Descriptional Complexity of Limited Automata

Martin Kutrib[a], Giovanni Pighizzini[b], Matthias Wendlandt[a]

[a] *Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany*
[b] *Dipartimento di Informatica, Università degli Studi di Milano Via Comelico 39, 20135 Milano, Italy*

Abstract

A k-limited automaton is a linear bounded automaton that may rewrite each tape cell only in the first k visits, where $k \geq 0$ is a fixed constant. It is known that these automata accept context-free languages only. We investigate the descriptional complexity of limited automata. Since the unary languages accepted are necessarily regular, we first study the cost in the number of states when finite automata simulate a unary k-limited automaton. For the conversion of a $4n$-state deterministic 1-limited automaton into one-way or two-way deterministic or nondeterministic finite automata, we show a lower bound of $n \cdot F(n)$ states, where F denotes Landau's function. So, even the ability to deterministically rewrite any cell only once gives an enormous descriptional power. For the simulation cost for removing the ability to rewrite each cell $k \geq 1$ times, more precisely, the cost for the simulation of sweeping unary k-limited automata by deterministic finite automata, we obtain a lower bound of $n \cdot F(n)^k$. The upper bound of the cost for the simulation by two-way deterministic finite automata is a polynomial whose degree is quadratic in k. If the k-limited automaton is rotating, the upper bound reduces to $O(n^{k+1})$ and the lower bound derived is $\Omega(n^{k+1})$ even for nondeterministic two-way finite automata. So, for rotating k-limited automata, the trade-off for the simulation is tight in the order of magnitude. Finally, we consider the simulation of k-limited automata over general alphabets by pushdown automata. It turns out that the cost is an exponential blow-up of the size. Furthermore, an exponential size is also necessary.

Keywords: Limited automata, descriptional complexity, simulation, finite automata, pushdown automata, unary languages.

1. Introduction

The cost for the simulation of one formal model by another is one of the main topics of descriptional complexity, where the cost is measured in close connec-

Email addresses: kutrib@informatik.uni-giessen.de (Martin Kutrib), pighizzini@di.unimi.it (Giovanni Pighizzini), matthias.wendlandt@informatik.uni-giessen.de (Matthias Wendlandt)

tion to the sizes of the models. Such simulations are of particular interest when both formal models capture the same family of languages. A fundamental result is that nondeterministic finite automata can be simulated by deterministic finite automata by paying the cost of exponentially many states (see, for example, [16]). Among the many models characterizing the regular languages, an interesting variant is the linear bounded automata where the time is restricted as well. It was shown by Hennie [5] that linear-time computations cannot accept non-regular languages. This result has been improved to $o(n \log n)$ time by Hartmanis [4]. In particular, the former result implies that a linear bounded automaton where any tape cell may be visited only a constant number of times accepts a regular language. Recent results [24] showed that the upper as well as the lower bound for converting a weight-reducing machine of this type, that is, each transition is required to lower the weight of the scanned symbol, into a deterministic finite automaton is doubly exponential. A related result [1] showed that if a two-way finite automaton is allowed to freely place a pebble on the tape, then again no non-regular language can be accepted, even if the time is unlimited. Doubly exponential upper and a lower bounds for the simulation by a deterministic finite automaton have been derived [16].

A variant of the machines studied by Hennie [5] was introduced by Hibbard [6]. He investigated linear bounded automata that may rewrite each tape cell only in the first k visits, where k is a fixed constant. However, afterwards the cells can still be visited any number of times (but without rewriting their contents). Hibbard [6] showed that the nondeterministic variant characterizes the context-free languages provided $k \geq 2$, while there is a tight and strict hierarchy of language classes depending on k for the deterministic variant. The latter means that the family of languages accepted with k rewrites is strictly included in the family of languages accepted with $k+1$ rewrites. One-limited automata, deterministic and nondeterministic, can accept only regular languages. From these results it follows that any *unary* k-limited automaton accepts a regular language only.

Recently, the study of limited automata from the descriptional complexity point of view has been initiated by Pighizzini and Pisoni [20, 21]. In [21] it was shown that the deterministic 2-limited automata characterize the deterministic context-free languages, which complements the result on nondeterministic machines. Furthermore, conversions between 2-limited automata and pushdown automata have been investigated. For the deterministic case the upper bound for the conversion from 2-limited automata to pushdown automata is doubly exponential. Conversely, the trade-off is shown to be polynomial. If the automata are nondeterministic, exponential upper and lower bounds are derived for the 2-limited automata to pushdown automata conversion. Comparisons between 1-limited automata and finite automata were done in [20]. In particular, a double exponential trade-off between nondeterministic 1-limited automata and one-way deterministic finite automata was shown. For deterministic 1-limited automata the conversion cost a single exponential increase in size. These results imply an exponential trade-off between nondeterministic and deterministic 1-limited automata, and they show that 1-limited automata can have less states

than equivalent two-way nondeterministic finite automata.

For a restricted variant of limited automata, so-called strongly limited automata, it was shown that context-free grammars as well as pushdown automata can be transformed in strongly limited automata and vice versa with polynomial cost [19].

Here, we first consider deterministic k-limited automata accepting unary languages. The descriptional complexity of unary regular languages has been studied in many ways. On one hand, many automata models such as one-way finite automata, two-way finite automata, pushdown automata, or context-free grammars for unary languages were investigated and compared to each other with respect to simulation results and the size of the simulation (see, for example, [3, 15, 18, 23]). On the other hand, many results concerning the state complexity of operations on unary languages have been obtained (see, for example, [7, 10, 14, 22]).

The results on the expressive power of limited automata imply that any unary language accepted by some k-limited automaton is regular. So, it is of interest to investigate the descriptional complexity in comparison with the models mentioned above. We establish upper and lower bounds for the conversion of unary deterministic k-limited automata to one-way and two-way finite automata. Moreover, the simulation of general k-limited automata by pushdown automata is considered. It turns out that the cost is an exponential blow-up of the size. From the case of 2-limited automata [21], it turns out that an exponential gap is also necessary.

2. Preliminaries

We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

Let $k \geq 0$ be an integer. A k-limited automaton is a restricted linear bounded automaton. It consists of a finite state control and a read-write tape whose initial content is the input word in between two endmarkers. At the outset of a computation, the automaton is in the designated initial state and the head of the tape scans the left endmarker. Depending on the current state and the currently scanned symbol on the tape, the automaton changes its state, rewrites the current symbol on the tape, and moves the head one cell to the left or one cell to the right. However, the rewriting is restricted such that the machine may rewrite each tape cell only in the first k visits. Subsequently, the cell can still be scanned but the content cannot be changed any longer. So, a 0-limited automaton is a two-way finite automaton. An input is accepted if the machine reaches an accepting state and halts.

The original definition of such devices by Hibbard [6] is based on string rewriting systems whose sentential forms are seen as configurations of automata. Let $u_1u_2\cdots u_{i-1}su_iu_{i+1}\cdots u_n$ be a sentential form that represents the tape contents $u_1u_2\cdots u_n$ and the current state s. Basically, in [6] rewriting rules

were provided of the form $su_i \to u'_i s'$, which means that the state changes from s to s', the tape cell to the right of s is scanned and rewritten from u_i to u'_i, the input head is moved to the right, and $u_{i-1}s \to s'u'_{i-1}$, which means that the state changes from s to s', the tape cell to the left of s is scanned and rewritten from u_{i-1} to u'_{i-1}, the input head is moved to the left. In this context, an automaton that changes its head direction on a cell scans the cell twice. By Pighizzini and Pisoni [20, 21] and below, limited automata are defined in a way that reflects this behavior.

Formally, a *(nondeterministic) k-limited automaton* (k-LA, for short) is a system $M = \langle S, \Sigma, \Gamma, \delta, \rhd, \lhd, s_0, F\rangle$, where S is the finite, nonempty set of *internal states*, Σ is the finite set of *input symbols*, Γ is the finite set of *tape symbols* partitioned into $\Gamma_k \cup \Gamma_{k-1} \cup \cdots \cup \Gamma_0$ where $\Gamma_0 = \Sigma$, $\rhd \notin \Gamma$ is the *left endmarker* and $\lhd \notin \Gamma$ is the *right endmarker*, $s_0 \in S$ is the *initial state*, $F \subseteq S$ is the set of *accepting states*, and $\delta : S \times (\Gamma \cup \{\rhd, \lhd\}) \to 2^{S \times (\Gamma \cup \{\rhd, \lhd\}) \times \{-1,1\}}$ is the transition function, where -1 means to move the head one cell to the left, 1 means to move it one cell to the right, and whenever $(s', y, d) \in \delta(s, \rhd)$ it holds $y = \rhd$, $d = 1$ and whenever $(s', y, d) \in \delta(s, \lhd)$ it holds $y = \lhd$, $d = -1$.

In order to implement the limited number of rewrite operations, δ is required to satisfy the following condition. For each $(s', y, d) \in \delta(s, x)$ with $x \in \Gamma_i$:

(1) if $i = k$ then $x = y$,

(2) if $i < k$ and $d = 1$ then $y \in \Gamma_j$ with $j = \min\{\lceil \frac{i}{2} \rceil \cdot 2 + 1, k\}$, and

(3) if $i < k$ and $d = -1$ then $y \in \Gamma_j$ with $j = \min\{\lceil \frac{i+1}{2} \rceil \cdot 2, k\}$.

It is worth mentioning that these conditions make the a priori global condition of a head turn on some cell local, until the cell cannot be rewritten anymore. The clever transformation of the original definition to the automata world [20, 21] gives that, if a cell content is from Γ_i then the head position is always to the right of that cell if i is odd, and it is to the left of the cell if i is even, as long as $i < k$.

A *configuration* of the k-LA M is a triple (s, v, h), where $s \in S$ is the current state, $v \in \rhd\Gamma^*\lhd$ is the current tape content, and $h \in \{0, 1, \ldots, |w| + 1\}$ gives the current head position, where w is the input word. If h is 0, the head scans the symbol $\rhd$, and if it is $|w| + 1$, then the head scans the symbol $\lhd$. The *initial configuration* for input w is set to $(s_0, \rhd w \lhd, 0)$. During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $n = |w|$, $a_0 = \rhd$, and $a_{n+1} = \lhd$. Then we set $(s, \rhd a_1 a_2 \cdots a_h \cdots a_n \lhd, h) \vdash (s', \rhd a_1 a_2 \cdots a'_h \cdots a_n \lhd, h + d)$ if and only if $(s', a'_h, d) \in \delta(s, a_h)$.

A k-LA *halts*, if the transition function is undefined for the current configuration. An input is *accepted* if in some computation the automaton halts at some time in an accepting state, otherwise it is rejected. The *language* $L(M)$ *accepted* by M is the set of all accepted inputs.

A k-LA is said to be *sweeping* if the direction of the head movement changes only on the endmarkers. It is *deterministic* (k-DLA, for short) when each

configuration has at most one successor, that is, $\delta : S \times (\Gamma \cup \{\rhd, \lhd\}) \to S \times (\Gamma \cup \{\rhd, \lhd\}) \times \{-1, 1\}$ is a partial function.

In order to clarify the notions we continue with an example that is later used for lower bounds.

Example 1. Let $k \geq 1$ and $n \geq 2$. The finite unary language $L = \{a^{n^{k+1}}\}$, which consists of one word only, is accepted by a sweeping k-limited automaton $M = \langle S, \{a\}, \Gamma, \delta, \rhd, \lhd, s_0, F\rangle$ with $n+2$ states and $2k+1$ tape symbols.

The principal idea of the construction is to sweep k times across the tape, where in each sweep $n-1$ out of every n non-marked symbols are marked. In this way, in the mth sweep it is checked whether the number of input symbols is a multiple of n^m. In the final $(k+1)$st sweep it is checked whether exactly n non-marked symbols exist. If yes, the length ℓ of the input satisfies $\frac{\ell}{n^k} = n$ which implies $\ell = n^{k+1}$.

