On multi-head automata with restricted nondeterminism

On Multi-head Automata with Restricted Nondeterminism

Daniel Reidenbach, Markus L. Schmid*

Department of Computer Science, Loughborough University, Loughborough, Leicestershire, LE11 3TU, United Kingdom

Abstract

In this work, we consider deterministic two-way multi-head automata, the input heads of which are nondeterministically initialised, i. e., in every computation each input head is initially located at some nondeterministically chosen position of the input word. This model serves as an instrument to investigate restricted nondeterminism of two-way multi-head automata. Our result is that, in terms of expressive power, two-way multi-head automata with nondeterminism in form of nondeterministically initialising the input heads or with restricted nondeterminism in the classical way, i. e., in every accepting computation the number of nondeterministic steps is bounded by a constant, do not yield an advantage over their completely deterministic counter-parts with the same number of input heads. We conclude this paper with a brief application of this result.

Keywords: Multi-head Automata, Restricted Nondeterminism

1. Introduction

Multi-head automata, in their one-way, two-way, deterministic and nondeterministic versions, have been intensely studied over the last decades (for a survey, see Holzer et al. [1]). They were first introduced by Rabin and Scott [2] and Rosenberg [3]. Although many results on multi-head automata have been reported since then, very basic questions still remain unsolved. One of these open problems is to determine whether or not, in the two-way case, nondeterminism is generally more powerful, i. e., whether or not the class of languages defined by two-way nondeterministic multi-head automata (2NFA) is strictly larger than the class of languages defined by two-way deterministic multi-head automata (2DFA). In other words, we ask whether we can remove the nondeterminism from an arbitrary 2NFA – compensated for, as appropriate, by enlarging its set of states and adding several input heads – without a detrimental effect on the computational power of the automaton. It is known that 2DFA and 2NFA characterise the complexity classes of deterministic logarithmic space (L) and nondeterministic logarithmic space (NL), respectively (see, e. g., Sudborough [4]). Thus, the above described problem is equivalent to the L-NL-Problem, i. e., the long-standing open question of whether or not L and NL coincide. This problem has been further narrowed down by Hartmanis [5] and Sudborough [4], such that in fact L = NL if and only if we can remove the nondeterminism from one-way nondeterministic two-head automata without changing the accepted language.

In order to gain further insights into the role of nondeterminism for a certain computational model, it is common to restrict the amount of nondeterminism (see, e. g., Fischer and Kintala [6], Kintala and Wotschke [7] and Kutrib [8]). With respect to multi-head automata, we can try to enlarge the set of languages defined by 2DFA by adding *some* amount of nondeterminism to the model of 2DFA and investigate the question of whether or not this leads to a strictly more powerful device. If such a new model really is more powerful and, in terms of expressive power, still contained in the set of 2NFA, then the L-NL-Problem is solved. If, on the other hand, we can show that our modification does not yield any advantages, then we have identified a special kind of nondeterminism that is not responsible for an increase of expressive power regarding 2DFA.

We follow this approach and introduce two-way deterministic multi-head automata, the input heads of which are nondeterministically initialised (IFA). More precisely, in every computation each input head is initially located at some nondeterministically chosen position in the input word; hence, the automaton, for each input head, guesses a position in the input word. Similarly, the first state is nondeterministically chosen from a given set of possible initial states. After this initialisation, the automaton behaves like a normal 2DFA, i. e., every transition is deterministic. This model clearly is nondeterministic, but its nondeterminism is restricted. Although it is easy to see that IFA are not more powerful than classical 2NFA, it is not obvious whether a 2NFA that, for some constant $m \in \mathbb{N}$, performs at most m nondeterministic steps in every accepting computation (2NFA_m), can simulate the special nondeterminism of initialising the input heads. This is due to the

*Corresponding author

Email addresses: D.Reidenbach@lboro.ac.uk (Daniel Reidenbach), M.Schmid@lboro.ac.uk (Markus L. Schmid)

fact that the natural way to move an input head to a nondeterministically chosen position of the input word is to move it to the right step by step, and, in *each* step, to guess whether it should be moved further on or stopped where it is. This procedure clearly requires a number of nondeterministic steps that depends on the guessed position of the input word and, thus, is not bounded by a constant. The question arises whether or not the model of IFA is more powerful than 2DFA and 2NFA$_m$. We answer this question in the negative by showing that the nondeterminism of 2NFA$_m$ and IFA can be completely removed, i. e., they can be transformed into 2DFA, without increasing their number of input heads.

