Restarting Automata with Restricted Utilization of Auxiliary Symbols*

Tomasz Jurdziński[1] and **Friedrich Otto**[2]

[1] Institute of Computer Science, University of Wrocław
51-151 Wrocław, Poland
tju@ii.uni.wroc.pl

[2] Fachbereich Mathematik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Abstract. The restarting automaton is a restricted model of computation that was introduced by Jančar et al. to model the so-called *analysis by reduction*, which is a technique used in linguistics to analyze sentences of natural languages. The most general models of restarting automata make use of auxiliary symbols in their rewrite operations, although this ability does not directly correspond to any aspect of the analysis by reduction. Here we put restrictions on the way in which restarting automata use auxiliary symbols, and we investigate the influence of these restrictions on their expressive power. In fact, we consider two types of restrictions. First, we consider the *number of auxiliary symbols* in the tape alphabet of a restarting automaton as a measure of its descriptional complexity. Secondly, we consider the *number of occurrences of auxiliary symbols* on the tape as a dynamic complexity measure. We establish some lower and upper bounds with respect to these complexity measures concerning the ability of restarting automata to recognize the (deterministic) context-free languages and some of their subclasses.

1 Introduction

The restarting automaton was introduced by Jančar et al. as a formal tool to model the *analysis by reduction*, which is a technique used in linguistics to analyze sentences of natural languages [3]. It consists of a stepwise simplification of a given sentence so that the (in)correctness of the sentence is not affected. It is applied primarily in languages that have a free word-order. Already several programs used in Czech and German (corpus) linguistics are based on the idea of restarting automata [13, 17].

A (two-way) restarting automaton, RLWW-automaton for short, is a device M that consists of a finite-state control, a flexible tape containing a word delimited by sentinels, and a read-write window of a fixed size. This window is moved

* This work was supported by a grant from the Deutsche Forschungsgemeinschaft. It was performed while Tomasz Jurdziński was visiting the University of Kassel.

along the tape by move-right and move-left operations until the control decides (nondeterministically) that the content of the window should be rewritten by some *shorter* string. In fact, the new string may contain auxiliary symbols that do not belong to the input alphabet. After the rewrite operation, M can continue to move its window until it either halts and accepts, or halts and rejects, or restarts, that is, it places its window over the left end of the tape, reenters the initial state, and continues with the computation. Thus, each computation of M can be described through a sequence of cycles.

Also various restricted versions of the restarting automaton have been considered. A *one-way restarting automaton*, RRWW-automaton for short, does not use any move-left operations. If, in addition, it is required to perform a restart step immediately after executing a rewrite operation, then it is an RWW-automaton. An RLWW-automaton which does not use any auxiliary symbols is called an RLW-automaton. If, in addition, each rewrite operation simply deletes some letters from the read-write window, then we have an RL-automaton. Similarly, RRW-, RR-, RW-, and R-automata are obtained, as well as the deterministic variants of all these models.

Many well-known classes of formal languages have been characterized in terms of restricted variants of the restarting automaton. For example, the deterministic R(R)WW-automaton characterizes the class of Church-Rosser languages [11, 12] of McNaughton et al. [10], the weakly monotone R(R)WW-automaton characterizes the class GCSL of growing context-sensitive languages [6] considered by Dahlhaus and Warmuth [1], the monotone R(R)WW-automaton characterizes the class CFL of context-free languages [4], and various types of deterministic monotone R(R)WW-automata characterize the class DCFL of deterministic context-free languages [4]. A restarting automaton is called *monotone* if the distance between the rewrite position and the right sentinel does not increase from one cycle to the next.

Here we place some restrictions on the way in which restarting automata make use of auxiliary symbols. This direction of research is motivated by the fact that originally the analysis by reduction does not involve the use of auxiliary symbols. On the other hand, the expressive power of restarting automata without auxiliary symbols is relatively weak, as not even all context-free languages can be recognized by them [4]. Thus, we introduce an intermediate level, at which auxiliary symbols can be used only in a restricted way. Actually, we consider two types of restriction. First we consider the number of auxiliary symbols in the tape alphabet as a measure of the *descriptional complexity* of the restarting automaton, and secondly we interpret the number of occurrences of auxiliary symbols on the tape as a *dynamic* complexity measure. As seen above many 'classical' classes of formal languages can be characterized by restricted variants of the rstarting automaton. Here we concentrate on the context-free languages and some of their subclasses establishing upper and lower bounds for them with respect to our new complexity measures concerning the utilization of auxiliary symbols.

In Section 2 we give the necessary definitions in short. In Section 3 we investigate the expressive power of deterministic restarting automata that use auxiliary symbols in a restricted way only. We establish in particular a lower bound for the number of occurrences of auxiliary symbols on the tape that are needed by deterministic RLWW-automata to accept certain context-free languages. The proof of this lower bound result, which is based on Kolmogorov complexity, is technically quite involved. Therefore, it is postponed to a separate section (Section 6). In Section 4, we study how many auxiliary symbols (in the alphabet or on the tape) are needed by nondeterministic RWW-automata to accept any context-free language, and we show that all k-linear languages ($k \geq 2$) are accepted by RLWW-automata with only two occurrences of a single auxiliary symbol. In the concluding section a number of open problems related to our work are presented.

2 Definitions

Throughout the paper ε will denote the empty word, and 2^S will denote the power set of a set S.

We now describe in short the type of restarting automaton we will be dealing with. More details can be found in [14].

A *two-way restarting automaton*, RLWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \mathbb{¢}, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, $\mathbb{¢}, \$ \notin \Gamma$ are symbols that serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read-write window*, and

$$\delta : Q \times \mathcal{PC}^{(k)} \to 2^{((Q \times (\{\mathsf{MVR},\mathsf{MVL}\} \cup \mathcal{PC}^{\leq (k-1)})) \cup \{\mathsf{RESTART},\mathsf{ACCEPT}\})}$$

is the *transition relation*. Here $\mathcal{PC}^{(k)}$ is the set of *possible contents* of the read-write window of M, where

$$\mathcal{PC}^{(i)} := (\mathbb{¢} \cdot \Gamma^{i-1}) \cup \Gamma^i \cup (\Gamma^{\leq i-1} \cdot \$) \cup (\mathbb{¢} \cdot \Gamma^{\leq i-2} \cdot \$) \quad (i \geq 0),$$

and

$$\mathcal{PC}^{\leq (k-1)} := \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

The transition relation describes five different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \neq \$$. If M is in state q and sees the string u in its read-write window, then this move-right step causes M to shift the read-write window one position to the right and to enter state q'. However, if the content u of the read-write window is only the symbol \$, then no shift to the right is possible.

2. A *move-left step* is of the form $(q', \mathsf{MVL}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$ does not start with the symbol $\mathbb{c}$. It causes M to shift the read-write window one position to the left and to enter state q'.
3. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{\leq (k-1)}$ such that $|v| < |u|$. It causes M to replace the content u of the read-write window by the string v, thereby shortening the tape, and to enter state q'. Further, the read-write window is placed immediately to the right of the string v. However, some additional restrictions apply in that the border markers $\mathbb{c}$ and $\$$ must not disappear from the tape nor that new occurrences of these markers are created. Further, the read-write window must not move across the right border marker $\$$, that is, if the string u ends in $\$$, then so does the string v, and after performing the rewrite operation, the read-write window is placed on the $\$$-symbol.
4. A *restart step* is of the form $\mathsf{RESTART} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to place its read-write window over the left end of the tape, so that the first symbol it sees is the left border marker $\mathbb{c}$, and to reenter the initial state q_0.
5. An *accept step* is of the form $\mathsf{ACCEPT} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

If $\delta(q, u) = \emptyset$ for some pair (q, u), then M necessarily halts, and we say that M *rejects* in this situation. Further, the transition relation must satisfy the additional requirement that rewrite steps and restart steps alternate in each computation of M, with a rewrite step coming first.

A *configuration* of M is a string $\alpha q \beta$ where $q \in Q$, and either $\alpha = \varepsilon$ and $\beta \in \{\mathbb{c}\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{\mathbb{c}\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 \mathbb{c} w \$$, where $w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0 \mathbb{c} w \$$ is an *initial configuration*. Thus, initial configurations are a particular form of restarting configurations.

In general, the automaton M is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side (q, u). If this is not the case, the automaton is *deterministic*.

Each computation of a two-way restarting automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the window moves along the tape performing MVR and MVL operations and a single rewrite operation until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. During a tail at most one rewrite operation may be executed.

We use the notation $u \vdash_M^c v$ to denote a cycle of M that begins with the restarting configuration $q_0 \mathbb{c} u \$$ and ends with the restarting configuration $q_0 \mathbb{c} v \$$; the relation $\vdash_M^{c^*}$ is the reflexive and transitive closure of $\vdash_M^c$. The relation $\vdash_M^c$ can be seen as the *single-step rewrite relation* induced by M, and $\vdash_M^{c^*}$ is the corresponding *rewrite relation*.

An input $w \in \Sigma^*$ is *accepted by* M, if there is a computation which, starting with the initial configuration $q_0 ¢ w \$$, finishes by executing an accept instruction. By $L(M)$ we denote the language consisting of all words accepted by M; we say that M *recognizes (accepts) the language* $L(M)$.

Various subclasses of RLWW-automta have been studied. They are obtained by combining two types of restrictions:

(a) Restrictions on the movement of the read-write window (expressed by the first part of the class name): RL- denotes no restriction, RR- denotes no MVL operations, R- denotes no MVL operations and each rewrite step is followed immediately by a restart step.
(b) Restrictions on the Rewrite instructions (expressed by the second part of the class name): -WW denotes no restriction, -W denotes no auxiliary symbols are available (that is, $\Gamma = \Sigma$), -ε denotes no auxiliary symbols are available and each rewrite step is simply a deletion (that is, if $(q', v) \in \delta(q, u)$ is a rewrite instruction of M, then v is obtained from u by deleting some symbols).

For example, RRW-automata do not use MVL instructions and they do not have auxiliary symbols.

By det-RLWW we denote the class of *deterministic* RLWW-automata, and analogously for the other types of restarting automata. Further, for each type X of automata, we denote the class of languages that are accepted by automata from that class by $\mathcal{L}(\mathsf{X})$.

Next we turn to the notion of monotonicity for restarting automata. As pointed out before, each computation of a restarting automaton proceeds in cycles, where each cycle contains exactly one application of a rewrite operation. Thus, each cycle C contains a unique configuration $\alpha q \beta$ in which a rewrite instruction is applied. The number $|\beta|$ is called the *right distance* of C, denoted by $D_r(C)$. We say that a *sequence of cycles* $S = (C_1, C_2, \cdots, C_n)$ is *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$. Now a computation of a restarting automaton M is called *monotone* if the corresponding sequence of cycles is monotone, and the restarting automaton M is *monotone* if all its computations that start with an initial configuration are monotone.

