





Linear-Time Limited Automata

Bruno Guillon^(✉)  and Luca Prigioniero 

Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy
{guillonb,prigioniero}@di.unimi.it

Abstract. The time complexity of 1-limited automata is investigated from a descriptional complexity view point. Though the model recognizes regular languages only, it may use quadratic time in the input length. We show that, with a polynomial increase in size and preserving determinism, each 1-limited automaton can be transformed into an halting linear-time equivalent one. We also obtain polynomial transformations into related models, including weight-reducing Hennie machines, and we show exponential gaps for converse transformations in the deterministic case.

1 Introduction

One classical topic of computer science is the investigation of computational models operating under restrictions. Finite automata or pushdown automata, for instance, can be considered as particular Turing machines in which the access to memory storage is limited. Other kinds of restrictions follow from finer analysis of the computational resources an abstract device requires to recognize certain languages. For example, in the case of Turing machines, classical complexity classes such as P, NP, LOGSPACE, *etc.* are defined by introducing a limit on the amount of resources, namely time or space, at disposal of the model.

Usually, such limitations reduce the expressive power. For instance, it is well-known that one-tape nondeterministic Turing machines operating within a space bounded by the length of the input, namely *linear bounded automata*, capture exactly the class of *context-sensitive languages*, *e.g.* [5]. Phenomena like this, where limiting an abstract model reduces its expressiveness to the level of some standard class, are of great interest, as they provide alternative characterizations of standard classes. Another example of this kind has been observed by Hennie in 1965. He indeed proved that deterministic one-tape Turing machines operating in *linear time* (*i.e.*, time $O(m)$ over inputs of length m), which can be converted into linear bounded automata operating in linear time, recognize exactly the class of *regular languages* [3]. The result has then been extended to the nondeterministic case [15], see also [8] for further improvements.¹ As a consequence, each *Hennie machine*, namely nondeterministic linear bounded automata working in linear time, is equivalent to some finite automaton. From the opposite point of view, this means that providing *two-way finite automata*

¹ In nondeterministic linear-time devices each accepting computation has linear length.

with the ability to overwrite the tape cells does not extend the expressiveness of the model, as long as the time is linearly bounded in the length of the input.

Unfortunately, Hennie proved that it is undecidable, given a deterministic one-tape Turing machine, to check whether it works in linear time over all input strings, namely, whether it is actually a Hennie machine. To avoid this drawback, Průša proposed a variant of Hennie machine, called *weight-reducing Hennie machine*, in which the time limitation is syntactic [12]. In this model, each visit of a cell should overwrite its content with a symbol in a decreasing way, with respect to some fixed order on the working alphabet. As a consequence, the number of visits of a cell by the head is bounded by some constant (*i.e.*, not depending on the input length) whence the device works in linear time over every input string.

By contrast to Hennie machines, the *d-scan limited automata* (or simply *d-limited automata*) introduced by Hibbard, restrict nondeterministic linear bounded automata by allowing overwriting of each tape cell during its first d visits only, for some fixed $d \geq 0$ [4]. Contrary to weight-reducing Hennie machines, the head is still allowed to visit a cell after the d -th visit, but cannot rewrite its content anymore. This allows to use super-linear time. Hence, *limited automata* (namely, *d-limited automata* for some d) live midway between linear-space Turing machines and weight-reducing Hennie machines. For each $d \geq 2$, Hibbard proved that *d-limited automata* recognize exactly the class of *context-free languages*. He furthermore showed the existence of an infinite hierarchy of deterministic *d-limited automata*, whose first level (*i.e.*, corresponding to deterministic 2-limited automata) has been later proved to coincide with the class of *deterministic context-free languages* [10]. (See [7] and references therein for further connections between limited automata and context-free languages.)

Clearly, 0-limited automata are no more than two-way finite automata. Hence, they characterize the class of regular languages. Wagner and Wechsung extended this result to the case $d = 1$: 1-limited automata recognize exactly the class of regular languages [16]. From that point, the question of the cost of their simulation by classical finite automata has been studied by Pighizzini and Pisoni in [9], where a tight doubly-exponential simulation by deterministic one-way finite automata is proved. This cost reduces to a single exponential when starting from a deterministic 1-limited automaton. Also, an exponential lower bound, using a single-letter input alphabet, has been obtained in [11], for the simulation of deterministic 1-limited automata by nondeterministic two-way finite automata.

Like *d-limited automata*, 1-limited automata can operate in super-linear time (*cf.* Example 1). This contrasts with Hennie machines which operate in linear time by definition. The question we address in this paper is whether this ability of 1-limited automata with respect to Hennie machines yields a gap between the two models in terms of the size of their representations.

We show that, with a polynomial increase in size, each 1-limited automaton can be transformed into an halting linear-time 1-limited automaton, or alternatively, into a weight-reducing Hennie machine, while preserving determinism.