2. Basic Definitions and Multi-head Automata

Let $\mathbb{N} := \{1, 2, 3, \ldots\}$. The symbol $\subseteq$ denotes the subset relation. For an arbitrary alphabet Σ, a *word* (*over* Σ) is a finite sequence of symbols from Σ, and ε stands for the *empty word*. The symbol Σ^+ denotes the set of all nonempty words over Σ, and $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two words u, v we write $u \cdot v$ or simply uv. We say that a word $v \in \Sigma^*$ is a *factor* of a word $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 \cdot v \cdot u_2$. If $u_1 = \varepsilon$ (or $u_2 = \varepsilon$), then v is a *prefix* of w (or a *suffix*, respectively). The notation $|K|$ stands for the size of a set K or the length of a word K. If we wish to refer to the symbol at a certain position in a word $w = a_1 \cdot a_2 \cdot \cdots \cdot a_n$, $a_i \in \Sigma$, $1 \leq i \leq n$, over some alphabet Σ, we use $w[i] := a_i$, $1 \leq i \leq n$. For an arbitrary class of automata models, e. g., the set DFA of deterministic finite automata, the expression "a DFA" refers to any automaton from DFA.

We assume the reader to be familiar with the concepts of automata theory and particularly multi-head automata (see, e. g., [1, 9]); thus, we define the automata models relevant to this paper just briefly. All automata considered in this paper are two-way models, i. e., their input heads can be moved in both directions. Hence, we drop the prefixes 1 and 2 used in Section 1 to distinguish between the one-way and two-way case.

A *Nondeterministic Multi-head Automaton* (denoted by NFA(k)) is a device $M := (k, Q, \Sigma, \delta, q_0, F)$, where $k \geq 1$ is the number of *input heads*, Q is a finite nonempty set of *states*, Σ is a finite nonempty alphabet of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states* and the *transition function* δ is a mapping $Q \times (\Sigma \cup \{¢, \$\})^k \to \mathcal{P}(Q \times \{-1, 0, 1\}^k)$, where $\mathcal{P}(S)$ denotes the power set of a set S. An *input* to M is any string of the form $¢w\$$, where $w \in \Sigma^*$ and the symbols $¢, \$$ (referred to as *left* and *right endmarker*, respectively) are not in Σ. Let $\delta(p, b_1, b_2, \ldots, b_k) \ni (q, m_1, m_2, \ldots, m_k)$. For each i, $1 \leq i \leq k$, we call the element b_i the *input symbol scanned by head* i and m_i the *instruction for head* i and, furthermore, we assume that $b_i = ¢$ implies $m_i \neq -1$ and $b_i = \$$ implies $m_i \neq 1$.

A *configuration* of an NFA(k) M on some input $¢w\$$ is a tuple containing a state and k positions in $¢w\$$. A configuration $c := (p, h_1, h_2, \ldots, h_k)$ can be changed into a configuration $c' := (q, h'_1, h'_2, \ldots, h'_k)$ (denoted by the relation $c \vdash_{M,w} c'$) if and only if there exists a transition $\delta(p, b_1, b_2, \ldots, b_k) \ni (q, m_1, m_2, \ldots, m_k)$ with $¢w\$[h_i] = b_i$ and $h'_i = h_i + m_i$, $1 \leq i \leq k$. To describe a *computation of M (on input $¢w\$$)* we use the reflexive and transitive closure of the relation $\vdash_{M,w}$, denoted by $\vdash^*_{M,w}$. The *initial configuration of M (on input $¢w\$$)* is the configuration $(q_0, 0, 0, \ldots, 0)$. An *accepting configuration of M (on input $¢w\$$)* is any configuration of form $(q_f, h_1, h_2, \ldots, h_k)$, $q_f \in F$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$. M accepts the word w if and only if $\widehat{c}_0 \vdash^*_{M,w} \widehat{c}_f$, where $\widehat{c}_0$ is the initial configuration, and $\widehat{c}_f$ is an accepting configuration. For any NFA(k) M, let $L(M)$ denote the set of words accepted by M.

A *Deterministic Multi-head Automaton* (denoted by DFA(k)) is an NFA(k) M, where, for every state q and for all $b_1, b_2, \ldots, b_k \in \Sigma \cup \{¢, \$\}$, $|\delta(q, b_1, b_2, \ldots, b_k)| \leq 1$. A *Nondeterministically Initialised Multi-head Automaton* (denoted by IFA(k)) is a DFA(k) M that has a set of possible initial states, denoted by I. An IFA(k) M accepts a word $w \in \Sigma^*$ if and only if $\widehat{c}_0 \vdash^*_{M,w} \widehat{c}_f$, where $\widehat{c}_f$ is some accepting configuration and $\widehat{c}_0$ is *any* configuration of form $(q, h_1, h_2, \ldots, h_k)$, where $q \in I$ and, for every i, $1 \leq i \leq k$, $0 \leq h_i \leq |w| + 1$. For every $f : \mathbb{N} \to \mathbb{N}$, an NFA($k$) that makes at most $f(|w|)$ nondeterministic moves in every *accepting computation* on input $¢w\$$ is said to have *restricted nondeterminism* and is denoted by NFA$_{f(n)}(k)$. If $f(n) = m$, for some constant $m \in \mathbb{N}$, then we write NFA$_m(k)$. For an arbitrary class of automata models A, $\mathcal{L}(A)$ refers to the set of languages accepted by some automaton in A, i. e., $\mathcal{L}(A) := \{L(M) \mid M \in A\}$.