Throughout the paper we will describe the behaviour of various restarting automata in detail. Instead of giving the corresponding transition relations, we will make use of the fact that the behaviour of an RLWW-automaton M can be described transparently by a finite set of so-called *meta-instructions* of the form $(E_1, u \rightarrow v, E_2)$ and (E, Accept), where E_1, E_2, and E are regular languages, which are called the *regular constraints* of the meta-instruction, and u and v are strings such that $u \rightarrow v$ stands for a Rewrite step of M. In a restarting configuration $q_0 ¢ w \$$, M nondeterministically chooses a meta-instruction. If $(E_1, u \rightarrow v, E_2)$ is chosen, then M halts and rejects, if w does not admit a factorization of the form $w = w_1 u w_2$ such that $¢ w_1 \in E_1$ and $w_2 \$ \in E_2$. Otherwise, one such factorization is chosen nondeterministically, and $q_0 ¢ w \$$ is transformed

into the restarting configuration $q_0 ¢ w_1 v w_2 \$$. If (E, Accept) is chosen, then M halts and accepts, if $¢w\$ \in E$, otherwise, M halts and rejects.

Similarly, the behaviour of an RWW-automaton M can be described through a finite sequence of *meta-instructions* of the form $(E, u \to v)$ and (E, Accept), where E is a regular language, and u and v are strings such that $u \to v$ stands for a Rewrite step of M. On trying to execute the meta-instruction $(E, u \to v)$, M will get stuck (and so reject) starting from the restarting configuration $q_0 ¢ w \$$, if w does not admit a factorization of the form $w = w_1 u w_2$ such that $¢w_1 \in E$. On the other hand, if w does admit a factorization of this form, then one such factorization is chosen nondeterministically, and $q_0 ¢ w \$$ is transformed into the restarting configuration $q_0 ¢ w_1 v w_2 \$$. On trying to execute the meta-instruction (E, Accept), M accepts if the tape content belongs to the language E and rejects otherwise.

Finally, we define new complexity measures for restarting automata with auxiliary symbols. For each type X of restarting automata with auxiliary symbols, and integers $i, j \in \mathbb{N}$, aux(j,i)-X, a-aux(j,i)-X, and g-aux(j,i)-X denote the class of restarting automata M of type X for which the number of auxiliary symbols in the tape alphabet does not exceed the number i and,

- for aux: the number of occurrences of auxiliary symbols in any configuration during *any* computation of M starting from an initial configuration is not larger than j;
- for a-aux: the number of occurrences of auxiliary symbols in any configuration during *any accepting* computation of M starting from an initial configuration is not larger than j;
- for g-aux: for each $x \in L(M)$, there exists an accepting computation of M such that the number of occurrences of auxiliary symbols in any configuration during that computation is not larger than j.

In some cases we may replace the constant j by a non-constant function, which is used to measure the number of occurrences of auxiliary symbols on the tape as a function of the length of the input.

As our main interest concerns those classes with only a single auxiliary symbol in the tape alphabet, we introduce the notation aux(j)-X as a shorthand for aux$(j,1)$-X.

Observe that, for each type $\mathsf{X} \in \{\mathsf{RL}, \mathsf{RR}, \mathsf{R}, \mathsf{det\text{-}RL}, \mathsf{det\text{-}RR}, \mathsf{det\text{-}R}\}$, XW and aux$(0)$-XWW denote essentially the same class of automata.

Proposition 1.
The following relationships hold for each $i \in \mathbb{N}$ and each function $j : \mathbb{N} \to \mathbb{N}$:

1. $\mathcal{L}(\mathsf{a\text{-}aux}(j,i)\text{-}\mathsf{det\text{-}XWW}) = \mathcal{L}(\mathsf{g\text{-}aux}(j,i)\text{-}\mathsf{det\text{-}XWW})$ *for all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RL}\}$.
2. $\mathcal{L}(\mathsf{Y\text{-}aux}(j,i)\text{-}\mathsf{RLWW}) = \mathcal{L}(\mathsf{Y\text{-}aux}(j,i)\text{-}\mathsf{RRWW})$ *for each* $\mathsf{Y} \in \{\mathsf{a\text{-}}, \mathsf{g\text{-}}\}$.
3. $\mathcal{L}(\mathsf{aux}(j,i)\text{-}\mathsf{RLWW}) \subseteq \mathcal{L}(\mathsf{aux}(2j,i)\text{-}\mathsf{RRWW})$.

Proof. (a) This statement immediately follows from the fact that a deterministic restarting automaton has only a single computation for each input word.

(b) For each RLWW-automaton M, there exists an RRWW-automaton M' such that M and M' use the same tape alphabet, they recognize the same language, and in each accepting computation M' executes exactly the same rewrite steps as M does in the corresponding computation [16]. More precisely, in each cycle of a computation M' guesses crossing tables for M and simultaneously verifies that its guesses are correct. In the affirmative M' has successfully simulated the corresponding cycle of M; otherwise M' has made a mistake, and therefore it terminates the simulation and halts without accepting. Thus, as long as M' simulates the computation of M correctly, both automata will always have the same number of occurrences of auxiliary symbols on their tapes.

(c) When M' makes an incorrect guess, then this can result in the introduction of at most j additional occurrences of auxiliary symbols, as M' only applies rewrite operations of M. Hence, in this case M' may have up to $2j$ occurrences of auxiliary symbols on its tape. □

We close this section with some additional notation.

Notation For a word x, we denote by $x[i]$ $(0 < i \leq |x|)$ the i-th symbol of x, and $x[i,j]$ denotes the factor $x[i] \dots x[j]$ for $0 < i \leq j \leq |x|$. Further, for all non-negative integers $i \leq j$, $[i,j] := \{\, l \in \mathbb{N} \mid i \leq l \leq j \,\}$. Throughout the paper we use LIN, DCFL, CFL, CRL, GCSL to denote the class of linear, deterministic context-free, context-free, Church-Rosser, and growing context-sensitive languages, respectively.

3 Deterministic Restarting Automata

In [18] a non-context-free language L_{lr} is presented such that $L_{lr} \in \mathcal{L}(\mathsf{det\text{-}RW})$. On the other hand, there exist context-free languages which are not even recognized by RRW-automata [4]. Thus, we have the following results.

Corollary 1.

(a) $\mathsf{DCFL} \subsetneq \mathcal{L}(\mathsf{aux}(0)\text{-}\mathsf{det\text{-}RWW})$.
(b) *The classes* CFL *and* $\mathcal{L}(\mathsf{aux}(0)\text{-}\mathsf{det\text{-}RWW})$ *are incomparable under inclusion.*

It is known that auxiliary symbols increase the expressive power of deterministic RWW-automata [4]. Here, we show that already a single occurrence of a single auxiliary symbol has that effect.

Proposition 2.
The language $L_{pow} := \{\, a^{2^n} \mid n \in \mathbb{N} \,\}$ *belongs to the class* $\mathcal{L}(\mathsf{aux}(1)\text{-}\mathsf{det\text{-}RWW})$.

Proof. Let $M = (Q, \{a\}, \{a, A\}, ¢, \$, q_0, 5, \delta)$ be the RWW-automaton that is described by the following set of meta-instructions:

$$
\begin{array}{l}
(1)\ (¢ \cdot \{a, a^2, a^4\} \cdot \$, \mathsf{Accept}),\\
(2)\ (¢ \cdot a^+, a^4\$ \to Aaa\$),\\
(3)\ (¢ \cdot a^+, a^4A \to Aa^2),\\
(4)\ (¢, a^4A \to a^2).
\end{array}
$$

It follows easily that M is deterministic, that $L(M) = L_{pow}$, and that no configuration of M that is reachable from an initial configuration ever contains more than a single occurrence of the auxiliary symbol A. □

Using the pumping lemma for restarting automata [16], it can be shown easily that L_{pow} is not accepted by any RLW-automaton. Thus, we obtain the following proper inclusions.

Corollary 2. *For each type* $\mathsf{X} \in \{\mathsf{det\text{-}R(R)WW}, \mathsf{det\text{-}RLWW}, \mathsf{R(R)WW}, \mathsf{RLWW}\}$,

$$\mathcal{L}(\mathsf{aux}(0)\text{-}\mathsf{X}) \subsetneq \mathcal{L}(\mathsf{aux}(1)\text{-}\mathsf{X}).$$

As shown in [14] (Section 5), det-RL-automata even accept some languages that are not growing context-sensitive. Hence, we see that the language class $\mathcal{L}(\mathsf{aux}(0)\text{-}\mathsf{det\text{-}RLWW})$ is not included in the class GCSL of growing context-sensitive languages. As GCSL includes the class of Church-Rosser languages, which coincides with the class $\mathcal{L}(\mathsf{det\text{-}RRWW})$, we obtain the following consequences.

Corollary 3. *For each* $i \in \mathbb{N}_+$ *and each function* $j : \mathbb{N} \to \mathbb{N}$,

$$\mathcal{L}(\mathsf{aux}(0)\text{-}\mathsf{det\text{-}RLWW}) \not\subset \mathcal{L}(\mathsf{aux}(j,i)\text{-}\mathsf{det\text{-}RRWW}) \subsetneq \mathcal{L}(\mathsf{aux}(j,i)\text{-}\mathsf{det\text{-}RLWW}).$$

Currently we do not know whether all context-free languages can be accepted by det-RLWW-automata. However, we can at least show that this is impossible when the number of occurrences of auxiliary symbols is restricted too much.

Proposition 3. *The language* $L_{pal2} := \{\, ww^R vv^R \mid w, v \in \{0,1\}^* \,\}$ *is not accepted by any deterministic* RLWW*-automaton that uses only* $o(n/\log^5 n)$ *occurrences of auxiliary symbols.*

The proof of this proposition, which is based on Kolmogorov complexity, is quite involved. Therefore it is postponed to Section 6. This result yields the following lower bound result.

Corollary 4. *If* CFL *is contained in the language class* $\mathcal{L}(\mathsf{aux}(j,i)\text{-}\mathsf{det\text{-}RLWW})$ *for some function* j *and some integer* i*, then* $j(n) \notin o(n/\log^5 n)$.

It is currently not known whether the deterministic RLWW-automaton is at all less expressive than the nondeterministic RLWW-automaton. However, as the language L_{pal2} is 2-linear, and as the class of 2-linear languages is included in aux(1)-RLWW (see Theorem 4), we have at least the following separation result.

Corollary 5. *For each function* $j(n) \in o(n/\log^5 n)$ *and each integer* $i > 0$,

$$\mathcal{L}(\mathsf{aux}(j,i)\text{-}\mathsf{det\text{-}RLWW}) \subsetneq \mathcal{L}(\mathsf{aux}(j,i)\text{-}\mathsf{RLWW}).$$

4 Nondeterministic Restarting Automata

Here we investigate the complexity of context-free languages with respect to the number of auxiliary symbols used. As R-automata can accept some languages that are not even growing context-sensitive [8], while some context-free languages cannot be accepted by RRW-automata [4], we have the following basic fact.

Corollary 6. *The language classes* $\mathcal{L}(\mathsf{aux}(0)\text{-}\mathsf{X})$ *and* CFL *are incomparable under set inclusion for each type* $\mathsf{X} \in \{\mathsf{RLWW}, \mathsf{RRWW}, \mathsf{RWW}\}$.

However, each context-free language can be accepted by an RWW-automaton that has only a single auxiliary letter.

Theorem 1. CFL *is included in* $\mathcal{L}(\mathsf{aux}(n,1)\text{-}\mathsf{RWW})$.

Proof. Let G be a context-free grammar in Chomsky normal form with the set N of nonterminals, let $m := |N|$, and let $\hat{L}$ be the set of all sentential forms that can be derived in G.