Formally, we set $S = \{s_0, s_1, \ldots, s_{n-1}, s_+, s_-\}$, $F = \{s_+\}$, and the tape alphabet $\Gamma = \{a, a_1, a'_1, a_2, a'_2, \ldots, a_k, a'_k\}$.

Whenever the head reaches an endmarker, M has to be in state s_0. Otherwise the computation halts and rejects. In this way it is verified whether the input length is a multiple of n^m.

1. $\delta(s_0, \rhd) = (s_0, \rhd, 1)$
2. $\delta(s_0, \lhd) = (s_0, \lhd, -1)$

The first sweep is realized by Transitions 3 and 4.

3. $\delta(s_i, a) = (s_{i+1}, a'_1, 1), \quad$ for $0 \leq i \leq n-2$
4. $\delta(s_{n-1}, a) = (s_0, a_1, 1)$

For sweep m with $2 \leq m \leq k$, Transitions 5 to 7 are used. Let $d = 1$ if m is odd and $d = -1$ if m is even.

5. $\delta(s_i, a_{m-1}) = (s_{i+1}, a'_m, d), \quad$ for $0 \leq i \leq n-2$
6. $\delta(s_{n-1}, a_{m-1}) = (s_0, a_m, d)$
7. $\delta(s_i, a'_{m-1}) = (s_i, a'_m, d), \quad$ for $0 \leq i \leq n-1$

Finally, in the $(k+1)$st sweep the states are reused in the same way to count up to n unmarked symbols. But after the first cycle state s_+ is entered instead of state s_0. If M reaches another unmarked symbol in state s_+, it rejects. Otherwise the computation halts and accepts on the endmarker in state s_+. Let $d = 1$ if $k+1$ is odd and $d = -1$ if $k+1$ is even.

8. $\delta(s_i, a_k) = (s_{i+1}, a_k, d), \quad$ for $0 \leq i \leq n-2$
9. $\delta(s_{n-1}, a_k) = (s_+, a_k, d)$
10. $\delta(s_i, a'_k) = (s_i, a'_k, d), \quad$ for $0 \leq i \leq n-1$
11. $\delta(s_+, a'_k) = (s_+, a'_k, d)$
12. $\delta(s_+, a_k) = (s_-, a_k, d)$ ■

As is often the case in connection with unary languages, Landau's function

$$F(n) = \max\{\mathrm{lcm}(c_1, c_2, \ldots, c_l) \mid l \geq 1, c_1, c_2, \ldots, c_l \geq 1 \text{ and } c_1 + c_2 + \cdots + c_l = n\}$$

plays a crucial role, where lcm denotes the *least common multiple*. It is well known that the c_i can always be chosen to be relatively prime. Moreover, an easy consequence of the definition is that the c_i can always be chosen such that $c_1, c_2, \ldots, c_l \geq 2$, $c_1 + c_2 + \cdots + c_l \leq n$, and $\mathrm{lcm}(c_1, c_2, \ldots, c_l) = F(n)$ (cf., for example, [17]). Since F depends on the irregular distribution of the prime numbers, we cannot expect to express $F(n)$ explicitly by n. The function itself was investigated by Landau [12, 13] who proved the asymptotic growth rate $\lim_{n\to\infty} \frac{\ln(F(n))}{\sqrt{n \cdot \ln(n)}} = 1$. The upper and lower bounds $F(n) \in e^{\sqrt{n \cdot \ln(n)}(1+o(1))}$ and $F(n) \in \Omega\left(e^{\sqrt{n \cdot \ln(n)}}\right)$ have been derived in [2, 27].

3. Simulation Cost of 1-DLA

We start with simulations of unary 1-DLA by finite automata. Upper bounds for general regular languages have been obtained as follows [20]. Any n-state 1-DLA can be simulated by a one-way deterministic finite automaton (1DFA) with no more than $n \cdot (n+1)^n$ states. The currently best lower bound for the simulations of *unary* 1-DLA by two-way nondeterministic finite automata (2NFA) was also obtained in [20], where it was shown that for infinitely many integers n there is a unary regular language recognized by an n-state, 3-tape-symbol 1-DLA such that each equivalent 2NFA requires a number of states which is quadratic in n. The next theorem improves this lower bound. It is worth mentioning that, to this end, the number of tape symbols is set to $n + 2$.

Theorem 2. *Let $n \geq 2$ be an integer. Then there is a unary $4n$-state and $(n+2)$-tape-symbol* 1-DLA *M, such that $n \cdot F(n)$ states are necessary for any* 2NFA *to accept the language $L(M)$.*

Proof. As mentioned above, there are positive integers $c_1, c_2, \ldots, c_l \geq 2$ such that $c_1 + c_2 + \cdots + c_l \leq n$ and $\mathrm{lcm}(c_1, c_2, \ldots, c_l) = F(n)$. The witness language is $L = \{a^{n \cdot F(n)}\}$. We construct a 1-DLA $M = \langle S, \{a\}, \Gamma, \delta, \triangleright, \triangleleft, s_0, \{p_+\}\rangle$ accepting L with at most $4n$ states and $n + 2$ tape symbols, where

$$\begin{aligned} \Gamma &= \{a, \sqcup\} \cup \{\, t_{i,j} \mid 1 \leq i \leq l, 0 \leq j \leq c_i - 1 \,\}, \text{ and} \\ S &= \{s_0, p_+, p_-, r_0, \ldots, r_{n-2}, r'_0, \ldots, r'_{n-3}, q_{l+1}, \ldots, q_n\} \\ &\quad \cup \{\, s_{i,j} \mid 1 \leq i \leq l, 0 \leq j \leq c_i - 1 \,\}. \end{aligned}$$

An input is accepted only if its length is a multiple of n. We consider the input to be partitioned into blocks of length n. The 1-DLA rewrites the input symbols in each block $xn+1, xn+2, \ldots, xn+n$, $x \geq 0$, by $t_{1,j_1}, t_{2,j_2}, \ldots, t_{l,j_l}, \sqcup, \ldots, \sqcup$, where $j_1 = (x+1) \bmod c_1, \ldots, j_l = (x+1) \bmod c_l$. So, a tape symbol $t_{i,j}$ occurring in the mth block means that $m \bmod c_i = j$. For each block this rewriting

is successive from left to right. To this end, the states $s_{i,j}$ are used. The idea of this part of the construction is as follows. When one of the first $l-1$ cells, say cell ℓ, of the new block has been rewritten (Transition 3), states r_h and r'_h are used to move the head back to cell $\ell+1$ of the previous block. Basically, state r_h or r'_h means to move the head back for another h cells. The content of cell $\ell+1$ of the previous block is then used to continue the block counting modulo $c_{\ell+1}$ (by states $s_{\ell+1,j}$, Transition 2) until the next still unwritten cell is reached (Transition 3). This is cell $\ell+1$ of the new block. After rewriting cell l of a block the states q_i are used to write the symbol $\sqcup$ to the remaining cells of the block (Transitions 4 and 5). Afterwards, states r_h and r'_h are again used to start the rewriting of the next block (Transition 6). Further roles played by the states r_h and r'_h are explained below. Let $y \in \Gamma \setminus \{\triangleleft, a\}$.

1. $\delta(s_0, \triangleright) = (s_{1,0}, \triangleright, 1)$
2. $\delta(s_{i,j}, y) = (s_{i,j}, y, 1)$, for $1 \le i \le l$ and $0 \le j \le c_i - 1$
3. $\delta(s_{i,j}, a) = (r_{n-2}, t_{i,(j+1) \bmod c_i}, -1)$, for $1 \le i \le l-1$ and $0 \le j \le c_i - 1$
4. $\delta(s_{l,j}, a) = (q_{l+1}, t_{l,(j+1) \bmod c_l}, 1)$, for $0 \le j \le c_l - 1$
5. $\delta(q_i, a) = (q_{i+1}, \sqcup, 1)$, for $l+1 \le i \le n-1$
6. $\delta(q_n, a) = (r_{n-2}, \sqcup, -1)$

The very first block is treated differently, since there is no predecessor block. However, whenever the head is moved back to the left endmarker, the index of states r_h and r'_h says how to continue the counting (Transitions 7 and 8).

7. $\delta(r_{n-i}, \triangleright) = (s_{i,0}, \triangleright, 1)$, for $2 \le i \le l$
8. $\delta(r'_{n-i}, \triangleright) = (s_{i,0}, \triangleright, 1)$, for $2 \le i \le l$

Now we turn to the end of the computation and the roles played by the states r_h and r'_h. Let w be the input. Its length $|w|$ is a multiple of n if and only if there is no partial block at the end. In addition, it is a multiple of $n \cdot F(n)$ if and only if $\frac{|w|}{n}$ is divisible by all the c_i, that is, $\frac{|w|}{n} \bmod c_i = 0$. In order to test whether the length of the input up to and including the current block is a multiple of $n \cdot F(n)$, it is sufficient to inspect the first l cells of the block. The test is positive if the content of cell ℓ is $t_{\ell,0}$, for all $1 \le \ell \le l$. This test is performed while M is in states r_h and r'_h, which move the head to the left. Moreover, since only the shortest input that meets the criteria may be accepted, M remembers a negative test result by changing from some state r_h to a primed version r'_{h-1} (Transitions 11, 12). Once in a primed state the head is moved back without further tests (Transition 13).

9. $\delta(r_h, \sqcup) = (r_{h-1}, \sqcup, -1)$, for $1 \le h \le n-2$
10. $\delta(r'_h, \sqcup) = (r'_{h-1}, \sqcup, -1)$, for $1 \le h \le n-3$
11. $\delta(r_h, t_{i,0}) = (r_{h-1}, t_{i,0}, -1)$, for $1 \le h \le n-2$, $1 \le i \le l$
12. $\delta(r_h, t_{i,j}) = (r'_{h-1}, t_{i,j}, -1)$,
    if $1 \le j \le c_i - 1$, for $1 \le h \le n-2$, $1 \le i \le l$
13. $\delta(r'_h, t_{i,j}) = (r'_{h-1}, t_{i,j}, -1)$, for $1 \le h \le n-3$, $1 \le i \le l$, $0 \le j \le c_i - 1$

When the head reaches its destination, M is in state r_0 or r'_0. If the destination is not the first cell of the block or the test was negative, M takes the cell contents to continue the counting (Transitions 14 and 15). If the destination is the first cell of the block and the test was positive, the first cell is tested as well. Depending on the result, either the rewriting of the next block is started (Transition 16) or state p_+ is entered (Transition 17).

14. $\delta(r_0, t_{i,j}) = (s_{i,j}, t_{i,j}, 1)$, for $2 \leq i \leq l$, $0 \leq j \leq c_i - 1$
15. $\delta(r'_0, t_{i,j}) = (s_{i,j}, t_{i,j}, 1)$, for $1 \leq i \leq l$, $0 \leq j \leq c_i - 1$
16. $\delta(r_0, t_{1,j}) = (s_{1,j}, t_{1,j}, 1)$, for $1 \leq j \leq c_1 - 1$
17. $\delta(r_0, t_{1,0}) = (p_+, t_{1,0}, 1)$

Once in state p_+ it is known that the input length, up to and including the current block, is the least multiple of $n \cdot F(n)$. So, it remains to be tested that there is no further input symbol a to the right of the block. By Transition 18 the head is moved to the right as long as there appears neither the input symbol a nor the right endmarker. If the right endmarker appears, the computation halts in the accepting state p_+. If there is a further a to the right of the current block, the rejecting state p_- is entered and the computation halts (Transition 19). Let $y \in \Gamma \setminus \{\lhd, a\}$.

