Oscar H. Ibarra | Juhani Karhumäki | Alexander Okhotin

On stateless multihead automata: hierarchies and the emptiness problem

Turku Centre *for* Computer Science

TUCS Technical Report
No 848, December 2007

On stateless multihead automata: hierarchies and the emptiness problem

Oscar H. Ibarra
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Juhani Karhumäki
Department of Mathematics, University of Turku, *and*
Turku Centre for Computer Science
Turku FIN–20014, Finland
karhumak@utu.fi

Alexander Okhotin
Academy of Finland, *and*
Department of Mathematics, University of Turku, *and*
Turku Centre for Computer Science
Turku FIN–20014, Finland
alexander.okhotin@utu.fi

TUCS Technical Report
No 848, December 2007

Abstract

We look at stateless multihead finite automata in their two-way and one-way, deterministic and nondeterministic variations. The transition of a k-head automaton depends solely on the symbols currently scanned by its k heads, and every such transition moves each head one cell left or right, or instructs it to stay. We show that stateless $(k+4)$-head two-way automata are more powerful than stateless k-head two-way automata. In the one-way case, we prove a tighter result: stateless $(k+1)$-head one-way automata are more powerful than stateless k-head one-way automata. Finally, we show that the emptiness problem for stateless 2-head two-way automata is undecidable.

Keywords: Multihead automata, stateless automata

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Introduction

Inspired by biologically-motivated models of computing [3, 4, 6], stateless multihead two-way finite automata and stateless multicounter machines were recently introduced by Yang, Dang and Ibarra [7]. These stateless machines are essentially one-state machines. The previous results [7] are mostly concerned with decidability/undecidability of decision problems such as emptiness and reachability. In this paper, we investigate the language accepting power of stateless multihead finite automata.

Denote two-way nondeterministic (deterministic) finite automata by 2NFA (2DFA), similarly denote their one-way variants by 1NFA (1DFA). We consider *stateless k-head 2NFAs* and define them as pairs of an alphabet Σ and a set of transitions δ. Let $¢, \$ \notin \Sigma$, be the left and right end markers. Each transition in δ is of the form $a_1 \dots a_k \to d_1 \dots d_k$, where $a_i \in \Sigma \cup \{¢, \$\}$ is the symbol scanned by i-th head, while $d_i \in \{\ell, s, r\}$ tells where each i-th head is to be moved (ℓ, s and r stand for *left*, *stay* and *right*, respectively). If there is at most one transition for every collection of symbols $a_1, \dots, a_k \in \Sigma^k$, we refer to such an automaton as a *stateless k-head 2DFA*. If none of the transitions move any heads to the left, such an automaton is called a *stateless k-head 1NFA (1DFA)*.

For an input string $w \in \Sigma^*$, machines work on a tape containing $¢w\$$ and start with all heads on the left end marker. At every step of the computation, the symbols $a_1, \dots, a_k$ currently scanned by all k heads are considered, any corresponding transition $a_1 \dots a_k \to d_1 \dots d_k \in \delta$ is chosen, and each i-th heads is moved according to d_i. If no such transition exists, the automaton rejects. If any of the heads falls off the tape, the automaton rejects as well. If the transition instructs all heads to stay, the automaton halts and accepts. The string is accepted if there exists a computation resulting in acceptance. As an example, the stateless 2-head 1DFA with instructions $¢¢ \to sr$, $¢a \to sr$, $¢b \to rr$, $ab \to rr$, and $b\$ \to ss$ recognizes the language $L = \{a^n b^{n+1} \,|\, n \geqslant 0\}$.

We shall also consider the well-known *k-head 2NFAs (2DFAs) with states*, which use a finite set of states Q, and in which transitions are quintuples $(q, a_1, \dots, a_k, q', d_1, \dots, d_k)$, with $a_i \in \Sigma \cup \{¢, \$\}$ and $d_i \in \{\ell, s, r\}$, and with $q, q' \in Q$ being the current and the next states of the automaton. The automaton starts on a tape containing $¢w\$$ with all heads over $¢$ and having an internal state $q_0 \in Q$. At every step of the computation such an automaton may apply only transitions labelled by the current state q, and along with moving the heads it enters state q'. The automaton accepts by entering a designated state $q_f \in Q$.

It is known that for both for multihead 2NFAs with states and for 2DFAs with states, $k+1$ heads are better than k heads [2]. For stateless machines, we would like to be able to show a similar result, i.e., that for $k \geqslant 1$, stateless $(k+1)$-head 2NFAs (resp., 2DFAs) are better than those with only k heads.

Although the case $k = 1$ is obvious, we are not able to give a proof for the general case at this time. Proving such a result using diagonalization (as in the case of automata with states [2]) seems quite difficult, as this would involve constructing a stateless multihead 2-NFA M that is capable of diagonalizing over all stateless k-head 2NFAs. However, it is not at all clear how M can accomplish this without states. Nevertheless, in Section 2, we show how to reduce the hierarchy problem for stateless multihead 2NFAs (resp., 2DFAs) to the hierarchy for multihead 2NFAs (resp. 2DFAs) with states. But the resulting hierarchy we obtain is not as tight, as we are only able to prove that stateless $(k + 4)$-head 2NFAs (resp., 2DFAs) are better than those with k heads.

