Forgetting 1-Limited Automata

Giovanni Pighizzini
Dipartimento di Informatica
Università degli Studi di Milano, Italy
pighizzini@di.unimi.it

Luca Prigioniero
Department of Computer Science
Loughborough University, UK
l.prigioniero@lboro.ac.uk

We introduce and investigate *forgetting* 1*-limited automata*, which are single-tape Turing machines that, when visiting a cell for the first time, replace the input symbol in it by a fixed symbol, so forgetting the original contents. These devices have the same computational power as finite automata, namely they characterize the class of regular languages. We study the cost in size of the conversions of forgetting 1-limited automata, in both nondeterministic and deterministic cases, into equivalent one-way nondeterministic and deterministic automata, providing optimal bounds in terms of exponential or superpolynomial functions. We also discuss the size relationships with two-way finite automata. In this respect, we prove the existence of a language for which forgetting 1-limited automata are exponentially larger than equivalent minimal deterministic two-way automata.

1 Introduction

Limited automata have been introduced in 1967 by Hibbard, with the aim of generalizing the notion of determinism for context-free languages [6]. These devices regained attention in the last decade, mainly from a descriptional complexity point of view, and they have been considered in several papers, starting with [14, 15]. (For a recent survey see [13].)

In particular, 1*-limited automata* are single-tape nondeterministic Turing machines that are allowed to rewrite the content of each tape cell only in the first visit. They have the same computational power as finite automata [24, Thm. 12.1], but they can be extremely more succinct. Indeed, in the worst case the size gap from the descriptions of 1-limited automata to those of equivalent one-way deterministic finite automata is double exponential [14].

In order to understand this phenomenon better, we recently studied two restrictions of 1-limited automata [17]. In the first restriction, called *once-marking* 1*-limited automata*, during each computation the machine can make only one change to the tape, just marking exactly one cell during the first visit to it. We proved that, under this restriction, a double exponential size gap to one-way deterministic finite automata remains possible.

In the second restriction, called *always-marking* 1*-limited automata*, each tape cell is marked during the first visit. In this way, at each step of the computation, the original content in the cell remains available, together with the information saying if it has been already visited at least one time. In this case, the size gap to one-way deterministic finite automata reduces to a single exponential. However, the information about which cells have been already visited still gives extra descriptional power. In fact, the conversion into equivalent two-way finite automata in the worst case costs exponential in size, even if the original machine is deterministic and the target machine is allowed to make nondeterministic choices.

A natural way to continue these investigations is to ask what happens if in each cell the information about the original input symbol is lost after the first visit. This leads us to introduce and study the subject of this paper, namely *forgetting* 1*-limited automata*. These devices are 1-limited automata in which, during the first visit to a cell, the input symbol in it is replaced with a unique fixed symbol. Forgetting

Rudolf Freund, Benedek Nagy (Eds.): 13th International Workshop on
Non-Classical Models of Automata and Applications (NCMA 2023)
EPTCS 388, 2023, pp. 95–109, doi:10.4204/EPTCS.388.10

automata have been introduced in the literature longtime ago [8]. Similarly to the devices we consider here, they can use only one fixed symbol to replace symbols on the tape. However, the replacement is not required to happen in the first visit, so giving the possibility to recognize more than regular languages. In contrast, being a restriction of 1-limited automata, forgetting 1-limited automata recognize only regular languages.

In this paper, first we study the size costs of the simulations of forgetting 1-limited automata, in both nondeterministic and deterministic versions, by one-way finite automata. The upper bounds we prove are exponential, when the simulated and the target machines are nondeterministic and deterministic, respectively. In the other cases they are superpolynomial. These bounds are obtained starting from the conversions of always-marking 1-limited automata into one-way finite automata presented in [17], whose costs, in the case we are considering, can be reduced using techniques and results derived in the context of automata over a one-letter alphabet [2, 11]. We also provide witness languages showing that these upper bounds cannot be improved asymptotically.

In the last part of the paper we discuss the relationships with the size of two-way finite automata, which are not completely clear. We show that losing the information on the input content can reduce the descriptional power. In fact, we show languages for which forgetting 1-limited automata, even if nondeterministic, are exponentially larger than minimal two-way deterministic finite automata. We conjecture that also the converse can happen. In particular we show a family of languages for which we conjecture that two-way finite automata, even if nondeterministic, must be significantly larger than minimal deterministic forgetting 1-limited automata.

2 Preliminaries

In this section we recall some basic definitions useful in the paper. Given a set S, $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ and a string $w \in \Sigma^*$, $|w|$ denotes the length of w, $|w|_a$ the number of occurrences of a in w, and Σ^k the set of all strings on Σ of length k.

We assume the reader to be familiar with notions from formal languages and automata theory, in particular with the fundamental variants of finite automata (1DFAS, 1NFAS, 2DFAS, 2NFAS, for short, where 1/2 mean *one-way*/*two-way* and D/N mean *deterministic*/*nondeterministic*, respectively). For any unfamiliar terminology see, e.g., [7].

A 1-*limited automaton* (1-LA, for short) is a tuple $A = (Q, \Sigma, \Gamma, \delta, q_I, F)$, where Q is a finite *set of states*, Σ is a finite *input alphabet*, Γ is a finite *work alphabet* such that $\Sigma \cup \{\triangleright, \triangleleft\} \subseteq \Gamma$, $\triangleright, \triangleleft \notin \Sigma$ are two special symbols, called the *left* and the *right end-markers*, $\delta : Q \times \Gamma \to 2^{Q \times (\Gamma \setminus \{\triangleright, \triangleleft\}) \times \{-1,+1\}}$ is the *transition function*, and $F \subseteq Q$ is a set of final states. At the beginning of the computation, the input word $w \in \Sigma^*$ is stored onto the tape surrounded by the two end-markers, the left end-marker being in position zero and the right end-marker being in position $|w| + 1$. The head of the automaton is on cell 1 and the state of the finite control is the *initial state* q_I.

In one move, according to δ and the current state, A reads a symbol from the tape, changes its state, replaces the symbol just read from the tape with a new symbol, and moves its head to one position forward or backward. Furthermore, the head cannot pass the end-markers, except at the end of computation, to accept the input, as explained below. Replacing symbols is allowed to modify the content of each cell only during the first visit, with the exception of the cells containing the end-markers, which are never modified. Hence, after the first visit, a tape cell is “frozen”. More technical details can be found in [14].

The automaton A accepts an input w if and only if there is a computation path that starts from the initial state q_I with the input tape containing w surrounded by the two end-markers and the head on the first input cell, and which ends in a *final state* $q \in F$ after passing the right end-marker. The device A is said to be *deterministic* (D-1-LA, for short) whenever $\#\delta(q,\sigma) \leq 1$, for every $q \in Q$ and $\sigma \in \Gamma$.

We say that the 1-LA A is a *forgetting* 1-LA (for short F-1-LA or D-F-1-LA in the deterministic case), when there is only one symbol Z that is used to replace symbols in the first visit, i.e., the work alphabet is $\Gamma = \Sigma \cup \{Z\} \cup \{\rhd, \lhd\}$, with $Z \notin \Sigma$ and if $(q,A,d) \in \delta(p,a)$ and $a \in \Sigma$ then $A = Z$.