We also observe that the 1-limited automaton resulting from this construction can be easily transformed into an equivalent one whose behavior can be divided into two phases: (1) an initial phase consisting in a left-to-right one-way traversal of the input, during which each input symbol is nondeterministically overwritten; (2) a second phase consisting in a read-only two-way computation. Similar behaviors have been considered in the context of *regular transductions* (i.e., transductions computed by, for instance, two-way transducers), because of their correspondence with global existential quantification in *monadic second order logic*, see, e.g. [1]. Using terminology from [1], we define the model of *two-way automaton with common guess* in order to capture these particular behaviors of 1-limited automata. Formally, such machines are not 1-limited automata, but the composition of an initial *common guess* (i.e., a nondeterministic marking of the input symbols using symbols from a finite alphabet, computed, for instance, by a 1-state *letter-to-letter one-way transducer*) with a two-way automaton working on the enriched alphabet. Reformulating the above-mentioned result, each 1-limited automaton can be simulated by a two-way automaton with common guess of polynomial size. Furthermore, the underlying two-way finite automaton in the resulting device is deterministic, when starting from a deterministic 1-limited automaton. (Be aware that a deterministic two-way automaton with common guess, is not a deterministic device, since it initially performs a common guess which is nondeterministic by definition.) A direct consequence of this last result, is that *reversing* a 1-limited automaton, i.e., transforming it into another one recognizing the reverse of its accepted language, has polynomial cost. This fails in the deterministic case, for which we exhibit an exponential lower bound. As a consequence, we obtain exponential lower bounds for the simulation of

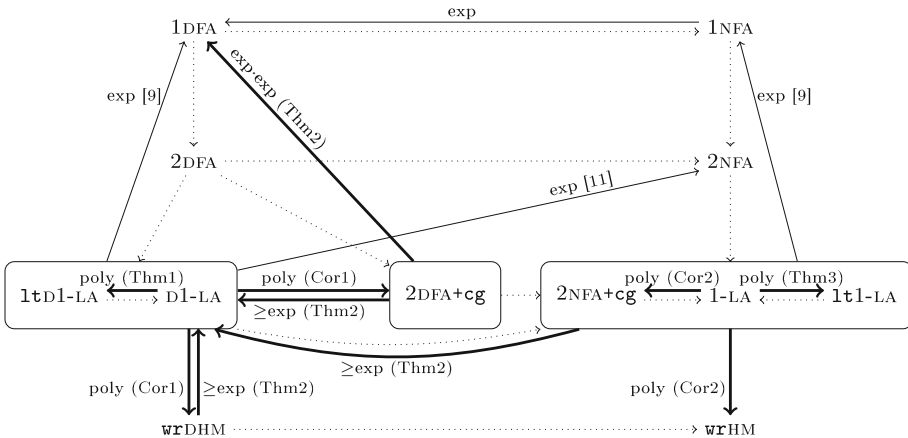


Fig. 1. Relationships between the main models studied in the paper. Here, **1t** and **wr** mean linear-time and weight-reducing, while **D1-LA** and **(D)HM** stand for deterministic 1-LA and (deterministic) Hennie machine, respectively. Deterministic and nondeterministic two-way automata with common guess are denoted by **2DFA+cg** and **2NFA+cg**. Dotted arrows indicate trivial connections while thick arrows indicate our results.

deterministic weight-reducing Hennie machines or deterministic two-way automata with common guess by deterministic 1-limited automata. The results are summarized in Fig. 1.

The paper is organized as follows. In Sect. 2 are gathered the main definitions and notations needed in the subsequent sections. The main ideas of the construction used for proving our results are detailed in Sect. 3, when considering the deterministic case. The results obtained therein are then extended to the nondeterministic case in Sect. 4.

2 Preliminaries

In this section we recall some basic definitions and notations useful in the paper. In particular, we assume the reader familiar with notions from formal languages and automata theory (see, *e.g.*, [5]). Given a set S , $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ , we denote by $|w|$ the length of a string $w \in \Sigma^*$, by w^R the reversal of w and by ϵ the empty string. For a language $L \subseteq \Sigma^*$, L^R denotes the *reversal* of L , namely $L^R = \{w^R \mid w \in L\}$.