In Section 3, we consider stateless multihead one-way machines. We show that stateless $(k + 1)$-head 1NFAs (resp., 1DFAs) are more powerful than stateless k-head 1NFAs (resp., 1DFAs), matching the known hierarchy for one-way machines with states. In Section 4, we show that the emptiness problem (deciding if the language accepted is empty) for stateless 2-head 2DFAs is undecidable, strengthening a recent result [7]. It remains an interesting open question whether this result can be shown to hold for stateless 2-head 1DFAs (or 1NFAs).

2 Hierarchy of two-way automata

We shall mainly establish hierarchies of stateless automata by simulating automata with states and using known hierarchy theorems for the latter automata. However, a rough infinite hierarchy of languages recognized by stateless multihead automata (with respect to heads) can be established directly, without using any previous work:

Proposition 2.1. *There is an infinite head-hierarchy of stateless multihead 1DFAs (resp., 1NFAs, 2NFAs, 2DFAs) over a unary alphabet.*

Proof. It is sufficient to show that for every $k \geqslant 1$, there is a language that cannot be accepted by any stateless k-head 2NFA but can be accepted by a stateless k'-head 1DFA for some $k' > k$.

For $k \geqslant 1$, define the singleton language $L_k = \{a^k\}$. L_k can be accepted by the stateless $(k + 1)$-head 1DFA with the following transitions:

$$\begin{aligned}&\text{¢}^{k+1} \to s^k r,\ \text{¢}^k a \to s^{k-1} rr,\ \text{¢}^{k-1} a^2 \to s^{k-2} rsr,\ \text{¢}^{k-2} a^3 \to s^{k-3} rssr, \ldots,\\ &\text{¢}a^k \to r s^{k-1} r,\ a^k\$ \to s^{k+1}.\end{aligned}$$

Clearly, for every k, there are at most a finite number of stateless k-head 2NFAs that we can define and, hence, only a finite number of distinct unary languages that can be accepted by such machines, and this number depends only on k. Let this number be $f(k)$. It follows that there is an $1 \leqslant i \leqslant f(k)+1$ such that L_i cannot be accepted by any stateless k-head 2NFA, but L_i can

be accepted by a stateless $(i+1)$-head 1DFA and, hence, also by a stateless $(f(k)+1)$-head 1DFA. $\square$

Let us now establish more precise separations. Our first hierarchy relies upon the following simulation:

Lemma 2.1. *Let M_1 be a k-head 2DFA (2NFA) with states, where $k \geqslant 1$. Let Σ be the input alphabet of M_1. Then there exists a stateless $(k+3)$-head 2DFA (2NFA, respectively) M_2 over $\Gamma \supset \Sigma$ and a string $x \in \Gamma^*$, such that $L(M_2) \cap x\Sigma^* = x \cdot L(M_1)$.*

Proof. Let M_1 have states $q_1, \ldots, q_n$, with initial state q_1 and unique halting/accepting state q_n. We assume that none of q_i's is in Σ. An input to M_1 is of the form $¢a_1 \ldots a_m\$$, with $m \geqslant 0$ and $a_i \in \Sigma$.

We show how to construct from M_1 a stateless $(k+3)$-head 2DFA or 2NFA M_2, which, when given $¢q_1 \ldots q_n a_1 \ldots a_n\$$, accepts if and only if M_1 accepts $¢a_1 \ldots a_n\$$, that is, the string x in the statement of the theorem is $q_1 \ldots q_n$. Given $¢q_1 \ldots q_n a_1 \ldots a_n\$$, M_2 simulates M_1 on the input $¢a_1 \ldots a_n\$$.

In the beginning, heads $k+1$ and $k+2$ stand over q_1, head $k+3$ remains at the left end marker, while heads $1, \ldots, k$ proceed to the beginning of the input. This is done by the following transitions:

$$¢^k ¢¢¢ \to r^k rrs$$
$$(q_i)^k q_1 q_1 ¢ \to r^k sss \quad (1 \leqslant i < n)$$

To simplify the notation, assume that the symbol q_n is the left end marker used by M_1 (instead of ¢). Then heads $1, \ldots, k$ assume their initial position at q_n.

Three extra heads of M_2 are used as follows. Head $k+3$ will stand either at ¢ or at q_1, thus storing a single bit, the number of steps of the simulated computation modulo 2. At the first step (as well as at every odd step), when head $k+3$ sees ¢, head $k+1$ scans the current state of M_1, while head $k+2$ is moving to the next state of M_1. At every even step, when head $k+3$ sees q_1, the roles of heads $k+1$ and $k+2$ are reversed: $k+2$ stands over the current state, while $k+1$ looks for the next state.