3. The Expressive Power of IFA(k) and NFA$_m$(k)

In this section, NFA$_{f(n)}(k)$, IFA(k) and DFA(k) are compared with respect to their expressive power. First, we note that by definition, for every $k \in \mathbb{N}$, $\mathcal{L}(\text{DFA}(k)) \subseteq \mathcal{L}(\text{IFA}(k)) \subseteq \mathcal{L}(\text{NFA}(k))$. This is due to the fact that, since the unrestricted nondeterminism of NFA(k) can be used to nondeterministically initialise the input heads and to guess an initial state, an arbitrary IFA(k) can be simulated by an NFA(k) and, on the other hand, we can easily transform any DFA(k) M into an equivalent IFA(k) by aborting every computation that does not start with configuration $(q_0, 0, 0, \ldots, 0)$, i. e., the initial configuration of M.

As already stated in Section 1, $\bigcup_k \mathcal{L}(\text{DFA}(k))$ coincides with L, the class of languages that can be accepted by deterministic Turing machines working with O($\log(n)$) space, where n is the length of the input. We can show that $\bigcup_k \mathcal{L}(\text{DFA}(k)) = \bigcup_k \mathcal{L}(\text{IFA}(k))$ and $\bigcup_k \mathcal{L}(\text{DFA}(k)) = \bigcup_{k,c} \mathcal{L}(\text{NFA}_{c\log(n)}(k))$ by showing how arbitrary IFA(k) and NFA$_{c\log(n)}(k)$ can be simulated by deterministic Turing machines with O($\log(n)$) space. We sketch these simulations very briefly. A deterministic Turing machine can simulate an IFA(k) M_1 by enumerating all possible initial configurations of M_1 and, for each such configuration, it

then simulates the *deterministic* computation of M_1 starting in this initial configuration. In order to investigate all possible initial configurations, M needs to keep track of the possible initial states of M_1 as well as of the input head positions. The numbers of input heads and possible initial states are constants, whereas each input head position can be stored within $\log(n)$ space.

In order to simulate an $\mathrm{NFA}_{c\log(n)}(k)$ M_2, the Turing machine M simply enumerates all possible binary strings $\alpha \in \{0,1\}^*$, $|\alpha| = c \times \log(n)$, and, for each such string α, it simulates M_2. If, in this simulation, M_2 performs the i^{th} nondeterministic step in its computation, then M chooses the next transition according to the i^{th} bit in α. This method can only be applied if the maximal nondeterministic branching factor of M_2 is 2, but it is straightforward to change it for the general case.

It follows that an arbitrary $\mathrm{IFA}(k)$ or $\mathrm{NFA}_{f(n)}(k)$ with $f(n) = \mathrm{O}(\log(n))$ can be transformed into an $\mathrm{DFA}(k')$. However, the details of such a transformation are not provided by the above sketched simulations and the question arises whether or not this can be done without increasing the number of input heads. In the following, we shall prove that, in fact, for every $k \in \mathbb{N}$, $\mathcal{L}(\mathrm{DFA}(k)) = \mathcal{L}(\mathrm{IFA}(k)) = \mathcal{L}(\mathrm{NFA}_{f(n)}(k))$, provided that $f(n)$ is a constant. Next, we show that, for every $k, m \in \mathbb{N}$, $\mathrm{NFA}_m(k)$ can be simulated by $\mathrm{IFA}(k)$.

Lemma 1. *Let $M \in \mathrm{NFA}_m(k)$, where $k, m \in \mathbb{N}$. There exists an $\mathrm{IFA}(k)$ M' such that $L(M) = L(M')$.*

Proof. There exists an $\widehat{m} \in \mathbb{N}$, such that we can transform M into an $\mathrm{NFA}_{\widehat{m}}(k)$ $\widehat{M} := (k, \widehat{Q}, \Sigma, \widehat{\delta}, q_0, \widehat{F})$ with $L(M) = L(\widehat{M})$ and, for every state $p \in \widehat{Q}$ and for all $b_1, b_2, \ldots, b_k \in \Sigma \cup \{¢, \$\}$, $|\widehat{\delta}(p, b_1, b_2, \ldots, b_k)| \leq 2$. This can be done by substituting a transition $\delta(p, b_1, b_2, \ldots, b_k)$ with $|\delta(p, b_1, b_2, \ldots, b_k)| = l > 2$, where δ is the transition function of M, by $l-1$ transitions that have exactly two nondeterministic choices. Obviously, this requires $l-1$ new states. In the following we assume some order on the two options of a nondeterministic transition, such that we can write nondeterministic transitions as ordered tuples rather than as sets.