For each $\alpha \in \hat{L}$, we consider a derivation tree for α. If α is sufficiently long, then there exists a subtree with at least $4m$ (and at most $8m$) leaves. The RWW-automaton guesses a subword of α which corresponds to such a subtree and replaces it by the encoding of the nonterminal appearing at the root of that subtree.

In order to use this technique when there is only one auxiliary symbol in the alphabet, we encode the i-th nonterminal of G by Aa^iA, where A is the only auxiliary symbol of the RWW-automaton considered and a is a fixed terminal symbol. As each rewrite step shortens the sentential form by at least $4m-1$ symbols, the rewrite steps remain length-reducing even when the above encoding for nonterminals of G is being used. □

If only the accepting computations with the smallest number of occurrences of auxiliary symbols are taken into account, then a technique of Hemaspaandra et al. for space efficient computations [2] can be used to derive the following result.

Theorem 2. CFL *is included in* $\mathcal{L}(\mathsf{g\text{-}aux}(\log n,1)\text{-}\mathsf{RWW})$.

Proof. First we describe an RWW-automaton which has several auxiliary symbols in its tape alphabet. Thereafter we will show how to reduce the number of auxiliary symbols to one.

Let L be a context-free language. From a context-free grammar in Chomsky normal form for L we easily obtain a grammar $G = (N, \Sigma, P, S)$ for L satisfying the following technical conditions:

1. the start symbol S does not occur on the righthand side of any rule of G,
2. for each rule $(X \to \alpha) \in P$, we have $|\alpha| \leq 2$, and if $|\alpha| \leq 1$, then $X = S$.

By $\hat{L}$ we denote the set of sentential forms that are derivable in G.

An RWW-automaton $M = (Q, \Sigma, N \cup \Sigma, \mathbb{¢}, \$, q_0, 2, \delta)$ for L is given through the following set of meta-instructions:

$$\begin{array}{ll} (1)\ (\mathbb{¢} \cdot x \cdot \$, \mathsf{Accept}) & \text{for all } x \in \hat{L},\ |x| < 2; \\ (2)\ (\mathbb{¢} \cdot (N \cup \Sigma)^*, \alpha \to X) & \text{for all } (X \to \alpha) \in P,\ |\alpha| = 2. \end{array}$$

Given an input $w \in \Sigma^*$, M performs a bottom-up parse of w. Hence, this automaton recognizes the language L.

We claim that, for each $w \in L$, there exists an accepting computation of M such that in each configuration during this computation, at most $\log |w|$ copies of auxiliary symbols occur on the tape. In other words, we claim that there exists a G-derivation $S \Rightarrow^* w$ such that each sentential form in this derivation contains at most $\log |w|$ nonterminals.

Let T be a G-derivation tree for w. For each vertex v of this tree, we define the *weight* $\omega(v)$ of v as the number of leaves in the subtree of T that is rooted at v, that is, $\omega(v)$ is the length of the factor of w that is derived in the G-derivation corresponding to T from the nonterminal associated to the vertex v. For proving the claim above, we consider that parse of T that satisfies the following condition: For each internal node v of T with two children v_1, v_2, the children are parsed one after the other; if $\omega(v_1) \geq \omega(v_2)$, then v_1 is parsed first; otherwise v_2 is parsed first.

This condition ensures that each sentential form encounted during the parsing of w contains at most $\log |w|$ many nonterminals. Indeed, let α be such a sentential form, let X_1 be a nonterminal occurring in α such that the vertex v_1 of T that is labelled with this particular occurrence of X_1 has maximal weight, and let $m := \omega(v_1)$. Thus, the subtree of T that is rooted at v_1 has already been parsed, but the subtree that is rooted at the father of v_1, say v, has not yet been parsed. This implies that no parsing steps have been performed for any part of the input that does not belong to the subtree that is rooted at v. In particular, all other nonterminals occurring in the sentential form α belong to the subtree of T that is rooted at the sibling v_2 of v_1. Further, the above parsing strategy implies that $\omega(v_2) \leq m$. Now we have two cases:

- if the nonterminal X_2 labelling the vertex v_2 already occurs in the sentential form α, then X_1 and X_2 are the only two nonterminals occurring in α;
- if X_2 does not yet occur in α, then we can continue our analysis for the subword of w that is generated by the subtree rooted at v_2. Thus, after at most $\log |w|$ many iterations we get to a subword of length 1, which means that in α, at most $\log |w|$ nonterminals occur.

Finally we can transform the RWW-automaton M above into an RWW-automaton M' that has only a single auxiliary symbol A. Let $X_1, \ldots, X_t$ be a linear ordering of the nonterminals of G. The nonterminal X_i is encoded as $A \cdot \mathsf{bin}(i)$, where $\mathsf{bin}(i)$ is a $t' := \lceil \log t \rceil$-bit encoding of i. For this encoding we use two fixed terminal symbols. Now the automaton M' is obtained from M by replacing each nonterminal X of G by the corresponding encoding in each of its

meta-instructions. In order to guarantee that each rewrite instruction of M' is length-reducing, M' simulates $t' + 2$ derivation steps of G per cycle. □

For the rest of the paper we restrict our attention to a particular class of context-free languages. A language L is called *k-linear* [19] if there is a context-free grammar $G = (N, \Sigma, P, S)$ for L that contains a starting rule of the form $S \to S_1 \dots S_k$ such that S does not occur in any other rule of G, and S_i is the starting symbol of a linear subgrammar $G_i = (N_i, \Sigma, P_i, S_i)$ for each $i \in \{1, \dots, k\}$. Further, $N_i \cap N_j = \emptyset$ for each $i \neq j$, and S_i does not occur on the righthand side of any rule of G_i $(1 \leq i \leq k)$. Thus, L is the concatenation $L_1 \cdot L_2 \cdot \ldots \cdot L_k$ of the linear languages $L_i := L(G_i)$ $(1 \leq i \leq k)$. By k-LIN we denote the class of k-linear languages.

Theorem 3. $\bigcup_{k \in \mathbb{N}} k\text{-LIN} \subsetneq \mathcal{L}(\text{aux}(2)\text{-RLWW})$.

Proof. Let L be a k-linear language, and let G be a k-linear grammar that generates L. First, we describe the idea of accepting L using only two occurrences of auxiliary symbols on the tape, but without restricting the number of auxiliary symbols in the alphabet.

For an input word x we first guess a G_1-derivation $S_1 \Rightarrow^* x_1$ for a prefix x_1 of x such that $x = x_1 \dots x_k$, $x_i \in L(G_i)$ for $i \in [1, k]$, in a bottom-up fashion. We start by choosing a production $X \to \alpha$ for $X \in N_1$ and $\alpha \in \Sigma^*$. That is, we perform a rewrite step $\alpha \to X$. Then we simulate consecutive steps of the derivation in reverse order by applying meta-instructions

$$(\text{¢}\Sigma^*, \alpha X \beta \to Y, \Sigma^*\$)$$

for $X, Y \in N_1$, $\alpha, \beta \in \Sigma^*$, corresponding to productions $Y \to \alpha X \beta$ of G_1.

When a tape content of the form $S_1 y$ is reached, where $y \in \Sigma^*$, we begin to simulate a G_2-derivation for $L_2 = L(G_2)$ by first executing the last step in a derivation of $x_2 \in L_2$. Thereafter, the tape contains two auxiliary symbols: $S_1 \in N_1$ and $X \in N_2$. This means that we have already found a prefix $x_1 \in L_1$ and started to simulate a G_2-derivation for $x_2 \in L_2$. So we can remove S_1. Further, we process consecutive factors analogously. In general, we can describe this behaviour by the following meta-instructions, where $u, y, v \in \Sigma^*$, $i \in [1, k]$:

$$\begin{array}{ll}
(\text{¢}\Sigma^*, u \to X, \Sigma^*\$) & \text{for } (X \to u) \in P_1, \\
(\text{¢}\Sigma^*, uXy \to Y, \Sigma^*\$) & \text{for } X, Y \in N_i, (Y \to uXy) \in P_i, \\
(\text{¢}S_i\Sigma^*, u \to X, \Sigma^*\$) & \text{for } X \in N_{i+1}, (X \to u) \in P_{i+1}, \\
(\text{¢}, S_i \to \varepsilon, \Sigma^* X \Sigma^*\$) & \text{for } X \in N_{i+1}, \\
(\text{¢}S_k\$, \mathsf{Accept}). &
\end{array}$$

However, this schema does not guarantee that the automaton is length-reducing, for example, a production $X \to y$ where $|y| \leq 1$ can be applied. Further, our aim is to use only one auxiliary symbol in the alphabet.

Without loss of generality we can assume that the grammar does neither contain any productions of the form $X \to Y$ for $|Y| \leq 1$ and $X \notin \{S_1, \dots, S_k\}$

nor of the form $S_i \to X$ for $i \in [1,k]$ and $X \in N$. Further, we can assume that Σ contains at least two symbols, say 0 and 1 (as context-free languages over a one-letter alphabet are regular). In order to apply the above strategy using only one auxiliary symbol, an occurrence of this auxiliary symbol will be followed by a binary encoding (of a fixed length) of the actual nonterminal of G. In order to make the resulting rewrite operations length-reducing, 'short' factors x_i will not be processed separately and for the remaining 'long' factors, we simulate several derivation steps by a single rewrite operation. In this way we will have sufficient space for the encodings.

Let $p := 2 \cdot \max(\lceil \log |N| \rceil, \lceil \log k \rceil)$, and let $X_1, \ldots, X_{|N|}$ be the nonterminals of G. For each occurrence of the only auxiliary symbol A of M on the tape, the p symbols following A will be interpreted as follows: the first $p/2$ symbols encode the number i of the nonterminal X_i, and the next $p/2$ symbols encode j, the index of the last factor x_j processed previously. For $X_i \in N$, we use $\mathsf{bin}(X_i)$ to denote the $(p/2)$-bit encoding of i, and for $i \in [1,k]$, $\mathsf{bin}(i)$ denotes the $(p/2)$-bit encoding of i.