18. $\delta(p_+, y) = (p_+, y, 1)$
19. $\delta(p_+, a) = (p_-, a, 1)$

From the construction it follows that M accepts the shortest input that is a multiple of $n \cdot F(n)$, that is, it accepts $a^{n \cdot F(n)}$. The numbers of states and tape symbols claimed follow also from the construction. So, it remains to be verified that no further inputs are accepted by M.

The only possibility to accept is in state p_+ on the right endmarker. Since state p_+ is entered only when the head is on the first cell of a block after the test was positive, we derive that the input is a multiple of $n \cdot F(n)$. Since there is no transition leading from state p_+ to any other state except for p_-, it follows that the input is the shortest word which is a multiple of $n \cdot F(n)$ and, thus, $L(M) = \{a^{n \cdot F(n)}\}$.

Finally, any two-way nondeterministic finite automaton that accepts a unary singleton language needs as least as many states as the length of the sole word in the language. □

Since even a 2NFA needs at least n states to accept the unary singleton language $\{a^n\}$, Theorem 2 reveals the same lower bound for one-way and two-way deterministic and nondeterministic finite automata.

Corollary 3. *Let $n \geq 2$ be an integer. Then there is a unary $4n$-state and $(n+2)$-tape-symbol* 1-DLA *M, such that $n \cdot F(n)$ states are necessary for any* 2DFA*,* 1DFA*, or* 1NFA *to accept the language $L(M)$.*

4. Simulation Cost of Sweeping and Rotating k-DLA

This section is devoted to deriving bounds on the cost for removing the ability to rewrite each cell $k \geq 1$ times, more precisely, the cost for the simulation of sweeping and rotating k-DLA by deterministic finite automata. We start with a lower bound for the simulation by 1DFA. Interestingly, this lower bound is greater than the lower bound $n \cdot F(n)^{k-1}$ known for the simulation of unary one-way k-head finite automata [11]. Both types of devices accept only regular unary languages, but only trivial bounds are currently known for the cost of their mutual simulations.

Theorem 4. *Let $k \geq 2$ and $n \geq 5$ be integers such that n is prime. Then there is a unary sweeping k-DLA M with at most $2n$ states and $2k$-tape-symbols such that $n \cdot F(n)^k$ states are necessary for any* 1DFA *to accept the language $L(M)$.*

Proof. For any constants $k \geq 2$ and prime $n \geq 5$, we construct a unary sweeping k-DLA $M = \langle S, \{a\}, \Gamma, \delta, \rhd, \lhd, s_0, F \rangle$. There are integers $c_1, c_2, \ldots, c_l \geq 2$ such that $c_1 + c_2 + \cdots + c_l \leq n$ and $\mathrm{lcm}(c_1, c_2, \ldots, c_l) = F(n)$. We set $p(1) = 0$, $q(1) = c_1 - 1$, $p(i) = q(i-1)+1$, $q(i) = p(i)+c_i-1$, for $2 \leq i \leq l$. So, we obtain in particular $q(l) \leq n-1$. Now we set $S = \{s_0, s_1, \ldots, s_{n-1}, s_{+,1}, s_{+,2}, \ldots, s_{+,l}\}$, $F = \{s_{+,1}, s_{+,2}, \ldots, s_{+,l}\}$, $\Gamma_0 = \{a\}$, $\Gamma_1 = \{a_1\}$, and $\Gamma_i = \{\sqcup_i, a_i\}$, for $2 \leq i \leq k$.

Let w be an input. In its first sweep, M rewrites every input cell with the symbol a_1. The purpose of the first sweep is to determine the value $|w| \bmod n$ (Transitions 1, 2). If the value does not belong to the set $\{p(1), p(2), \ldots, p(l)\}$, the computation halts and rejects (Transition 3).

1. $\delta(s_0, \rhd) = (s_0, \rhd, 1)$
2. $\delta(s_i, a) = (s_{(i+1) \bmod n}, a_1, 1)$, for $0 \leq i \leq n-1$
3. $\delta(s_{p(j)}, \lhd) = (s_{p(j)}, \lhd, -1)$, for $1 \leq j \leq l$

The principal idea of the further computation is as follows. In the first sweep a value $p(j)$ is determined. Now M fixes the j and uses k further sweeps to test whether the length of the input is a multiple of c_j^k. A detailed analysis of the language accepted follows after the construction. In the next $k-1$ sweeps only the states $s_{p(j)}$ to $s_{q(j)}$ are used. During a sweep every c_jth non-blank symbol is kept non-blank, while all the others are rewritten by a blank (Transitions 4, 5, and 6). If the number of non-blank symbols found during the sweep is not a multiple of c_j, that is, M reaches the opposite endmarker not in state $s_{p(j)}$, the computation halts and rejects (Transitions 7, 8). The following transitions are defined for all $1 \leq j \leq l$. They are used for the mth sweep, $2 \leq m \leq k$, where $d = 1$ if m is odd, and $d = -1$ if m is even.

4. $\delta(s_i, a_{m-1}) = (s_{(i+1)}, \sqcup_m, d)$, for $p(j) \leq i \leq q(j) - 1$
5. $\delta(s_{q(j)}, a_{m-1}) = (s_{p(j)}, a_m, d)$
6. $\delta(s_i, \sqcup_{m-1}) = (s_i, \sqcup_m, d)$, for $0 \leq i \leq q(l)$
7. $\delta(s_{p(j)}, \rhd) = (s_{p(j)}, \rhd, 1)$
8. $\delta(s_{p(j)}, \lhd) = (s_{p(j)}, \lhd, -1)$

After the kth sweep no further rewriting is possible. However, M continues with one more sweep for which the states $s_{p(j)}$ to $s_{q(j)}$ and $s_{+,j}$ are used, where $s_{+,j}$ just replaces $s_{p(j)}$ after the first cycle. Let $d = 1$ if $k+1$ is odd, and $d = -1$ if $k+1$ is even.

9. $\delta(s_i, a_k) = (s_{(i+1)}, a_k, d)$, for $p(j) \leq i \leq q(j) - 1$
10. $\delta(s_{q(j)}, a_k) = (s_{+,j}, a_k, d)$
11. $\delta(s_i, \sqcup_k) = (s_i, \sqcup_k, d)$, for $0 \leq i \leq q(l)$
12. $\delta(s_{+,j}, a_k) = (s_{p(j)+1}, a_k, d)$
13. $\delta(s_{+,j}, \sqcup_k) = (s_{+,j}, \sqcup_k, d)$

Finally, if M reaches the endmarker with state $s_{+,j}$, the input is accepted since the transition function is undefined for $s_{+,j}$ on endmarkers and $s_{+,j} \in F$.

Now we turn to determining the language $L(M)$. Let $\ell = |w|$ be the length of the input. The first sweep is used to count ℓ modulo n. If the head arrives at the right endmarker in any state not in $\{s_{p(1)}, s_{p(2)}, \ldots, s_{p(l)}\}$, the computation halts and rejects. Let us assume the state is $s_{p(j)}$, for $1 \leq j \leq l$. Then we know $\ell = x_1 \cdot n + p(j)$, for some $x_1 \geq 0$.

For sweep $2 \leq m \leq k$, if the head arrives at the endmarker in any state not equal to $s_{p(j)}$, the computation halts and rejects. Otherwise, the number of non-blank cells has been divided by c_j and the number of non-blank cells found during the sweep is a multiple of c_j. So, we have $\ell = z_m \cdot c_j^{m-1}$, for some $z_m \geq 0$. If M accepts after a further sweep, it has checked once more whether the number of non-blank cells found during the sweep is a multiple of c_j. Therefore, we derive $\ell = x_2 \cdot c_j^k$, for some $x_2 \geq 0$. The further reasoning is as for k-head finite automata [11]. We recall it for the sake of completeness.

Together we have that the length of the input has to meet the two properties $\ell = x_1 \cdot n + p(j)$ and $\ell = x_2 \cdot c_j^k$. Since $n \geq 5$ is prime and c_j is less than n, the numbers n and c_j are relatively prime. We conclude that n and c_j^k are relatively prime as well. So, there is an $x' \geq 1$ such that $x'c_j^k$ is congruent 1 modulo n. We derive that there is a y' such that $x'c_j^k = y'n + 1$. This implies $p(j)x'c_j^k = p(j)y'n + p(j)$ and, thus, there exists an ℓ having the properties mentioned above. By extending the length of the input by multiples of nc_j^k an infinite set of input lengths ℓ meeting the properties is derived. More precisely, given such an ℓ, the difference to the next input length longer than ℓ meeting the properties has to be a multiple of n and a multiple of c_j^k. Since both numbers are relatively prime, it has to be a multiple of nc_j^k. The language L_j consisting of all input lengths having these two properties is regular, and every 1DFA accepting unary L_j has a cycle of at least nc_j^k states.

The language $L(M)$ is the union of the languages L_j, $1 \leq j \leq l$. Since all c_j and n are pairwise relatively prime, all c_j^k and n are pairwise relatively prime. So, an immediate generalization of the proof of the state complexity for the union of two unary 1DFA languages [30] shows that every 1DFA accepting $L(M)$ has a cycle of at least $\operatorname{lcm}\{\, nc_j^k \mid 1 \leq j \leq l \,\} = n(c_1 c_2 \cdots c_l)^k = n \cdot F(n)^k$ states. □

Now we turn to an upper bound that shows that removing the ability to rewrite each cell $k \geq 1$ times, but keeping the two-way head movement, cost only a polynomial number of states. From the resulting unary sweeping 2DFA an upper bound for one-way devices can be derived by the known bounds for removing the two-way head movement.

Theorem 5. *Let $k, n \geq 1$ be integers and M be a unary n-state sweeping k-DLA. Then $O(n^{\frac{k^2+k}{2}})$ states are sufficient for a sweeping* 2DFA *to accept the language $L(M)$. The* 2DFA *can effectively be constructed from M.*

Proof. Let w be some input long enough and M be a unary n-state sweeping k-DLA. At first we consider the structure of the tape inscription after the first $m \leq k$ sweeps. Since M is sweeping and has n states, its first sweep starts by rewriting the leftmost at most n input symbols. Then the behavior becomes cyclic with a cycle length of at most n. At the end, there may appear an incomplete cycle whose length is again at most n. So, after the first sweep the tape inscription has the form

$$x_1 x_2 \cdots x_{l_1} (y_1 y_2 \cdots y_{j_1})^* z_1 z_2 \cdots z_{r_1}$$

where all symbols x_i, y_i, and z_i are from Γ_1, and $l_1, j_1, r_1 \leq n$. Let in the following the block $x_1 x_2 \cdots x_{l_1}$ be the *left block*, the block $y_1 y_2 \cdots y_{j_1}$ be the *middle block*, and the block $z_1 z_2 \cdots z_{r_1}$ be the *right block*. In general, the blocks and their lengths depend on M and on the length of the input (for the first sweep only the right block depends on the length of the input, but for further sweeps all blocks do).

In order to argue inductively, assume that after the mth sweep, $m \leq k-1$, the tape inscription is of the form

$$x'_1 x'_2 \cdots x'_{l_m} (y'_1 y'_2 \cdots y'_{j_m})^* z'_1 z'_2 \cdots z'_{r_m}$$

where all symbols x'_i, y'_i, and z'_i are from Γ_m, $l_m, r_m \leq n + n^2 + \cdots + n^m$, and $j_m \leq n^m$. Depending on the parity of m, during the $(m+1)$st sweep M first rewrites the left or right block. When it reaches the middle block, it again becomes cyclic after another at most $j_m \cdot n$ steps, where the cycle length is at most n^{m+1}. At the end, M enters the opposite, that is right or left, block. Again, there may appear an incomplete cycle of the middle block that gives the maximal number of symbols added to the opposite block. So, after the $(m+1)$st sweep the tape inscription is of the form

$$x''_1 x''_2 \cdots x''_{l_{m+1}} (y''_1 y''_2 \cdots y''_{j_{m+1}})^* z''_1 z''_2 \cdots z''_{r_{m+1}}$$

where all symbols x''_i, y''_i, and z''_i are from Γ_{m+1}, $j_{m+1} \leq n^{m+1}$, and the lengths of the left and right block are $l_{m+1}, r_{m+1} \leq n + n^2 + \cdots + n^{m+1}$.