We shall now construct an $\mathrm{IFA}(k)$ M' with $L(M') = L(\widehat{M})$. Let $M' := (k, Q', \Sigma, \delta', I, F')$. Before we formally define M', we informally explain its behaviour. The automaton M' initially chooses one out of $2^{\widehat{m}}$ copies of the initial state q_0 of $\mathrm{NFA}_{\widehat{m}}(k)$ $\widehat{M}$. Each of these $2^{\widehat{m}}$ initial states of M' uniquely corresponds to $\widehat{m}$ nondeterministic binary guesses that may be performed in a computation of $\widehat{M}$. This is done by storing a binary sequence of length $\widehat{m}$ in the initial states of M'. After M' initially guesses one of the initial states, it simulates the computation of $\widehat{M}$. Deterministic steps are performed in exactly the same way and whenever $\widehat{M}$ nondeterministically chooses one out of two possible transitions, then M' chooses the next transition according to the first bit of the binary sequence currently stored in the state and this first bit is then removed. We shall now give the formal definitions.

The set of states is defined by $Q' := \{q^{(\alpha)} \mid q \in \widehat{Q}, \alpha \in \{0,1\}^*, |\alpha| \leq \widehat{m}\}$, the set of initial states is defined by $I := \{q_0^{(\alpha)} \mid \alpha \in \{0,1\}^*, |\alpha| = \widehat{m}\}$ and the set of accepting states is defined by $F' := \{q^{(\alpha)} \mid q \in \widehat{F}, \alpha \in \{0,1\}^*, |\alpha| \leq \widehat{m}\}$. For every deterministic transition $\widehat{\delta}(p, b_1, b_2, \ldots, b_k) = \{(q, m_1, m_2, \ldots, m_k)\}$ of $\widehat{M}$ and for every $\alpha \in \{0,1\}^*$, $|\alpha| \leq \widehat{m}$, we define

$$\delta'(p^{(\alpha)}, b_1, b_2, \ldots, b_k) := (q^{(\alpha)}, m_1, m_2, \ldots, m_k)\,.$$

For every nondeterministic transition $\widehat{\delta}(p, b_1, b_2, \ldots, b_k) = ((q_1, m_{1,1}, m_{1,2}, \ldots, m_{1,k}),\ (q_2, m_{2,1}, m_{2,2}, \ldots, m_{2,k}))$ and for every $\alpha \in \{0,1\}^*$, $|\alpha| \leq \widehat{m} - 1$, we define

$$\delta'(p^{(0\cdot\alpha)}, b_1, b_2, \ldots, b_k) := (q_1^{(\alpha)}, m_{1,1}, m_{1,2}, \ldots, m_{1,k})\,,$$
$$\delta'(p^{(1\cdot\alpha)}, b_1, b_2, \ldots, b_k) := (q_2^{(\alpha)}, m_{2,1}, m_{2,2}, \ldots, m_{2,k})\,.$$

This particularly means that if $|\widehat{\delta}(p, b_1, b_2, \ldots, b_k)| = 2$, then $\delta'(p^{(\varepsilon)}, b_1, b_2, \ldots, b_k)$ is undefined. Furthermore, in every initial state $q_0^{(\alpha)}$, M' must check whether all input heads scan the left endmarker and reject if this is not the case.

Let $q \in \widehat{F}$, $\alpha \in \{0,1\}^*$ and, for every i with $1 \leq i \leq k$, $0 \leq h_i \leq |w| + 1$. It follows directly from the definition that, on every input $¢w\$$, $\widehat{M}$ reaches $(q, h_1, h_2, \ldots, h_k)$ from $(q_0, 0, 0, \ldots, 0)$ by applying binary nondeterministic choices according to α if and only if, for some $\alpha' \in \{0,1\}^*$ with $|\alpha \cdot \alpha'| = \widehat{m}$, M' reaches $(q^{(\alpha')}, h_1, h_2, \ldots, h_k)$ from $(q_0^{(\alpha\cdot\alpha')}, 0, 0, \ldots, 0)$ on input $¢w\$$. Consequently, $L(\widehat{M}) = L(M')$. □

