

Converting Nondeterministic Two-Way Automata into Small Deterministic Linear-Time Machines*

Bruno Guillon^{a,1}, Giovanni Pighizzini^{b,†,2}, Luca Prigioniero^{c,2},
and Daniel Průša^{d,‡,3}

¹*LIMOS, Université Clermont-Auvergne, France*

²*Dipartimento di Informatica, Università degli Studi di Milano, Italy*

³*Faculty of Electrical Engineering, Czech Technical University, Prague*

Abstract

In 1978 Sakoda and Sipser raised the question of the cost, in terms of size of representations, of the transformation of two-way and one-way nondeterministic automata into equivalent two-way deterministic automata. Despite all the attempts, the question has been answered only for particular cases (*e.g.*, restrictions of the class of simulated automata or of the class of simulating automata). However the problem remains open in the general case, the best-known upper bound being exponential. We present a new approach in which unrestricted non-deterministic finite automata are simulated by deterministic models extending two-way deterministic finite automata, paying a polynomial increase of size only. Indeed, we study the costs of the conversions of

*This work contains, in an extended form, some material and results which were previously presented in a preliminary form in conference papers [20] and [5].

^abruno.guillon@uca.fr

^bpighizzini@di.unimi.it

^cprigioniero@di.unimi.it

^dprusapa1@cmp.felk.cvut.cz

[†]Partially supported by Gruppo Nazionale per il Calcolo Scientifico (GNCS-INdAM).

[‡]Supported by the Czech Science Foundation, grant 19-21198S.

nondeterministic finite automata into some variants of one-tape deterministic Turing machines working in linear time, namely Hennie machines, weight-reducing Turing machines, and weight-reducing Hennie machines. All these variants are known to share the same computational power: they characterize the class of regular languages.

1 Introduction

One-way deterministic finite automata (1DFAs) are the canonical acceptor for the class of regular languages. By allowing nondeterministic transitions (1NFAs) or/and movements of the head in both directions on the input tape, so obtaining *two-way deterministic* and *nondeterministic finite automata* (2DFAs/2NFAs), the computational power does not increase [21, 24]. Other extensions of finite automata have been proved to capture the same class of languages, such as *constant-height pushdown automata* [2, 4], *straight-line programs* [2], *1-limited automata* [28, 18, 19], or, as will be of interest for this work, *linear-time one-tape Turing machines* [8, 20, 5, 6].¹

A natural question concerning models that share the same computational power is the comparison of the sizes of their descriptions. In particular, the cost of the elimination of nondeterminism is a standard problem. For instance, it is a classical result that an exponential increase in size is sufficient and, in the worst case, necessary for the conversion of 1NFAs to 1DFAs [21]. However, already for two-way automata, the famous Sakoda and Sipser question concerning the size blowups from 1NFAs or 2NFAs to 2DFAs is a much more intricate problem. For both conversions, Sakoda and Sipser conjectured that the costs are exponential [22]. The question has been solved in some special cases that can be grouped in three classes: by considering restrictions on the simulating machines (*e.g.*, *sweeping* [25], *oblivious* [10], *few reversals* 2DFAs [11]), by considering restrictions on languages (*e.g.*, *unary case* [3]), by considering restrictions on the simulated machines (*e.g.*, *outer-nondeterministic* automata [1, 13]). However, in spite of all attempts, in the general case the question remains open (for further references see [17]). Here, we consider a different approach: in order to obtain a polynomial simulation, we enlarge the family of simulating machines. To this end, we study size blowups for the conversion of 1NFAs and 2NFAs into several variants of *linear-time one-tape deterministic Turing machines*, which all characterize regular languages. These variants and their properties have been investi-

¹Actually, the model considered by Hennie was deterministic. Several extensions of this result, including that to the nondeterministic case and greater time lower bounds for nonregular language recognition, have been stated in the literature [27, 7, 15, 16, 26].