The behaviour at odd steps is implemented as follows. For each transition $(q_i, a_1, \ldots, a_k, q_{i'}, d_1, \ldots, d_k)$ of M_1, where $q_i, q_j \in Q$, $a_1, \ldots, a_k \in \Sigma \cup \{q_n, \$\}$ and $d_1, \ldots, d_k \in \{\ell, s, r\}$, define the following transition of M_2:

$$a_1 \ldots a_k q_i q_j ¢ \to \begin{cases} s^k srs & \text{if} \quad q_j < q_{i'} \\ s^k s\ell s & \text{if} \quad q_j > q_{i'} \\ d_1 \ldots d_k ssr & \text{if} \quad q_j = q_{i'} \end{cases}$$

That is, while head $k+2$ scans a state other than $q_{i'}$, it moves towards $q_{i'}$, while other heads wait and continue scanning their symbols. This allows us to know the exact state of M_1 during the entire movements of head $k+2$.

Once head $k+2$ reaches $q_{i'}$, the transition of M_1 is simulated in a single step of M_2, and at the same time head $k+3$ is moved from $\¢$ to q_1, thus indicating that it is head $k+2$ that currently sees the state of M_1, while head $k+1$ can be anywhere and should move towards the next state of M_1.

The behaviour at even steps of the computation of M_1 is implemented in M_2 symmetrically:

$$a_1 \dots a_k q_j q_i q_1 \to \begin{cases} s^k rss & \text{if } q_j < q_{i'} \\ s^k \ell ss & \text{if } q_j > q_{i'} \\ d_1 \dots d_k ss\ell & \text{if } q_j = q_{i'} \end{cases}$$

Finally, once M_1 enters the accepting state q_n, M_2 should accept as well, that is, and for all $q_j \in Q$ and $a_1, \dots, a_k \in \Sigma \cup \{q_n, \$\}$,

$$a_1 \dots a_k q_n q_j \¢ \to s^k sss$$
$$a_1 \dots a_k q_j q_n q_1 \to s^k sss$$

This completes the construction of M_2, which is applicable both to deterministic and nondeterministic cases. □

It is known that $(k+1)$-head 2DFAs are more powerful than k-head 2DFAs [2], and the same result holds for 2NFAs. This gives an infinite hierarchy (with respect to heads) of stateless multihead two-way DFAs.

Theorem 2.1. *For $k \geqslant 1$, stateless $(k+4)$-head 2DFAs (2NFAs) are more powerful than stateless k-head 2DFAs (2NFAs, respectively).*

Proof. Let $L \subseteq a^*$ be a language defined by Monien [2], which is accepted by a $(k+1)$-head 2DFA M_1 with states (2NFA, respectively), but cannot be accepted by any k-head 2DFA with states (2NFA, respectively). Let M_2 be the corresponding $(k+4)$-head two-way stateless machine defined in Lemma 2.1, which recognizes $L' \subseteq \Gamma^*$ with $L' \cap x\Sigma^* = xL \subseteq xa^*$.

Suppose L' can be accepted by a stateless k-head 2DFA (2NFA) M_3. We can then construct from M_3 a k-head 2DFA (2NFA) with states M_4 accepting the original language L. The input to M_4 is $\¢a^d\$$. M_4 simulates the computation of M_3 on $\¢xa^d\$$, but since x is not on its input, M_4 simulates the moves of the k heads on x in its finite-state control. Hence, L can be accepted by a k-head 2DFA (2NFA) with states. This is a contradiction, which shows that L' is a desired example. □

It is an interesting open question whether Theorem 2.1 can be made tighter. Note that if one can improve the simulation in Lemma 2.1 so that M_2 needs less than $k+3$ heads, one can do this.

Next, we show that any language accepted by a multihead 2DFA (resp., 2NFA) with states can be accepted by a stateless multihead 2DFA (resp., 2NFA) at the price of more heads. The proof is based upon the following simulation, which is similar to the one by Yang, Dang and Ibarra [7].

Lemma 2.2. *Every language accepted by a k-head 2DFA (resp., 2NFA) with n states is accepted by a stateless $(k + \lceil \log_2 n \rceil)$-head 2DFA (resp., 2NFA).*

Proof. Consider an arbitrary k-head 2DFA (resp., 2NFA) M with states $q_0, \ldots, q_{n-1}$. We construct a stateless DFA (resp., 2NFA) M' to simulate the k-head 2DFA M. The automaton M' has $k + \lceil \log_2 n \rceil$ heads: heads $1, \ldots, k$ operate exactly as the corresponding heads of M, while the additional heads $k+1, \ldots, k + \lceil \log_2(n+1) \rceil$ are used to keep track of the state. At every moment, the position of heads $k+1, \ldots, k + \lceil \log_2(n+1) \rceil$ represents a number between 0 and $n-1$ in binary notation: if head $k+i$, with $1 \leq i \leq \lceil \log_2 n \rceil$, is at the left end marker marker, we consider the i-th bit as 0, and if it is at the next symbol to the left (whether it is the first symbol of the input, or the right end marker if the input is empty), we consider this bit as 1. This number represents the index of the current state of M.