From Lemma 1, we can immediately conclude that, for every $k, m \in \mathbb{N}$, the class of languages described by $\mathrm{NFA}_m(k)$ is included in the class of languages given by $\mathrm{IFA}(k)$:

Theorem 2. *For every $k \in \mathbb{N}$ and $m \in \mathbb{N}$, $\mathcal{L}(\mathrm{NFA}_m(k)) \subseteq \mathcal{L}(\mathrm{IFA}(k))$.*

Before we can show our second result, i. e., $\mathrm{IFA}(k)$ can be simulated by $\mathrm{DFA}(k)$, we need to define a few more concepts. First, every $\mathrm{IFA}(k)$ can be transformed into an equivalent one that has exactly one unique accepting configuration and it halts as soon as this configuration is entered:

Definition 3. *Let $M \in \mathrm{IFA}(k)$, $k \in \mathbb{N}$, and let F be the set of accepting states of M. M is* well-formed *if and only if $F = \{q_f\}$, $(q_f, 0, 0, \ldots, 0)$ is the only possible accepting configuration that can be reached in any computation of M and no transition $\delta(q_f, b_1, b_2, \ldots, b_k)$, $b_i \in \Sigma \cup \{¢, \$\}$, $1 \leq i \leq k$, is defined.*

We observe that every $\mathrm{IFA}(k)$ can be transformed into an equivalent well-formed one by introducing a new state that serves as the only accepting state and that is not entered before all input heads have been moved to the left endmarker.

Proposition 4. *Let $M \in \mathrm{IFA}(k)$, $k \in \mathbb{N}$. Then there exists a well-formed* $\mathrm{IFA}(k)$ M' *with* $L(M) = L(M')$.

Next, we define a special configuration graph for computations of $\mathrm{IFA}(k)$, a concept that has already been introduced by Sipser in [10], where it has been applied to space-bounded Turing machines.

Definition 5. *Let M be a well-formed* $\mathrm{IFA}(k)$, $k \in \mathbb{N}$, *and let $w \in \Sigma^*$. Let $G'_{M,w} := (V'_{M,w}, E'_{M,w})$, where $V'_{M,w} := \{(q, h_1, h_2, \ldots, h_k) \mid q \in Q, 0 \leq h_i \leq |w| + 1, 1 \leq i \leq k\}$ and $E'_{M,w} := \{(c_1, c_2) \mid c_2 \vdash_{M,w} c_1\}$. The* backward configuration graph of M on w, *denoted by $G_{M,w}$, is the connected component of $G'_{M,w}$ that contains $(q_f, 0, 0, \ldots, 0)$.*

Since the vertex $(q_f, 0, 0, \ldots, 0)$ of the backward configuration graph of a well-formed $\mathrm{IFA}(k)$ M cannot have an incoming edge and since all the transitions of M are deterministic, we can conclude that the backward configuration graph is a tree rooted by $(q_f, 0, 0, \ldots, 0)$. Therefore, from now on, we shall use the term *backward configuration tree*. For arbitrary $\mathrm{IFA}(k)$ M and $w \in \Sigma^*$, the backward configuration tree can also be used to decide on the acceptance of w by M:

Proposition 6. *Let M be a well-formed* $\mathrm{IFA}(k)$, $k \in \mathbb{N}$, *and let I be the set of initial states of M. For every $w \in \Sigma^*$, $w \in L(M)$ if and only if there exists a path from $(q_f, 0, 0, \ldots, 0)$ to some vertex $(q_0, h_1, h_2, \ldots, h_k)$, $q_0 \in I$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$, in the backward configuration tree of M on w.*

We can now state our next result, i. e., for every $k \in \mathbb{N}$, every $\mathrm{IFA}(k)$ can be transformed into an equivalent $\mathrm{DFA}(k)$. We shall prove this statement by applying a technique developed by Sipser in [10] in order to prove that every space bounded deterministic Turing machine can be transformed into a halting deterministic Turing machine with the same space bound. Furthermore, this technique has also been used by Muscholl et al. [11] in order to show a similar result for deterministic tree-walking automata and by Geffert et al. [12] in order to complement deterministic two-way automata. More precisely, we show for an arbitrary $\mathrm{IFA}(k)$ M, how a $\mathrm{DFA}(k)$ M' can be constructed that, on any input $¢w\$$, searches the backward configuration tree of M on w for a path from $(q_f, 0, 0, \ldots, 0)$ to some $(q_0, h_1, h_2, \ldots, h_k)$, where q_0 is an initial state of M. It is not obvious how M' can do this, since the size of the backward configuration tree of M on w does not only depend on the constant size of M, but also on the size of the current input $¢w\$$.