Two-way finite automata are limited automata in which no rewritings are possible; one-way finite automata can scan the input in a one-way fashion only. A finite automaton is, as usual, a tuple (Q,Σ,δ,q_I,F), where, analogously to 1-LAs, Q is the finite set of states, Σ is the finite input alphabet, δ is the transition function, q_I is the initial state, and F is the set of final states. We point out that for two-way finite automata we assume the same accepting conditions as for 1-LAs.

Two-way machines in which the direction of the head can change only at the end-markers are said to be *sweeping* [22].

In this paper we are interested in comparing the size of machines. The *size* of a model is given by the total number of symbols used to write down its description. Therefore, the size of 1-LAs is bounded by a polynomial in the number of states and of work symbols, while, in the case of finite automata, since no writings are allowed, the size is linear in the number of instructions and states, which is bounded by a polynomial in the number of states and in the number of input symbols. We point out that, since F-1-LAs use work alphabet $\Gamma = \Sigma \cup \{Z\} \cup \{\rhd, \lhd\}$, $Z \notin \Sigma$, the relevant parameter for evaluating the size of these devices is their number of states, differently than 1-LAs, in which the size of the work alphabet is not fixed, i.e., depends on the machine.

We now shortly recall some notions and results related to number theory that will be useful to obtain our cost estimations. First, given two integers m and n, let us denote by $\gcd(m,n)$ and by $\mathrm{lcm}(m,n)$ their *greatest common divisor* and *least common multiple*, respectively.

We remind the reader that each integer $\ell > 1$ can be factorized in a unique way as product of powers of primes, i.e., as $\ell = p_1^{k_1} \cdots p_r^{k_r}$, where $p_1 < \cdots < p_r$ are primes, and $k_1, \ldots, k_r > 0$.

In our estimations, we shall make use of the *Landau's function* $F(n)$ [9, 10], which plays an important role in the analysis of simulations among different types of unary automata (e.g. [2, 4, 11]). Given a positive integer n, let

$$F(n) = \max\{\mathrm{lcm}(\lambda_1, \ldots, \lambda_r) \mid \lambda_1 + \cdots + \lambda_r = n\},$$

where $\lambda_1, \ldots, \lambda_r$ denote, for the time being, arbitrary positive integers. Szalay [23] gave a sharp estimation of $F(n)$ that, after some simplifications, can be formulated as follows:

$$F(n) = e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}.$$

Note that the function $F(n)$ grows less than e^n, but more than each polynomial in n. In this sense we say that $F(n)$ is a *superpolynomial function*.

As observed in [5], for each integer $n > 1$ the value of $F(n)$ can also be expressed as the maximum product of powers of primes, whose sum is bounded by n, i.e.,

$$F(n) = \max\{p_1^{k_1} \cdots p_r^{k_r} \mid p_1^{k_1} + \cdots + p_r^{k_r} \leq n,\ p_1, \ldots, p_r \text{ are primes, and } k_1, \ldots, k_r > 0\}.$$

3 Forgetting 1-Limited Automata vs. One-Way Automata

When forgetting 1-limited automata visit a cell for the first time, they replace the symbol in it with a fixed symbol Z, namely they forget the original content. In this way, each input prefix can be rewritten in a unique way. As already proved for *always-marking* 1-LAs, this prevents a double exponential size gap in the conversion to 1DFAs [17]. However, in this case the upper bounds obtained for always-marking 1-LAs, can be further reduced, using the fact that only one symbol is used to replace input symbols:

Theorem 1 *Let M be an n-state* F-1-LA. *Then M can be simulated by a* 1NFA *with at most $n \cdot (5n^2 + F(n)) + 1$ states and by a complete* 1DFA *with at most $(2^n - 1) \cdot (5n^2 + F(n)) + 2$ states.*

Proof. First of all, we recall the argument for the conversion of 1-LAs into 1NFAs and 1DFAs presented [14, Thm. 2] that, in turn, is derived from the technique to convert 2DFAs into equivalent 1DFAs, presented in [21], and based on *transitions tables*.

Let us start by supposing that $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is an n-state 1-LA.

Roughly, transition tables represent the possible behaviors of M on "frozen" tape segments. More precisely, given $z \in \Gamma^*$, the *transition table* associated with z is the binary relation $\tau_z \subseteq Q \times Q$, consisting of all pairs (p, q) such that M has a computation path that starts in the state p on the rightmost symbol of a tape segment containing $\triangleright z$, ends reaching the state q by leaving the same tape segment to the right side, i.e., by moving from the rightmost cell of the segment to the right, and does not visit any cell outside the segment.

A 1NFA A can simulate M by keeping in the finite control two components:

- The transition table corresponding to the part of the tape at the left of the head. This part has been already visited and, hence, it is frozen.
- The state in which the simulated computation of M reaches the current tape position.

Since the number of transition tables is at most 2^{n^2}, the number of states in the resulting 1NFA A is bounded by $n \cdot 2^{n^2}$.

Applying the subset construction, this automaton can be converted into an equivalent deterministic one, with an exponential increasing in the number of states, so obtaining a double exponential number of states in n. In the general case, this number cannot be reduced due to the fact that different computations of A, after reading the same input, could keep in the control different transition tables, depending on the fact that M could replace the same input by different strings.

We now suppose that M is a F-1-LA. In this case each input string can be replaced by a unique string. This would reduce the cost of the conversion to 1DFAs to a single exponential. Indeed, it is possible to convert the 1NFA A obtained from M into an equivalent 1DFA that keeps in its finite control the *unique* transition table for the part of the tape scanned so far (namely, the same first component as in the state of A), and the set of states that are reachable by M when entering the current tape cell (namely, a set of states that can appear in the second component of A, while entering the current tape cell). This leads to an upper bound of $2^n \cdot 2^{n^2}$ states for the resulting 1DFA. We can make a further improvement, reducing the number of transition tables used during the simulation. Indeed we are going to prove that only a subset of all the possible 2^{n^2} transition tables can appear during the simulation.

Since only a fixed symbol Z is used to replace input symbols on the tape, the transition table when the head is in a cell depends only on the position of the cell and not on the initial tape content.

For each integer $m \geq 0$, let us call τ_m the transition table corresponding to a frozen tape segment of length m, namely the transition table when the head of the simulating one-way automaton is on the tape cell $m+1$. We are going to prove that the sequence $\tau_0, \tau_1, \ldots, \tau_m, \ldots$ is ultimately periodic, with period length bounded by $F(n)$ and, more precisely, $\tau_m = \tau_{m+F(n)}$ for each $m > 5n^2$.