The automaton M' starts with all heads on the left end marker; the position of heads $k+1, \ldots, k + \lceil \log_2(n+1) \rceil$ represents the state q_0, that is, the initial state of M. At every step of the computation, M' simulates a single transition of M. It can see the current state of M from the symbols observed by heads $k+1, \ldots, k + \lceil \log_2(n+1) \rceil$. Then M' moves its heads $1, \ldots, k$ with all its hends on the left end marker according to the transition table of M, and at the same time moves its heads $k+1, \ldots, k + \lceil \log_2(n+1) \rceil$ to encode the next state of M. We omit the details of the simulation of the instructions of M. □

Theorem 2.2. *Stateless multihead 2DFAs (resp., 2NFAs) are equivalent to multihead 2DFAs (resp., 2NFAs) with states, which are, in turn, equivalent to* $\log n$ *space-bounded deterministic (resp., nondeterministic) Turing machines.*

Since over a unary alphabet, $(k+1)$-head 1DFAs (resp., 1NFAs) with states are better than k-head 1DFAs (resp, 1NFAs) with states [2], we again obtain, as a corollary, that there is an infinite head-hierarchy of stateless multihead 2DFAs (resp., 2NFAs) over a unary alphabet.

3 Stateless Multihead One-way Automata

We now look at stateless multihead 1DFAs (resp., 1NFAs) and show a tight hierarchy. Our starting point is the result of Rosenberg [5], who showed that the language

$$L_k = \{u_{\frac{k(k-1)}{2}} \# u_{\frac{k(k-1)}{2}-1} \# \ldots \# u_2 \# u_1 \# v_1 \# v_2 \# \ldots \# u_{\frac{k(k-1)}{2}-1} \# u_{\frac{k(k-1)}{2}} \mid u_i, v_i \in \{a,b\}^*, u_i = v_i\}$$

is recognized by a k-head 1DFA with states. Yao and Rivest [8] have further established that this language cannot be recognized by any $(k-1)$-head 1NFA with states. Using a variant of this language, we show a tight hierarchy for stateless multihead one-way automata.

Theorem 3.1. *There is a language that is accepted by a stateless k-head 1DFA that cannot be accepted by any $(k-1)$-head 1NFA with states.*

Proof. Let $m = \frac{k(k-1)}{2}$. Consider the language

$$L'_k = \{u_m \dagger_{m-1} u_{m-1} \dagger_{m-2} \dots \dagger_1 u_1 \ddagger_1 v_1 \ddagger_2 v_2 \ddagger_3 \dots \ddagger_m v_m \mid u_i, v_i \in \{a,b\}^*, u_i = v_i\}$$

over the alphabet $\Sigma_k = \{a, b, \dagger_1, \dots, \dagger_{m-1}, \ddagger_1, \dots, \ddagger_m\}$.

We construct a stateless k-head 1DFA M_1 which accepts all the strings in L'_k plus some extraneous strings not in L'_k. This is because M_1 cannot check the number, locations, and the markings (symbols different from a, b). However, as we shall see, these extraneous strings will not affect the correctness of the proof.

The construction, which is done inductively on k, is an adaptation of the method of Rosenberg [5]. While Rosenberg essentially relies on internal states, in our stateless construction the automaton is guided by the numbers attached to the markers.

Basis $k = 2$: the language $\{w\ddagger_1 w \mid w \in \{a,b\}^*\}$ is recognized by a 2-head 1-DFA with the following transitions: $¢¢ \to rs$, $a¢ \to rs$, $b¢ \to rs$, $\ddagger_1 ¢ \to rr$, $aa \to rr$, $bb \to rr$, $\$\ddagger_1 \to ss$.

Induction step. The computation proceeds as follows. At the first phase, heads are moved to their initial positions: head k goes to $\ddagger_{m-k+2}$, each head i $(2 \leqslant i \leqslant k-1)$ proceeds to $\dagger_{m-i+1}$, while head 1 stays at the start marker. At the second phase, head k moves across the substrings $v_{m-k+2}, \dots, v_m$, and as it starts from each $\ddagger_{m-i+1}$ to read v_{m-i+1}, head i simultaneously starts from $\dagger_{m-i+1}$ and reads u_{m-i+1}. Finally, at the third phase heads $1, \dots, k-1$ are moved to $\dagger_{m-k+1}$, from where the inner part of the string will be tested for membership in L'_{k-1} as claimed in the induction hypothesis. The third phase has a special form for $k = 3$.

The first phase is implemented by moving heads $2, \dots, k$ together, and once the destination of each head is reached, this head is left behind and the rest of the heads continue their movement, until k reaches its final point. This is done using following transitions:

$$\begin{aligned}
¢¢^{k-1} &\to sr^{k-1}, \\
¢\dagger_{m-1} \dots \dagger_{m-i+2} x x^{k-i} &\to s^{i-1} r r^{k-i} \quad \big(i \in \{2, \dots, k-1\},\ x \in \{\dagger_{m-i+2}, a, b\}\big), \\
¢\dagger_{m-1} \dots \dagger_{m-k+2} x &\to s^{k-1} r \quad \big(x \in \{\dagger_{m-k+1}, \dots, \dagger_1, \ddagger_1, \dots, \ddagger_{m-k+1}, a, b\}\big).
\end{aligned}$$

Note that the sequence $\dagger_{m-1} \dots \dagger_{m-i+2}$ is empty when $i = 2$. After these transitions are applied, the heads stand as follows:

The movement of heads in phase two is defined in the following way:

$$\begin{aligned}
¢\dagger_{m-1} \dots \dagger_{m-i+2} x \dagger_{m-i-1} \dots \dagger_{m-k+1} y &\to s^{i-1} r s^{k-i-1} r \\
&\quad (\text{for all } i \in \{2, \dots, k-1\} \text{ and } xy \in \{\dagger_{m-i+1}\ddagger_{m-i+1}, aa, bb\}), \\
x\dagger_{m-2} \dots \dagger_{m-k+1} y &\to r s^{k-2} r \quad (\text{for all } xy \in \{¢\ddagger_m, aa, bb\}),
\end{aligned}$$

Let us first define the third phase for the case $k = 3$. The tape contains $\text{¢}u_3\dagger_2 u_2\dagger_1 u_1\ddagger_1 v_1\ddagger_2 v_2\ddagger_3 v_3\$$, and after the second phase head 1 is over $\dagger_2$, head 2 is over $\dagger_1$ and head 3 is over \$. Now head 2 is to be moved to $\ddagger_1$, which is done by transitions $\dagger_2 x\$ \to srs$ with $x \in \{\dagger_1, a, b\}$, and then head 1 is moved to $\dagger_1$ using transitions $x\ddagger_1\$ \to rss$ with $x \in \{\dagger_2, a, b\}$. It remains to compare u_1 to v_1. Instead of applying the induction hypothesis, for $k = 3$ it is easier to implement this comparison again using transitions $xy\$ \to rrs$, for all $xy \in \{\dagger_1\ddagger_1, aa, bb\}$. Acceptance is done by $\ddagger_1\ddagger_2\$ \to sss$.

Let us now define phase three for $k \geqslant 4$. All heads should catch up with head $k-1$, which is currently over $\dagger_{m-k+1}$. The heads are moved one by one in the following order: first $k-2$, then $k-3$, and so on until head 1. The following transitions implement this:

$$\dagger_{m-1} \dots \dagger_{m-i+1} x (\dagger_{m-k+1})^{k-i-2}\$ \to s^{i-1} r s^{k-i-1}$$
$$\text{(for all } i \in \{1, \dots, k-2\} \text{ and } x \in \{\dagger_{m-i+1}, \dots, \dagger_{m-k+2}, a, b\})$$

Once the first three phases check the conditions $u_i = v_i$ for all $i \in \{m, m-1, \dots, m-k+2\}$ and put heads $1, \dots, k-1$ over $\dagger_{m-k+1}$, it remains to check the membership of the string $u_{m-k+1}\dagger_{m-k+2} \dots \dagger_1 u_1\ddagger_1 v_1 \dots \ddagger_{m-k+1} v_{m-k+1}$ in L'_{k-1}, By the induction hypothesis, there exists a $(k-1)$-head DFA recognizing this language. Let $T \subseteq (\Sigma_{k-1})^{k-1} \times \{s, r\}^{k-1}$ be its set of transitions. For every transition $c_1 \dots c_{k-1} \to d_1 \dots d_{k-1}$ in this automaton, the constructed automaton contains the transition $c'_1 \dots c'_{k-1}\$ \to d_1 \dots d_{k-1} s$, where

$$c'_i = \begin{cases} \dagger_{m-k+1}, & \text{if } c_i = \text{¢} \\ \ddagger_{m-k+2}, & \text{if } c_i = \$ \\ c_i, & \text{otherwise} \end{cases}$$

The resulting automaton recognizes L'_k.

Now suppose $L(M_1)$ is accepted by a $(k-1)$-head 1NFA M_2 with states. Then, we can construct from M_2 a $(k-1)$-head 1NFA M_3 with states accepting the original language L_k as follows: When M_3 is given $\text{¢}w\$$ (note that the $\dagger$ and $\ddagger$ markings are not in w), M_3 simulates the computation of M_2, but uses its finite-state to remember the markings and their order and insert these markings at the appropriate places for the heads to simulate. Note also that M_3 can make sure that it is only simulating the computation of M_2 on strings with valid format. Hence L can be accepted by a $(k-1)$-head 1NFA with states. This is impossible. It follows that there is a language accepted by a stateless k-head 1DFA that cannot be accepted by k-head 1NFA. □

Corollary 3.1. *Stateless k-head 1DFAs (resp, 1NFAs) are strictly more powerful than stateless $(k-1)$-head 1DFAs (resp., 1NFAs).*

Let us now recall another result by Yao and Rivest [8], who constructed a language recognized by a 2-head 1NFA with states but not recognized by a k-head 1DFA with states for any k. The following stronger statement involving stateless 1NFAs can be established:

Theorem 3.2. *There exists a language recognized by a stateless 2-head 1NFA, which is not recognized by any k-head 1DFA with states for any k.*

Proof. Yao and Rivest [8] give the following example:

$$L = \{\#w_1x_1 \dots \#w_nx_n | n \geqslant 0, w_i \in \{a,b\}^*, x_i \in \{0,1\}^*, \exists i \exists j : w_i = w_j, x_i \neq x_j\}$$

Let $\Sigma = \{\dagger, \ddagger, a, b, 0, 1, ¢, \$\}$ and consider a variant of the above language:

$$L' = \{\dagger\ddagger w_1x_1 \dots \dagger\ddagger w_nx_n | n \geqslant 0, w_i \in \{a,b\}^*, x_i \in \{0,1\}^*, \exists i \exists j : w_i = w_j, x_i \neq x_j\}$$

Let us prove that this language is also not recognized by any k-head 1DFA with states. Suppose the contrary; then, given a k-head 1DFA with states for this language, one can easily construct a k-head 1DFA for L, which contradicts the result of Yao and Rivest [8, Th.4].