A *two-way nondeterministic finite automaton* (2NFA) is defined as a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite *set of states*, Σ is a finite *input alphabet*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is a *set of final states*, and $\delta : Q \times \Sigma_{\triangleright\triangleleft} \rightarrow 2^{Q \times \{-1, 0, +1\}}$ is a *nondeterministic transition function* where $\Sigma_{\triangleright\triangleleft}$ denotes the set $\Sigma \cup \{\triangleright, \triangleleft\}$ with the two special symbols $\triangleright, \triangleleft \notin \Sigma$ respectively called the *left* and the *right endmarkers*. The input is written on the tape surrounded by the two endmarkers, the left endmarker being at the position zero. Hence, on input w , the right endmarker is at position $|w| + 1$. In one move, \mathcal{A} reads an input symbol, changes its state, and moves the input head one position backward, forward or keeps it in position depending on whether δ returns -1 , $+1$ or 0 , respectively. Furthermore, the head cannot violate the endmarkers, except at the end of computation, to accept the input, as now explained. The machine *accepts* the input, if there exists a computation path starting from the initial state q_0 with the head on the first tape cell (*i.e.*, scanning the left endmarker) and ending in a final state $q \in F$ after violating the right endmarker. The language accepted by \mathcal{A} is denoted by $L(\mathcal{A})$. A 2NFA \mathcal{A} is said to be *deterministic* (2DFA), whenever $\#\delta(q, \sigma) \leq 1$, for any $q \in Q$ and $\sigma \in \Sigma_{\triangleright\triangleleft}$. It is called *one-way* if its head can never move backward, *i.e.*, if no transition returns -1 . By 1NFAs and 1DFAs we denote one-way nondeterministic and deterministic finite automata, respectively.

A *1-limited automaton* (1-LA) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q , Σ , q_0 and F are defined as for 2NFAs, Γ is a finite *working alphabet* such that $\Sigma \subset \Gamma$, $\delta : Q \times \Gamma_{\triangleright\triangleleft} \rightarrow 2^{Q \times \Gamma_{\triangleright\triangleleft} \times \{-1, 0, +1\}}$ is the *nondeterministic transition function* where $\Gamma_{\triangleright\triangleleft}$ denotes the set $\Gamma \cup \{\triangleright, \triangleleft\}$ with $\triangleright, \triangleleft \notin \Gamma$ the left and the right endmarkers as for 2NFAs. In one move, according to δ , \mathcal{A} reads a symbol from the tape, changes its state, replaces the symbol just read by a new symbol, and moves its head one position backward or forward or keeps it in place. However, replacing symbols is subject to some restrictions, which,

essentially, allow to modify the content of a cell during the first visit only. Technically, symbols from Σ shall be replaced by symbols from $\Gamma \setminus \Sigma$, while symbols from $\Gamma_{\triangleright\triangleleft} \setminus \Sigma$ are never overwritten. In particular, at any time, both special symbols \triangleright and \triangleleft occur exactly once on the tape and exactly at the respective left and right boundaries. *Acceptance* for 1-LAS as well as *deterministic 1-LAS* are defined exactly as for 2NFAS, and the language accepted by a given 1-LA \mathcal{A} is denoted by $L(\mathcal{A})$.

Some 1-LAS have a particular behavior, which can be decomposed into two phases. In the first phase, they nondeterministically rewrite the content of the whole tape during a left-to-right traversal of the input. Then, in the second phase, they perform a two-way read-only computation over the overwritten tape. To formally define this kind of 1-LAS, we introduce the following model. A *2NFA (resp. 2DFA) with common guess* (*2NFA+cg, resp. 2DFA+cg*)² is a tuple $\langle \mathcal{A}, \Sigma, \Delta \rangle$ where Σ and Δ are two alphabets and \mathcal{A} is a 2NFA (resp. 2DFA) over the product alphabet $\Sigma \times \Delta$. The model is aimed to recognize languages from Σ^* . Its dynamics is defined as for two-way automata, but a nondeterministic pre-computation initially marks each input symbol with a symbol from Δ . Hence, the read-only automaton \mathcal{A} has access to both the input symbol and the guessed additional information. The language accepted, denoted $L(\langle \mathcal{A}, \Sigma, \Delta \rangle)$, is defined as the projection, denoted π_1 , of $L(\mathcal{A})$ to the alphabet Σ , i.e., $L(\langle \mathcal{A}, \Sigma, \Delta \rangle) = \pi_1(L(\mathcal{A}))$. In other words, a word is accepted by $\langle \mathcal{A}, \Sigma, \Delta \rangle$ if for some guess, the enriched word is accepted by \mathcal{A} . We point out that, due to the common guess, 2DFA+cg's are nondeterministic devices.

For each of the above-defined models, a *configuration* is represented as a string $z \cdot p \cdot z'$, meaning that p is the current state, $zz' \in \triangleright \Pi^* \triangleleft$ is the content of the tape (here Π denotes the alphabet Σ , Γ , or $\Sigma \times \Delta$ depending on the model under consideration) and the head is scanning the first symbol of z' . The transition relation between configurations is denoted by \vdash , and its reflexive-transitive closure by \vdash^* . Notice that, in case $|z'| = 0$, the machine has reached the end of the computation. We also represent *partial configurations* as $u \cdot p \cdot v$, where p is the current state and $uv \in \{\epsilon, \triangleright\} \Pi^* \{\epsilon, \triangleleft\}$ is a factor of the tape content. The relations \vdash and \vdash^* naturally extend onto partial configurations.