We conclude that after the kth sweep the lengths l_k of the left and r_k of the right block are limited by $k \cdot n^k$, and the length j_k of the middle block is limited by n^k. These lengths and the structures of the tape inscriptions after

each sweep depend on M and the length of the input. However, in total there are only finitely many different structures that may appear: During the first sweep, M may be in one of n states when it leaves the last copy of the middle block. This implies that all inputs long enough are partitioned into at most n classes. To each class a fixed structure can be associated and the structure can be precomputed from M. In the second sweep, each class is further partitioned into subclasses that are defined by the state in which M leaves the last copy of the middle block. Again, for each subclass its associated fixed tape structure can be precomputed. Arguing in the same way, in every further sweep up to sweep k, each subclass is further partitioned into subclasses that are defined by the state in which M leaves the last copy of the middle block, and for each subclass its associated fixed tape structure can be precomputed.

Next we turn to the construction of an equivalent deterministic sweeping finite automaton M'. Basically, the idea is to use a subset of states for every possible input tape structure that allow M' to mimic M.

The first sweep of M is simulated by M' directly. To this end, the 2DFA M' takes at most n states. When M' reaches the right endmarker in some state s_1, this state gives the class the input belongs to and, thus, the tape structure including the values of l_1, j_1, and r_1. Depending on s_1, now M' starts the second sweep with a subset of states that mimic the behavior of M on the right block of the tape inscription. Since M' only sees the original unary tape content, to this end a number of states corresponding to the length of the block is necessary. So, at most $n + n^2$ states are used in this subset. When this sequence of states has been passed through, M' uses another subset of states to process the cycle of the middle block. Since the length of the cycle is at most n^2, at most n^2 states are used in this subset. While M' runs through this cycle it cannot detect the position on the tape where the left block begins, since it only sees the original unary input. Therefore, M' will reach the left endmarker in some state s_2' of the cycle. However, the subset of states used in the cycle is only used for the particular tape structure computed by M in the second cycle on the same input. Therefore, by exploiting the precomputations it can be determined in which state M' has left the last copy of the middle block. Since the content of the left block is precomputed too, it can further be determined whether M would have halted and accepted on the left block. In the latter case, s_2' is made an accepting state and the transition function is undefined for state s_2' on the left endmarker $\triangleright$. In this way, M' accepts as well. In the former case, depending on the state in which M would have left the left endmarker, now M' starts the third sweep with a new subset of states that mimic the behavior of M on the left block of the tape inscription. So, this second sweep takes at most $n + n^2 + n^2 = 2n^2 + n$ states.

However, a different subset of states for the second sweep is required for each of the n states that may appear when the last copy of the middle block is left in the first sweep. So, altogether, at most $n \cdot (2n^2 + n) \leq c_2 \cdot n^3$ states are sufficient for M' to mimic the second sweeps of M, where c_2 is a constant. Arguing in the same way, for each fixed state M' can be in at the end of the second sweep, the third sweep can be mimiced with at

most another $n + n^2 + n^3 + n^3 = 2n^3 + n^2 + n$ states. So, altogether, at most $c_2 \cdot n^3 \cdot (2n^3 + n^2 + n) \leq c_3 \cdot n^6$ states are sufficient for M' to mimic the third sweeps of M, where c_3 is a constant. For the kth sweeps we obtain altogether at most $c_k \cdot n^{\frac{k^2+k}{2}}$ states, where c_k is a constant.

Summing up $n + c_2 \cdot n^3 + c_3 \cdot n^6 + \cdots + c_k \cdot n^{\frac{k^2+k}{2}}$ the number of states needed by M' to mimic the first k sweeps of M is a polynomial of degree $\frac{k^2+k}{2}$.

In addition, possible further sweeps of M have to be considered. Since after the kth sweep the tape inscription is fixed but is read from both ends, the states for the kth sweeps can be doubled to sweep across the tape in the opposite direction. Moreover, to simulate further sweeps the states used for the kth sweeps and their reversals can be used together with an additional register that maintains the n states of M.

Finally, all input words not long enough have to be considered, this means, all input words whose lengths are at most $O(k \cdot n^k)$. In order to deal with these words, M' performs a sweep with at most $O(k \cdot n^k)$ states. All of these words are accepted at the end of this very first sweep. If a word is longer, subsequently the deterministic sweeping finite automaton described above is simulated.

We derive that M' takes no more than $O(n^{\frac{k^2+k+2}{2}})$ states. □

Example 1 provides the witness language $L = \{a^{n^{k+1}}\}$ for a lower bound. Since L is a unary singleton, every 2NFA, 2DFA, 1NFA, or 1DFA needs at least n^{k+1} states to accept it. Since the example shows that L is accepted by some sweeping $(n+2)$-state k-DLA, the following lower bound follows.

Theorem 6. *Let $k \geq 1$ and $n \geq 2$ be integers. Then there is a unary sweeping $(n+2)$-state, $(2k+1)$-tape-symbol k-DLA M, such that n^{k+1} states are necessary for any* 2NFA*,* 2DFA*,* 1NFA*, or* 1DFA *to accept the language $L(M)$.*

The quadratic degree of the polynomial for the upper bound shown in Theorem 5 is essentially due to the fact that the non-unary tape contents after the first sweep cannot be recomputed. Instead, the computation has to be simulated by states that reflect the contents. The problem with the recomputation is caused by the alternating directions of the sweeps. So, a recomputation would require reversibility of the single sweeps. But in general these sweeps have an irreversible nature. Further restrictions of sweeping two-way automata studied in the literature are so-called rotating automata [8]. A *rotating* k-DLA is a sweeping k-DLA whose head is reset to the left endmarker every time the right endmarker is reached. So, the computation of a rotating machine can be seen as on a circular input with a marker between the last and first symbol. While every unary 2DFA can be made sweeping by adding one more state [9], and unary sweeping 2DFA can be made rotating without increasing the number of states, for unary 2DFA all these modes are almost the same. However, the situation might be different for limited automata. The next theorem shows that the simulation of rotating k-DLA by 2DFA is cheaper even for arbitrary alphabets. Moreover, it will turn out that the upper and lower bounds are tight in the order of magnitude. The degree of the polynomials is the same.

Theorem 7. *Let $k, n \geq 1$ be integers and M be an n-state rotating k-DLA. Then $O(n^{k+1})$ states are sufficient for a (rotating or sweeping)* 2DFA *to accept the language $L(M)$. The* 2DFA *can effectively be constructed from M.*

PROOF. Let $M = \langle S, \Sigma, \Gamma, \delta, \rhd, \lhd, s_0, F \rangle$ be an n-state rotating k-DLA. An equivalent sweeping 2DFA $M' = \langle S', \Sigma, \delta', s'_0, F' \rangle$ is constructed as follows. Basically, M' simulates M in left-to-right sweeps. To this end, states with up to $k+1$ registers are used in which computations of M are simulated. The first register is used to simulate the first input scan of M once and again in any left-to-right sweep of M'. When the first input scan of M ends, the state in which the second scan begins is known. So, M' starts to simulate the second scan once and again in the second register in any left-to-right sweep. The same behavior is applied to the first k registers. Subsequently, another register is used to simulate all further scans of M in which the tape inscription does not change anymore. Whenever M' reaches the right endmarker, it moves its head back to the left endmarker.

For a formal construction the state set S' of M' is defined as $S' = S_1 \cup \overleftarrow{S_1}$, where $S_1 = \bigcup_{i=1}^{k+1} S^i$ and $\overleftarrow{S_1} = \{ \overleftarrow{s} \mid s \in S_1 \}$. The states in S_1 are used to simulate M and the states in $\overleftarrow{S_1}$ are used to move the head back to the left. The set of final states F' is $\{ (s_1, s_2, \ldots, s_i), \overleftarrow{(s_1, s_2, \ldots, s_i)} \mid 1 \leq i \leq k+1, s_i \in F \}$, and the initial state s'_0 is set to s_0.

The next two transitions are used to move the head from the right endmarker to the left. For all $s \in S_1$ and $a \in \Sigma$:

1. $\delta'(s, \lhd) = (\overleftarrow{s}, -1)$
2. $\delta'(\overleftarrow{s}, a) = (\overleftarrow{s}, -1)$

The following transitions simulate scan m of M with $1 \leq m \leq k$ by a left-to-right sweep of M'. For all $s_1, s_2, \ldots, s_{k+1} \in S$ and $a_1 \in \Sigma$:

3. $\delta'(s'_0, \rhd) = (s', 1)$, where $\delta(s_0, \rhd) = (s', \rhd, 1)$
4. $\delta'(s_1, s_2, \ldots, s_m, a_1) = (s'_1, s'_2, \ldots, s'_m, 1)$,
   where $(s'_j, a_{j+1}, 1) = \delta(s_j, a_j)$, for $1 \leq j \leq m$
5. $\delta'(\overleftarrow{s_1, s_2, \ldots, s_m}, \rhd) = (s'_1, s'_2, \ldots, s'_m, s'_{m+1}, 1)$,
   where $(s'_j, \rhd, 1) = \delta(s_{j-1}, \rhd)$, for $1 \leq j \leq m+1$

So, in every left-to-right step on an input symbol all the symbols that are written during the previous scans are recomputed. For all scans m with $m \geq k+1$ the tape inscription does not change anymore. They are simulated as follows.

6. $\delta'(s_1, s_2, \ldots, s_{k+1}, a_1) = (s'_1, s'_2, \ldots, s'_{k+1}, 1)$,
   where $(s'_j, a_{j+1}, 1) = \delta(s_j, a_j)$, for $1 \leq j \leq k$,
   and $(s'_{k+1}, a_{k+1}, 1) = \delta(s_{k+1}, a_{k+1})$
7. $\delta'(\overleftarrow{s_1, s_2, \ldots, s_{k+1}}, \rhd) = (s'_1, s'_2, \ldots, s'_{k+1}, 1)$,
   where $(s'_j, \rhd, 1) = \delta(s_{j-1}, \rhd)$, for $1 \leq j \leq k$,
   and $(s'_{k+1}, \rhd, 1) = \delta(s_{k+1}, \rhd)$

When M halts on a non-endmarker in the ith step of the jth scan, then M' will halt in the ith step of the jth left-to-right sweep. When M halts on the endmarker, M' will halt on the left endmarker. The last state component of M' is the currently simulated state of M. So, by definition of the set of accepting states, M accepts if and only if M' accepts. For the number of states of M' we obtain $|S'| = 2\sum_{i=1}^{k+1} n^i \leq 2(k+1)n^{k+1} \in O(n^{k+1})$.

So far we have constructed M' to be sweeping. For the case that M' has to be rotating, essentially the same construction can be used. However, moving the head back to the left endmarker is now for free. That is, the states from the set $\overleftarrow{S_1}$ are unnecessary and Transitions 5 and 7 have to be adjusted straightforwardly. □

The construction of the sweeping k-DLA accepting the singleton language $L = \{a^{n^{k+1}}\}$ given in Example 1 can easily be modified to the construction of an equivalent rotating k-DLA. So, we have the following lower bound that matches the upper bound in the order of magnitude.