Construct a stateless 2-head 1NFA that recognizes L' modulo intersection with $(\dagger\ddagger\{a,b\}^*\{0,1\}^*)^*$. In general, this automaton operates similarly to the 2-head 1NFA with states sketched by Yao and Rivest, and uses double markers to simulate a few internal states. In the beginning, head 1 nondeterministically chooses an instance of $\ddagger$, using transitions

$$\sigma ¢ \to rs \quad (\text{for all } \sigma \in \{¢, \dagger, \ddagger, 0, 1, a, b\}).$$

Next, head 1 waits over $\ddagger$, while head 2 nondeterministically chooses another instance of $\ddagger$ as follows:

$$\ddagger\sigma \to sr \quad (\text{for all } \sigma \in \{¢, \dagger, \ddagger, 0, 1, a, b\}).$$

Once head 1 scans $\ddagger$ in front of w_ix_i, while head 2 scans $\ddagger$ before w_jx_j, both heads synchronously move to the right, ensuring that $w_i = w_j$:

$$\begin{aligned} \ddagger\ddagger &\to rr, \\ aa &\to rr, \\ bb &\to rr. \end{aligned}$$

Once the symbols from x_i and x_j are encountered, the heads proceed further as long as these strings remain identical:

$$\begin{aligned} 00 &\to rr, \\ 11 &\to rr. \end{aligned}$$

If any symbols in x_i and x_j do not match, the string is immediately accepted:

$$\begin{aligned} 01 &\to ss, \\ 10 &\to ss. \end{aligned}$$

If one of these substrings is shorter than the other, then one head arrives to †, while the other still reads symbols; in this case the automaton also accepts:

$$\begin{aligned} \dagger 0 &\to ss, \\ \dagger 1 &\to ss, \\ 0\dagger &\to ss, \\ 1\dagger &\to ss. \end{aligned}$$

If x_i and x_j are identical, then both heads come to † simultaneously, and since the transition by †† is undefined, the automaton rejects.

Let L'' be the language recognized by this automaton and suppose it is recognized by a k-head 1DFA with states for some $k \geqslant 1$. Then one can construct a k-head 1DFA with states for $L'' \cap (\dagger\ddagger\{a,b\}^*\{0,1\}^*)^* = L'$, which contradicts the claim proved above. □

Next we show, that even for unary inputs, multihead 1DFAs are surprisingly powerful:

Theorem 3.3. *For every $m \geqslant 1$, the singleton language $L_m = \{\ a^{2^m-1}\ \}$ can be accepted by a stateless $(2m+1)$-head 1DFA.*

Proof. Of $2m+1$ heads used by the automaton, head 1 is the main head, and the rest of the heads form m pairs $(i, i+m)$. At the first step, heads $1, 2, \ldots, m+1$ (that is, the main head and the first head from each pairs) are moved to position 1, while heads $m+2, \ldots, 2m+1$ (second components of all pairs) remain in position 0.

Then heads $(m+1, 2m+1)$ (that is, the last pair), which are only one position apart, are moved towards the end of the string, until $m+1$ sees the end marker. From here, heads $1, 2, \ldots, m$ (the main head and the first components of all unused pairs) move synchronously with head $2m+1$, until head $2m+1$ sees the end marker. For the last pair, this will take only one step, and after that heads $1, 2, \ldots, m$ will be at position 2, heads $m+2, \ldots, 2m$ will be at the start marker, while heads $m+1$ and $2m+1$ will be at the end marker.

Then the next pair $(m, 2m)$ is taken, and the same sequence of steps is repeated. Note that the distance between these heads is now 2. The result is that heads m and $2m$ are moved to the end, while heads $1, 2, \ldots, m-1$ are moved to position 4. This is continued with the rest of the pairs, until the following configuration is reached: heads 1 and 2 are in position 2^{m-1}, head $m+2$ is in position 0, the rest of the heads are at the end marker.

From here, heads 2 and $m+2$ are moved towards the end of the string, until head 2 sees the end marker. At this point, heads 1 and $m+2$ are at the same position if and only if the length of the string is $2^m - 1$. After that head $m+2$ is moved together with head 1, and the input is accepted if and only if these two heads arrive to the end at the same time. This happens if and only if the input has length $2^m - 1$. □

4 The Emptiness Problem

It has been shown by Yang, Dang and Ibarra [7] that the emptiness problem (is the language accepted by a given machine empty?) for stateless 3-head 1DFAs is undecidable. It remains open whether this result holds for stateless 2-head 1DFAs (or 1NFAs). In this section, we show that the emptiness problem for stateless 2-head machines is undecidable if two-way movement is allowed.