For each model under consideration, we evaluate its size as the total number of symbols used to define it. Hence, the *size* of n -state 2NFAS, 1-LAS or 2NFA+cg's are given by some polynomial in the parameters n , $\#\Sigma$, and possibly $\#\Gamma$ or $\#\Delta$.

Example 1. We consider the language

$$L_n = \{x_0 x_1 \cdots x_k \mid k \in \mathbb{N}, x_i \in \{a, b\}^n, \#\{i > 0 \mid x_i = x_0\} \text{ is odd}\}.$$

A deterministic 1-LA \mathcal{A}_n may recognize L_n as follows. It first overwrites the factor x_0 , replacing each input symbol with a marked copy. Then, \mathcal{A}_n repeats a subroutine which overwrites a factor x_i with some fixed symbol \sharp , while checking in the meantime whether x_i equals x_0 or not. This can be achieved as follows. Before overwriting the j -th symbol of x_i , first, \mathcal{A}_n , with the help of a

² 2DFA+cg's also correspond to *synchronous two-way deterministic finite verifiers* [6].

counter modulo n , moves the head leftward to the position j of x_0 and stores the unmarked scanned symbol σ in its finite control; second, it moves the head rightward until reaching the position j of x_i , namely, the leftmost position that has not been overwritten so far. At this point, \mathcal{A}_n compares the scanned symbol (*i.e.*, the j -th symbol of x_i) with σ (*i.e.*, the j -th symbol of x_0). By counting modulo 2 the number of factors equal to x_0 , and finally checking that the input string has length multiple of n , \mathcal{A}_n can decide the membership of the input to L_n .

It is possible to implement \mathcal{A}_n with a number of states linear in n and $\#\Sigma + 1$ working symbols. Since for each position of a factor x_i , $i > 0$, the head has to move back to the factor x_0 , we observe that \mathcal{A}_n works in quadratic time in the length of the input string.

3 A Linear-Time Simulation for Deterministic 1-LAs

If a linear-space Turing machine can visit a tape cell only a constant number of times, it necessarily works in linear time. Conversely, Turing machines working in linear time (*i.e.*, Hennie machines), have been shown to visit each tape cell only a constant number of times during a computation [3]. This contrasts with the case of 1-LAs, which can use quadratic time, as shown in Example 1. However, our main contribution states that, with a polynomial increase in size of the model, we can recover the above property, and therefore obtain equivalent 1-LAs working in linear time.

Theorem 1. *For each deterministic 1-LA \mathcal{A} , there exists an equivalent deterministic 1-LA \mathcal{A}' satisfying:*

1. \mathcal{A}' has polynomial size with respect to \mathcal{A} ;
2. in every computation of \mathcal{A}' , each tape cell is visited a number of times which is bounded by some polynomial in the size of \mathcal{A} ;
3. \mathcal{A}' works in linear time: on every input string w , it halts within $O(|w|)$ steps.

Proof. Clearly, Item 3 is implied by Item 2. It remains to prove that Item 2 can be achieved, while keeping a polynomial size of the device, namely Item 1. The key idea is to ensure that, in any computation, the simulating device works in a “virtual” window of fixed size, that is shifted along the configuration in a one-way manner. More precisely, in every computation and for each cell c , there exists a time t after which c is not visited anymore and furthermore, at this time, the rightmost cell that has been overwritten is at bounded distance to the right of c . This requires to detect local loops, that, thanks to this local window boundaries on space, can be done by using a polynomial number of states.

In [9], the authors presented a construction to simulate any deterministic 1-LA \mathcal{A} by a 1DFA \mathcal{A}'' , using classic ideas from the simulation of 2DFAs by 1DFAs [13]. The main ingredient is to store in the finite control of \mathcal{A}'' , a “transition table” describing the possible behaviors of \mathcal{A} that may occur to the left of the current head position. Since the part of the tape to the left of the current head position has necessarily already been visited, its “frozen” content

belongs to $\triangleright(\Gamma \setminus \Sigma)^*$. Hence, the above-mentioned behaviors to the left of the current head position, are read-only computations. To represent them, for each word $z'X \in \triangleright(\Gamma \setminus \Sigma)^*$ with $|X| = 1$, a function $\tau_{z'X} : Q \rightarrow Q$, where Q denotes the set of states of \mathcal{A} , is considered. The function maps a state p to a state q , if, starting from the state p with the head scanning the last symbol of $z'X$, \mathcal{A} eventually reaches the state q one cell to the right of $z'X$. Formally,

$$\tau_{z'X}(p) = q \quad \text{if} \quad z' \cdot p \cdot X \vdash^* z'X \cdot q.$$

By determinism of \mathcal{A} , $\tau_{z'X}$ is a function which is partial in general.³ We complete it by setting all undefined images to some fixed symbol $\perp \notin Q$. With the information of $\tau_{z'X}$, \mathcal{A}'' has no need to read the part of the tape containing $z'X$, that is, to move its head leftward. Moreover, as acceptance is done by violating the right endmarker, if the configuration $z' \cdot p \cdot X$ occurs in some accepting computation, then $\tau_{z'X}(p) \in Q$. Hence, \mathcal{A}'' cannot miss accepting computations. Finally, given a string $x \in (\Gamma \setminus \Sigma)^*$, we can construct $\tau_{z'Xx}$ from $\tau_{z'X}$ by scanning x .