Finally, let $r := \max\{\, |\alpha| \mid (X \to \alpha) \in P \,\} + p$. The automaton M will proceed according to the following strategy:

1. If the tape does not contain any occurrences of the auxiliary symbol, and if the length of the tape content is not longer than $k \cdot r$, then M decides whether the input belongs to L in a tail computation.
2. If the tape does not contain any occurrences of the auxiliary symbol, but the length of the tape content exceeds the number $k \cdot r$, then M guesses a minimal index j such that $|x_j| > r$. Next M guesses a derivation $X \Rightarrow_G^* u$ such that $p + 1 < |u| \leq r$, $u \in \Sigma^*$, and $X \in N$, M finds an occurrence of the factor u within the tape content, and executes the rewrite step $u \to A\mathsf{bin}(X)\mathsf{bin}(0)$.
3. To simulate a derivation step in a single factor, M has a meta-instruction of the form
$$(\mathfrak{c}\Sigma^*, uA\mathsf{bin}(Y)\mathsf{bin}(j)v \to A\mathsf{bin}(X)\mathsf{bin}(j), \Sigma^*\$)$$
for each production $X \to uYv$, where $X, Y \in N_i$ and $i > j \geq 0$.
4. To finish the derivation of a factor x_i, M has a meta-instruction of the form
$$(\mathfrak{c}, yuA\mathsf{bin}(Y)\mathsf{bin}(j-1)v \to A\mathsf{bin}(S_i)\mathsf{bin}(i), \Sigma^*\$)$$
for each $y = x_j x_{j+1} \ldots x_{i-1}$ such that $x_l \in L_l$ and $|x_l| \leq r$ for $l \in [j, i-1]$, and for each production $S_i \to uYv$.
5. To start the processing of a new factor, M guesses the next value $j > i$ such that $|x_j| > r$, where i is the index of the previously processed factor. Next M chooses a derivation $X \Rightarrow_G^* u$ such that $p + 1 < |u| \leq r$, $X \in N_j$, finds the factor u on the tape, and executes the meta-instruction
$$(\mathfrak{c}A\mathsf{bin}(S_i)\mathsf{bin}(i)\Sigma^*, u \to A\mathsf{bin}(X)\mathsf{bin}(i), \Sigma^*\$).$$
6. In order to remove an occurrence of the auxiliary symbol which is not needed anymore from the tape (together with the encoding of the nonterminal which follows this symbol), M uses the meta-instructions
$$(\mathfrak{c}, A\mathsf{bin}(S_i)\mathsf{bin}(i) \to \varepsilon, \Sigma^* A\mathsf{bin}(X)\mathsf{bin}(i)\Sigma^*\$)$$

for $X \in N_j$, $j > i$.

7. Finally, for each $y = x_j x_{j+1} \dots x_{i-1}$ and $y' = x_{i+1} \dots x_k$ such that $x_l \in L_l$ and $|x_l| \leq r$ for $l \in [j,k] - \{i\}$, $i \in [1,k]$, M has the meta-instruction

$$(\mathfrak{c}yA\mathsf{bin}(S_i)\mathsf{bin}(j-1)y'\$, \mathsf{Accept}).$$

The above meta-instructions define a length-reducing RLWW-automaton which recognizes $L(G)$ and which uses at most two occurrences of the single auxiliary symbol A.

As already $\mathcal{L}(\mathsf{aux}(0)\text{-RLWW})$ contains non-context-free languages, the above inclusion is a proper one. □

From the proof above we obtain the following consequence.

Corollary 7. $\mathsf{LIN} \subsetneq \mathcal{L}(\mathsf{aux}(1)\text{-RLWW})$.

Actually, this result can be extended as follows, improving on Theorem 3 at least for the case $k = 2$.

Theorem 4. $2\text{-}\mathsf{LIN} \subsetneq \mathcal{L}(\mathsf{aux}(1)\text{-RLWW})$.

Proof. First recall from Corollary 6 that already $\mathsf{aux}(0)$-RWW-automata accept some non-context-free languages. Hence, if 2-LIN is included in $\mathcal{L}(\mathsf{aux}(1)\text{-RLWW})$, then this is a proper inclusion. Below we show that, for each language $L \in 2\text{-}\mathsf{LIN}$, there exists an RLWW-automaton M' with only a single auxiliary symbol such that M' recognizes the language L, and it uses at most one occurrence of its auxiliary symbol in each configuration that is reachable from an initial configuration.

Let $G = (N, \Sigma, P, S)$ be a 2-linear grammar with the starting rule $S \to S_1 S_2$ and linear subgrammars $G_i = (N_i, \Sigma, P_i, S_i)$, $i = 1, 2$. We can assume without loss of generality that G does not contain any productions of the form $X \to Y$ for $|Y| \leq 1$ and $X \notin \{S_1, S_2\}$ or of the form $S_i \to X$ for $i \in \{1, 2\}$ and $X \in N$.

First, we present a restarting automaton M which recognizes $L = L(G)$ and which uses only a single occurrence of an auxiliary symbol in any configuration that is reachable from an initial configuration, but which has many auxiliary symbols in its tape alphabet. In addition, M will not satisfy the condition that each rewrite step is length-reducing. Therefore, we describe in a second stage the changes that are necessary to get rid of all but one auxiliary symbol from the tape alphabet, and that make each rewrite step length-reducing.

Let $N, \bar{N}, \hat{N}, N'$ be four disjoint copies of the set of nonterminals of the grammar G, and let $c \in \mathbb{N}_+$ be a constant that will be specified later.

Input words of length at most $2c + 1$ are accepted or rejected by M in tail computations. For an input word x satisfying $|x| \geq 2c + 2$, M must determine whether x admits a factorization $x = x_1 x_2$ such that $x_1 \in L(G_1)$ and $x_2 \in L(G_2)$. As a first step, M chooses nondeterministically one of the cases (i) $|x_1|, |x_2| > c$, (ii) $|x_1| \leq c$ (and so $|x_2| > c$), or (iii) $|x_2| \leq c$ (and so $|x_1| > c$).

In cases (ii) and (iii) M guesses a G_i-derivation of x_i from S_i in reverse order for the factor x_i satisfying $|x_i| > c$ using only a single occurrence of an auxiliary symbol from the set $\bar{N}_i$, verifying that the remaining factor x_j, $j \neq i$, belongs to $L(G_j)$ in the final step.

Finally, in case (i) M works as follows. First, it guesses a G_1-derivation for x_1 in reverse order, using auxiliary symbols from the set N_1. After having completed this derivation, M simulates a G_2-derivation

$$\begin{aligned} S_2 \Rightarrow\ & s_0X_1r_0 \Rightarrow s_0s_1X_2r_1r_0 \\ \Rightarrow^*\ & s_0s_1 \dots s_{m-1}X_mr_{m-1} \dots r_1r_0 \\ \Rightarrow\ & s_0s_1 \dots s_{m-1}s_mr_{m-1} \dots r_1r_0 \end{aligned}$$

of $x_2 = s_0s_1 \dots s_{m-1}s_mr_{m-1} \dots r_1r_0$ from S_2, where $X_{i-1} \to s_{i-1}X_ir_{i-1}$ is the i-th derivation step for $1 \leq i \leq m$, $X_0 := S_2$, and $X_m \to s_m$ is the last step. For remembering the position of the nonterminal X_i within the current content of the tape, M uses a finite number of symbols at the suffix of the tape content to encode this position. Unfortunately, M cannot apply any rewrite steps to the suffix as long as $i \leq j$, where j is the minimal index for which $r_j \neq \varepsilon$. Therefore, the initial part of length j of the above G_2-derivation is treated in a different way. For taking care of this initial part, M uses the following meta-instructions that employ auxiliary symbols from the set $\hat{N}$:

$$\begin{aligned} &(10)\ (\mathfrak{c}, S_1 \to \hat{S}_2, \Sigma^*\$), \\ &(20)\ (\mathfrak{c}, \hat{X}_is_i \to \hat{X}_{i+1}, \Sigma^*\$). \end{aligned}$$

Let $p := \max\{3, \lceil \log |N| \rceil, \lceil \log(\alpha c) \rceil\}$, where $\alpha := \max\{|\beta| \mid (X \to \beta) \in P\}$. Below we use the notation $\mathsf{bin}(X)$ to denote a p-bit encoding of $X \in N$ (according to some fixed linear ordering) or of $X \in \mathbb{N}$. By $\mathsf{bin}(\star)$ we denote an arbitrary string of length p over the input alphabet.

The simulation of the derivation step $X_j \to s_jX_{j+1}r_j$, $s_j, r_j \neq \varepsilon$, will require more than one rewrite step in our simulation. This follows from the fact that the distance between the factors s_j and r_j which are to be removed can be arbitrarily large. In order to solve this problem, a fixed number of input symbols immediately to the right of the auxiliary symbol and a fixed number of input symbols immediately to the left of the sentinel \$ will be used to encode information about the derivation step to be simulated and to coordinate the rewrite steps. More precisely,

- the input symbol following the auxiliary symbol and the rightmost input symbol on the tape will encode information about the progress of the simulation of the current derivation step;
- the next p input symbols following the auxiliary symbol will encode the length of the factor r_j which has to be removed (when the automaton is in the appropriate stage of the simulation).

This technique requires some extra space for storing encodings. To make up for this, M will simulate at least c derivation steps at once.

Let j be the minimal index for which $|r_j| > 0$ in the above G_2-derivation. For this part of the simulation, M uses the subalphabet N_2'. We distinguish between two cases.

1. If $|r_j| = 1$, then M does not change the suffix, but it puts an indicator into the prefix that is 'related' to r_j:

$$(30)\ (\text{¢}, \hat{X}_j s_j \to X'_{j+1}\mathsf{neg}(r_j)\mathsf{bin}(1), \Sigma^* r_j \$),$$

   where, for $a \in \Sigma$, $\mathsf{neg}(a)$ denotes an arbitrary symbol $b \in \Sigma$ satisfying $b \neq a$.
2. If $|r_j| > 1$, let $a := r_j[|r_j|]$ (that is, a is the last symbol of r_j):

$$(40)\ (\text{¢}, \hat{X}_j s_j \to X'_{j+1} a\mathsf{bin}(|r_j|), \Sigma^* r_j \$),$$
$$(50)\ (\text{¢} X'_{j+1} a\mathsf{bin}(|r_j|)\Sigma^*, r_j \to \mathsf{neg}(a), \$).$$

Thereafter M simulates subderivations of the form $X \Rightarrow^c s'Yr'$, where $|s'| > 0$, using at most two cycles for each such subderivation and employing the symbols from N_2'. Let $X \Rightarrow^c s'Yr'$ be the actual subderivation to be simulated, where $|s'| > 0$. Again we distinguish two cases.

1. If $|r'| > 0$, then the following meta-instructions are used:

$$(60)\ (\text{¢}, X' a\mathsf{bin}(j)s' \to Y'\mathsf{neg}(a)\mathsf{bin}(|r'|), \Sigma^* r'\mathsf{neg}(a)\$) \ \text{ for any } j,$$
$$(70)\ (\text{¢}, Y' a\mathsf{bin}(|r'|)\Sigma^*, r'a \to \mathsf{neg}(a), \$).$$

2. If $|r'| = 0$, then the following meta-instructions are used:

$$(80)\ (\text{¢}X' a\mathsf{bin}(j)s' \to Y' a\mathsf{bin}(0), \Sigma^* \mathsf{neg}(a)\$).$$

The symbol a following the auxiliary symbol and the symbol b preceding the right sentinel \$ are used here as follows. If these two symbols are equal, then a rewrite step is to be applied to the suffix deleting a factor r' the length of which is encoded by the factor $\mathsf{bin}(|r'|)$ following the symbol a. If the two symbols differ, then the simulation of the next derivation step shall be started (that is, a rewrite step will be applied to the prefix).

However, if $s' = \varepsilon$ in the subderivation $X \Rightarrow^c s'Yr'$, then the rewrite operations in meta-instructions (30), (40), and (60) are not length-reducing; in fact, they are in general not even weight-reducing. Hence, in this situation we move the encoded information necessary for simulating the above subderivation to the suffix of the tape inscription. To distinguish this case from the previous one we make use of a new auxiliary symbol E that will be placed into the prefix. Still, the rewrite step introducing E will not be length-reducing, either, but we will solve this issue in the final part of the proof.