The proof is based on the analysis of computation paths in unary 2NFAs carried on in [11, Section 3]. Indeed, we can see the parts of the computation on a frozen tape segment as computation paths of a unary 2NFA. More precisely, by definition, for $p, q \in Q$, $\tau_m(p, q) = 1$ if and only if there is a computation path C that enters the frozen tape segment of length m from the right in the state p and, after some steps, exits the segment to the right in the state q. Hence, during the path C the head can visit only frozen cells (i.e., the cells in positions $1,\ldots,m$) of the tape, and the left end-marker. There are two possible cases:

- *In the computation path C the head never visits the left end-marker.*
  A path of this kind is also called *left U-turn*. Since it does not depend on the position of the left end-marker, this path will also be possible, suitably shifted to the right, on each frozen segment of length $m' > m$. Hence $\tau_{m'}(p,q) = 1$ for each $m' \geq m$. Furthermore, it has been proven that if there is a left U-turn which starts in the state p on cell m, and ends in state q, then there exists another left U-turn satisfying the same constraints, in which the head never moves farther than n^2 positions to the left of the position m [11, Lemma 3.1]. So, such a "short" U-turn can be shifted to the left, provided that the tape segment is longer than n^2.

  Hence, in this case $\tau_m(p,q) = 1$ implies $\tau_{m'}(p,q) = 1$ for each $m' > n^2$.

- *In the computation path C the head visits at least one time the left end-marker.*
  Let $s_0, s_1, \ldots, s_k$, $k \geq 0$, be the sequence of the states in which C visits the left end-marker. We can decompose C in a sequence of computation paths $C_0, C_1, \ldots, C_k, C_{k+1}$, where:
  - C_0 starts from the state p with the head on the cell m and ends in s_0 when the head reaches the left end-marker. C_0 is called *right-to-left traversal* of the frozen segment.
  - For $i = 1,\ldots,k$, C_i starts in state s_{i-1} with the head on the left end-marker and ends in s_i, when the head is back to the left end-marker. C_i is called *right U-turn*. Since, as seen before for left U-turns, each right U-turn can always be replaced by a "short" right U-turn, without loss of generality we suppose that C_i does not visit more than n^2 cells to the right of the left end-marker.
  - C_{k+1} starts from the state s_k with the head on the left end-marker and ends in q, when the head leaves the segment, moving to the right of the cell m. C_{k+1} is called *left-to-right traversal* of the frozen segment.

  From [11, Theorem 3.5], there exists a set of positive integers $\{\ell_1,\ldots,\ell_r\} \subseteq \{1,\ldots,n\}$ satisfying $\ell_1 + \cdots + \ell_r \leq n$ such that for $m \geq n$, if a frozen tape segment of length m can be (left-to-right or right-to-left) traversed from a state s to a state s' then there is an index $i \in \{1,\ldots,r\}$ such that, for each $\mu > \frac{5n^2-m}{\ell_i}$, a frozen tape segment of length $m + \mu\ell_i$ can be traversed (in the same direction) from state s to state s'. This was proved by showing that for $m > 5n^2$ a traversal from s to s' of a segment of length m can always be "pumped" to obtain a traversal of a segment of length $m' = m + \mu\ell_i$, for $\mu > 0$, and, furthermore, the segment can be "unpumped" by taking $\mu < 0$, provided that the resulting length m' is greater than $5n^2$.

  Let ℓ be the least common multiple of $\ell_1,\ldots,\ell_r$. If $m > 5n^2$, from the original computation path C, by suitably pumping or unpumping the parts C_0 and C_{k+1}, and without changing C_i, for $i = 1,\ldots,k$, for each $m' = m + \mu\ell > 5n^2$, with $\mu \in \mathbb{Z}$, we can obtain a computation path that enters a frozen segment of length m' from the right in the state p and exits the segment to the right in the state q.

By summarizing, from the previous analysis we conclude that for all $m, m' > 5n^2$, if $m \equiv m' \pmod{\ell}$ then $\tau_m = \tau_{m'}$. Hence, the transition tables used in the simulation are at most $5n^2 + \ell$. Since, by definition, ℓ cannot exceed $F(n)$, we obtain the number of different transitions tables that are used in the simulation is bounded by $5n^2 + 1 + F(n)$.

According with the construction outlined at the beginning of the proof, from the F-1-LA M we can obtain a 1NFA A that, when the head reaches the tape cell $m+1$, has in the first component of its finite control the transition table τ_m, and in the second component the state in which the cell $m+1$ is entered for the first time during the simulated computation. Hence the total number of states of A is bounded by $n \cdot (5n^2 + 1 + F(n))$.

We observe that, at the beginning of the computation, the initial state is the pair containing the transition matrix τ_0 and the initial state of M. Hence, we do not need to consider other states with τ_0 as first component, unless τ_0 occurs in the sequence $\tau_1, \ldots, \tau_{5n^2+F(n)}$. This allows to reduce the upper bound to $n \cdot (5n^2 + F(n)) + 1$

If the simulating automaton A is a 1DFA, then first component does not change, while the second component contains the set of states in which the cell $m+1$ is entered for the first time during all possible computations of M. This would give a $2^n \cdot (5n^2 + F(n)) + 1$ state upper bound. However, if the set in the second component is empty then the computation of M is rejecting, regardless what is the remaining part of the input and what has been written on the tape. Hence, in this case, the simulating 1DFA can enter a sink state. This allows to reduce the upper bound to $(2^n - 1) \cdot (5n^2 + F(n)) + 2$. □

Optimality: The Language $\mathscr{L}_{n,\ell}$

We now study the optimality of the state upper bounds presented in Theorem 1. To this aim, we introduce a family of languages $\mathscr{L}_{n,\ell}$, that are defined with respect to integer parameters $n, \ell > 0$.

Each language in this family is composed by all strings of length multiple of ℓ belonging to the language L_{MF_n} which is accepted by the n-state 1NFA $A_{MF_n} = (Q_n, \{a,b\}, \delta_n, q_0, \{q_0\})$ depicted in Figure 1, i.e., $\mathscr{L}_{n,\ell} = L_{MF_n} \cap (\{a,b\}^\ell)^*$.

The automaton A_{MF_n} was proposed longtime ago by Meyer and Fischer as a witness of the exponential state gap from 1NFAs to 1DFAs [12]. Indeed, it can be proved that the smallest 1DFA accepting it has exactly 2^n states. In the following we shall refer to some arguments given in the proof of such result presented in [20, Thm. 3.9.6].

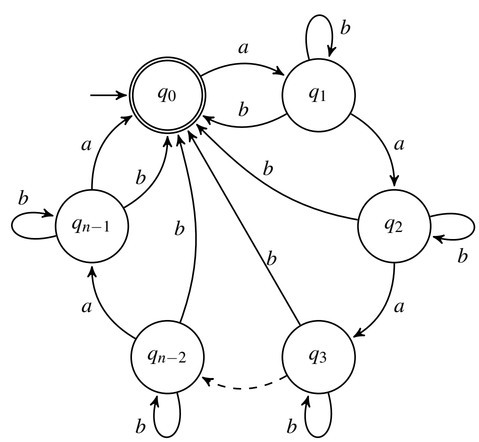

Figure 1: The 1NFA A_{MF_n} accepting the language of Meyer and Fischer.

Let us start by presenting some simple state upper bounds for the recognition of $\mathscr{L}_{n,\ell}$ by one-way finite automata.