Theorem 4.1. *The emptiness problem for stateless 2-head 2DFAs is undecidable, even when each head makes only one reversal on the input tape.*

The proof is by reduction from the emptiness problem for a restricted class of 2-head 1DFAs with states. Let us define this class.

Definition 4.1. *A 2-head 1DFA with states, with initial offset and with simultaneous movement of heads is a sextuple* $(\Sigma, \#, Q, q_0, \delta, q_f)$*, where* $\# \in \Sigma$ *is a designated symbol,* $q_0, q_f \in Q$ *are the initial and the accepting states,* $\delta : Q \times \Sigma \times \Sigma \to Q$ *is the transition function.*

Given an input of the form $u\#v$*, with* $u \in \Sigma^+ \setminus \{\#\}^*$ *and* $v \in \Sigma^*$*, the automaton starts in state* q_0 *with head 1 over the first symbol of* u *and head 2 over* $\#$ *in front of* v*. If the automaton is in state* q*, the first head scans* a *and the second head scans* b*, the automaton goes to state* $\delta(q, a, b)$ *and both heads are moved to the right by one square. The input is accepted if and only if the state when head 2 reaches the end of the string is* q_f*.*

In a typical case, u will be much shorter than v, and eventually head 1 will reach the marker $\#$. It will process it uniformly with the rest of the symbols, according to the transition function.

Lemma 4.1. *The emptiness problem for the class of 2-head 1DFAs with states given in Definition 4.1 is undecidable.*

Proof. Let us define a variant of the language of valid accepting computations of a Turing machine T operating over the input alphabet Γ. The configuration of T on the input $w \in \Gamma^*$ at step i using workspace s is given by a string of length s over some auxiliary alphabet Ω. Denote this string by $C_T(w, s, i)$. Then the language of computation histories is defined as

$$\begin{aligned}\mathrm{VALC}(T) = \{&C_T(w,s,0)\#C_T(w,s,1)\natural\ldots\natural C_T(w,s,n) \mid \\ &\text{at each } i\text{-th step T uses at most } s \text{ squares} \\ &C_T(w,s,n) \text{ is an accepting configuration}\}\end{aligned}$$

The exact form of C_T can be defined so that this language can be recognized by a two-head automaton as is Definition 4.1. On the other hand, $\mathrm{VALC}(T) = \varnothing$ if and only if $L(T) = \varnothing$. Since the emptiness of a Turing machine is undecidable, so is the given decision problem. □

Proof of Theorem 4.1. The proof is a reduction from the emptiness problem for the automata given in Definition 4.1.

Let $A = (\Sigma, \#, Q, q_{init}, \delta, q_f)$ be such an automaton, let $\Sigma' = \Sigma \times Q \times Q \times \{1, 2\}$. Let $w = a_1 \dots a_{m-1} \# a_{m+1} \dots a_n$ be a string given to A, let q_i $(m \leqslant i \leqslant n)$ be the state of A after 2nd head reads a_i. Then $q_m = q_{init}$. For convenience, define $q_0 = q_1 = \ldots = q_{m-1} = q_{init}$ (though head 2 never reads $a_0, \dots a_{m-1}$). Then the computation of A on w is represented by the following string over Σ':

$$x_n^{(1)} x_n^{(2)} x_{n-1}^{(1)} x_{n-1}^{(2)} \dots x_1^{(1)} x_1^{(2)}, \quad \text{with } x_i^{(j)} = (a_i, q_{i-1}, q_i, j) \tag{1}$$

Each quadruple (a_i, q_{i-1}, q_i, j) represents A with its heads 1 and 2 in positions $i - m$ and i, respectively, with symbol a_i under head 2, currently being in state q_{i-1} and about to enter state q_i. Note that the order of symbols is reversed, and each symbol of w is represented by an "odd" and an "even" symbol, which differ only in the last component.

Construct a stateless 2-head 2DFA B over Σ' to accept the language of all strings of this form corresponding to the strings accepted by A. At the first stage of the computation of B, its heads go together to the end of the input, with head 2 always being one square ahead of head 1. While travelling like this, the heads check the general form (1) of the computation. This behaviour is implemented by the following transitions:

$$¢¢ \to sr \tag{2}$$

$$¢(a, q_f, q', 1) \to rr \quad (a \in \Sigma; q, q' \in Q) \tag{3}$$

$$(a, q, q', 1)(a, q, q', 2) \to rr \quad (a \in \Sigma; q, q' \in Q) \tag{4}$$

$$(a, q', q'', 1)(b, q, q', 2) \to rr \quad (a, b \in \Sigma; q, q', q'' \in Q) \tag{5}$$

Transition (3) checks the last state for being accepting. If an odd and an even symbol in some pair have different data, then (4) will not be applicable and the input will be rejected. Similarly, if two consecutive pairs violate the sequence of states, then (5) is not applicable.