For this case we add the following meta-instructions, where $b := r'[|r'|]$:

$$(90)\ (\text{¢}, X' a\mathsf{bin}(j) \to Eb\mathsf{neg}(a)\mathsf{bin}(X), \Sigma^* r'\mathsf{neg}(a)\$),$$
$$(100)\ (\text{¢}Eb\mathsf{neg}(a)\mathsf{bin}(X)\Sigma^*, r'\mathsf{neg}(a) \to \mathsf{bin}(Y)\mathsf{bin}(0)ba, \$).$$

Observe that $|s'| = 0$ implies that $|r'| \geq c$. Notice further that here we do not have an occurrence of an auxiliary symbol in the suffix. Instead the nonterminal Y is encoded using input symbols. Similarly the nonterminal X is encoded in the prefix, in difference to the situation in meta-instructions (10)–(80), where different nonterminals correspond to different auxiliary symbols.

As long as there is an occurrence of E on the tape, M continues to simulate subderivations $X \Rightarrow^c s'Yr'$ satisfying $|r'| > 0$ using the suffix of the tape inscription to encode the necessary information. It will switch back to using the encoding by symbols from N_2' in the prefix as soon as a subderivation of the form $X \Rightarrow^c s'Yr'$ satisfying $|r'| = 0$ is to be simulated.

Before listing the appropriate meta-instructions, we need to explain the way in which the two input symbols ba that follow immediately to the right of the symbol E and the two input symbols $b'a'$ immediately preceding the \$-symbol are used during this phase of the simulation. By comparing ba to $b'a'$ we determine the current status of the simulation. There are four cases:

(α) $ba = b'a'$	: M is just moving from the N_2'-mode to the E-mode of encoding, but the suffix has not yet been adjusted (see (90));
(β) $ba = \mathsf{neg}(b')a'$	: the prefix is to be adjusted next;
(γ) $ba = b'\mathsf{neg}(a')$	: the next derivation step is to be started (see (100);
(δ) $ba = \mathsf{neg}(b')\mathsf{neg}(a')$	: M is to switch back to the N_2'-mode of encoding.

Let $X \Rightarrow^c s'Yr'$ be the next subderivation to be simulated, and assume that the tape content is of the following form for some string $w \in \Sigma^*$:

$$(110)\ \text{¢}Eba\mathsf{bin}(\star)s'wr'\mathsf{bin}(X)\mathsf{bin}(\star)b\mathsf{neg}(a)\$.$$

From this tape content we extract the following information:

(a) E indicates that the nonterminal is encoded in the suffix,
(b) ba immediately to the right of E and $b\mathsf{neg}(a)$ immediately preceding \$ indicate that the simulation of the next subderivation of length c is to start,
(c) $\mathsf{bin}(X)$ encodes the nonterminal X,
(d) the factors $\mathsf{bin}(\star)$ to the right of Eba and $\mathsf{bin}(\star)$ to the right of $\mathsf{bin}(X)$ are strings of length p over the input alphabet, which do currently not contain any important information. However, they are used to 'reserve' space for the next step of the simulation. In particular, the place currently occupied by the second factor $\mathsf{bin}(\star)$ will be used to encode the length of the factor s' which will be removed in the next subderivation.

For $|r'| > 0$, we use the following meta-instructions for M, where we distinguish between two cases depending on $|s'|$.

1. For $|s'| > 0$, the following meta-instructions are used:

$$(120)\ (\text{¢}Eba\mathsf{bin}(\star)s'\Sigma^*, r'\mathsf{bin}(X)\mathsf{bin}(\star)b\mathsf{neg}(a) \to \mathsf{bin}(Y)\mathsf{bin}(|s'|)\mathsf{neg}(b)a, \$),$$
$$(130)\ (\text{¢}E, ba\mathsf{bin}(\star)s' \to \mathsf{neg}(b)\mathsf{neg}(a)\mathsf{bin}(\star), \Sigma^*\mathsf{bin}(Y)\mathsf{bin}(|s'|)\mathsf{neg}(b)a\$),$$

2. while for $|s'| = 0$, the following meta-instruction is used:

$$(140)\ (\mathfrak{c}Eba\mathsf{bin}(\star)\Sigma^*, r'\mathsf{bin}(X)\mathsf{bin}(\star)b\mathsf{neg}(a) \to \mathsf{bin}(Y)\mathsf{bin}(\star)b\mathsf{neg}(a), \$).$$

In (140) the factors $\mathsf{bin}(\star)$ that occur on the lefthand side and on the right-hand side of the rewrite operation are identical, that is, this factor is not altered by the current rewrite operation.

Finally, we consider the case that $|r'| = 0$. Notice that in this case $|s'| \geq c$. In this situation the simulation is to switch back to the N_2'-mode of encoding, that is, an auxiliary symbol from N_2' is used in the prefix, while in the suffix only the last symbol preceding the \$-symbol is used to coordinate the simulation (see meta-instructions (30)–(80)). Accordingly, we introduce the following meta-instruction:

$$(150)\ (\mathfrak{c}, Eba\mathsf{bin}(\star)s' \to Y'\mathsf{neg}(a)\mathsf{bin}(|\mathsf{bin}(X)\mathsf{bin}(\star)| + 1), \Sigma^*\mathsf{bin}(X)\mathsf{bin}(\star)b\mathsf{neg}(a)\$).$$

After executing this meta-instruction, the factor $\mathsf{bin}(X)\mathsf{bin}(\star)b$ in the suffix is no longer appropriate, as in the N_2'-mode of encoding only a single symbol in the suffix contains any non-input information. However, as the symbol following Y' equals the rightmost symbol preceding the \$-symbol, meta-instruction (70) will remove this factor, as its length $2p + 1$ has been encoded as

$$\mathsf{bin}(|\mathsf{bin}(X)\mathsf{bin}(\star)| + 1) = \mathsf{bin}(2p + 1)$$

at the corresponding place in the prefix.

As decribed above M will correctly accept the language $L = L(G)$. However, M is not length-reducing (see meta-instructions (10), (30), (40), (90), (100)), and it has many auxiliary symbols in its tape alphabet. Therefore, we now transform M into a length-reducing automaton with only a single auxiliary symbol.

First we fix unique binary encodings enc for the auxiliary symbols of M. These encodings will not be of fixed length, but they will be prefix-free. Then we transform M into a length-reducing automaton M' with only a single auxiliary symbol F in its tape alphabet by replacing each occurrence of each auxiliary symbol X (in the above meta-instructions) by $F \cdot \mathsf{enc}(X)$. Below we list all the conditions that these encodings and the constants p and c must satisfy in order to make M' length-reducing. Afterwards we will show how to choose the encodings and the constants satisfying all these conditions.

1. The encoding of the auxiliary symbols is to be prefix-free, that is, whenever X and Y are two different auxiliary symbols of M, then $\mathsf{enc}(X)$ is not a prefix of $\mathsf{enc}(Y)$. Further, $|\mathsf{enc}(X)| < c/2$ for all auxiliary symbols of M, as within c derivation steps, we need to make room for two such encodings.
2. $|\mathsf{enc}(X)| < |\mathsf{enc}(Y)|$ for all $X \in \hat{N}$ and $Y \in N$ (see (10)).
3. For each type of auxiliary symbol $\mathcal{R} \in \{N, \bar{N}, \hat{N}, N'\}$ of M, we require that $|\mathsf{enc}(X)| = |\mathsf{enc}(Y)|$ holds for all $X, Y \in \mathcal{R}$.
4. $|\mathsf{enc}(X)| > |\mathsf{enc}(Y)| + 1 + p$ for all $X \in \hat{N}$ and $Y \in N'$ (see (30) and (40)).

5. $|\mathsf{enc}(X)| > |\mathsf{enc}(E)| + 1 + p$ for all $X \in N'$ (see (90)).
6. $c > 2p + 1$ (see (100)).
7. $|\mathsf{enc}(E)| + 2 + c > |\mathsf{enc}(X)| + 1$ for all $X \in N'$ (see (150)).
8. $\lceil \log(2p + 1) \rceil < p$ (see (150)).

Finally we show how to satisfy these conditions. Let $n, \hat{n}, n'$ denote the lengths of the encodings of the auxiliary symbols from the sets $N, \hat{N}, N'$, respectively, and let e denote the length of the encoding of E. Then we obtain the following inequalities from the conditions above:

- $c/2 > n > \hat{n} > n' + 1 + p$,
- $e + 1 + c > n' > e + 1 + p$,
- $c > 2p + 1$.

Also $p = \max\{3, \lceil \log |N| \rceil, \lceil \log(\alpha c) \rceil\}$, where $\alpha = \max\{\, |\beta| \mid (X \to \beta) \in P \,\}$.

We first fix a linear ordering of the nonterminals N of G. Then, for each $X \in N$, $\mathsf{bin}(X)$ denotes the p-bit encoding of the index of X in the linear ordering of N. We now choose the encodings as follows:

1. for $X \in N$, we choose $\mathsf{enc}(X) := 00001 \cdot 0^{p+2} \cdot \mathsf{bin}(X)$;
2. for $\hat{X} \in \hat{N}$, we take $\mathsf{enc}(X) := 0001 \cdot 0^{p+2} \cdot \mathsf{bin}(X)$;
3. for $X' \in N'$, we take $\mathsf{enc}(X') := 0010 \cdot \mathsf{bin}(X)$;
4. $\mathsf{enc}(E) := 01$.

It remains to determine the constant c. From the requirements above we see that the constants c and p must satisfy the conditions:

$$c/2 > n = 5 + 2 + p + p \text{ and } p \geq \lceil \log(\alpha c) \rceil.$$

If we take $c := 4(7 + 2p)$, then it remains to ensure that

$$p \geq \lceil \log(\alpha c) \rceil = \lceil \log(4\alpha(7 + 2p)) \rceil.$$

Obviously there exists a constant p_0 such that this inequality is satisfied for each $p > p_0$. Thus, with these values for p and c, the resulting restarting automaton M' is length-reducing, and it accepts the language $L = L(G)$ using only a single occurrence of its only auxiliary symbol F in each configuration that is reachable from an initial configuration. This completes the proof of Theorem 4. □

5 Conclusions and Open Problems

We have seen that two occurrences of a single auxiliary symbol suffice to accept every k-linear language, and that for $k = 2$, already a single occurrence suffices. On the other hand, we have seen that a bounded number of occurrences of auxiliary symbols does not suffice to accept all context-free languages by deterministic RLWW-automata. However, many problems concerning the new measures remain open. For example, is there an infinite hierarchy with respect to the number of

auxiliary symbols in the tape alphabet? Or is it possible to show that a single auxiliary symbol is always sufficient by using appropriate encodings? What can be said in general on the number of occurrences of auxiliary symbols on the tape? Is there an infinite hierarchy with respect to the number of occurrences of auxiliary symbols? Other interesting questions concern the context-free languages. For example, is there a constant d such that each context-free language is accepted by a nondeterministic RLWW-automaton that uses at most d occurrences of auxiliary symbols? Recall that each *deterministic* context-free language is accepted by a monotone det-R-automaton [4], that is, for these languages no auxiliary symbols are required at all.