Theorem 2 *For every two integers $n, \ell > 0$, there exists a complete 1DFA accepting $\mathscr{L}_{n,\ell}$ with $(2^n - 1) \cdot \ell + 1$ states and a 1NFA with $n \cdot \ell$ states.*

Proof. We apply the subset construction to convert the 1NFA A_{MF_n} into a 1DFA with 2^n states and then, with the standard product construction, we intersect the resulting automaton with the trivial ℓ-state automaton accepting $(\{a,b\}^\ell)^*$. In this way we obtain a 1DFA with $2^n \cdot \ell$ states for $\mathscr{L}_{n,\ell}$. However, all the states obtained from the sink state, corresponding to the empty set, are equivalent, so they can be replaced by a unique sink state. This allows to reduce the number of states to $(2^n - 1) \cdot \ell + 1$.

In the case of 1NFAs we apply the product construction to A_{MF_n} and the ℓ-state automaton accepting $(\{a,b\}^\ell)^*$, so obtaining a 1NFA with $n \cdot \ell$ states. □

We now study how to recognize $\mathscr{L}_{n,\ell}$ using two-way automata and F-1-LAs. In both cases we obtain sweeping machines.

Theorem 3 *Let $\ell > 0$ be an integer that factorizes $\ell = p_1^{k_1} \cdots p_r^{k_r}$ as a product of prime powers and $o = r \bmod 2$. Then:*

- *$\mathscr{L}_{n,\ell}$ is accepted by a sweeping 2NFA with $n + p_1^{k_1} + \cdots + p_r^{k_r} + o$ states, that uses nondeterministic transitions only in the first sweep.*
- *$\mathscr{L}_{n,\ell}$ is accepted by a sweeping F-1-LA with $\max(n, p_1^{k_1} + \cdots + p_r^{k_r} + o)$ states that uses nondeterministic transitions only in the first sweep.*
- *$\mathscr{L}_{n,\ell}$ is accepted by a sweeping 2DFA with $2n + p_1^{k_1} + \cdots + p_r^{k_r} + o$ states.*

Proof. In the first sweep, the 2NFA for $\mathscr{L}_{n,\ell}$, using n states, simulates the 1NFA A_{MF_n} to check if the input belongs to L_{MF_n}. Then, it makes one sweep for each $i = 1, \ldots, r$ (alternating a right-to-left sweep with a left-to-right sweep), using $p_i^{k_i}$ states in order to check whether $p_i^{k_i}$ divides the input length. If the outcomes of all these tests are positive, then the automaton accepts. When r is even, the last sweep ends with the head on the right end-marker. Then, moving the head one position to the right, the automaton can reach the accepting configuration. However, when r is odd, the last sweep ends on the left end-marker. Hence, using an extra state, the head can traverse the entire tape to finally reach the accepting configuration.

A F-1-LA can implement the same strategy. However, to check if the tape length is a multiple of ℓ, it can reuse the n states used in the first sweep, plus $p_1^{k_1} + \cdots + p_r^{k_r} + o - n$ extra states when $n < p_1^{k_1} + \cdots + p_r^{k_r} + o$. This is due to the fact that the value of the transition function depends on the state and on the symbol in the tape cell and that, in the first sweep, all the input symbols have been replaced by Z.

Finally, we can implement a 2DFA that recognizes $\mathscr{L}_{n,\ell}$ by firstly making r sweeps to check whether $p_i^{k_i}$ divides the input length, $i = 1, \ldots, r$. If so, then the automaton, after moving the head from the left to the right end-marker in case of r even, makes a further sweep from right to left, to simulate a 1DFA accepting the reversal of L_{MF_n}, which can be accepted using $2n$ states [19]. If the simulated automaton accepts, then the machine can make a further sweep, by using a unique state to move the head from the left end-marker to the right one, and then accept. The total number of states is $2n + p_1^{k_1} + \cdots + p_r^{k_r} + 2 - o$. This number can be slightly reduced as follows: in the first sweep (which is from left to right) the automaton checks the divisibility of the input length by $p_1^{k_1}$; in the second sweep (from right to left) the automaton checks the membership to L_{MF_n}; in the remaining $r - 1$ sweeps (alternating left-to-right with right-to-left sweeps), it checks the divisibility for $p_i^{k_i}$, $i = 2, \ldots, r$. So, the total number of sweeps for these checks is $r + 1$. This means that, when r is even, the last sweep ends on the right end-marker and the machine can immediately move to the accepting configuration. Otherwise the head needs to cross the input from left to right, using an extra state. □

As a consequence of Theorem 3, in the case of F-1-LAs we immediately obtain:

Corollary 1 *For each $n > 0$ the language $\mathscr{L}_{n,F(n)}$ is accepted by a* F-1-LA *with at most $n+1$ states.*

Proof. If $F(n) = p_1^{k_1} \cdots p_r^{k_r}$ then $p_1^{k_1} + \cdots + p_r^{k_r} \le n \le F(n)$. Hence, the statement follows from Theorem 3. □

We are now going to prove lower bounds for the recognition of $\mathscr{L}_{n,\ell}$, in the case n and ℓ are relatively primes.

Let us start by considering the recognition by 1DFAs.

Theorem 4 *Given two integers $n, \ell > 0$ with $\gcd(n,\ell) = 1$, each* 1DFA *accepting $\mathscr{L}_{n,\ell}$ must have at least $(2^n - 1) \cdot \ell + 1$ states.*

Proof. Let $Q_n = \{q_0, q_1, \ldots, q_{n-1}\}$ be the set of states of A_{MF_n} (see Figure 1). First, we briefly recall some arguments from the proof presented in [20, Thm. 3.9.6]. For each subset S of Q_n, we define a string w_S having the property that $\delta_n(q_0, w_S) = S$. Furthermore, it is proved that all the strings so defined are pairwise distinguishable, so obtaining the state lower bound 2^n for each 1DFA equivalent to A_{MF_n}. In particular, the string w_S is defined as follows:

$$w_S = \begin{cases} b & \text{if } S = \emptyset; \\ a^i & \text{if } S = \{q_i\}; \\ a^{e_k - e_{k-1}} b a^{e_{k-1} - e_{k-2}} b \cdots a^{e_2 - e_1} b a^{e_1}, & \text{otherwise;} \end{cases} \tag{1}$$

where in the second case $S = \{q_i\}$, $0 \le i < n$, while in the third case $S = \{q_{e_1}, q_{e_2}, \ldots, q_{e_k}\}$, $1 < k \le n$, and $0 \le e_1 < e_2 < \cdots < e_k < n$.

To obtain the claimed state lower bound in the case of the language $\mathscr{L}_{n,\ell}$, for each nonempty subset S of Q_n and each integer j, with $0 \le j < \ell$, we define a string $w_{S,j}$ which is obtained by suitably padding the string w_S in such a way that the set of states reachable from the initial state by reading $w_{S,j}$ remains S and the length of $w_{S,j}$, divided by ℓ, gives j as reminder. Then we shall prove that all the so obtained strings are pairwise distinguishable. Unlike (1), when defining $w_{S,j}$ we do not consider the case $S = \emptyset$.

In the following, let us denote by $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ a function satisfying $f(i,j) \bmod n = i$ and $f(i,j) \bmod \ell = j$, for $i, j \in \mathbb{N}$. Since $\gcd(n,\ell) = 1$, by the Chinese Reminder Theorem, such a function always exists.