In [9], the table of size $(n+1)^n$ corresponding to the function $\tau_{z'X}$ was stored in the finite control of the simulating 1DFA and it was updated at each step. This yielded an exponential number of states for the simulating device (that was shown to be necessary for the considered simulation). Here, as our simulating device \mathcal{A}' is a deterministic 1-LA, we take advantage of its ability to write on the tape, and we store the table onto the n cells following the last position of $z'X$. Thus, the i -th position to the right of the tape part containing $z'X$ will contain the image of the i -th state of \mathcal{A} by $\tau_{z'X}$, as part of its written symbol. However, updating the table when moving to the right is done block by block rather than cell by cell, for a decomposition of the input into blocks of length $\#Q$. We consider the cell containing the left endmarker as a whole block, while the last block containing the right endmarker may be shorter than $\#Q$.

We now gather the two ideas presented above. Let $Q = \{q_0, q_1, \dots, q_{n-1}\}$ be the set of states of \mathcal{A} , $\perp \notin Q$ be a fixed symbol and Q_\perp denote the set $Q \cup \{\perp\}$. At any time in a computation of \mathcal{A} we consider a “virtual window” of size $2n$ which covers two successive blocks of length $n = \#Q$. The right block covered by the window contains the leftmost cell that has not been visited so far, to which we refer as *current frontier*. A typical situation is depicted in Fig. 2.

In order to simulate \mathcal{A} , the linear-time 1-LA overwrites each block with a word $\tilde{x} \in ((\Gamma \setminus \Sigma) \times Q_\perp)^n$ whose projection to $(\Gamma \setminus \Sigma)$ is the word x written by \mathcal{A} on the corresponding block, and the projection to Q_\perp is exactly the table τ_z , where z is the content of the tape to the left of the block. (In Fig. 2, $z = \triangleright w$ when considering the left block covered by the window, whose “frozen” content is x .) Roughly, when the window covers such a block as left part, \mathcal{A}' has to fill

³ $\tau_{z'X}(p)$ is undefined if one of the two following cases of the computation starting in $z' \cdot p \cdot X$ occurs: either, after a finite number of steps, no successive transition is defined (incompleteness of \mathcal{A}), or the computation eventually enters a deterministic loop (non-haltingness of \mathcal{A}).

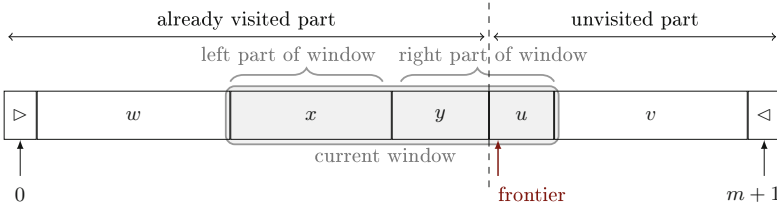


Fig. 2. Typical description of the window during a computation of \mathcal{A} : m denotes the length of the input word, the current frontier occurs in the right block as first position of u , $w \in ((\Gamma \setminus \Sigma)^n)^*$, $x \in (\Gamma \setminus \Sigma)^n$, $y \in (\Gamma \setminus \Sigma)^*$, $u \in \Sigma^+$ with $|yu| = n$, and $v \in \Sigma^*$.

the next block, cell by cell, with τ_{zx} . To this end, it has read-only access to the left block, containing all the required information, namely τ_z and x . In parallel, \mathcal{A}' should also recover the simulated computation of \mathcal{A} . As soon as the right block is completely filled, the window is shifted to the right, in such a way that it covers the block just treated (as left part) and its successor (as right part).

We now describe the formal implementation of the above-explained procedure. By using a state component of size $2n$, named *relative position*, \mathcal{A}' can store the exact position of its head relative to the current window. We represent it as a pair (i, s) , where $i \in \{0, \dots, n - 1\}$ is the position in the scanned block of length n and $s \in \{L, R\}$ is equal to L (resp. R) if the head is scanning a position in the left (resp. right) block of the window. We suppose that the component is updated at each head move. Using this component, \mathcal{A}' can avoid moving to the left of the current window. More precisely, from a relative position $(0, L)$ (i.e., the leftmost position covered by the window), in order to simulate a backward move of \mathcal{A} from p to q , \mathcal{A}' enters a special mode to determine the state $\tau_z(q)$ (if it exists), where z is the content of the tape to the left of the window. Hence, it simulates not only the backward step from p to q , but also the complete computation segment to the left of the window from state q , namely, it simulates $z'X \cdot p \vdash z' \cdot q \cdot X \vdash^* z'X \cdot \tau_z(q)$, where $z = z'X$. This special mode, called *readFromTable*, which starts and ends in relative position $(0, L)$, consists in a simple read of the table τ_z that has been written on the left block covered by the current window (in the factor \tilde{x} corresponding to the factor x in Fig. 2).