6 The Proof of Proposition 3

To prove Proposition 3 we will make use of the notion of Kolmogorov complexity and its properties [9]. Here we consider the Kolmogorov complexity $K(x)$ of words x over a finite alphabet Σ satisfying $s := |\Sigma| > 1$. In the following all logarithms used will be taken with respect to base s. A word $x \in \Sigma^+$ is called *random* if $K(x) > |x| - 4\log|x|$, and it is called *incompressible* if $K(x) \geq |x|$.

Proposition 3 claims that the language L_{pal2} is not accepted by any deterministic RLWW-automaton that uses only $o(n/\log^5 n)$ occurrences of auxiliary symbols. So, for the sake of contradiction, we assume that there exists a det-RLWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ for L_{pal2} that uses only $o(n/\log^5 n)$ occurrences of auxiliary symbols.

For deriving a contradiction, we will analyze the (accepting) computation of M on an input of the form $w_1 w_1^R w_2^R w_2$, where $|w_1| = |w_2|$, and $w := w_1 w_2$ satisfies the conditions that $n := |w|$ is sufficiently large and that w is incompressible (that is, $K(w) \geq |w|$). We will show that as long as a long infix x of $w_1 w_1^R w_2^R w_2$ remains unchanged during this computation, which means that no rewrite operation is performed inside the factor x, each rewrite operation is executed inside the prefix or inside the suffix in which rewrite steps were already executes before, or within a distance of at most $\log^4 n$ from that prefix or that suffix.

As each rewrite step shortens the tape, the above property implies that the prefix w_1 or the suffix w_2 is shortened significantly within a certain number of cycles, while no rewrite operation is applied to the infix $w_1^R w_2^R$. On the other hand, the rewrite steps add some extra information to the prefix and to the suffix, as they introduce occurrences of auxiliary symbols. Each occurrence of an auxiliary symbol is uniquely determined by its position on the tape and its number in a linear ordering of Γ. Hence, it can be encoded by a word from Σ^+ of length $\log n + \log|\Gamma|$. As the number of occurrences of auxiliary symbols on the tape is limited, however, they cannot make up for the loss of information that follows from the reduction in length. Thus, we will finally shorten the suffix w_2 to a word that can be encoded by less than $n/2 - \omega(\log n)$ symbols without applying any rewrite step to the infix w_2^R, or we will shorten the prefix w_1 to a word that can be encoded by less than $n/2 - \omega(\log n)$ symbols without applying any rewrite

step to the infix w_1^R. As M accepts the language L_{pal2}, both cases yield a unique 'short' representation of w, which contradicts the incompressibility of w. Below we present this proof in more detail.

Proposition 4. *Let $M = (Q, \Sigma, \Gamma, \mathrm{¢}, \$, q_0, k, \delta)$ be a* det-RLWW*-automaton, and let $u_1, u_2, v \in \Sigma^*$ and $u_1', u_2' \in \Gamma^*$ such that $|u_1| = |u_2| = n > n_0$ and $K(u_i) > n - 4\log n$ for $i = 1, 2$, where n_0 is a fixed constant. Then the rewrite position of M in the cycle which begins with the restarting configuration $q_0\mathrm{¢}u_1'u_1vu_2u_2'\$$ is located inside the prefix of length $|u_1'| + \log^4 n$ or inside the suffix of length $|u_2'| + \log^4 n$ of $u_1'u_1vu_2u_2'$.*

The proof of Proposition 4 is rather long and complicated. Therefore, we postpone it to Subsection 6.1. In order to apply Proposition 4 repeatedly (see below), we need the following technical result.

Proposition 5. ([9] p. 110) *Let w be an incompressible word of length n, where n is sufficiently large. Then each subword x of w of length equal to or greater than $\log^2 n$ satisfies the condition $K(x) > |x| - 3\log n$. In particular, if $|x| > n/4$, then x is random.*

Let $w = w_1w_2 \in \Sigma^n$ be incompressible, where $|w_1| = |w_2|$, and n is sufficiently large. We analyze the configuration that M reaches from the initial configuration $q_0\mathrm{¢}w_1w_1^Rw_2^Rw_2\$$ within $t := \frac{n}{4\log^4 n}$ cycles. From Proposition 4 and Proposition 5 it follows that all t rewrite operations that are performed during these cycles are applied to the prefix of length $t\log^4 n$ and to the suffix of length $t\log^4 n$. As $t\log^4 n \leq n/4$, we see that all these rewrite steps are applied to the prefix $\mathrm{¢}w_1$ and to the suffix $w_2\$$, while no rewrite step is applied to the infix $w_1^Rw_2^R$. Hence, the prefix $\mathrm{¢}w_1$ has been transformed into $\mathrm{¢}w_1'$, and the suffix $w_2\$$ has been transformed into $w_2'\$$ for some words $w_1', w_2' \in \Gamma^*$ satisfying the inequality

$$|w_1'| + |w_2'| \leq n - t = n - \frac{n}{4\log^4 n}\,.$$

In addition, the number of occurrences of auxiliary symbols in w_1' and w_2' is in $o(n/\log^5 n)$ by our assumption on M. That is, we can assume that the number of occurrences of auxiliary symbols is smaller than $n/(32\log^5 n)$, if n is sufficiently large.

We now construct a short representation for $w_1'w_2'$. Let π_Σ be the projection from Γ^* onto Σ^* defined by $\pi_\Sigma(a) := a$ for all $a \in \Sigma$ and $\pi_\Sigma(A) := \varepsilon$ for all $A \in \Gamma \setminus \Sigma$. We describe $w_1'w_2'$ through the word $\pi_\Sigma(w_1'w_2')$ and the number $|w_1'|$, and in addition, for each occurrence of an auxiliary symbol A in $w_1'w_2'$ we describe its position in $w_1'w_2'$ and its value. For $\pi_\Sigma(w_1'w_2')$ we need space

$$|\pi_\Sigma(w_1'w_2')| = |w_1'w_2'| - |w_1'w_2'|_{\Gamma\setminus\Sigma} \leq n - \frac{n}{4\log^4 n},$$

and for each occurrence of an auxiliary symbol we need at most space $2\log n$. Hence, for all auxiliary symbols together we need at most space

$$\frac{n}{32\log^5 n} \cdot 2\log n = \frac{n}{16\log^4 n}\,.$$

Thus, in this way we obtain a representation for $w_1'w_2'$ of length

$$\frac{n}{1} - \frac{n}{4\log^4 n} + \frac{n}{16\log^4 n} + 4\log n \leq \frac{n}{1} - \frac{n}{8\log^4 n} .$$

Here the additional space $4\log n$ is needed to store the lengths of w_1' and w_2' and to make sure that the encodings of the various pieces are clearly separated from each other.

Using this representation of $w_1'w_2'$ we will now construct a compact representation for w, which will then contradict our assumption that w is incompressible. This description of w will consist of:

- The above representation of $w_1'w_2'$ of size not larger than $n - n/(8\log^4 n)$.
- The description of the det-RLWW automaton M.
- Two 'transition tables' T_1, T_2. The first of these transition tables determines, for each state q of M, the state q' of M and the direction (left or right) in which M leaves the infix $w_1^R w_2^R$ when it enters it from the left being in state q; the second table describes the corresponding information for the case that M enters the infix $w_1^R w_2^R$ from the right. More precisely, these tables only list the indicated information as it was encountered during the first t cycles of the above computation.
- The index of the word $w = w_1w_2$ in the lexicographic order of the set of words that are *equivalent* with respect to M, w_1', w_2', T_1, and T_2. This set consists of all words $uv \in \Sigma^n$ ($|u| = |v|$) that satisfy the following conditions:
  - the transition tables T_1 and T_2 agree with the behavior of M on the word $u^R v^R$;
  - M accepts $uu^R v^R v$ (that is, $uu^R v^R v \in L_{pal2}$), and the computation of M on $uu^R v^R v$ contains a subcomputation of the form

    $$uu^R v^R v \vdash_M^* w_1' u^R v^R w_2'$$

    in which M only crosses the infix $u^R v^R$ as described in the transition tables T_1 and T_2.

The first element of the above description is of size at most $n - n/(8\log^4 n)$, the second and the third have constant size. The fourth element is of size logarithmic with respect to the size of the set of words equivalent with respect to M, w_1', w_2', T_1, and T_2. Hence, in order to derive a contradiction to the incompressibility of w, it is sufficient to show that the size of the set of words equivalent with respect to M, w_1', w_2', T_1, and T_2 is $o(|\Sigma|^{n/(8\log^4 n)})$. This will be shown below.

Proposition 6. *1. If $uv \in \Sigma^n$, $|u| = |v|$, is equivalent with respect to M, w_1', w_2', T_1, and T_2, then M accepts $w_1u^Rv^Rw_2$, that is, $w_1u^Rv^Rw_2 \in L_{pal2}$.*

2. There exists a constant c satisfying the following condition. Let n be sufficiently large, let $x = x_1x_2$, where $|x_1| = |x_2| = n/2$, be an incompressible word, and let $z = x_1yx_2 \in L_{pal2}$ for some $y \in \Sigma^n$. Then each factorization $z = z_1z_2$, where z_1, z_2 are palindromes, satisfies the condition $-c \leq |z_1| - n \leq c$.

Proof. (1) As $u^R v^R$ and $w_1^R w_2^R$ have the same transition tables T_1, T_2, starting from the initial configuration $q_0 ¢ w_1 u^R v^R w_2 \$$, M reaches the restarting configuration $q_0 ¢ w_1' u^R v^R w_2' \$$, from which it will finally accept, as $uu^R v^R v \vdash_M^* w_1' u^R v^R w_2'$ and $uu^R v^R v \in L_{pal2}$. Hence, $w_1 u^R v^R w_2 \in L_{pal2}$.

(2) Let $z = z_1 z_2$ be a factorization of $z = x_1 y x_2$ such that $|z_1| \neq |z_2|$, $|z_1| = 2m$, and z_1, z_2 are palindromes. Without loss of generality we can assume that $|z_1| < |z_2|$. Then $z_1 = v v^R$ for some word v satisfying $|v| = m < n/2 = |x_1|$. If $|z_1| = 2m \geq n/2 = |x_1|$, then this implies that $x_1 = v_0 v_1 v_1^R$ for some factorization $v = v_0 v_1$, where $|v_1| = n/2 - m$ and $v_0 = 2m - n/2$. Now we can encode $x = x_1 x_2$ by giving the following information:

- v_1, the length of v_1, and the additional information describing that v_1^R is a subword of x which ends at position $n/2$, where n is the length of x;
- $v_0 x_2$ together with the information that this is the remaining part of x.

Note that in this representation we save $|v_1|$ positions, but we have to add a description of how to combine v_1 and the remaining part of this encoding and of the length of v_1. However, this description takes size d plus at most $2 \log |v_1|$ many positions . Thus, if $|v_1| > 2 \log |v_1| + d$, we get a contradiction to the incompressibility of x. However, this inequality is satisfied each time $||z_1| - n| > c$ for some constant c.

Finally, if $|z_1| = 2m < n/2 = |x_1|$, then $z_1 = v v^R$ is a prefix of x_1, that is, $x_1 = v v^R x_3$ for some word x_3 of length $|x_3| = n/2 - 2m$. If $|z_1| > n/4$, then we can use this factorization in the same way as above to save $|v| = m > n/8$ many positions in the representation of x. On the other hand, if $|z_1| \leq n/4$, then $x_3 y x_2 = z_2$ being a palindrome implies that the suffix x_3 of x_1 of length $n/2 - |z_1| \geq n/4$ is the reversal of the suffix of x_2 of the corresponding length. Again this can be used to obtain a compression of x. □

Finally, we obtain the required result.