For each non-empty subset S of Q_n and each integer j, with $0 \le j < \ell$, the string $w_{S,j}$ is defined as:

$$w_{S,j} = \begin{cases} a^{f(i,j)} & \text{if } S = \{q_i\}; \\ a^{e_k - e_{k-1}} b a^{e_{k-1} - e_{k-2}} b \cdots a^{e_2 - e_1} b^{H\ell - k - e_k + 2 + j} a^{e_1}, & \text{otherwise;} \end{cases} \tag{2}$$

where in the first case $S = \{q_i\}$, $0 \le i < n$, while in the second case $S = \{q_{e_1}, q_{e_2}, \ldots, q_{e_k}\}$, $1 < k \le n$, $0 \le e_1 < e_2 < \cdots < e_k < n$, and $H \ge 1$ is a fixed integer such that $H\ell > 2n$ (this last condition is useful to have $H\ell - k - e_k + 2 + j > 0$, in such a way that the last block of b's is always well defined and not empty).

We claim and prove the following facts:

1. $|w_{S,j}| \bmod \ell = j$.
   If $S = \{q_i\}$, then by definition $|w_{S,j}| \bmod \ell = f(i,j) \bmod \ell = j$. Otherwise, according to the second case in (2), $S = \{q_{e_1}, q_{e_2}, \ldots, q_{e_k}\}$ and $|w_{S,j}| = e_k - e_{k-1} + 1 + e_{k-1} - e_{k-2} + 1 + \cdots + e_2 - e_1 + H\ell - k - e_k + 2 + j + e_1$, which is equal to $H\ell + j$.
2. $\delta_n(q_0, w_{S,j}) = S$.
   In the automaton A_{MF_n}, all the transitions on the letter a are deterministic. Furthermore, by reading

the string a^x, $x > 0$, from the state q_0, the only reachable state is $q_{x \bmod n}$. Hence, for the first case $S = \{q_i\}$ in (2) we have $\delta_n(q_0, w_{S,j}) = \{q_{f(i,j) \bmod n}\} = \{q_i\}$.

For the second case, we already mentioned that $\delta_n(q_0, w_S) = S$. Furthermore $w_{S,j}$ is obtained from w_S by replacing the rightmost b by a block of more than one b. From the transition diagram of A_{MF_n} we observe that from each state q_i, with $i > 0$, reading a b the automaton can either remain in q_i or move to q_0. Furthermore, from q_0 there are no transitions on the letter b. This allows to conclude that the behavior does not change when one replaces an occurrence of b in a string with a sequence of more than one b. Hence, $\delta_n(q_0, w_{S,j}) = \delta_n(q_0, w_S) = S$.

3. *For $i = 0, \ldots, n-1$ and $x \geq 0$, $\delta_n(q_i, a^x) = \{q_{i'}\}$ where $i' = 0$ if and only if $x \bmod n = n - i$. Hence a^x is accepted by some computation path starting from q_i if and only if $x \bmod n = n - i$.*

   It is enough to observe that all the transitions on the letter a are deterministic and form a loop visiting all the states. More precisely $i' = (i + x) \bmod n$. Hence, $i' = 0$ if and only if $x \bmod n = n - i$.

We now prove that all the strings $w_{S,j}$ are pairwise distinguishable. To this aim, let us consider two such strings $w_{S,j}$ and $w_{T,h}$, with $(S, j) \neq (T, h)$. We inspect the following two cases:

- $S \neq T$. Without loss of generality, let us consider a state $q_s \in S \setminus T$. We take $z = a^{f(n-s,\ell-j)}$. By the previous claims, we obtain that $w_{S,j} \cdot z \in L_{MF_n}$, while $w_{T,h} \cdot z \notin L_{MF_n}$. Furthermore, $|w_{S,j} \cdot z| \bmod \ell = (j + \ell - j) \bmod \ell = 0$. Hence $w_{S,j} \cdot z \in (\{a,b\}^\ell)^*$. This allows to conclude that $w_{S,j} \cdot z \in \mathscr{L}_{n,\ell}$, while $w_{T,h} \cdot z \notin \mathscr{L}_{n,\ell}$.
- $j \neq h$. We choose a state $q_s \in S$ and, again, the string $z = a^{f(n-s,\ell-j)}$. Exactly as in the previous case we obtain $w_{S,j} \cdot z \in \mathscr{L}_{n,\ell}$. Furthermore, being $j \neq h$ and $0 \leq j, h < \ell$, we get that $|w_{T,h} \cdot z| \bmod \ell = (h + \ell - j) \bmod \ell \neq 0$. Hence $w_{T,h} \cdot z \notin (\{a,b\}^\ell)^*$, thus implying $w_{T,h} \cdot z \notin \mathscr{L}_{n,\ell}$.

By summarizing, we have proved that all the above defined $(2^n - 1) \cdot \ell$ strings $w_{S,j}$ are pairwise distinguishable. We also observe that each string starting with the letter b is not accepted by the automaton A_{MF_n}.[1] This implies that the string b and each string $w_{S,j}$ are distinguishable. Hence, we are able to conclude that each 1DFA accepting $\mathscr{L}_{n,\ell}$ has at least $(2^n - 1) \cdot \ell + 1$ states. □

Concerning 1NFAs, we prove the following:

Theorem 5 *Given two integers $n, \ell > 0$ with $\gcd(n, \ell) = 1$, each* 1NFA *accepting $\mathscr{L}_{n,\ell}$ must have at least $n \cdot \ell$ states.*

Proof. The proof can be easily given by observing that $X = \{(a^i, a^{n \cdot \ell - i}) \mid i = 0, \ldots, n \cdot \ell - 1\}$ is a *fooling set* for $\mathscr{L}_{n,\ell}$ [1]. Hence, the number of states of each 1NFA for $\mathscr{L}_{n,\ell}$ cannot be lower than the cardinality of X. □

As a consequence of Theorems 4 and 5 we obtain:

Theorem 6 *For each prime $n > 4$, every* 1DFA *and every* 1NFA *accepting $\mathscr{L}_{n,F(n)}$ needs $(2^n - 1) \cdot F(n) + 1$ and $n \cdot F(n)$ states, respectively.*

Proof. First, we prove that $\gcd(n, F(n)) = 1$ for each prime $n > 4$. To this aim, we observe that by definition $F(n) \geq 2 \cdot (n-2)$ for each prime n. Furthermore, if $n > 4$ then $2 \cdot (n-2) > n$. Hence $F(n) > n$ for each prime $n > 4$. Suppose that $\gcd(n, F(n)) \neq 1$. Then n, being prime and less than $F(n)$, should

[1] We point out that two strings that in A_{MF_n} lead to the emptyset are not distinguishable. This is the reason why we did not considered strings of the form $w_{\emptyset,j}$ in (2).

divide $F(n)$. By definition of $F(n)$, this would imply $F(n) = n$; a contradiction. This allows us to conclude that $\gcd(n, F(n)) = 1$, for each prime $n > 4$.

Using Theorems 4 and 5, we get that, for all such n's, a 1DFA needs at least $(2^n - 1) \cdot F(n) + 1$ states to accept $\mathscr{L}_{n,F(n)}$, while an equivalent 1NFA needs at least $n \cdot \ell$ states. □

As a consequence of Theorem 6, for infinitely many n, the 1DFA and 1NFA for the language $\mathscr{L}_{n,F(n)}$ described in Theorem 2 are minimal.