In addition to the relative position, \mathcal{A}' stores in its finite control the relative position of the current frontier, to which we refer as *relative frontier*. Since this position always occurs in the right block of the window, it is enough to represent it as an index $\rho \in \{0, \dots, n - 1\}$. Much like the relative position component, we suppose that it is updated each time a new cell is visited. Observe that such updates are increments modulo n . As explained below, incrementing $\rho = n - 1$ means shifting the window n cells to the right. Using both ρ and (i, s) , \mathcal{A}' can ensure that entering a cell for the first time, may be done only once all necessary information (that is required to determine the symbol to write on the cell at its first visit) has been gathered. This information corresponds to a pair $(q, \tau_{zx}(q_\rho))$, where q is the state entered by \mathcal{A} when visiting for the first time the frontier

cell in the simulated computation, and zx is the content of the tape to the left of the right block of the current window.

We now describe the subroutine `simulateLeft` that is used for recovering this information. The procedure takes a state p as argument and starts from and ends in some relative position (i, s) which indicates either one cell to the left of the right block, or one cell to the left of the frontier. Notice that the two positions coincide when $\rho = 0$. Hence, (i, s) belongs to $\{(n - 1, L), (\rho - 1, R)\}$. Denoting $\gamma \in (\Gamma \setminus \Sigma)$ the symbol contained in this cell, the procedure returns $q \in Q_\perp$ such that $\tau_{zx'\gamma}(p) = q$, where zx' is the content of the tape to the left of the relative position (i, s) . (In Fig. 2, $z = \triangleright w$ and $x'\gamma$ is equal to x or to xy , respectively if (i, s) is equal to $(n - 1, L)$ or to $(\rho - 1, R)$.) During the computation, `simulateLeft` has access to the content of the window up to position (i, s) . It basically performs a direct simulation of \mathcal{A} on the corresponding part of the tape, which uses the procedure `readFromTable` in order to simulate (in one step) computations that occur to the left of the window, as explained above. Moreover, if $zx' \cdot p \cdot \gamma \vdash^* zx'\gamma \cdot q$ in \mathcal{A} , namely if $q = \tau_{zx'\gamma}(p) \neq \perp$, then it halts in the relative position (i, s) , before the last step of the simulated computation. At this point, it is possible to determine the return value q . The direct simulation might fail for two reasons: (1) the simulation halts on some previous position (incompleteness of \mathcal{A}); (2) the simulation enters a deterministic loop and never exits (non-haltingness of \mathcal{A}). For the first case, it is sufficient to enter a sub-mode which moves the head to the right until reaching the relative position (i, s) and returns \perp . The second case requires to detect loops. Since the simulating computation takes place in a read-only window of size at most $2n$, any loop-free computation has length bounded by some polynomial of degree 2. Hence, by using a clock of size $O(n^2)$, we can enforce the procedure to halt. Only runs that halted before this time limit may return a state value, while “killed” runs will return \perp .⁴

Hence, before visiting for the first time the cell at relative position (ρ, R) :

first, \mathcal{A}' calls `simulateLeft`(p) from the position $(\rho - 1, R)$, and saves its return value q in its finite control, where p is the state that was entered by \mathcal{A} when visiting for the first time the cell at relative position $(\rho - 1, R)$ in the simulated computation;
then, \mathcal{A}' calls `simulateLeft`(q_ρ) from the position $(n - 1, L)$, and stores the returned value r in its finite control.

Once \mathcal{A}' has gathered the pair of states $(q, \tau_{zx}(q_\rho)) = (q, r)$, it moves to the cell at relative position (ρ, R) , and reads the input symbol $\sigma \in \Sigma \cup \{\triangleleft\}$. If $\sigma = \triangleleft$, \mathcal{A}' calls `simulateLeft`(q) and accepts, after violating the endmarker, if the return value is a final state of \mathcal{A} (it rejects otherwise). If $\sigma \neq \triangleleft$, \mathcal{A}' writes the symbol (r, γ) where $\gamma \in (\Gamma \setminus \Sigma)$ is the symbol returned by $\delta(q, \sigma)$, and repeats

⁴ We could do a finer construction, based on Sipser’s backward construction [14], which has linear cost (without counting the relative position and relative frontier components) instead of the expensive $O(n^3)$ cost of the clocked simulation presented here. For an adaptation to finite automata, see [2].

the procedure with the updated relative frontier. In the case $\rho = n - 1$, the window is shifted to the right, in such a way that the head is positioned on the rightmost cell of its left block. This is formally done by setting the relative position to $(n - 1, L)$ and the relative frontier to $(0, R)$.