Theorem 8. *Let $k \geq 1$ and $n \geq 2$ be integers. Then there is a unary rotating $(n+2)$-state, $(2k+1)$-tape-symbol k-DLA M, such that n^{k+1} states are necessary for any* 2NFA*,* 2DFA*,* 1NFA*, or* 1DFA *to accept the language $L(M)$.*

5. From *k*-Limited to Pushdown Automata

From the results by Hibbard, it is known that for each $k \geq 2$, k-LAs recognize exactly the context-free languages, that is, they are equivalent to nondeterministic pushdown automata (PDAs). The proof by Hibbard was given in several steps. Roughly, a simulation of 2-LAs by PDAs was provided. Furthermore, for $k > 2$ it was shown that each k-LA can be expressed as the combination of a transducer with a $(k-1)$-LA. In turns, the combination of a transducer with a PDA can be simulated by another PDA. By repeatedly applying these constructions, from each k-LA it is possible to obtain an equivalent PDA.

Here we provide a direct simulation from which we are able to show that the resulting PDA has a description whose size is exponential with respect to the size of the description of the given k-LA. We point out that an exponential size is also necessary. In fact, an exponential gap between the sizes of 2-LAs and equivalent PDAs is already known [21].

The simulation we present is an extension of the one given by Pighizzini and Pisoni [21] for 2-LAs. Furthermore, we also use some ideas from [29], where the authors gave a construction to convert (for each function f) any one-tape Turing machine that on inputs of length n can rewrite each tape cell only in the first $f(n)$ visits into an equivalent PDA equipped with an $f(n)$-space bounded auxiliary tape. In the particular case $f(n) = k$ this gives a simulation of k-LAs by PDAs. However, the argument used in [29] requires a preliminary transformation of the given machine in a special form and, from that construction, we do not get any information on the size of the resulting PDA.

The fundamental tools used in our simulation are *crossing sequences* and *transition tables*. Both of them have been introduced at the beginning of automata theory to prove, in two different and independent ways, the equivalence between two-way and one-way automata [25, 26].

We recall that given an accepting computation on an input w of a two-way automaton A, we can associate with each boundary between consecutive tape cells the sequence of states entered by the automaton when the boundary is crossed by the head. Such a sequence is called a *crossing sequence*. Given an input symbol a, by inspecting the transition function of A it is possible to verify whether or not two crossing sequences are "compatible" with respect to a, which means that they may appear at the left and at the right boundaries of a tape cell containing a in some accepting computation. A one-way machine simulating A can be obtained by guessing consecutive crossing sequences while scanning the input tape, and locally verifying the compatibility of crossing sequences guessed for the left and right boundaries of each tape cell. Since for each accepted input there exists an accepting computation such that each crossing sequence does not contain any repeated state, it turns out that this can be implemented in a finite control, that is, by a one-way finite automaton.

We could try to use crossing sequences also for k-LAs. However, there are two main differences. First of all, each tape cell is rewritten in the first k visits. Since the rewriting depends on "local" information, it is possible to take it into account by suitably extending the compatibility relation between crossing sequences (see, for example, [5]). The second and more relevant difference is that even in the shortest accepting computation we could have crossing sequences of unbounded length (for example, linear in the length of the input), so they cannot be stored in a finite control. To overcome this problem, in our simulation we will represent only some initial parts of crossing sequences. Roughly, these parts correspond to *active visits*, that is, to computation steps that modify cell contents (the transitions used in these steps will be also called *active transitions*). The lengths of these parts are bounded by $k + 1$.

For the remaining parts of the computations, which are the steps working on "frozen" tape cells (that is cells that cannot be further rewritten), we make use of *transition tables*. Essentially, the transition table associated with a string (written in a frozen tape segment) is a relation which specifies for each possible entrance in the segment (that is, each pair defined by a state used to enter the tape segment and a side, left or right) the possible exits.

In the restricted case of sweeping k-LAs, each computation can be split into two parts. In the first part, corresponding to the first k sweeps, the tape is modified. This part can be simulated by inspecting from left to right all tape cells, by nondeterministically generating, for each cell, the sequence of transitions used in the active visits, and by verifying that the crossing sequence obtained in this way at the left boundary of the cell coincides with the crossing sequence which has been obtained for the right boundary of the previous cell. At the end of this part all the tape is frozen. The remaining part of the computation can be verified using the transition table corresponding to the final tape content. Notice that this simulation produces a finite state device. Indeed, the language

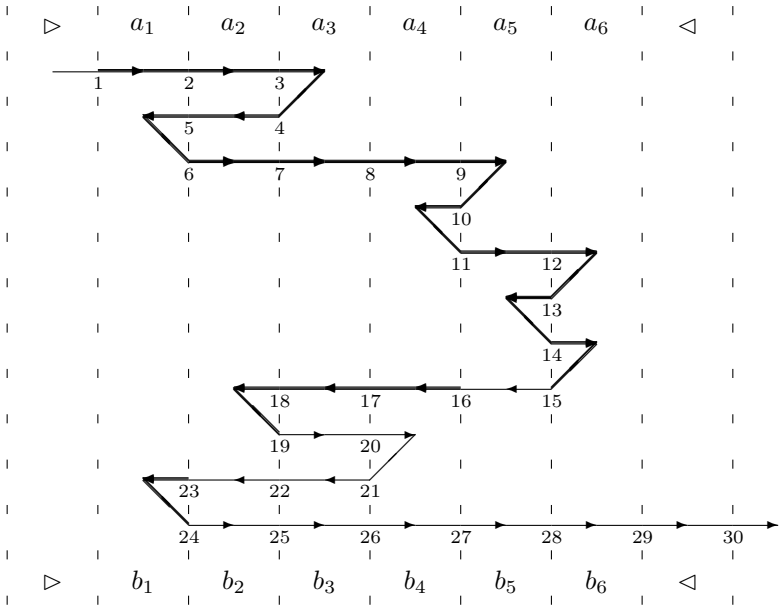

Figure 1: The trajectory of the head in an accepting computation of a 4-LA. The top and bottom lines represent the initial and the final tape content, respectively. The parts in boldface represent active transitions. The numbers count computation steps. The crossing sequence at a cell boundary can be obtained by replacing those numbers (from top to bottom) by the states reached in the corresponding steps (for example, the crossing sequence at the boundary between cell 4 and 5 consists of states $q_9, q_{10}, q_{11}, q_{16}, q_{27}$, where q_j denotes the state reached at step j).

accepted by any sweeping k-LA is regular.

In the general case, the situation is more complicated because, due to head reversals, it is not possible to decompose the computation in two parts as in the sweeping case. In particular, transitions on frozen cells could be followed in a computation by active transitions on other cells and vice versa (see Figure 1 for an example). A pushdown store will be helpful to solve this problem.

From now on, let M be the k-LA we have to simulate. We remind the reader that each computation starts in the initial state with the head on the leftmost tape cell which contains the left endmarker. For technical reasons, we assume that M accepts the input in any state *after violating* the right endmarker. No other moves can be done in this situation, that is, once the head is moved to the right of the right endmarker the computation stops. Finally, without loss of generality, we can also assume that in each accepting computation all the cells become frozen, that is, the final tape content is a string from Γ_k^* surrounded by the endmarkers. This can be implemented by increasing the number of states by a constant value.

Active visits and crossing sequences

Our simulation tries to construct an accepting computation of M on the given input string by (implicitly) guessing, for each tape cell, the sequence of transitions used in the active visits to the cell. These transitions define a crossing sequence at the left boundary of the cell, the final cell content, and a crossing sequence at the right boundary. To formalize this idea, we now introduce and then we will discuss the following definition:

Definition 9. *Given $a \in \Sigma$, let* active(a) *be the subset of $S^* \times \Gamma_k \times S^*$ consisting of all triples $(s_1 s_2 \cdots s_l, b, t_1 t_2 \cdots t_r)$, with $s_1, s_2, \ldots, s_l, t_1, t_2, \ldots, t_r \in S$, $l \geq 1$, $r \geq 0$, $b \in \Gamma_k$, such that there is an integer h, $1 \leq h \leq k$, states $p_1, p_2, \ldots, p_{2h} \in S$, symbols $\gamma_0, \gamma_1, \ldots, \gamma_{h-1} \in \Gamma \setminus \Gamma_k$, $\gamma_h \in \Gamma_k$, $d_0 = -1$, $d_1, d_2, \ldots, d_h \in \{-1, +1\}$, satisfying:*

(i) $\gamma_0 = a$, $\gamma_h = b$, *and* $(p_{2i}, \gamma_i, d_i) \in \delta(p_{2i-1}, \gamma_{i-1})$, *for* $i = 1, \ldots, h$,

(ii) $s_1 s_2 \cdots s_l$ *is the subsequence of* $p_1 p_2 \cdots p_{2h}$ *consisting of all* p_j*'s such that* $d_{\lfloor j/2 \rfloor} = -1$,

(iii) $t_1 t_2 \cdots t_r$ *is the remaining subsequence consisting of all* p_j*'s such that* $d_{\lfloor j/2 \rfloor} = +1$.

Furthermore, in order to consider endmarkers, the definition is extended as follows:

- active$(\rhd) = \{(s_0, \rhd, t) \mid (t, \rhd, +1) \in \delta(s_0, \rhd)\}$, *and*
- active$(\lhd) = \{(s, \lhd, t) \mid (t, \lhd, +1) \in \delta(s, \lhd)\} \cup \{(st, \lhd, \lambda) \mid (t, \lhd, -1) \in \delta(s, \lhd)\}$.

We point out that the transitions considered in Definition 9(i) give a sequence of possible active visits to a tape cell, whose initial content is the input symbol a.[1] According to the definition of transition function, the value d_i, $i = 1, \ldots, h$, represents the head movement in the ith transition, so specifying the side on which the cell is left by the head after the ith transition. This is also the side from which the cell is re-entered in the next visit, if any. Since in the first visit a cell is always entered from the left, we set $d_0 = -1$. Hence, transition i occurs after entering the cell in the state p_{2i-1} from the d_{i-1}-side and leaves it in the state p_{2i} to the d_i-side. So, when $d_i = -1$ ($d_{i-1} = -1$, resp.), the state p_{2i} (p_{2i-1}, resp.) belongs to the crossing sequence at the left of the cell, otherwise it belongs to the crossing sequence at the right. The sequences of states $s_1 s_2 \cdots s_l$ and $t_1 t_2 \cdots t_r$ so defined (items (ii) and (iii)) are initial parts of crossing sequences at the left and at the right boundary of the cell, in an accepting computation in which the cell is rewritten using these active transitions. (See Figure 2 for some examples.) Furthermore, in the sequence $s_1 s_2 \cdots s_l$, all the

[1] We remind the reader that, by definition, the transitions reversing the head direction are counted as double visits. For this reason $h \leq k$.