By combining the results in Corollary 1 and Theorem 6, we obtain that the costs of the simulations of F-1-LAs by 1NFAs and 1DFAs presented in Theorem 1 are asymptotically optimal:

Corollary 2 *For infinitely many integers n there exists a language which is accepted by a* F-1-LA *with at most $n+1$ states and such that all equivalent* 1DFA*s and* 1NFA*s require at least $(2^n - 1) \cdot F(n) + 1$ and $n \cdot F(n)$ states, respectively.*

4 Deterministic Forgetting 1-Limited Automata vs. One-Way Automata

In Section 3 we studied the size costs of the conversions from F-1-LAs to one-way finite automata. We now restrict our attention to the simulation of deterministic machines. By adapting to this case the arguments used to prove Theorem 1, we obtain a superpolynomial state bound for the conversion into 1DFAs, which is not so far from the bound obtained starting from nondeterminstic machines:

Theorem 7 *Let M be an n-state* D-F-1-LA. *Then M can be simulated by a* 1DFA *with at most $n \cdot (n + F(n)) + 2$ states.*

Proof. We can apply the construction given in the proof of Theorem 1 to build, from the given D-F-1-LA M, a one-way finite automaton that, when the head reaches the tape cell $m+1$, has in its finite control the transition table τ_m associated with the tape segment of length m and the state in which the cell is reached for the first time. Since the transitions of M are deterministic, each tape cell is reached for the first time by at most one computation and the resulting automaton is a (possible partial) 1DFA, with no more than $n \cdot (5n^2 + F(n)) + 1$ states. However, in this case the number of transition tables can be reduced, so decreasing the upper bound. In particular, due to determinism and the unary content in the frozen part, we can observe that left and right U-turns cannot visit more than n tape cells. Furthermore, after visiting more than n tape cells, a traversal is repeating a loop. This allows to show that the sequence of transition matrices starts to be periodic after the matrix τ_n, i.e, for $m, m' > n$, if $m \equiv m' \pmod{F(n)}$ then $\tau_m = \tau_{m'}$. Hence, the number of different transition tables used during the simulation is at most $n + 1 + F(n)$, and the number of states of the simulating (possibly partial) 1DFA is bounded by $n \cdot (n + F(n)) + 1$. By adding one more state we finally obtain a complete 1DFA. □

Optimality: The Language $\mathscr{J}_{n,\ell}$

We now present a family of languages for which we prove a size gap very close to the upper bound in Theorem 7. Given two integers $n, \ell > 0$, let us consider:

$$\mathscr{J}_{n,\ell} = \{w \in \{a,b\}^* \mid |w|_a \bmod n = 0 \text{ and } |w| \bmod \ell = 0\}.$$

First of all, we observe that it is not difficult to recognize $\mathscr{J}_{n,\ell}$ using a 1DFA with $n \cdot \ell$ states that counts the number of a's using one counter modulo n, and the input length using one counter modulo ℓ. This number of states cannot be reduced, even allowing nondeterministic transitions:

Theorem 8 *Each* 1NFA *accepting* $\mathscr{J}_{n,\ell}$ *has at least* $n \cdot \ell$ *states.*

Proof. Let $H > \ell + n$ be a multiple of ℓ. For $i = 1, \ldots, \ell$, $j = 0, \ldots, n-1$, consider $x_{ij} = a^j b^{H+i-j}$ and $y_{ij} = b^{H-i-n+j} a^{n-j}$. We are going to prove that the set

$$X = \{(x_{ij}, y_{ij}) \mid 1 \leq i \leq \ell, 0 \leq j < n\}$$

is an *extended fooling set* for $\mathscr{J}_{n,\ell}$. To this aim, let us consider $i, i' = 1, \ldots, \ell$, $j, j' = 0, \ldots, n-1$. We observe that the string $x_{ij}y_{ij}$ contains n a's and has length $j + H + i - j + H - i - n + j + n - j = 2H$ and hence it belongs to $\mathscr{J}_{n,\ell}$. For $i, i' = 1, \ldots, \ell$, if $i \neq i'$ then the string $x_{ij}y_{i'j} \notin \mathscr{J}_{n,\ell}$ because it has length $2H + i - i'$ which cannot be a multiple of ℓ. On the other hand, if $j < j'$, the string $x_{ij}y_{i'j'}$ contains $j + n - j' < n$ many a's, so it cannot belong to $\mathscr{J}_{n,\ell}$, ◻

Concerning the recognition of $\mathscr{J}_{n,\ell}$ by F-1-LAs we prove the following:

Theorem 9 *Let* $\ell > 0$ *be an integer that factorizes* $\ell = p_1^{k_1} \cdots p_s^{k_r}$ *as product of prime powers,* $o = r$ mod 2*, and* $n > 0$*. Then* $\mathscr{J}_{n,\ell}$ *is accepted by a sweeping* 2DFA *with* $n + p_1^{k_1} + \cdots + p_r^{k_r} + o$ *states and by a sweeping* D-F-1-LA *with* $\max(n, p_1^{k_1} + \cdots + p_r^{k_r} + o)$ *states.*

Proof. A 2DFA can make a first sweep of the input, using n states, to check if the number of a's in the input is a multiple of n. Then, in further r sweeps, alternating right-to-left with left-to-right sweeps, it can check the divisibility of the input length by $p_i^{k_i}$, $i = 1, \ldots, r$. If r is odd this process ends with the head on the left end-marker. Hence, in this case, when all tests are positive, a further sweep (made by using a unique state) is used to move the head from the left to the right end-marker and then reach the accepting configuration.

We can implement a D-F-1-LA that uses the same strategy. However, after the first sweep, all input symbols are replaced by Z. Hence, as in the proof of Theorem 3, the machine can reuse the n states of the first sweep. So, the total number of states reduces to $\max(n, p_1^{k_1} + \cdots + p_r^{k_r} + o)$. ◻

As a consequence of Theorem 9, we obtain:

Corollary 3 *For each integer* $n > 0$ *the language* $\mathscr{J}_{n,F(n)}$ *is accepted by a* D-F-1-LA *with at most* $n+1$ *states.*

By combining the upper bound in Corollary 3 with the lower bound in Theorem 8, we obtain that the superpolynomial cost of the simulation of D-F-1-LAs by 1DFAs given in Theorem 7 is asymptotically optimal and it cannot be reduced even if the resulting automaton is nondeterministic:

Corollary 4 *For each integer* $n > 0$ *there exists a language accepted by a* D-F-1-LA *with at most* $n+1$ *states and such that all equivalent* 1DFA*s and* 1NFA*s require at least* $n \cdot F(n)$ *states.*

5 Forgetting 1-Limited vs. Two-Way Automata

Up to now, we have studied the size costs of the transformations of F-1-LAs and D-F-1-LAs into one-way automata. We proved that they cannot be significantly reduced, by providing suitable witness languages. However, we can notice that such languages are accepted by two-way automata whose sizes are not so far from the sizes of F-1-LAs and D-F-1-LAs we gave. So we now analyze the size relationships between forgetting and two-way automata. On the one hand, we show that forgetting input symbols can dramatically reduce the descriptional power. Indeed, we provide a family of languages for which F-1-LAs are exponentially larger than 2DFAs. On the other hand, we guess that also in the opposite direction at least a superpolynomial gap can be possible. To this aim we present a language accepted by a D-F-1-LA of size $O(n)$ and we conjuncture that each 2NFA accepting it requires more than $F(n)$ states.