Once head 2 reaches the end marker, with head 1 lagging behind by one symbol, the heads exchange their positions using the transition

$$(a, q_{init}, q_{init}, 2)\$ \to r\ell, \tag{6}$$

and then head 1 stays over the end marker, while head 2 proceeds to the left until it encounters #:

$$\$(a, q_{init}, q_{init}, i) \to s\ell \quad (a \in \Sigma \setminus \{\#\}, i \in \{1, 2\})$$
$$\$(\#, q_{init}, q, 2) \to \ell s \quad (q \in Q)$$

At this point, head 1 is over the last symbol before \$, which should be of the form $(a, q_{init}, q_{init}, 2)$, while head 2 scans the leftmost symbol $(\#, q_{init}, q, 2)$.

At this time, both heads are reading even symbols, and they start simultaneously moving left, maintaining equal parity of the symbols they scan. This allows the transitions in this phase to be distinct from the previously defined transitions. The following transitions simulate the operation of A:

$$(b, q'', q''', i)(a, q, q', i) \to \ell\ell \quad (a, b \in \Sigma; q, q', q'', q''' \in Q; \delta(q, a, b) = q'; i \in \{1, 2\})$$

If all transitions are correct, head 2 will eventually reach the start marker, where B accepts:

$$(b, q'', q'', 2)\text{¢} \to ss \quad (b \in \Sigma; q'', q''' \in Q)$$

It remains to consider the cases when the input is ill-formed. Suppose the general form (1) is violated, that is, let the string be of the form

$$x_n^{(1)} x_n^{(2)} x_{n-1}^{(1)} x_{n-1}^{(2)} \dots x_i^{(j)} y \dots \tag{7}$$

where symbols up to $x_i^{(j)}$ are as in (1), while y is not as required. Then B eventually reaches a configuration with head 1 over $x_i^{(j)}$ and head 2 over y.

Suppose it is the alternation of even and odd symbols that has been violated. Then $x_i^{(j)} = (a_i, q_{i-1}, q, j)$ and $y = (b, q', q'', j)$, and the transition is either undefined, or it is of the form $(a, q, q', i)(b, q'', q''', i) \to \ell\ell$. In the latter case, both heads move left by one symbol and reach their previous configuration, from which they will again move over $x_i^{(j)}$ and y. Thus the computation goes into an infinite loop.

If the string prematurely ends with an odd symbol, then eventually head 1 will scan $(a, q, q', 1)$, while head 2 will scan \$. The transition (6) will not be applicable, and the input will be rejected. In this way the syntax of the input string will be checked before the simulation of A starts, and hence syntactic garbage will not cause mistakes in the simulation. □

Although we are not able to resolve at this time the question of whether or not the emptiness problem for stateless 2-head 1DFAs (or 1NFAs) is undecidable, we can show an interesting result using the following lemma.

Lemma 4.2 (Domaratzki [1]). *Let Σ be an alphabet, let $\Sigma' = \{a' \mid a \in \Sigma\}$ be its copy and define a homomorphism $h : \Sigma^* \to (\Sigma')^*$ by $h(a) = a'$ for all $a \in \Sigma^*$. Then the language $\bigcup_{w \in \Sigma^*} w \sqcup\!\sqcup h(w)$ is recognized by a stateless 2-head 1DFA.*

Theorem 4.2. *There is a fixed stateless 2-head 1DFA M_1 over a 4-letter alphabet, such that it is undecidable to determine, given a DFA M_2, whether or not $L(M_1) \cap L(M_2) = \varnothing$.*

Proof. Let $\Sigma = \{a, b\}$ and consider the twin shuffle language L_1 as in Lemma 4.2, defined over the alphabet $\{a, b, a', b'\}$. Let $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ be an instance of PCP over $\{a, b\}$, and consider the regular language

$$L_2 = \big(\bigcup_{i=1}^{m} u_i h(v_i)\big)^+.$$

The intersection $L_1 \cap L_2$ is empty if and only if this is a yes-instance. Since PCP is undecidable, this proves undecidability of the emptiness of intersection. □

It remains an interesting open question whether the emptiness problem for stateless 2-head 1DFAs (or 1NFAs) is undecidable.

References

[1] M. Domaratzki, Personal communication, August 2007.

[2] B. Monien. Two-way multihead automata over a one-letter alphabet. *RAIRO Informatique theoretique*, 14(1): 67–82, 1980.

[3] Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[4] Gh. Păun. *Membrane Computing, An Introduction.* Springer-Verlag, 2002.

[5] A. L. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10:5 (1966), 388–394.

[6] L. Yang, Z. Dang, and O. H. Ibarra. Bond computing systems: a biologically inspired and high-level dynamics model for pervasive computing. Proceedings of the 6th International Conference on Unconventional Computation (UC'07), Lecture Notes in Computer Science, 2007.

[7] L. Yang, Z. Dang, and O. H. Ibarra. On stateless automata and P systems. *Pre-Proceedings of Workshop on Automata for Cellular and Molecular Computing*, August 2007.

[8] A. C. Yao and R. L. Rivest. $k + 1$ heads are better than k. *Journal of the ACM*, 25:2 (1978), 337–340.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978–952–12–1967–2
ISSN 1239-1891