Proposition 7. *The size of the set of words equivalent with respect to M, w_1', w_2', T_1, and T_2 is bound by a constant.*

Proof. By the above proposition, the following condition is satisfied for each word uv (with $|u| = |v| = n/2$) equivalent with respect to M, w_1', w_2', T_1, and T_2:

$$w_1 u^R v^R w_2 \in L_{pal2}.$$

Moreover, if $z = w_1 u^R v^R w_2 \in L_{pal2}$, then there exists a factorization $z = z_1 z_2$ such that z_1, z_2 are palindromes and $||z_1| - |z_2|| \leq c$, where c is a constant independent of n.

We now show that, for each $c' \in \mathbb{N}$ such that $c' \leq c$, the number of words uv as above for which there exists a factorization of $w_1 u^R v^R w_2$ into $z_1 z_2$ as above such that $||z_1| - |z_2|| = c'$ is bound by a constant independent of n. This gives the intended result. Indeed, as c is a constant independent of n, we can assume that c' is much smaller than $n/2$. Thus, w_1 is a prefix of z_1, and w_2 is a suffix

of z_2. Assume that $|z_1| = |z_2| + c'$, that is, $|z_1| = n + c'/2$ and $|z_2| = n - c'/2$. As z_1 is a palindrome, we see that the suffix of length $n/2$ of z_1 is determined by its prefix w_1 of length $n/2$. Hence, only the part in the middle of z_1 of length $c'/2$ is not determined by w_1. On the other hand, z_2 is completely determined by its suffix w_2 of length $n/2$. Thus, the number of possible values for $u^R v^R$ is bound from above by the number $|\Sigma|^{c'}$. □

6.1 Proof of Proposition 4

Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a deterministic RLWW-automaton. From the results in [7, 15] it follows that there exists a deterministic RLWW-automaton M' that satisfies all of the following conditions:

1. $L(M') = L(M)$,
2. for all $x, y \in \Gamma^*$, $x \vdash^c_{M'} y$ if and only if $x \vdash^c_M y$, that is, M and M' execute exactly the same cycles, and
3. in each cycle, M' works in two *stages*. In the first stage it behaves like a deterministic one-way finite-state acceptor moving its window from the left sentinel ¢ to the right sentinel \$. In the second stage, it works in so-called *phases*, the first of which starts at the rightmost position. In a phase that starts at position i of the tape, M' executes first a number of MVL steps, then it performs the same number of MVR steps taking its window back to position i, and finally it makes another MVL step (to position $i - 1$), which ends this phase. Hence, M' does not move its window to the right of position i during the phase which starts at that position. These phases continue until M' finds a rewrite position at the end of some phase j, applies the required rewrite operation, and restarts immediately thereafter.

We will analyze the computations of the automaton M'. Let A_1 be a deterministic one-way finite-state acceptor that corresponds to the behaviour of M' during the first stage of each cycle, and let A_2 be a two-way finite-state acceptor that corresponds to the left-right-movement of M' during the phases of the second stage of each cycle, that is, starting at position i, A_2 performs a number of MVL steps, then the same number of MVR steps, taking its head back to position i.

To complete the proof of Proposition 4 we need a few technical results. The first of these deals with the behaviour of deterministic two-way finite-state acceptors.

Proposition 8. [5] *Let A be a deterministic two-way finite-state acceptor with p states and tape alphabet $\Gamma \supset \Sigma \neq \emptyset$. Then there is a word $b = b_L b_R \in \Sigma^*$ of length at most $2p^2$ such that the following conditions are satisfied:*

- *if the head of A enters b from the left, then it either*
  - *leaves b to the left without reaching b_R, or*
  - *it loops inside b_L, or*

  - *it leaves b to the right, and for no $x \in \Sigma^*$ does the head of A leave b to the left while working inside bx (but it may leave b to the left if it encounters a symbol which does not belong to Σ);*
- *if the head of A enters b from the right, then the corresponding properties hold.*

The next result states that each word v occurs as a subword in each random word of sufficient length.

Proposition 9. *Let $v \in \Sigma^c$ for some constant $c \in \mathbb{N}$. Then there exists an integer $n_0 \in \mathbb{N}$ such that, for each $n > n_0$, for each random word $w \in \Sigma^n$, and for each subword x of w of length $\log^2 n$, v is a subword of x.*

Proof. Let x be any subword of w of length $\log^2 n$, where w is a random word of length n. Assume that v does not appear in x at all. This means that x belongs to the following set

$$S := \{\, y \in \Sigma^{\log^2 n} \mid v \text{ does not appear at position } 1 + i \cdot c \text{ in } y \text{ for all } i = 0, 1, \ldots, ((\log^2 n)/c) - 1 \,\}.$$

Let s be the cardinality of Σ. Then the set S contains at most $(s^c - 1)^{(\log^2 n)/c}$ elements, while the set of all words of length $\log^2 n$ over Σ contains $s^{\log^2 n}$ elements. Thus, we can encode x by giving its number in the lexicographic ordering of the set S and by providing the word v (which determines the set S). This encoding takes space

$$\log((s^c - 1)^{(\log^2 n)/c}) + c = \frac{\log(s^c - 1)}{c} \cdot \log^2 n + c\,.$$

This encoding saves linear space with respect to the length of x, which is linear with respect to $\log^2 n$.

Finally, assume that $w = y_1 x y_2$. Then we can describe w by

(a) giving explicitely y_1, y_2, and their lengths,
(b) and by the above representation of x.

Item (a) needs some extra space, but only of logarithmic size, while item (b) allows to save space $c' \cdot \log^2 n$ for some constant c', which yields a contradiction to the assumption that w is random. □

The next result is concerned with the behaviour of a deterministic one-way finite-state acceptor on a random word of sufficient length.

Proposition 10. *Let A be a deterministic one-way finite-state acceptor with tape alphabet $\Gamma \supseteq \Sigma \neq \emptyset$. Then there exists a constant $n_0 \in \mathbb{N}$ such that, for each integer $n > n_0$, and each random word $w \in \Sigma^n$, the following condition is satisfied for each word $v \in \Sigma^+$:*

Assume that A is in state q when it enters wv from the left, and that it reaches state q' when its head is located inside v. Then A already encounters state q' while its head is still inside the prefix of w of length $\log^2 n$.

Proof. Assume that there exists a deterministic one-way finite-state acceptor $A = (Q, \Gamma, q_0, F, \delta)$ and a nonempty subalphabet $\Sigma \subseteq \Gamma$ for which the above statement does not hold, and take $p := |Q|$ and $s := |\Sigma|$. Then, for each integer $n_0 > 0$, there exists a random word $w \in \Sigma^+$ of length $|w| > n_0$ such that, for some word $v \in \Sigma^+$ and some states $q, q' \in Q$, when A is in state q when it enters the word wv from the left, then it reaches state q' when its head is located inside v, but this state is not encountered as long as the head is still inside the prefix of w of length $\log^2 n$.

Now let w be a sufficiently long random word satisfying the above condition, let $q, q' \in Q$ be the corresponding states, and let $v \in \Sigma^+$ be the corresponding right factor. Further, let $x := x_1 \dots x_{(\log^2 n)/p}$ be the prefix of w of length $\log^2 n$, where $|x_i| = p$ for each $i \in [1, (\log^2 n)/p]$. Note that state q' is accessible from each state q'' which A enters while its head is inside x. Hence, for each state q'' encountered in this way, there exists a word $u_{q''} \in \Sigma^p$ such that A reaches state q' with its head inside $u_{q''}$, when it starts its computation in state q'' with its head on the first letter of $u_{q''}$.

For each index $i = 1, \dots, (\log^2 n)/p$, let q_i be the state of A when its head enters the factor x_i. From $q = q_1$ and the above properties we can conclude that $x_1 \neq u_q$. Hence, the knowledge of q_1 allows us to describe x_1 through its number (according to the lexicographical ordering) in the set $S_1 := \Sigma^p \setminus \{u_q\}$. Unfortunately, due to the size of S_1 this description would not save any space at all. However, from this description we can determine the state q_2 and the set $S_2 := \Sigma^p \setminus \{u_{q_2}\}$ of possible values for the factor x_2. Further, the index of x_2 in the set S_2 allows to determine x_2 and the state q_3. Continuing in this way, the state q_i determines the set S_i, and the index of the word x_i in S_i determines the word x_i and the state q_{i+1}.

Hence, we can encode the prefix x through the state $q = q_1$ and the sequence of indices $i_1, \dots, i_{(\log^2 n)/p}$ such that i_j is the index of the factor x_j in the set S_j. In order to make this representation space efficient, we concatenate all these indices into a single code word $I := i_1 i_2 \dots i_{(\log^2 n)/p}$. As $i_j \in [1, s^p - 1]$, the set of possible values for I is of size $(s^p - 1)^{(\log^2 n)/p}$. Finally, we encode I by its index in the set of all possible values of I, which requires

$$\log(s^p - 1)^{(\log^2 n)/p} = \frac{\log(s^p - 1)}{p} \cdot \log^2 n$$

symbols. Recall that $q = q_1$ and I together describe x uniquely.

Based on the above description of x, we now construct an encoding of w by combining the following items:

- a description of A, the state $q = q_1$, and the description of the above method of encoding of x (which is of constant size);
- the above encoding of the index I of size $\frac{\log(s^p-1)}{p} \cdot \log^2 n$;
- the suffix u of w of length $n - \log^2 n$.

In order to combine these parts into a single word, we have to add information about the lengths of the first two parts (see [9] for self-delimiting descriptions),

which needs at most $8 \log \log n$ more positions. Thus, the length of our encoding of w is

$$c + 8 \log \log n + n - (1 - \frac{\log(s^p - 1)}{p}) \cdot \log^2 n = c + 8 \log \log n + n - c' \cdot \log^2 n,$$

where c and c' are constants. Note that $c' := (1 - \frac{\log(s^p-1)}{p})$ is a constant that is smaller than one. Thus, if we take n_0 to be the smallest integer for which

$$c + 8 \log \log n_0 < \frac{c' \cdot \log^2 n_0}{2}$$

holds, then, for each random word of length $n > n_0$, we obtain a compression by $\frac{c' \cdot \log^2 n}{2}$ positions. This, however, is more than $4 \log n$ for sufficiently large n, thus contradicting the assumption that w is random. □

Finally, we need the following result on deterministic two-way finite-state acceptors, which is easily proved by considering crossing sequences.

Proposition 11. *Let A be a deterministic two-way finite-state acceptor with tape alphabet Γ, let Q_A be the set of states of A, and let $q, q' \in Q_A$. Assume that there exists a word $u = u_1 u_2$ such that, starting from the configuration qu, A reaches state q' while its head is inside u_2, without moving its head outside of u during this computation. Then there exists a word u_2' of length $4 \cdot |Q_A|!$ such that, starting from the configuration qu_1u_2', A reaches state q' while its head is inside u_2', without moving its head outside of u_1u_2' during this computation.*

The analogous property holds for the symmetric case that A enters u from the right and reaches state q' while its head is inside u_1.