From Two-way to Forgetting 1-Limited Automata

For each integer $n > 0$, let us consider the following language

$$\mathscr{E}_n = \{w \in \{a,b\}^* \mid \exists x \in \{a,b\}^n, \exists y,z \in \{a,b\}^* : w = x \cdot y = z \cdot x^R\},$$

i.e., the set of strings in which the prefix of length n is equal to the reversal of the suffix. As we shall see, it is possible to obtain a 2DFA with $O(n)$ states accepting it. Furthermore, each equivalent F-1-LA requires 2^n states.

To achieve this result, first we give a lower bound technique for the number of states of F-1-LAs, which is inspired by the *fooling set technique* for 1NFAs [1].

Lemma 1 *Let $L \subseteq \Sigma^*$ be a language and $X = \{(x_i, y_i) \mid i = 1,\ldots,n\}$ be a set of words such that the following hold:*

- $|x_1| = |x_2| = \cdots = |x_n|$,
- *$x_i y_i \in L$, for $i = 1,\ldots,n$,*
- *$x_i y_j \notin L$ or $x_j y_i \notin L$, for $i, j = 1,\ldots,n$ with $i \neq j$.*

Then each F-1-LA*s accepting L has at least n states.*

Proof. Let M be a F-1-LAs accepting L. Let C_i be an accepting computation of M on input $x_i y_i$, $i = 1,\ldots,n$. We divide C_i into two parts C_i' and C_i'', where C_i' is the part of C_i that starts from the initial configuration and ends when the head reaches for the first time the first cell to the right of x_i, namely the cell containing the first symbol of y_i, while C_i'' is the remaining part of C_i. Let q_i be the state reached at the end of C_i', namely the state from which C_i'' starts.

If $q_i = q_j$, for some $1 \leq i, j \leq n$, then the computation obtained concatenating C_i' and C_j'' accepts the input $x_i y_j$. Indeed, at the end of C_i' and of C_j', the content of the tape to the left of the head is replaced by the same string $Z^{|x_i|} = Z^{|x_j|}$. So M, after inspecting x_i, can perform exactly the same moves as on input $x_j y_j$ after inspecting x_j and hence it can accept $x_i y_j$. In a similar way, concatenating C_j' and C_i'' we obtain an accepting computation on $x_j y_i$. If $i \neq j$, then this is a contradiction.

This allows to conclude that n different states are necessary for M. □

We are now able to prove the claimed separation.

Theorem 10 *The language $\mathscr{E}_n$ is accepted by a* 2DFA *with $O(n)$ states, while each* F-1-LA *accepting it has at least 2^n states.*

Proof. We can build a 2DFA that on input $w \in \Sigma^*$ tests the equality between the symbols in positions i and $|w| - i$ of w, for $i = 1,\ldots,n$. If one of the tests fails, then the automaton stops and rejects, otherwise it finally accepts. For each i, the test starts with the head on the left end-marker and the value of i in the finite control. Then, the head is moved to the right, while decrementing i, to locate the ith input cell and remember its content in the finite control. At this point, the head is moved back to the left end-marked, while counting input cells to restore the value of i. The input is completely crossed from left to right, by keeping this value in the control. When the right end-marker is reached, a similar procedure is applied to locate the symbol in position $|w| - i$, which is then compared with that in position i, previously stored in the control. If the two symbols are equal, then the head is moved again to the right end-marker, while restoring i. If $i = n$, then the machine moves in the accepting configuration, otherwise the value of i is incremented and the head is moved to the left end-marker to prepare the next test. From the above description we can conclude that $O(n)$ states are enough for a 2DFA to accept $\mathscr{E}_n$.

For the lower bound, we observe that the set $X = \{(x, x^R) \mid x \in \{a,b\}^n\}$, whose cardinality is 2^n, satisfies the requirements of Lemma 1. □

From Forgetting 1-limited to Two-way Automata

We wonder if there is some language showing an exponential, or at least superpolynomial, size gap from F-1-LAs to two-way automata. Here we propose, as a possible candidate, the following language, where $n, \ell > 0$ are integers:

$$\mathcal{H}_{n,\ell} = \{ub^n v \mid u \in (a+b)^* a, v \in (a+b)^*, |u|_a \bmod n = 0, \text{ and } |u| \bmod \ell = 0\}.$$

We prove that $\mathcal{H}_{n,F(n)}$ can be recognized by a D-F-1-LA with a number of states linear in n.

Theorem 11 *For each integer $n > 1$ the language $\mathcal{H}_{n,F(n)}$ is accepted by a* D-F-1-LA *with $O(n)$ states.*

Proof. A D-F-1-LA M can start to inspect the input from left to right, while counting modulo n the a's. In this way it can discover each prefix u that ends with an a and such that $|u|_a \bmod n = 0$. When such a prefix is located, M verifies whether $|u|$ is a multiple of $F(n)$ and it is followed by b^n. We will discuss how to do that below. If the result of the verification is positive, then M moves to the accepting configuration, otherwise it continues the same process.

Now we explain how the verification can be performed. Suppose $F(n) = p_1^{k_1} \cdots p_r^{k_r}$, where $p_1^{k_1}, \ldots, p_r^{k_r}$ are prime powers. First, we point out that when the verification starts, exactly the first $|u|$ tape cells have been rewritten. Hence, the rough idea is to alternate right-to-left and left-to-right sweeps on such a portion of the tape, to check the divisibility of $|u|$ by each $p_i^{k_i}$, $i = 1, \ldots, r$. A right-to-left sweep stops when the head reaches the left end-marker. On the other hand, a left-to-right sweep can end only when the head reaches the first cell to the right of the frozen segment. This forces the replacement of the symbol in it with the symbol Z, so increasing the length of the frozen segment by 1. In the next sweeps, the machine has to take into account how much the frozen segment increased. For instance, after checking divisibility by $p_1^{k_1}$ and by $p_2^{k_2}$, in the next sweep the machine should verify that the length of the frozen segment, modulo $p_3^{k_3}$, is 1. Because the machine has to check r divisors and right-to-left sweeps alternate with left-to-right sweeps, when all r sweeps are done, exactly $\lfloor r/2 \rfloor$ extra cells to the right of the original input prefix u are frozen. Since $n > r/2$, if the original symbol in all those cells was b, to complete the verification phase the machine has to check whether the next $n - \lfloor r/2 \rfloor$ not yet visited cells contain b. However, the verification fails if a cell containing an a or the right end-marker is reached during some point of the verification phase. This can happen either while checking the length of the frozen segment or while checking the last $n - \lfloor r/2 \rfloor$ cells. If the right end-marker is reached, then the machine rejects. Otherwise it returns to the main procedure, i.e., resumes the counting of the a's.