Lemma 7. *Let $M \in \mathrm{IFA}(k)$, $k \in \mathbb{N}$. There exists a* $\mathrm{DFA}(k)$ M', *such that $L(M) = L(M')$.*

Proof. Let $\widehat{M} \in \mathrm{IFA}(k)$, $k \in \mathbb{N}$, be arbitrarily chosen. By Proposition 4, we can conclude that there exists a well-formed $\mathrm{IFA}(k)$ $M := (k, Q, \Sigma, \delta, I, \{q_f\})$ with $L(M) = L(\widehat{M})$. By Proposition 6, for every $w \in \Sigma^*$, we can decide on whether $w \in L(M)$ by searching the backward configuration tree of M on w for a path from $(q_f, 0, 0, \ldots, 0)$ to some vertex of form $(q_0, h_1, h_2, \ldots, h_k)$, $q_0 \in I$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$. Consequently, in order to prove the lemma, it is sufficient to show that this task can be carried out by a $\mathrm{DFA}(k)$ M' if $¢w\$$ is the input. More precisely, M' needs to perform a Depth-First-Search on the backward configuration tree of M on w starting at the root. Obviously, it is not possible to store the entire tree in the finite state control of M', as this tree grows with the input length. However, we shall see that it is possible for M' to construct the necessary parts of the tree "on-the-fly" without having to store too much information in the states. We shall explain the main idea in more detail.

For an arbitrary $w \in \Sigma^*$, let $(q, h_1, h_2, \ldots, h_k)$ be an arbitrary vertex of $G_{M,w}$. The situation that M' visits this vertex is represented in the following way: The input heads of M' scan the positions h_i, $1 \leq i \leq k$, of the input $¢w\$$ and q, the state of M, is stored in the current state of M'. In order to avoid confusion, this state q shall be called the *currently stored state*. Initially, q_f is the the currently stored state, which, according to the above mentioned interpretation of how M' visits vertices of the backward configuration tree, particularly means that the initial configuration of M' corresponds to $(q_f, 0, 0, \ldots, 0)$, i. e., the root of the backward configuration tree. Now, M' has to visit the next vertex of $G_{M,w}$ according to a Depth-First-Search traversal. Let $(p, h'_1, h'_2, \ldots, h'_k)$ be this next vertex; so there is an edge $((q, h_1, h_2, \ldots, h_k), (p, h'_1, h'_2, \ldots, h'_k))$ in the backward configuration tree, which, by definition, implies $(p, h'_1, h'_2, \ldots, h'_k) \vdash_{M,w} (q, h_1, h_2, \ldots, h_k)$. Hence, in order to move from vertex $(q, h_1, h_2, \ldots, h_k)$ to vertex $(p, h'_1, h'_2, \ldots, h'_k))$, M' must simulate a step of M, but in the opposite direction.

The main difficulty with this procedure is that, for any vertex v in $G_{M,w}$, there may be several children to visit and, thus, we have to choose one of them and, furthermore, the next time we visit v we need to know which children have already been visited to decide which one to choose next. To this end we define a *rank* for all *possible* children of a vertex in $G_{M,w}$, and an order of these ranks. To implement the Depth-First-Search, M' then enumerates all *possible* children of the currently visited vertex v with respect to their rank and visits them. As soon as the subtree rooted by some child u of v has been completely searched, we move back to v and, in order to pick the next child of v to visit, we need to know the rank of u. Obviously, for every vertex, we cannot directly store the ranks of all its children visited so far in the finite state control. However, there is exactly one transition that changes M from the child to the parent. Therefore, we interpret this transition as the rank of u, which allows us to restore the rank while moving from the child u back to the parent v. Next, we shall formally define the set of ranks and then explain their

role for the construction of M' in more detail:

$$\Gamma := \{\langle p, m_1, m_2, \ldots, m_k, q\rangle \mid p, q \in Q, m_i \in \{-1, 0, 1\}, 1 \leq i \leq k\}\,.$$