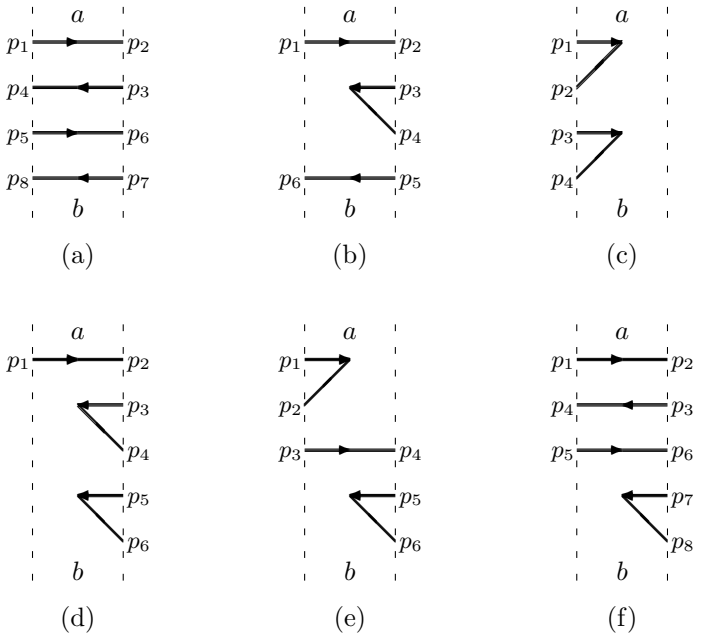

Figure 2: One tape cell of a 4-DLA with $2-4$ active transitions that rewrite the initial cell content a by b. The corresponding tuples in active(a) are: (a) $(p_1p_4p_5p_8, b, p_2p_3p_6p_7)$, (b) $(p_1p_6, b, p_2p_3p_4p_5)$, (c) $(p_1p_2p_3p_4, b, \lambda)$, (d) $(p_1, b, p_2p_3p_4p_5p_6)$, (e) $(p_1p_2p_3, b, p_4p_5p_6)$, (f) $(p_1p_4p_5, b, p_2p_3p_6p_7p_8)$.

states in odd positions are reached while entering the cell from the left, while all states in even positions are reached while leaving the cell to the left. On the other hand, in the sequence $t_1t_2\cdots t_r$, all the states in odd positions are reached while leaving the cell to the right, while all states in even positions are reached while entering the cell from the right. Since to accept the input the machine has to visit each tape cell at least one time, we have $l \geq 1$. However, all active visits to a cell could move the head to the left. For this reason, the value of r can also be 0. Notice that $l + r$ is an even number. If l and r are odd, then the last active transition is the one entering the state t_r and leaving the cell to the right. In a similar way, if l and r are even, then the last active transition is the one entering s_l and leaving the cell to the left.

Finally, we point out that the sets active($\triangleright$) and active($\triangleleft$) have been introduced for technical reasons and represent the first possible visits to the cells containing the endmarkers.

Further notations

We are now almost ready to present our simulation. We only need a few further notions.

- We consider relations from $S \times \{-1, +1\} \times S \times \{-1, +1\}$, called *transition tables*. We associate with each $w \in (\Gamma_k \cup \{\triangleright, \triangleleft\})^*$ a transition table τ_w

such that $(q, d', p, d'') \in \tau_w$ if and only if M has a computation path which starts in the state q on the d'-side of a tape segment containing w and leaves the segment in the state p to the d''-side, where, as usual, -1 and $+1$ are used to indicate *left* and *right*, respectively. In the case $w \neq \lambda$ *a path which starts on the* d'*-side* is a path starting on the d'-most cell of the segment. In the degenerate case $w = \lambda$, the only possible computation paths consist only of one state and no transitions. It is useful to stipulate that these paths enter the empty segment from one side and leave it on the opposite side. Hence, $(q, d', p, d'') \in \tau_\lambda$ if and only if $q = p$ and $d'' = -d'$.

Notice that the number of possible transition tables is finite.

- We can define the composition of transition tables, denoted by the symbol $\cdot$, in such a way that $\tau_{wz} = \tau_w \cdot \tau_z$, for $w, z \in (\Gamma_k \cup \{\rhd, \lhd\})^*$.

- We are also interested in computation paths starting at some point inside a tape segment. Given two frozen tape segments u_l and u_r, suppose that they are adjacent, so they form a tape segment $u = u_l u_r$. Given a state q, we consider the set of pairs $(p, d) \in S \times \{-1, +1\}$ such that M has a computation path which starts in the state q on the right side of the segment u_l and ends in the state p leaving the entire segment u to the d-side. This set of pairs depends only on the transition tables T_l and T_r associated with the segments u_l and u_r, and on the starting state q. Hence, it will be denoted as $\mathrm{exit}(T_l, q, T_r)$.

In the simulation we will also make use of the following macros:

- nSelect(P): returns a nondeterministically selected element belonging to the set P. If P is empty then the computation rejects. In the case P is a set of n-tuples and we need to restrict the selection to tuples having specified values in some components, we write $\mathrm{nSelect}(P_{|\alpha})$, where α is an n-component vector, specifying the values of such components and having the symbol $\cdot$ in the remaining positions. For instance $\mathrm{nSelect}(T_{|(p,+1,\cdot,\cdot)})$ selects a vector in T with first component p and second component $+1$. If T does not contain any vector satisfying this restriction then the computation rejects.

- read() returns the tape symbol on the currently scanned cell and moves the head one position to the right.

The simulation

Let us start to present a high level description of the PDA M' simulating the given k-LA M. The PDA M' scans the input string w from left to right and, step by step, it guesses and collects information on the parts of computation involving the already scanned input prefix. This information is stored on the stack. When the head reaches the ith input symbol a, M' guesses a triple $(s_1 s_2 \cdots s_l, b, t_1 t_2 \cdots t_r) \in \mathrm{active}(a)$, corresponding to a sequence of active visits to the cell i of the tape of M. Notice that any further transition on this cell

Algorithm 1: The simulation

```
1  stack initially empty, head on cell 1
2  (s_0, ▷, t_1 t_2 ··· t_r) ← nSelect(active(▷))                    // r = 1
3  T ← τ_▷
4  repeat                                                            // main loop
5  |  push(t_1 t_2 ··· t_r T)
6  |  T ← τ_λ
7  |  a ← read()
8  |  (s_1 s_2 ··· s_l, b, t_1 t_2 ··· t_r) ← nSelect(active(a))
9  |  s ← pop()
10 |  if s ≠ s_1 then REJECT
11 |  for j ← 1 to ⌊(l − 1)/2⌋ do connectLeft(s_{2j}, s_{2j+1})
12 |  if l is odd then  T ← T · τ_b
13 |  else
14 |  |  s ← nSelect(S)
15 |  |  t ← nSelect(S)
16 |  |  connectLeft(s_l, s)
17 |  |  T ← T · τ_b
18 |  |  connectLeft(s, t)
19 |  |  r ← r + 1
20 |  |  t_r ← t
21 |  if stack top is a table then
22 |  |  T′ ← pop()
23 |  |  T ← T′ · T
24 until a = ◁
25 if stack is empty then ACCEPT
26 else REJECT
```

should be compliant with the transition table τ_b, associated with the symbol b finally written on this cell. Then, using the information on the stack, M' verifies that the sequence $s_1 s_2 \cdots s_l$ matches the parts of computation which have been guessed in the previous steps while inspecting the input prefix of length $i-1$. If the verification is successful, then M' leaves on the stack the information on the possible transitions on the cell i *after* the active visits, by saving the transition table τ_b (or, as we will see, a transition table corresponding to a frozen tape segment ending in cell i), and the states $t_1, t_2, \ldots, t_r$ which are now *"pending"* and should be connected, in the future simulation steps, with some parts of the computation visiting the cells to the right of cell i.

More into details, the PDA M' implements Algorithm 1, also using the macro in Algorithm 2. To simplify the exposition, we suppose that the right endmarker $\lhd$ is appended to the input of M'. Hence, if L is the language accepted by the given k-LA M, then M' will accept the language $L\lhd$. This requirement will be removed by guessing the symbol $\lhd$ (proof of Theorem 11).

The symbols on the pushdown store are states and transition tables. At the beginning of the computation the stack is empty. The algorithm is written in such a way that at each iteration (after the execution of line 5) the stack contains alternately nonempty sequences of states and transition tables, where the sequence of states on the top has an odd length, while the other sequences have an even length. (For examples see Figure 3 and Figure 4.) Hence, the

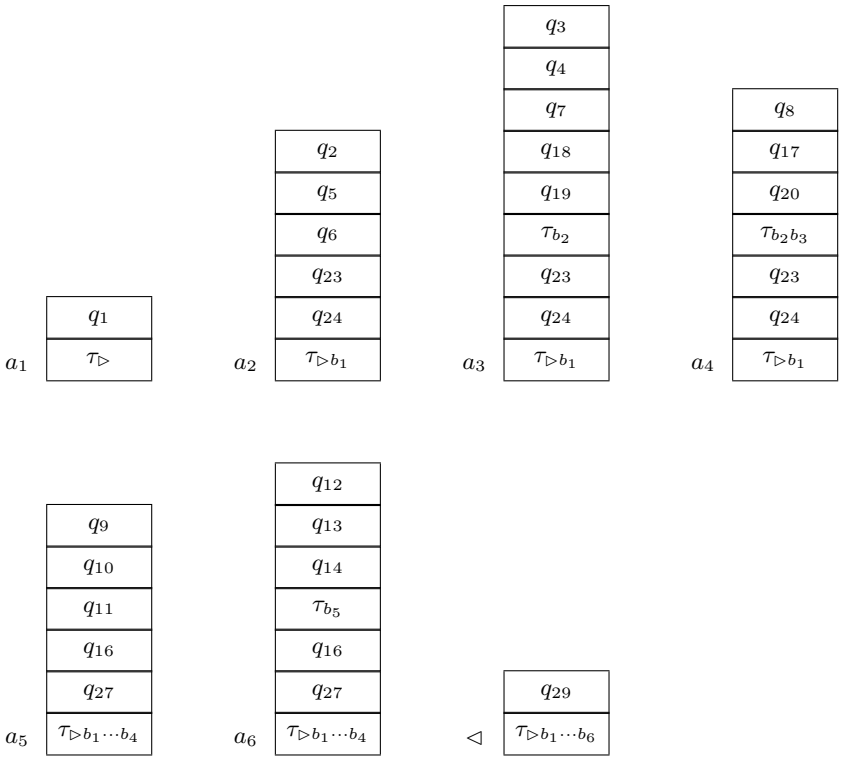

Figure 3: Snapshots of the PDA M' during the simulation of the computation in Figure 1. Each snapshot is taken after executing line 5 of Algorithm 1 and shows the symbol scanned by the head and the pushdown content.

stack content, from top to bottom, is a string

$$t\alpha_k T_k \alpha_{k-1} T_{k-1} \alpha_{k-2} \cdots \alpha_1 T_1$$

where $t \in S$, for $j = 1, \ldots, k$, $\alpha_j \in S^*$ and $|\alpha_j|$ is even, with $|\alpha_j| \geq 2$ when $j < k$, and T_j is a transition table.

The transition tables $T_1, T_2, \ldots, T_k$ correspond to a factorization of the final content of the part of the tape so far inspected, while $\alpha_1, \alpha_2, \ldots, \alpha_k$ are the sequences of states pending at the right boundaries of the corresponding tape segments. In other words, there exist indices $0 = i_0 < i_1 < \cdots < i_k \leq |w| + 1$ such that $i_k = i$ is the current head position and, for $j = 1, \ldots, k$, $T_j = \tau_{z_j}$, where z_j is the string finally written on cells $i_{j-1}, \ldots, i_j - 1$.