Let us describe the initial configuration of \mathcal{A}' . At the beginning of the computation, the head is scanning the left endmarker, which is considered as the left block of the current window. Hence, the initial relative position and relative frontier are set to $(n - 1, L)$ and $(0, R)$, respectively. Since the head of \mathcal{A} cannot move backward from the left endmarker, the procedure `simulateLeft` never calls the subroutine `readFromTable`, as long as the window is in its initial place.

We have shown how \mathcal{A}' simulates \mathcal{A} in an halting manner, by shifting a virtual window to the right along its computation, and by restricting local head moves to the current window. Moreover, \mathcal{A}' only uses a polynomial number of states in n , and working alphabet $(\Gamma \setminus \Sigma) \times Q_{\perp} \cup \Sigma$. We now evaluate the time used by \mathcal{A}' . Let fix a cell c . As \mathcal{A}' is loop-free, each time the head visits c it must have a different state or a different tape content. A tape modification between two visits of c is restricted to cells from the right block of the current window containing c . The number of successive tape modifications in a window is linear in n (after n overwritings, the window is shifted), and c may occur in two successive windows. Thus, the number of visits to the cell c is bounded by some polynomial in n . The number of visits to each cell is hence bounded by a polynomial in n . As a consequence, \mathcal{A}' operates in linear time with respect to the input length. \square

Linear-time 1-LAS are particular cases of Hennie machines, hence, it follows from the above result that any 1-LA can be transformed into a Hennie machine of size polynomial in the size of the 1-LA. Using Item 2 we can actually obtain the stronger result that the obtained 1-LA can be transformed into a *weight-reducing Hennie machine*. Informally, weight-reducing Hennie machines are Hennie machines in which each overwriting is decreasing with respect to some fixed order on the working alphabet. As a consequence, after overwriting a cell with a minimal symbol, such a machine cannot visit the cell again. See [12] for formal definition and study of the model. It is also possible to modify the construction of Theorem 1 in order to obtain an equivalent 2DFA+cg.

Corollary 1. *For each deterministic 1-LA, there exists an equivalent deterministic weight-reducing Hennie machine or halting 2DFA+cg of size polynomial in the size of the 1-LA.*

Concerning the converse simulation, using the language L_n from Example 1, we can prove an exponential gap in the deterministic case.

Theorem 2. *Let L_n be the language of Example 1. Hence*

$$L_n^R = \{x_k x_{k-1} \cdots x_0 \mid k > 0, x_i \in \{a, b\}^n, \#\{i > 0 \mid x_i = x_0\} \text{ is odd}\}.$$

Then,

1. L_n^R is accepted by a 2DFA+cg, a linear-time nondeterministic 1-LA, or a deterministic weight-reducing Hennie machine of size polynomial in n ;
2. any 1DFA recognizing L_n^R requires 2^{2^n} states;
3. any deterministic 1-LA recognizing L_n^R requires $O(2^n)$ states.

Proof (outline). Example 1 describes a deterministic 1-LA recognizing L_n , whose size is linear in n . By applying Corollary 1, we respectively obtain equivalent weight-reducing Hennie machine and 2DFA+cg of polynomial size. Both models can be transformed with a constant increase in size, in order to accept the reverse of the language, thus proving Item 1. Using a distinguishability argument, we can prove Item 2. Finally, Item 3 can be deduced from this previous point and the exponential blowup from deterministic 1-LA to 1DFA given in [9]. \square

4 A Linear-Time Simulation for Nondeterministic 1-LAS

The techniques presented in the proof of Theorem 1 can be used for the nondeterministic case. However, this time, the table τ_z does no longer represent a partial function, but a binary relation on Q , whose size is thus 2^{n^2} . To handle this size increase, we store the table on the n^2 cells following the last cell of the tape part containing z (one bit by cell). The “virtual” window described in the previous section is hence extended to the size $2n^2$.

Theorem 3. *For each nondeterministic 1-LA \mathcal{A} , there exists an equivalent nondeterministic 1-LA \mathcal{A}' , satisfying:*

1. \mathcal{A}' has polynomial size with respect to \mathcal{A} ;
2. in every computation of \mathcal{A}' , each tape cell is visited a number of times which is bounded by some polynomial in the size of \mathcal{A} ;
3. \mathcal{A}' works in linear time: on every input string w , every branch of the computation of \mathcal{A}' halts within $O(|w|)$ steps.

Proof. Again, Item 2 implies Item 3. The proof that Item 2 can be achieved while preserving a polynomial size, namely Item 1, is analogous to those given in the deterministic case (Theorem 1). We emphasize the main differences below.