As mentioned above, a rank $\langle p, m_1, m_2, \ldots, m_k, q\rangle$ corresponds to a transition of M, i. e., the transition that changes M from state p to q and moves the input heads according to $m_1, m_2, \ldots, m_k$. Let $v := (q, h'_1, h'_2, \ldots, h'_k)$ and $u := (p, h_1, h_2, \ldots, h_k)$ be two arbitrarily chosen configurations of M on input $¢w$\$. We say that u is an *actual child of v with rank* $\langle p, m_1, m_2, \ldots, m_k, q\rangle$ if, for every i, $1 \leq i \leq k$, $m_i = h'_i - h_i$, and $\delta(p, w[h_1], w[h_2], \ldots, w[h_k]) = (q, m_1, m_2, \ldots, m_k)$. If, for every i, $1 \leq i \leq k$, $m_i = h'_i - h_i$, but $\delta(p, w[h_1], w[h_2], \ldots, w[h_k]) \neq (q, m_1, m_2, \ldots, m_k)$, then u is a *ghost child of v with rank* $\langle p, m_1, m_2, \ldots, m_k, q\rangle$. Obviously, u is an actual child of v if and only if u is also a child of v in the backward configuration tree of M on w, whereas ghost children do not exist in the backward configuration tree. However, it shall be very convenient to allow M' to visit ghost children and to interpret the backward configuration tree to contain ghost children as well. We also need an order over the set of ranks, but, as any such order is sufficient for our purpose, we simply assume that an order is given and we define a mapping $\text{next} : \Gamma \to \Gamma \cup \{0\}$, such that, for every $r \in \Gamma$ that is not the last rank in the order, $\text{next}(r)$ is the successor of r and $\text{next}(r) = 0$ if r is the last rank. Now we are ready to formalise the constructions described above.

We assume that M' visits vertex $v := (p, h_1, h_2, \ldots, h_k)$ of the backward configuration tree right now, i. e., p is the currently stored state and the input heads scan positions $h_1, h_2, \ldots, h_k$ of the input $¢w$\$. We distinguish two operational modes of M': Either M' just moved to v from its parent (*mode* 1) or it just moved back to v from one of its children (*mode* 2). In order to distinguish and change between these two different modes, M' uses an indicator implemented in the finite state control.

If M' is in mode 1, then it just moved from the parent vertex $u := (q, h'_1, h'_2, \ldots, h'_k)$ to v. We assume that when this happens, the rank $r_v := \langle p, m_1, m_2, \ldots, m_k, q\rangle$ of v is already stored in the finite state control. By consulting the transition function δ of M, M' can check whether or not $\delta(p, w[h_1], w[h_2], \ldots, w[h_k]) = (q, m_1, m_2, \ldots, m_k)$, i. e., it checks whether or not v is an actual child or a ghost child. If v is a ghost child, then M goes back to u by changing the currently stored state back to q, moving the input heads according to $m_1, m_2, \ldots, m_k$ and changing into mode 2. This is possible, since all necessary information for this step is provided by the rank r. If, on the other hand, v is an actual child, then M' stores the smallest possible rank $r_{\min}$ in the finite state control and visits the child of v with rank $r_{\min}$ while staying in mode 1.

If M' is in mode 2, then it has just been moved back to v from some child v' and we assume that the rank $r_{v'}$ of v' is stored in the finite state control. Now, if $\text{next}(r_{v'}) = 0$, then all children of v have been visited, thus, M' must go back to the parent vertex of v and stay in mode 2. Furthermore, this has to be done in a way that the rank of v is restored. Again, let $u := (q, h'_1, h'_2, \ldots, h'_k)$ be the parent vertex of v. By definition, the rank of v is $r_v := \langle p, m_1, m_2, \ldots, m_k, q\rangle$, where, for every i, $1 \leq i \leq k$, $m_i = h'_i - h_i$, and, since v is an actual child, $\delta(p, w[h_1], w[h_2], \ldots, w[h_k]) = (q, m_1, m_2, \ldots, m_k)$. Hence, all required information to restore the rank of v is provided by the transition function δ and the currently stored state p. So M' stores rank r_v in the finite state control and moves back to vertex v by changing the currently stored state to q and moving the input heads according to m_i, $1 \leq i \leq k$.

If, on the other hand, there exists a child of v with rank $\text{next}(r_{v'}) = \langle q', m'_1, m'_2, \ldots, m'_k, p\rangle$ that has not yet been visited, then $\text{next}(r_{v'})$ is stored in the finite state control and M' visits the child corresponding to rank $\text{next}(r_{v'})$. This is done by changing the currently stored state from p to q' and moving the input heads exactly in the opposite direction as given by $m'_1, m'_2, \ldots, m'_k$, i. e., for every i, $1 \leq i \leq k$, the instruction for head i is $(-m'_i)$. Furthermore, M' changes into mode 1.

