ as long as $\tilde{c}_\nu \geqq 2^{k.n}$ holds. In order to change σ_ν it uses again the rotation technique described in detail in the proof of lemma 4. Note that M has one counter free and therefore it can compare the content of one of its counters with $2^{k.n}$ whenever it wants it. It is clear that M can change the $\tilde{h}$, $\tilde{c}_\nu$ and σ_ν, $0 \leqq \nu \leqq j-1$, according to the changes performed by M.

In this way $\tilde{M}$ simulates each step of M and we can construct $\tilde{M}$ in such a way that it accepts $T_k(L)$. It remains to show that there exists a constant $d \in N$ such that $\tilde{c}_\nu \leqq d \cdot 2^{k.n}$ holds during the whole simulation for all $\nu = 1, \ldots, j-1$.

Suppose (*b*) holds, that means the head of M scans one of its endmarkers or one of the counters is zero and suppose that $\tilde{c}_\nu \leqq 2^{k.n}$ for all $\nu = 1, \ldots, j-1$. (This is true when M starts its computation.) Let us estimate the number r of moves which are performed by $\tilde{M}$ before $\tilde{M}$ reaches again a configuration of the type (*b*). During this computation the moves of M are determined uniquely by its states. Suppose M has p states and s is the state in the beginning of the computation we investigate here. Then $r \leqq p$ or there exist numbers $q_0, q_1 \leqq p$ and a state t such M reaches the states

$$s \to \underset{q_0 \text{ moves}}{\ldots} \to t \to \underset{q_1 \text{ moves}}{\ldots} \to t.$$

Let $\delta_\nu^0, \delta_\nu^1, 0 \leqq \nu \leqq j-1$, be the distances by which the head position and the first $j-1$ counters of M are changed during the first q_0 moves and during the following q_1 moves, respectively. During this computation $\tilde{M}$ moves its head and its first $j-1$ counters in the same way as M does and therefore (since M halts) $\delta_0^1 \neq 0$ or there exists $\nu_0 \in \{1, \ldots, j-1\}$ such that $\delta_{\nu_0}^1 < 0$.

Set $\mu = 0$ or $\mu = \nu_0$, respectively.

[When we want to accentuate the dependence of μ, δ_ν^0, δ_ν^1 on s we will write $\mu(s)$, $\delta_\nu^0(s)$, $\delta_\nu^1(s)$.]

Therefore $\tilde{M}$ reaches again a configuration of the type (*b*) after at most

$$r = \frac{2^{k.n} + \delta_\mu^0}{\delta_\mu^1} \cdot q_1 \text{ moves.}$$

The contents of the counters $\nu \in \{1, \ldots, j-1\} - \{\mu\}$ are bounded by $2^{k.n} + \delta_\nu^0 + \delta_\nu^1 r$. And for sufficiently large n $2^{k.n} + \delta_\nu^0 + \delta_\nu^1 . r \leqq d . 2^{k.n}$ for some $d \in \mathbb{N}$.

Whenever $\tilde{M}$ reaches a configuration of the type (*b*) it changes the numbers stored by its first $j-1$ counters into numbers

$$\tilde{c}_\nu \leqq 2^{k.n} \quad \text{for all } \nu = 1, \ldots, j-1,$$

and therefore because of the above considerations

$$\tilde{c}_\nu \leqq d \cdot 2^{k.n} \quad \text{holds for all } \nu = 1, \ldots, j-1$$

during the whole computation of $\tilde{M}$.

Furthermore the counters of M are always bounded by $2^{(k+1).n}$. This implies that $\sigma_\nu < 2^n$ holds for all $\nu = 1, \ldots, j-1$ during the whole computation of $\tilde{M}$. □

THEOREM 3: *For all* $j \in N$:

$$C_{\{0\}}(j) \not\subseteq C_{\{0\}}(j+1) \quad \textit{and} \quad NC_{\{0\}}(j) \not\subseteq NC_{\{0\}}(j+2).$$

Proof: The nondeterministic case was already proved in the beginning of this section. The deterministic case follows from lemma 6 in the same way as theorem 1 follows from lemma 4. □

It should be noted that the proof of lemma 6 really uses the determinism of the automata. The argument holds also in the nondeterministic case if one is able to show that for any input string there exists a computation where the number of moves performed without reaching an endmarker is bounded in the same way as in the proof of lemma 6.

REFERENCES

1. O. H. IBARRA, *On Two-Way Multihead Automata*, J. Comp. and System Sc., Vol. 7, 1973, pp. 28-37.
2. B. MONIEN, *Transformational Methods and Their Application to Complexity Problems*, Acta Informatica, Vol. 6, 1976, pp. 95-108; *Corrigenda*, Acta Informatica, Vol. 8, 1977, pp. 383-384.
3. J. I. SEIFERAS, *Techniques for Separating Space Complexity Classes*, J. Comp. and System Sc., Vol. 14, 1977, pp. 73-99.

4. J. I. SEIFERAS, *Relating Refined Space Complexity Classes*, J. Comp. and System Sc., Vol. 14, 1977, pp. 100-129.
5. I. H. SUDBOROUGH, *Some Remarks on Multihead Automata*, R.A.I.R.O. Informatique théorique, Vol. 11, 1977, pp. 181-195.
6. A. C. YAO and R. L. RIVEST, $k+1$ *Heads are Better Than* k, J. Ass. Comp. Machinery, Vol. 25, 1978, pp. 337-340.