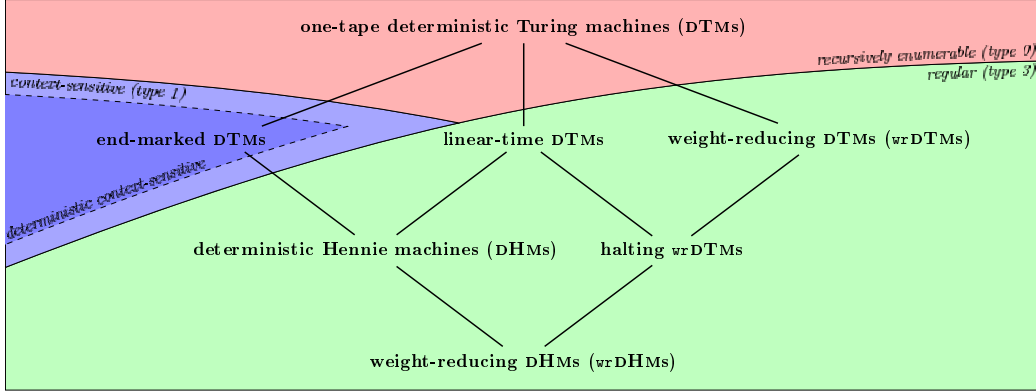


Figure 1: Variants of one-tape deterministic Turing machines and their expressive power confronted with the Chomsky hierarchy. In particular, end-marked DTMs are known as *deterministic linear bounded automata* in the literature, and recognize the so-called *deterministic context-sensitive languages*, a subclass of context-sensitive languages, see [29]. It is still unknown if such inclusion is strict.

gated in [5, 6]. We now give a short description of them (see Figure 1).

As proven by Hennie, linear-time one-tape deterministic Turing machines recognize regular languages only [8]. However, it cannot be decided whether or not a one-tape Turing machine actually works in linear time. This negative result remains true in the restricted case of *end-marked machines*, namely one-tape deterministic Turing machines that do not have any extra space, besides the tape portion which initially contains the input. End-marked machines working in linear time will be called *deterministic Hennie machines*. To overcome the above-mentioned “negative” results, a syntactical restriction on deterministic one-tape Turing machines, called *weight-reducing machines*, has been considered. This restriction enforces computations to either be infinite or to halt within a linear number of steps in the input length. In contrast to Hennie machines, this model benefits from nice properties. In particular, it can be decided whether a one-tape Turing machine is weight-reducing, and whether a weight-reducing Turing machine is halting whence works in linear time. Furthermore, haltingness of the model can always be obtained paying a polynomial size increase only. However, its space is not limited to the portion of the tape that contains the input at the beginning of the computation, namely the device is not end-marked. Considering end-marked linear-time machines satisfying the syntactical restriction of weight-reducing machines, we obtain *weight-reducing Hennie machines*.

Our main result is that each 2NFA \mathcal{A} can be simulated by a one-tape deterministic Turing machine which works in linear time (with respect to the input length) and which has polynomial size with respect to the size of \mathcal{A} . We point out that the resulting machine can use extra space, besides the tape segment which initially contains the input. Next, the machine is halting and *weight-reducing*, thus implying a linear execution time. Hence, nondeterminism can be eliminated with at most a polynomial size increase, obtaining a linear execution time in the input length, and provided the ability to rewrite tape cells and to use some extra space.

We then investigate what happens when the latter capability is removed, namely if the machine does not have any further tape storage, *i.e.*, it is a Hennie machine. We prove that even under this restriction it is still possible to obtain a machine of polynomial size, that is each 2NFA can be transformed into an equivalent Hennie machine of polynomial size. However, the machine resulting from our construction is not weight reducing, unless we require that it agrees with the given 2NFA only on sufficiently long inputs. We do not have this problem in the unary case, namely for a one-letter input alphabet, where we prove that each unary 2NFA can be simulated by a weight-reducing Hennie machine of polynomial size. Similar results are obtained for the transformation of 1NFAs into variants of one-tape deterministic machines.

The paper is organized as follows. In Section 2 we present some fundamental notions and definitions, included those related to the computational models we are interested in, and we state some basic properties. In Section 3 we present our main simulation result: each n -state 2NFA can be transformed into an equivalent halting weight-reducing machine of size polynomial in n . In Section 4 we discuss how the simulation changes if the resulting machine is required to be a Hennie machine. Finally, in Section 5 we revise the results of Sections 3 and 4 under the assumption that the simulated automata are *one-way* instead of being two-way.

2 Preliminaries

In this section we recall some basic definitions and notations. We also describe the main computational models considered in the paper and we give some preliminary results.

We assume the reader familiar with notions from formal languages and automata theory (see, *e.g.*, [9]). Given a set S , $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ , $|w|$ denotes the length of a string $w \in \Sigma^*$, w_i denotes the i -th symbol of w , $i = 1, \dots, |w|$, and ε denotes the empty string.