The machine uses a counter modulo n for the a's. In the verification phase this counter keeps the value 0. The device first has to count the length of the frozen part modulo $p_i^{k_i}$, iteratively for $i = 1, \ldots, r$, and to verify that the inspected prefix is followed by b^n, using again a counter. Since $p_1^{k_1} + \cdots + p_r^{k_r} \leq n$, by summing up we conclude that the total number of states is $O(n)$. ∎

By using a modification of the argument in the proof of Theorem 8, we can show that each 1NFA accepting $\mathcal{H}_{n,F(n)}$ cannot have less than $n \cdot F(n)$ states.[2] We guess that such a number cannot be substantially reduced even having the possibility of moving the head in both directions. In fact, a two-way automaton using $O(n)$ states can easily locate on the input tape a "candidate" prefix u. However, it cannot remember in which position of the tape u ends, in order to check $|u|$ in several sweeps of u. So we do not see how the machine could verify whether $|u|$ is a multiple of $F(n)$ using less than $F(n)$ states.

[2] It is enough to consider the set $X' = \{(x_{ij}, y_{ij} b^n) \mid 1 \leq i \leq \ell, 0 \leq j < n\}$, instead of X.

6 Conclusion

We compared the size of forgetting 1-limited automata with that of finite automata, proving exponential and superpolynomial gaps. We did not discuss the size relationships with 1-LAs. However, since 2DFAs are D-1-LAs that never write, as a corollary of Theorem 10 we get an exponential size gap from D-1-LAs to F-1-LAs. Indeed, the fact of having a unique symbol to rewrite the tape content dramatically reduces the descriptional power.

We point out that this reduction happens also in the case of F-1-LAs accepting languages defined over a one-letter alphabet, namely unary languages. To this aim, for each integer $n > 0$, let us consider the language $(a^{2^n})^*$. This language can be accepted with a D-1-LA having $O(n)$ states and a work alphabet of cardinality $O(n)$, and with a D-1-LA having $O(n^3)$ states and a work alphabet of size not dependent on n [16, 18]. However, each 2NFA accepting it requires at least 2^n states [16]. Considering the cost of the conversion of F-1-LAs into 1NFAs (Theorem 1), we can conclude that such a language cannot be accepted by any F-1-LA having a number of states polynomial in n.

References

[1] Jean-Camille Birget (1992): *Intersection and Union of Regular Languages and State Complexity*. Inf. Process. Lett. 43(4), pp. 185–190, doi:10.1016/0020-0190(92)90198-5.

[2] Marek Chrobak (1986): *Finite Automata and Unary Languages*. Theor. Comput. Sci. 47(3), pp. 149–158. Available at http://dx.doi.org/10.1016/0304-3975(86)90142-8. Errata: [3].

[3] Marek Chrobak (2003): *Errata to: Finite automata and unary languages: [Theoret. Comput. Sci. 47 (1986) 149-158]*. Theor. Comput. Sci. 302(1-3), pp. 497 – 498, doi:10.1016/S0304-3975(03)00136-1.

[4] Viliam Geffert (2007): *Magic numbers in the state hierarchy of finite automata*. Inf. Comput. 205(11), pp. 1652–1670. Available at http://dx.doi.org/10.1016/j.ic.2007.07.001.

[5] Viliam Geffert & Giovanni Pighizzini (2012): *Pairs of Complementary Unary Languages with "Balanced" Nondeterministic Automata*. Algorithmica 63(3), pp. 571–587, doi:10.1007/s00453-010-9479-9.

[6] Thomas N. Hibbard (1967): *A Generalization of Context-Free Determinism*. Inf. Control. 11(1/2), pp. 196–238, doi:10.1016/S0019-9958(67)90513-X.

[7] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

[8] Petr Jancar, Frantisek Mráz & Martin Plátek (1993): *A Taxonomy of Forgetting Automata*. In: MFCS'93, Lecture Notes in Computer Science 711, Springer, pp. 527–536, doi:10.1007/3-540-57182-5_44.

[9] Edmund Landau (1903): *Über die Maximalordnung der Permutation gegebenen Grades*. Archiv der Mathematik und Physik 3, pp. 92–103.

[10] Edmund Landau (1909): *Handbuch der Lehre von der Verteilung der Primzahlen I*. Teubner, Leipzig/Berlin.

[11] Carlo Mereghetti & Giovanni Pighizzini (2001): *Optimal Simulations between Unary Automata*. SIAM J. Comput. 30(6), pp. 1976–1992, doi:10.1137/S009753979935431X.

[12] Albert R. Meyer & Michael J. Fischer (1971): *SWAT 1971*. IEEE Computer Society, pp. 188–191, doi:10.1109/SWAT.1971.11.

[13] Giovanni Pighizzini (2019): *Limited Automata: Properties, Complexity and Variants*. In: DCFS 2019, Lecture Notes in Computer Science 11612, Springer, pp. 57–73, doi:10.1007/978-3-030-23247-4_4.

[14] Giovanni Pighizzini & Andrea Pisoni (2014): *Limited Automata and Regular Languages*. Int. J. Found. Comput. Sci. 25(7), pp. 897–916, doi:10.1142/S0129054114400140.

[15] Giovanni Pighizzini & Andrea Pisoni (2015): *Limited Automata and Context-Free Languages*. Fundam. Inform. 136(1-2), pp. 157–176, doi:10.3233/FI-2015-1148.

[16] Giovanni Pighizzini & Luca Prigioniero (2019): *Limited automata and unary languages*. *Inf. Comput.* 266, pp. 60–74, doi:10.1016/j.ic.2019.01.002.

[17] Giovanni Pighizzini & Luca Prigioniero (2023): *Once-Marking and Always-Marking* 1*-Limited Automata*. In: *AFL 2023*, *Electronic Proceedings in Theoretical Computer Science* 386, pp. 215–227, doi:10.4204/EPTCS.386.17.

[18] Giovanni Pighizzini & Luca Prigioniero (2023): *Two-way Machines and de Bruijn Words*. In: *CIAA 2023*, *Lecture Notes in Computer Science* 14151, pp. 254–65, doi:10.1007/978-3-031-40247-0_19.

[19] Giovanni Pighizzini, Luca Prigioniero & Simon Šádovský (2022): *1-Limited Automata: Witness Languages and Techniques*. *J. Autom. Lang. Comb.* 27(1-3), pp. 229–244, doi:10.25596/jalc-2022-229.

[20] Jeffrey O. Shallit (2008): *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, doi:10.1017/CBO9780511808876.

[21] John C. Shepherdson (1959): *The Reduction of Two-Way Automata to One-Way Automata*. *IBM J. Res. Dev.* 3(2), pp. 198–200, doi:10.1147/rd.32.0198.

[22] Michael Sipser (1980): *Lower Bounds on the Size of Sweeping Automata*. *J. Comput. Syst. Sci.* 21(2), pp. 195–202, doi:10.1016/0022-0000(80)90034-3.

[23] Mihály Szalay (1980): *On the maximal order in S_n and S_n^**. *Acta Arithmetica* 37, pp. 321–331, doi:10.4064/aa-37-1-321-331.

[24] Klaus W. Wagner & Gerd Wechsung (1986): *Computational complexity*. D. Reidel Publishing Company, Dordrecht.