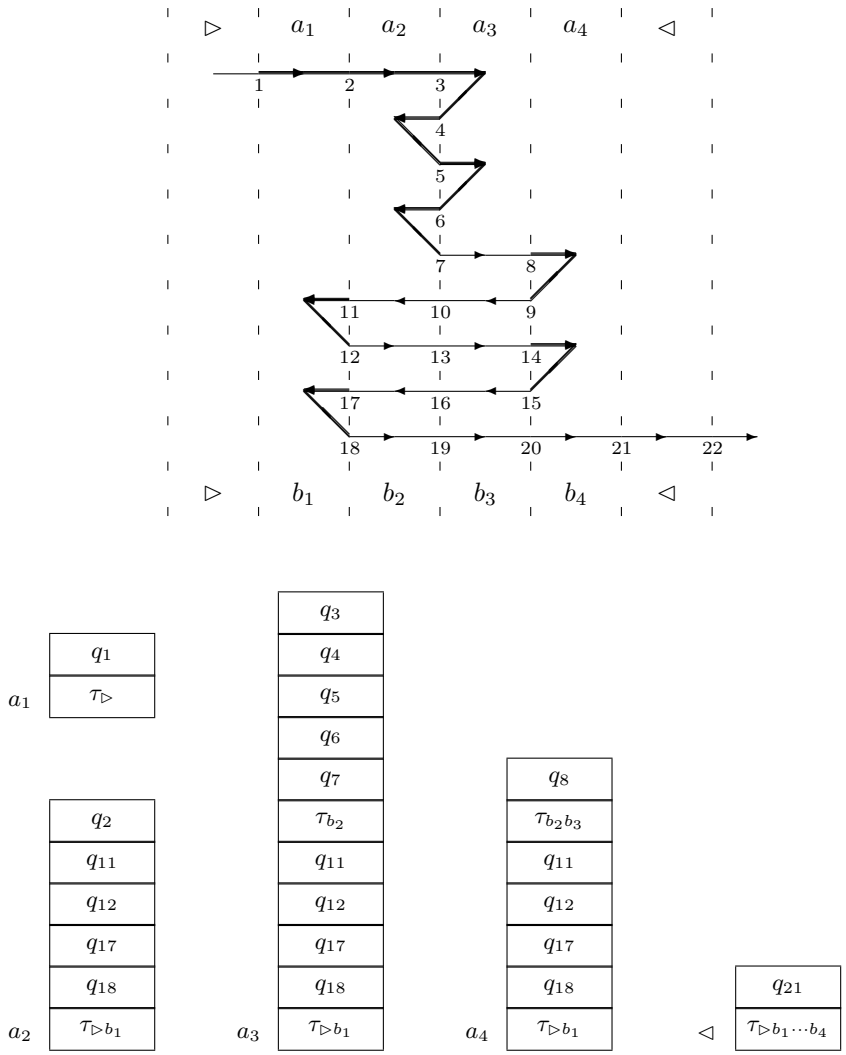

Figure 4: Another example of accepting computation of a 4-LA M and the corresponding snapshots of the PDA M' simulating it, taken after executing line 5 of Algorithm 1.

Each α_j contains the states which are pending at the boundary between cells $i_j - 1$ and i_j. These states should be entered alternately while moving from cell i_j to cell $i_j - 1$ and while re-entering cell i_j. The computation of M should enter the cell under inspection, that is, cell i_k, for the first time in the state stored on the top. For instance, the third snapshot in Figure 3 represents the stack content before inspecting the cell containing a_3. Hence, $k = 2$, $T_1 = \tau_{\triangleright b_1}$, $T_2 = \tau_{b_2}$, $\alpha_1 = q_{23}q_{24}$, $\alpha_2 = q_4 q_7 q_{18} q_{19}$, $t = q_3$, $i_0 = 0$, $i_1 = 2$, $i_2 = 3$. The states q_{23} and q_{24} are pending at the boundary between cells 1 and 2; the cell 3 should be entered for the first time in state q_3. The next times the head moves

from cell 3 to 2 and vice versa, the states should be q_4, q_7, q_{18}, and q_{19}.

During the simulation, the finite state control keeps a transition table in a variable T. This table is called the *current transition table* and it is the transition table associated with the final content of the *current tape segment*, which is the longest tape segment ending at the left boundary of the currently scanned cell and such that for all cell boundaries inside the segment there are no pending states.[2] At the beginning of each iteration of the main loop (lines 4–24), the current transition table and the states $t_1, t_2, \ldots, t_r$ which are pending at the right of the last inspected cell are pushed on the stack with T at the bottom and t_1 at the top. As we will see, r is always odd.

At the beginning of the first iteration the current tape segment consists of the cell containing the left endmarker. For this reason, the value assigned to T, before executing the loop, is the transition table $\tau_{\triangleright}$. Furthermore, $r = 1$ and t_1 is a state entered in the first computation step, while moving from the left endmarker to the right.

After saving $t_1 t_2 \cdots t_r T$ on the stack, the current tape segment becomes the one starting to the right of the last inspected cell, so it has length 0. For this reason, on line 6 the transition table τ_λ is assigned to the variable T.

The current cell is inspected by choosing $(s_1 s_2 \cdots s_l, b, t_1 t_2 \cdots t_r) \in \text{active}(a)$, where $a \in \Sigma$ is the cell content (line 8). It should be verified that $s_1 s_2 \cdots s_l$ matches the information saved on the stack. To this aim, first of all the state s_1 which is reached while entering the cell for the first time is compared with the state on the stack top (the "previous" t_1), rejecting if they are different (line 10). Then, the remaining states $s_2, \ldots, s_l$ in the sequence are inspected with respect to the information saved on the stack. Each state s_{2j} with even index corresponds to a transition moving from the current cell to the left, while the state s_{2j+1}, for $2j + 1 \leq l$, should be reached by the first among the following transitions which re-enters the current tape cell. Hence, for each j such that $1 < 2j + 1 \leq l$, M' has to verify that there is a computation path which starts to the left of the current tape cell in the state s_{2j} and reaches the current tape cell in the state s_{2j+1}, without visiting it in between. This is done on line 11 by the macro connectLeft. This macro uses the information stored in the stack and, if necessary, it can enlarge the current tape segment to the left, thus modifying the content of T. A detailed presentation of the macro is postponed. For the moment, we can say that in the special case of T containing the empty transition table τ_λ and two states t' and t'' at the stack top, connectLeft(s', s'') verifies that $s' = t'$ and $s'' = t''$ (rejecting in negative case), popping t' and t'' off the stack, without changing the content of T.

Example 10. In the simulation of the accepting computation in Figure 1, in the initial phase (line 2) M' selects $(s_0, \triangleright, q_1) \in \text{active}(\triangleright)$, so guessing that,

[2] With other words, these are no pending states at the right boundaries of cells other than the rightmost one in the segment or, equivalently, at the left boundaries of the cells other than the leftmost one. Hence, the current tape segment should start either at the cell containing the left endmarker, or at one cell having some pending states at its left boundary.

after reading the left endmarker, cell 1 is entered for the first time in state q_1. To prepare the inspection of cell 1, at the beginning of the first iteration this state is pushed on the stack with the transition table $\tau_\triangleright$ associated with the left endmarker. On line 8, the tuple $(q_1, b_1, q_2q_5q_6q_{23}q_{24})$ is guessed and, on line 10, it is verified that the first (and only) state in the first component, i.e., q_1, matches the state saved on the top of the stack. Since $l = 1$ and, thus, there are no active transitions from this cell moving to the left, the loop on line 11 ends without executing any iteration. ■

When l is odd, with this procedure all the sequence of states $s_1s_2 \cdots s_l$ is matched with the previously guessed parts of computation (lines 10–11). So, what remains to be done is to enlarge the current tape segment, "appending" to it the current tape cell. This is done by multiplying the variable T (whose content could have been changed, as we will discuss later, by the macro connectLeft) by the transition matrix τ_b associated with the final cell content (line 12). It may happen that at the end of the above described operations (or of the operations described later in the case of l even), the stack top contains a transition table. This means that there are no more pending states for the corresponding tape segment which can then be safely attached to the left of the current one. This is done by updating T (lines 21–23).

Example 10 (cont.) After the assignment on line 6, the content of the variable T has not been changed. Hence, by multiplying it by the transition table τ_{b_1} associated with the symbol finally written on cell 1, the content of T becomes τ_{b_1}. Furthermore, since the stack top contains the table $\tau_\triangleright$, on line 23 the content of T is further modified assigning to it $\tau_{\triangleright b_1}$. At the beginning of the next iteration, this table is pushed on the stack and, over it, the sequence of states $q_2q_5q_6q_{23}q_{24}$. In the second iteration, the tuple $(q_2q_5q_6, b_2, q_3q_4q_7q_{18}q_{19}) \in \text{active}(a_2)$ is guessed. The matching between the sequence of states in the first component and the information previously saved on the stack is verified on lines 10–11. (In this case, after using q_2 on line 10, connectLeft(q_5, q_6) pops the two elements on the top of the stack and verifies that they are q_5 and q_6, without any change on T, whose content remains τ_λ.) Since even in this case l is odd, it only remains to update T and to start the next iteration, by saving $q_3q_4q_7q_{18}q_{19}\tau_{b_2}$ before inspecting the next input symbol. ■

When l is even, the state s_l represents a move to the left. If the computation is accepting, then at some time the current cell should be re-entered from the left. The algorithm guesses the state s reached while re-entering the current cell for the first time and the state t that will be reached when, after that moment, the cell will be left for the first time to the right. The state t will be appended to the sequence $t_1t_2 \cdots t_r$, so obtaining an odd length sequence of pending states, to be stored in the stack at the beginning of the following iteration. Before doing that, the algorithm has to verify the guesses of s and t. The state s is verified by calling connectLeft(s_l, s) (lines 14–16). Notice that when the cell is re-entered in the state s, its content is already frozen. For this reason, on

Algorithm 2: The macro connectLeft(q, q')

```
27 (q, +1, p, d) ← nSelect(T_{|(q,+1,·,·)})
28 while d = −1 do
29 |  Y ← pop()
30 |  if Y = p then
31 |  |  p' ← pop()
32 |  |  (p', −1, p, d) ← nSelect(T_{|(p',−1,·,·)})
33 |  else if Y is a table then
34 |  |  (p, d) ← nSelect(exit(Y, p, T))
35 |  |  T ← Y · T
36 |  else REJECT
37 if p ≠ q' then REJECT
```

line 17, the variable T is updated, as in the case of l odd, by multiplying its content by the table τ_b associated with the final cell content. At this point, t is verified by calling connectLeft(s, t). Lines 19–20 are used to append t to the sequence at the right of the cell that, in this way, becomes of odd length.

Example 10 (cont.) In the third iteration, the tuple $(q_3q_4q_7q_{18}, b_3, q_8q_{17}) \in \text{active}(a_3)$ is selected. As in the previous iteration, on lines 10–11 the matching between q_3, q_4, q_7 and the states saved on the top of the stack is verified. However, as l is even, it remains to verify the last state q_{18} in the left component. M' guesses $s = q_{19}$ as the state in which the cell is re-entered from the left and $t = q_{20}$ as the state that will be used to leave the cell to the right for the first time after the sequence of active visits. The correctness of the first guess is checked by calling connectLeft(q_{18}, q_{19}) on line 16. Since q_{18} and q_{19} are at the top of the stack, the outcome of the verification is positive. After updating the variable T, the correctness of the second guess is checked by calling connectLeft(q_{19}, q_{20}) on line 18. In this case, the required path consists of just one transition moving to the right from the current cell of M, now containing the symbol b_3. The macro connectLeft makes this verification, just using the transition table τ_{b_3} which, after the assignment on line 17, is stored in T. Finally, on lines 21–23, the value of T is updated by multiplying it to the left by table τ_{b_2} which was on the top of the stack, obtaining $\tau_{b_2b_3}$. ■

Now we describe the macro connectLeft (Algorithm 2).

Given two states $q, q' \in S$ the macro has to check the existence of a computation path which starts in the state q on the right side of the current tape segment, visits only cells in the current tape segment or to the left of it, and ends in the state q' leaving the current tape segment to the right.

First of all, a tuple $(q, +1, p, d)$ is selected from the table T. This tuple corresponds to a path inside the current segment, starting in the state q on the right side, and leaving the segment in the state p to the d-side. There are two

possibilities:[3]

- If $d = +1$, that is, the segment is left to the right, the macro has only to verify that $p = q'$. This is done on line 37, by rejecting if the test fails. In this case the body of the loop on lines 28–36 is ignored.

- If $d = -1$ then the segment is left to the left. The state p is expected to be on the top of the stack, as pending. This is verified on line 30. This means that the cell boundary at the left of the current segment is traversed from the right in the state p and the next traversal of it should be from left to right in the state p', which was saved on the stack below p (line 31). At this point, the original computation of M re-enters the current segment from the left side in state p'. To compute the next exit from the segment, a tuple $(p', -1, p, d)$ is selected from T and the same process is repeated.