Now we return to the proof of Proposition 4. Let $b_{M'}$ be a word that satisfies the conditions of Proposition 8 for the automaton A_2, and let $n_{M'}$ be the constant from Proposition 9 corresponding to the word $b_{M'}$. Thus, if w is any random word of length $|w| > n_{M'}$, then $b_{M'}$ occurs in each subword of w of length $\log^2 n$.

Let $u_1, u_2, v \in \Sigma^*$ and $u_1', u_2' \in \Gamma^*$ such that $|u_1| = |u_2| = n > n_{M'}$ and $K(u_1), K(u_2) > n - 4 \log n$, that is, u_1 and u_2 are random words. Further, we assume that the integer n is sufficiently large as to satisfy the conditions of Proposition 10.

We will analyze the cycle of M' that begins with the tape content $u_1'u_1vu_2u_2'$. Let $u_2 = y_2x_2$, where x_2 is the suffix of u_2 of length $\log^4 n$, and let $u_1 = x_1y_1$, where x_1 is the prefix of u_1 of length $\log^4 n$. If M' performs a rewrite step inside the prefix $u_1'x_1$ or inside the suffix x_2u_2', then there is nothing to prove. So assume that M' performs the rewrite step of the current cycle neither inside $u_1'x_1$ nor inside x_2u_2'. Based on this assumption we will now derive a short description for u_2, which will contradict the fact that u_2 is a random word.

Assume that M' has just finished the phase of the second stage for the leftmost position of u_2'. Let $x_2 = x_{2,m}x_{2,m-1} \dots x_{2,1}$, where $m = (\log^2 n)/2$ and $|x_{2,i}| = 2 \log^2 n$ for each $i \in [1, m]$. From Proposition 9 we conclude that, for each $i \in [1, m]$, $b_{M'}$ occurs as a subword in the prefix of length $\log^2 n$ of $x_{2,i}$ as

well as in the suffix of length $\log^2 n$ of $x_{2,i}$. From the fact that M' finds a rewrite position of M we see that there exists a particular state of M' that M' does not enter before during the current cycle.

To complete the proof we need the following additional propositions.

Proposition 12. *The read-write window of M' does not reach the part of the tape containing u'_1 during any phase of the second stage, which corresponds to a position of x_2.*

Proof. According to our assumption the rewrite position of the current cycle is inside the factor y_1vy_2. Hence, the current cycle of M' ends with a phase that corresponds to some position inside y_1vy_2. Assume that the head of A_2 reaches u'_1 during a phase that corresponds to a position inside x_2. Then in the second part of this phase, in which A_2 performs MVR steps, it is working as a deterministic one-way finite-state acceptor, moving across the complete factor x_1. During this part of the computation A_2 does not enter the state which indicates the end of the phase. However, as u_1 is a random word of sufficient length, this contradicts Proposition 10. □

Proposition 13. *For each factorization of x_2 of the form $x_2 = yb_{M'}y'$, the window of M' does not reach the factor y during the second stage of the current cycle before the phase that starts at the rightmost position of $b_{M'}$.*

Proof. Each time the window of M' reaches the factor $b_{M'}$ during the second stage before the phase that starts at the rightmost position of $b_{M'}$, M' is running the automaton A_2. According to the choice of $b_{M'}$, if A_2 leaves $b_{M'}$ to the left after entering it from the right, there is no possibility for A_2 to cross this particular factor $b_{M'}$ again from left to right as long as it works on a word over the alphabet Σ (see Proposition 8). However, we see from Proposition 12 that during this stage M' will not reach the factor u'_1, which is the only prefix of the current tape content containing non-input symbols. □

Now we describe the compression of the word u_2:

1. The factor y_2 is stored.
2. For $j \in [2, m]$, let $x_{2,j} = x'z_jb_{M'}x''$, where $|z_j| = 4 \cdot |Q|!$, and x'' does not contain $b_{M'}$ as a factor. Recall that $x_{2,j}$ contains at least two occurrences of this factor. Of $x_{2,j}$, we only store $x'b_{M'}x''$, that is, we do not store the factor z_j.
3. The above factors z_j ($j \in [2, m]$) are encoded as follows. According to Proposition 13, M' does not move its window to z_j until it executes the phase that corresponds to the rightmost position of the occurrence of $b_{M'}$ following the factor z_j. Further, we know that starting with this phase M' will finally reach the state q' that expresses the fact that the rewrite position of the current cycle has been found. Let q_j be the state of M' at the beginning of the phase that corresponds to the rightmost position of this particular occurrence of $b_{M'}$. By Proposition 11 there exists a word $\tilde{z}_j$ of length $4 \cdot |Q|! = |z_j|$ such that, starting in state q_j at the last position of the word $\tilde{z}_jb_{M'}$, M' reaches

state q' without leaving the part of the tape containing the word $\tilde{z}_j b_{M'}$. It follows in particular that $z_j \neq \tilde{z}_j$, as M' reaches state q' only outside of the part of the tape containing the word $x_{2,j}$.

Let $z := z_1 z_2 \dots z_m$. If we knew the states $q_1, q_2, \dots, q_m$, then we would be able to determine the words $\tilde{z}_j$ ($j \in [1, m]$). Let $s := |\Sigma|$ and $r := |z_j| = 4 \cdot |Q|!$. Then the word z above belongs to the set Z of size $(s^r - 1)^m$ of words of length $r \cdot m$. Now we encode z by its number in the lexicographical ordering of this set. This encoding has length $\log_s (s^r - 1)^m$, which is smaller than the length of z by $d \cdot \log^2 n$, where $d := \frac{1}{2}(r - \log_s (s^r - 1))$ is a constant that is independent of n, as $m = (\log^2 n)/2$.

Such a representation of u_2, which saves $d \cdot \log^2 n$ symbols, would contradict the assumption that u_2 is random. However, we need to know the states $q_1, q_2, \dots, q_m$. Now we show how to obtain this encoding without the knowledge of these states. Let q_0 be the state of M' at the beginning of the phase corresponding to the rightmost position of x_2. Starting from this state at the rightmost position of x_2, M' will not leave the part of the tape containing the suffix of x_2 including the rightmost occurrence of $b_{M'}$ until it starts the phase that corresponds to the rightmost position of this particular factor $b_{M'}$. Thus, we are able to determine q_1, which allows us to encode the factor z_1 and to continue with the computation. Similarly, knowing q_i we are able to encode z_i and to determine the state q_{i+1} for all $i \in [2, m]$.

In summary, we have obtained a representation of u_2 of size $|u_2| - d \cdot \log^2 n + O(\log n)$. The additional $O(\log n)$ factor follows from the fact that we have to include information about the lengths of the consecutive parts of the above encoding. However, if n is sufficiently large, the obtained compression of u_2 contradicts our assumption that u_2 is random, which completes the proof of Proposition 4.

Acknowledgement. The authors thank František Mráz from Charles University, Prague, for many very interesting and fruitful discussions on restarting automata in general and the topic of this paper in particular.

References

1. E. Dahlhaus and M. Warmuth. Membership for growing context-sensitive grammars is polynomial. *Journal of Computer and System Sciences* 33 (1986) 456–472.
2. L.A. Hemaspaandra, P. Mukherji and T. Tantau. Overhead-Free Computation, DCFLs, and CFLs. CoRR (The Computing Research Repository) cs.CC/0410035: (2004). A preliminary version appeared in: Z. Ésik and Z. Fülöp (eds.), *DLT'03, Proc.*, *LNCS 2710*, pages 325–336. Springer, Berlin, 2003.
3. P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In: H. Reichel (ed.), *FCT'95, Proc.*, *LNCS 965*, pages 283–292. Springer, Berlin, 1995.
4. P. Jančar, F. Mráz, M. Plátek and J. Vogel. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics* 4 (1999) 283-292.

5. T. Jurdzinski, M. Kutylowski and K. Lorys. Multi-party finite computations. In: T. Asano, H. Imai, D.T. Lee, Sh. Nakano and T. Tokuyama (eds.), *COCOON'99, Proc.*, *LNCS 1627*, pages 318–329. Springer, Berlin, 1999.
6. T. Jurdziński, K. Loryś, G. Niemann and F. Otto. Some results on RWW- and RRWW-automata and their relationship to the class of growing context-sensitive languages. *Mathematische Schriften Kassel* 14/01, Universität Kassel, 2001. Also: *Journal of Automata, Languages and Combinatorics*, to appear.
7. T. Jurdziński, F. Mráz, M. Plátek and F. Otto. Deterministic two-way restarting automata don't need auxiliary symbols if they are (right-, left, or right-left-) monotone. *Mathematische Schriften Kassel* 7/04, Universität Kassel, 2004.
8. T. Jurdziński, F. Otto, F. Mráz and M. Plátek. On the complexity of 2-monotone restarting automata. *Mathematische Schriften Kassel* 4/04, Universität Kassel, 2004. An extended abstract appeared in: C. Calude, E. Calude and M. Dinneen (eds.), *DLT'04, Proc.*, *LNCS 3340*, pages 237–248. Springer, Berlin, 2004.
9. M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.
10. R. McNaughton, P. Narendran and F. Otto. Church-Rosser Thue systems and formal languages. *Journal of the Association for Computing Machinery* 35 (1988) 324–344.
11. G. Niemann and F. Otto. Restarting automata, Church-Rosser languages, and representations of r.e. languages. In: G. Rozenberg and W. Thomas (eds.), *Developments in Language Theory - Foundations, Applications, and Perspectives, DLT'99, Proc.*, pages 103–114. World Scientific, Singapore, 2000.
12. G. Niemann and F. Otto. Further results on restarting automata. In: M. Ito and T. Imaoka (eds.), *Words, Languages and Combinatorics III, Proc.*, pages 352–369. World Scientific, Singapore, 2003.
13. K. Oliva, P. Květoň and R. Ondruška. The computational complexity of rule-based part-of-speech tagging. In: V. Matoušek and P. Mautner (eds.), *TSD 2003, Proc.*, *LNCS 2807*, pages 82–89. Springer, Berlin, 2003.
14. F. Otto. Restarting Automata - Notes for a Course at the 3rd International PhD School in Formal Languages and Applications. *Mathematische Schriften Kassel* 6/04, Universität Kassel, 2004.
15. F. Otto and T. Jurdziński. On left-monotone restarting automata. *Mathematische Schriften Kassel* 17/03, Universität Kassel, 2003. An extended abstract appeared in: C. Calude, E. Calude and M. Dinneen (eds.), *DLT'04, Proc.*, *LNCS 3340*, pages 249–260. Springer, Berlin, 2004.
16. M. Plátek. Two-way restarting automata and j-monotonicity. In: L. Pacholski and P. Ružička (eds.), *SOFSEM'2001, Proc.*, *LNCS 2234*, pages 316–325. Springer, Berlin, 2001.
17. M. Plátek, M. Lopatková and K. Oliva. Restarting automata: motivations and applications. In: M. Holzer (ed.), *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten', Proc.*, pages 90–96. Institut für Informatik, Technische Universität München, 2003.
18. M. Plátek, F. Otto, F. Mráz and T. Jurdziński. Restarting automata and variants of j-monotonicity. *Mathematische Schriften Kassel* 9/03, Universität Kassel, 2003.
19. A. Salomaa. *Formal Languages*. Academic Press, 1973.