In the procedure above, it can happen that the next child to visit has a rank that requires input heads to be moved to the left of the left endmarker or to the right of the right endmarker. By definition of an IFA(k), such a child can only be a ghost child, thus, we can simply ignore it and proceed with the next rank. As soon as a vertex of form $(q_0, h_1, h_2, \ldots, h_k)$, $q_0 \in I$, $0 \leq h_i \leq |w| + 1$, $1 \leq i \leq k$, is visited, M' accepts and if, in mode 2, M moves back to $(q_f, 0, 0, \ldots, 0)$ from the child with the highest rank, then M' rejects w. This proves $L(M) = L(M')$. □

From Lemma 7, we can immediately conclude the following theorem:

Theorem 8. *For every $k \in \mathbb{N}$, $\mathcal{L}(\text{IFA}(k)) \subseteq \mathcal{L}(\text{DFA}(k))$.*

Theorems 2 and 8 imply that, for every $k, m \in \mathbb{N}$, $\mathcal{L}(\text{NFA}_m(k)) \subseteq \mathcal{L}(\text{IFA}(k)) \subseteq \mathcal{L}(\text{DFA}(k))$ and, by combining this result with the fact that, by definition, for every $k, m \in \mathbb{N}$, $\mathcal{L}(\text{DFA}(k)) \subseteq \mathcal{L}(\text{NFA}_m(k))$, we obtain the following corollary:

Corollary 9. *For every $k, m \in \mathbb{N}$,*

$$\mathcal{L}(\text{NFA}_m(k)) = \mathcal{L}(\text{IFA}(k)) = \mathcal{L}(\text{DFA}(k))\,.$$

Thus, with reference to the questions addressed in Section 1, we conclude that if nondeterminism yields an actual advantage, in terms of the expressive power of two-way multi-head automata, then this nondeterminism must be unrestricted. The proof of this insight is facilitated by the use of IFA(k), which, in contrast to $\text{NFA}_m(k)$, provide the neat property of initially performing only one nondeterministic step followed by a completely deterministic computation.

We shall conclude this work by a brief application of the results above. To this end, we consider so-called pattern

languages, which have first been introduced by Angluin in [13] (for a survey see [14]). A *pattern* is a string comprising variables and terminal symbols. The *pattern language* of a pattern α is the set of all words that can be obtained by uniformly substituting the variables in α by terminal strings. For example, the pattern $\beta := x_1 \cdot \mathtt{a} \cdot \mathtt{b} \cdot x_2 \cdot x_1$ (where x_1, x_2 are variables and $\mathtt{a}$, $\mathtt{b}$ are terminal symbols) describes the set of all words w that, for some words u and v, can be factorised into $w = u \cdot \mathtt{a} \cdot \mathtt{b} \cdot v \cdot u$. Hence, words of the pattern language given by β can be recognised by guessing a factorisation of the word and checking it for the above described properties.

We can implement this approach using IFA(k) and state without proof that every pattern language given by some pattern containing k different types of variables can be accepted by an IFA($2(k+1)$). Naturally, the initial nondeterminism of the IFA($2(k+1)$) is used to guess a factorisation of the input word, which is then validated in the deterministic computation that follows. Other approaches to the membership problem for pattern languages discussed in the literature also make use of methods that are intrinsically nondeterministic. In contrast to this, by applying the results from above, we can conclude that for every pattern α the corresponding pattern language can be accepted by a *deterministic* two-way multi-head automaton with a number of input heads linear in the number of variables of α.

Acknowledgements

The authors wish to thank the anonymous referees for their detailed and helpful remarks and suggestions, which significantly strengthened the results and improved the readability of this paper.

[1] M. Holzer, M. Kutrib, A. Malcher, Complexity of multi-head finite automata: Origins and directions, Theoretical Computer Science 412 (2011) 83–96.

[2] M. O. Rabin, D. Scott, Finite automata and their decision problems, IBM Journal of Research and Development 3.

[3] A. L. Rosenberg, On multi-head finite automata, IBM Journal of Research and Development 10.

[4] I. Sudborough, On tape-bounded complexity classes and multi-head finite automata, Journal of Computer and System Sciences 10 (1975) 62–76.

[5] J. Hartmanis, On non-determinancy in simple computing devices, Acta Informatica 1 (1972) 336–344.

[6] P. C. Fischer, C. M. R. Kintala, Real-time computations with restricted nondeterminism, Mathematical Systems Theory 12 (1979) 219–231.

[7] C. M. R. Kintala, D. Wotschke, Amounts of nondeterminism in finite automata, Acta Informatica 13 (1980) 199–204.

[8] M. Kutrib, Refining nondeterminism below linear time, Journal of Automata, Languages and Combinatorics 7 (2002) 533–547.

[9] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2000.

[10] M. Sipser, Halting space-bounded computations, Theoretical Computer Science 10 (1980) 335–338.

[11] A. Muscholl, M. Samuelides, L. Segoufin, Complementing deterministic tree-walking automata, Information Processing Letters 99 (2006) 33–39.

[12] V. Geffert, C. Mereghetti, G. Pighizzini, Complementing two-way finite automata, Information and Computation 205 (2007) 1173–1187.

[13] D. Angluin, Finding patterns common to a set of strings, in: Proc. 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 130–141.

[14] A. Mateescu, A. Salomaa, Patterns, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 1, Springer, 1997, pp. 230–242.