  However, while reconnecting the computation with pending states, it may happen that the stack top contains a table Y instead of a state. This means that all previously pending states at the left boundary of the current segment have been successfully reconnected to the computation and Y is the transition table of a frozen segment which ends at that boundary. At this point, we have to determine what happens by entering the segment represented by Y from the right in the state p. We are interested to know in which state and direction the *whole* segment corresponding to Y and T is left. To this aim, the values of p and d are updated using a pair chosen in $\mathrm{exit}(Y, p, T)$ (line 34). Since there are no more pending states inside the whole segment, the current segment can be enlarged to the left, attaching the segment represented by Y. This is done by updating the table T with its product to the left by the table Y (line 35).

Example 10 (cont.) The fourth iteration starts by pushing $q_8 q_{17} q_{20} \tau_{b_2 b_3}$ on the stack. The tuple $(q_8 q_{17}, b_4, q_9 q_{10} q_{11} q_{16}) \in \mathrm{active}(a_4)$ is selected. At the beginning, it is verified that the first state in the left sequence, which is q_8, matches the state at the top of the stack. Since $l = 2$, the loop on line 11 does not have to verify any pair of states. Then the block from lines 14–20 is executed, by firstly guessing $s = q_{20}$ and $t = q_{27}$. The verification $\mathrm{connectLeft}(q_{17}, q_{20})$ on line 16 is easily done, as in the previous iteration, using the two states at the top of the stack. In fact, since T contains the empty transition table τ_λ, the only possible choice on line 27 is $(q_{17}, +1, q_{17}, -1)$, which assigns q_{17} to p and -1 to d. Hence, the body of the loop on lines 28–36 should be executed. First, the stack top, which is q_{17}, is assigned to the variable Y. Hence, the test on line 30 is positive and on line 31 the current stack top, the state q_{20}, is assigned to p'. Again, being $T = \tau_\lambda$, the only possible choice on line 32 is $(q_{20}, -1, q_{20}, +1)$, which assigns q_{20} to p and $+1$ to d and leads to the end of the loop. Since the test on line 37 is negative, the macro ends without rejecting.

[3] Actually, in the case $T = \tau_\lambda$, the only possible tuple is $(q, +1, q, -1)$.

To verify the existence of a path starting in q_{20} with the head scanning the current tape cell, possibly moving in the cells to the left, and finally leaving the current cell in q_{27} to the right, connectLeft(q_{20}, q_{27}) is called on line 18. First we notice that, as a consequence of previous steps, the stack content is $\tau_{b_2b_3}q_{23}q_{24}\tau_{\triangleright b_1}$ and the variable T contains τ_{b_4}. The macro, using the information stored on the stack, reconnects the parts of the path from q_{20} to q_{27} in the computation depicted on Figure 1 in the following way.

On line 27, the tuple $(q_{20}, +1, q_{21}, -1)$ is selected. The loop is entered, by removing the stack top, which is the transition table $\tau_{b_2b_3}$. This means that all pending states at the right boundary of the segment represented by this table have been inspected. So, the current segment can be enlarged to the left by including that segment. To this aim, the current transition table is multiplied to the left by $\tau_{b_2b_3}$, so obtaining $\tau_{b_2b_3b_4}$ (line 35). However, before doing that, the exit from the resulting segment needs to be computed. This is done on line 34, selecting as (p, d) the pair $(q_{23}, -1)$, which represent a path leaving the segment to the left. This leads to a further iteration of the loop body, in which the state in the variable p coincides with the one that, at the beginning of the iteration, is popped off the stack (line 29). This state represents the entering point from the right in a further tape segment located at the left of the current segment and which is left in the state which is popped off the stack on line 31 (q_{24}), to re-enter the current segment from the left. Finally, on line 32, a new exit is selected, by choosing the tuple $(q_{24}, -1, q_{27}, +1)$. Since $d = +1$, the execution of the loop ends. Furthermore, the state so obtained matches the one expected when the macro was called. Thus the test on line 37 is negative and the execution of the macro ends without rejecting.

Summarizing, as a side effect of the execution of the macro the value of T is changed and now it contains $\tau_{b_2b_3b_4}$, the stack content is $\tau_{\triangleright b_1}$ and, after the execution of line 20 of Algorithm 1, $t_1t_2\cdots t_r$ is $q_9q_{10}q_{11}q_{16}q_{27}$. On lines 21–23 the value of T is updated to enlarge the current segment, including the table on the stack top. The fifth iteration starts by saving, on line 5, information on the stack (see Figure 3). The execution is continued in a similar way with the inspection of the remaining tape cells. ■

If Algorithm 1 correctly guessed an accepting computation, then after inspecting the right endmarker no pending states are left on the stack and the current tape segment should coincide with the entire tape. So, its transition table is stored in the variable T while ending the execution of the main loop, with no information on the stack. For this reason, the algorithm ends by accepting the input if and only if the stack is empty.

As a consequence of the simulation, we obtain the following:

Theorem 11. *For any integer $k \geq 1$, each n-state k-*LA *which accepts the input by violating the right endmarker and rewriting all input cells by symbols from Γ_k, can be simulated by a* PDA *M' with $2^{O(n^2)}$ states and a pushdown alphabet of $n + 2^{4n^2}$ symbols, such that in each transition at most $k + 2$ symbols are pushed on the stack.*

PROOF. First of all, we observe that if the given k-LA M accepts a language L, then the PDA resulting from Algorithm 1 accepts the language $L\triangleleft$. However, it can be easily modified in order to accept L, by guessing the end of the input at the beginning of each iteration of the main loop.

To estimate the size of the resulting PDA, we observe that the number of possible transition tables is 2^{4n^2}. At each step the finite state control of M' has to remember the current transition table, the current input symbol, two state sequences whose total lengths are in $O(k)$, the symbol finally written on the tape cell, a fixed number of variables ranging on the set of states and on the set $\{-1, +1\}$, plus control flow information. Hence the total number of states is bounded by the product of 2^{4n^2} by a polynomial in n. This gives $2^{O(n^2)}$ many states.

The pushdown alphabet consists of states and transition tables. Hence its cardinality is $n + 2^{4n^2}$. Push operations are executed only on line 5 of Algorithm 1 where it can be observed that at most $k + 1$ states and one transition table are pushed in one single operation. □

Corollary 12. *For any $k \geq 1$, each k-*LA *M can be simulated by a* PDA *whose description has a size which is exponential in the size of the description of M. Furthermore, an exponential size is also necessary.*

PROOF. The upper bound is an immediate consequence of Theorem 11. For $k \geq 2$, the lower bound follows from an exponential gap between the size of 2-LAs and equivalent pushdown automata proved in [21].

Now we discuss the case $k = 1$ for which limited automata recognize only regular languages [28]. There is an exponential gap between the size of 1-LAs and the size of equivalent nondeterministic finite automata. The gap becomes doubly exponential when the simulating automaton is required to be deterministic. These results have been proved in [20] by presenting for each $n \geq 1$ a witness language L_n which is accepted by a 1-LA whose description has size $O(n)$. The argument used in [21, Thm. 4.3] to prove the exponential separation between 2-LAs and PDAs can be immediately adapted to show that each PDA accepting L_n should have size at least exponential in n. □

Acknowledgments. The authors would like to thank the anonymous referees for valuable remarks and useful suggestions.

References

[1] Blum, M., Hewitt, C., 1967. Automata on a 2-dimensional tape. In: Symposium on Switching and Automata Theory (SWAT 1967). IEEE, pp. 155–160.

[2] Ellul, K., 2004. Descriptional complexity measures of regular languages. Master's thesis, University of Waterloo, Canada.

[3] Geffert, V., Mereghetti, C., Pighizzini, G., 2003. Converting two-way nondeterministic unary automata into simpler automata. Theoret. Comput. Sci. 295, 189–203.

[4] Hartmanis, J., 1968. Computational complexity of one-tape Turing machine computations. J. ACM 15, 325–339.

[5] Hennie, F. C., 1965. One-tape, off-line Turing machine computations. Inform. Control 8, 553–578.

[6] Hibbard, T. N., 1967. A generalization of context-free determinism. Inform. Control 11, 196–238.

[7] Holzer, M., Kutrib, M., 2003. Unary language operations and their nondeterministic state complexity. In: Developments in Language Theory (DLT 2002). Vol. 2450 of LNCS. Springer, pp. 162–172.

[8] Kapoutsis, C. A., Královič, R., Mömke, T., 2012. Size complexity of rotating and sweeping automata. J. Comput. System Sci. 78, 537–558.

[9] Kunc, M., Okhotin, A., 2011. On deterministic two-way finite automata over a unary alphabet. Tech. Rep. 950, Turku Centre for Computer Science.

[10] Kunc, M., Okhotin, A., 2012. State complexity of operations on two-way finite automata over a unary alphabet. Theoret. Comput. Sci. 449, 106–118.

[11] Kutrib, M., Malcher, A., Wendlandt, M., 2014. Simulations of unary one-way multi-head finite automata. Int. J. Found. Comput. Sci. 25, 877–896.

[12] Landau, E., 1903. Über die Maximalordnung der Permutationen gegebenen Grades. Archiv der Math. und Phys. 3, 92–103.

[13] Landau, E., 1909. Handbuch der Lehre von der Verteilung der Primzahlen. Teubner, Leipzig.

[14] Mera, F., Pighizzini, G., 2005. Complementing unary nondeterministic automata. Theoret. Comput. Sci. 330, 349–360.

[15] Mereghetti, C., Pighizzini, G., 2001. Optimal simulations between unary automata. SIAM J. Comput. 30, 1976–1992.

[16] Meyer, A. R., Fischer, M. J., 1971. Economy of description by automata, grammars, and formal systems. In: Symposium on Switching and Automata Theory (SWAT 1971). IEEE, pp. 188–191.

[17] Nicolas, J.-L., 1968. Sur l'ordre maximum d'un élément dans le groupe S_n des permutations. Acta Arith. 14, 315–332.

[18] Pighizzini, G., 2009. Deterministic pushdown automata and unary languages. Int. J. Found. Comput. Sci. 20, 629–645.

[19] Pighizzini, G., 2014. Strongly limited automata. In: Non-Classical Models of Automata and Applications (NCMA 2014). Vol. 304 of books@ocg.at. Austrian Computer Society, pp. 191–206.

[20] Pighizzini, G., Pisoni, A., 2014. Limited automata and regular languages. Int. J. Found. Comput. Sci. 25, 897–916.

[21] Pighizzini, G., Pisoni, A., 2015. Limited automata and context-free languages. Fund. Inform. 136, 157–176.

[22] Pighizzini, G., Shallit, J., 2002. Unary language operations, state complexity and Jacobsthal's function. Int. J. Found. Comput. Sci. 13, 145–159.

[23] Pighizzini, G., Shallit, J., Wang, M.-w., 2002. Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. J. Comput. System Sci. 65, 393–414.

[24] Průša, D., 2014. Weight-reducing Hennie machines and their descriptional complexity. In: Language and Automata Theory and Applications (LATA 2014). Vol. 8370 of LNCS. Springer, pp. 553–564.

[25] Rabin, M. O., Scott, D., 1959. Finite automata and their decision problems. IBM J. Res. Dev. 3, 114–125.

[26] Shepherdson, J. C., 1959. The reduction of two-way automata to one-way automata. IBM J. Res. Dev. 3, 198–200.

[27] Szalay, M., 1980. On the maximal order in S_n and S_n^*. Acta Arithm. 37, 321–331.

[28] Wagner, K., Wechsung, G., 1986. Computational Complexity. Reidel, Dordrecht.

[29] Wechsung, G., Brandstädt, A., 1979. A relation between space, return and dual return complexities. Theoret. Comput. Sci. 9, 127–140.

[30] Yu, S., 2001. State complexity of regular languages. J. Autom. Lang. Comb. 6, 221–234.