First, for $z \in \triangleright(\Gamma \setminus \Sigma)^*$, a pair (p, q) belongs to the **relation** τ_z , if and only if $z' \cdot p \cdot X \vDash z'X \cdot q$ where $z'X = z$ with $|X| = 1$. Hence, the virtual window is extended to the size $2n^2$, in order to store a table of size n^2 (one bit by cell) representing a relation $\tau_z \subseteq Q \times Q$. The working alphabet of \mathcal{A}' is therefore set to $(\Gamma \setminus \Sigma) \times \{0, 1\} \cup \Sigma$ where Γ and Σ are the working and input alphabets of \mathcal{A} .

Second, the `readFromTable` subroutine, which takes a state p as argument, returns a nondeterministically chosen state q such that the pair (p, q) has image 1 in the table stored in the left block of the window.⁵ Also the subroutine `simulateLeft` is nondeterministic, as it performs a direct simulation of \mathcal{A} , and

⁵ We implicitly fix a bijection from $\{0, \dots, n^2 - 1\}$ to Q^2 .

possibly calls the subroutine `readFromTable`. As a consequence, the table stored on the left block of the current window does not encode the complete relation τ_z but a subset of it. The reason behind this loss of information, is that checking the nonexistence of computations from $z' \cdot p \cdot X$ to $z'X \cdot q$ requires a universal quantification over computations, for which nondeterminism (which corresponds to existential quantification over computations) is unsuitable. Nevertheless, the relation encoded is a subset of τ_z and when making perfect guesses, it is equal to τ_z . Thus, no accepting computation is missed.

We point out that the simulating automaton can still use a clock of polynomial size (but this time, Sipser's construction cannot apply) to limit the time of direct simulations of \mathcal{A} that necessarily occur in the space bounded locally by the virtual window. Doing so, the resulting 1-LA is halting, has polynomial size with respect to \mathcal{A} , and works in linear time. \square

By analogy to Corollary 1, it follows from Theorem 3:

Corollary 2. *For each 1-LA, there exists an equivalent weight-reducing Hennie machine or halting 2NFA+cg of polynomial size.*

This last result is of particular interest. Indeed, 2NFA+cg's are particular cases of 1-LAS. (It is not the case for 2DFA+cg's with respect to deterministic 1-LAS.) Hence, Corollary 2 gives a kind of normal form for nondeterministic 1-LAS.

Acknowledgement. We are very indebted to Giovanni Pighizzini for suggesting the problem and for many stimulating conversations.

References

1. Bojańczyk, M., Daviaud, L., Guillon, B., Penelle, V.: Which classes of origin graphs are generated by transducers. In: ICALP 2017. LIPIcs, vol. 80, pp. 114:1–114:13 (2017)
2. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. *Inf. Comput.* **205**(8), 1173–1187 (2007)
3. Hennie, F.C.: One-tape, off-line Turing machine computations. *Inf. Comput.* **8**(6), 553–578 (1965)
4. Hibbard, T.N.: A generalization of context-free determinism. *Inf. Comput.* **11**(1/2), 196–238 (1967)
5. Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Boston (1979)
6. Kapoutsis, C.A.: Predicate characterizations in the polynomial-size hierarchy. In: Beckmann, A., Csuhaj-Varjú, E., Meer, K. (eds.) *CiE 2014*. LNCS, vol. 8493, pp. 234–244. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08019-2_24
7. Kutrib, M., Pighizzini, G., Wendlandt, M.: Descriptive complexity of limited automata. *Inf. Comput.* **259**(2), 259–276 (2018)
8. Pighizzini, G.: Nondeterministic one-tape off-line Turing machines. *J. Autom. Lang. Comb.* **14**(1), 107–124 (2009)
9. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. *Int. J. Found. Comput. Sci.* **25**(07), 897–916 (2014)

10. Pighizzini, G., Pisoni, A.: Limited automata and context-free languages. *Fundamenta Informaticae* **136**(1–2), 157–176 (2015)
11. Pighizzini, G., Prigioniero, L.: Limited automata and unary languages. In: Charlier, É., Leroy, J., Rigo, M. (eds.) *DLT 2017*. LNCS, vol. 10396, pp. 308–319. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62809-7_23
12. Průša, D.: Weight-reducing hennie machines and their descriptive complexity. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) *LATA 2014*. LNCS, vol. 8370, pp. 553–564. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04921-2_45
13. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.* **3**(2), 198–200 (1959)
14. Sipser, M.: Halting space-bounded computations. *Theor. Comput. Sci.* **10**(3), 335–338 (1980)
15. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time Turing machines. *Theor. Comput. Sci.* **411**(1), 22–43 (2010)
16. Wagner, K.W., Wechsung, G.: *Computational Complexity*. D. Reidel Publishing Company, Dordrecht (1986)