Finite automata are computational devices equipped with a finite control and a finite read-only tape which is scanned by an input head. A *one-way nondeterministic finite automaton* (1NFA) is defined as a quintuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, 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 $\delta : Q \times \Sigma \rightarrow 2^Q$ is a nondeterministic transition function. At each step, according to its current state p and the symbol σ scanned by the head, \mathcal{A} enters one nondeterministically-chosen state from $\delta(p, \sigma)$ and moves the input head rightward to the next symbol. The machine *accepts* the input if there exists a computation starting from the initial state q_0 with the head on the leftmost input symbol and ending in a final state $q \in F$ after having read the whole input. A 1NFA \mathcal{A} is said to be *deterministic* (1DFA), whenever $\#\delta(q, \sigma) \leq 1$, for any $q \in Q$ and $\sigma \in \Sigma$.

Providing 1NFAs (resp., 1DFAs) with the ability of moving the head back and forth, we obtain *two-way nondeterministic* (resp., *deterministic*) *finite automata* (2NFAs, resp., 2DFAs). They are defined by extending the transition function so that a left (-1) or right ($+1$) head direction is indicated in each instruction. Furthermore, to prevent the head to fall out the input, the device is *end-marked*, in the following sense. Two special symbols \triangleright and \triangleleft not belonging to Σ , called the *left* and the *right endmarker*, respectively, surround each input word, and enforce the computation to stay between them (except at the end of computation when accepting, as described below). More precisely, on input w the tape contains $\triangleright w \triangleleft$, the left endmarker being at position 0 and the right endmarker being at position $|w| + 1$. By $\Sigma_{\triangleright, \triangleleft}$ we denote the set $\Sigma \cup \{\triangleright, \triangleleft\}$. Formally, the transition function of a two-way automaton is $\delta : Q \times \Sigma_{\triangleright, \triangleleft} \rightarrow 2^{Q \times \{-1, +1\}}$ such that, for each transition $(q, d) \in \delta(p, \sigma)$, if $\sigma = \triangleright$ then $d = +1$, and if $\sigma = \triangleleft$ then $d = -1$ or $q \in F$. In this way, the head cannot violate the endmarkers, except at the end of computation to accept. The machine *accepts* the input if there exists a computation starting from the initial state q_0 with the head on the 0-th tape cell (i.e., scanning the left endmarker) and ending in a final state $q \in F$ after violating the right endmarker.

The other main computational model we consider is the *deterministic one-tape Turing machine* (DTM). Such a machine is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ where Q is the set of states, Σ is the input alphabet, Γ is the working alphabet including both Σ and the special blank symbol, denoted by \emptyset , that cannot be written by the machine, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Gamma \rightarrow Q \times (\Gamma \setminus \{\emptyset\}) \times \{-1, +1\}$ is the partial deterministic transition function. In one step, depending on its current state p and on the symbol σ read by the head, a DTM changes its state to q , overwrites the corresponding tape cell with τ and moves the head one cell to the left

when $d = -1$ or to the right when $d = +1$, if $\delta(p, \sigma) = (q, \tau, d)$. Since δ is partial, it may happen that no transition can be applied. In this case, we say that the machine *halts*. At the beginning of computation the input string w resides on a segment of a bi-infinite tape, called *initial segment*, and the remaining infinity of cells contain the blank symbol. The computation over w starts in the initial state with the head scanning the leftmost non-blank symbol. The input is *accepted* if the machine eventually halts in a final state.

Let $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ be a DTM. A *configuration* of \mathcal{A} is given by the current state, the tape contents, and the position of the head. If the head is scanning a non-blank symbol, we describe it by zqu where $zu \in \Gamma^*$ is the finite non-blank contents of the tape, $u \neq \varepsilon$, and the head is scanning the first symbol of u . Otherwise, we describe it by $q\emptyset z$ or zq according to whether the head is scanning the first blank symbol to the left or to the right of the non-blank tape contents z , respectively. If, from a configuration zqu the device may enter in one step a configuration $z'q'u'$, we say that $z'q'u'$ is a *successor* of zqu , denoted $zqu \vdash z'q'u'$. A *halting configuration* is a configuration that has no successor. The reflexive and transitive closure of \vdash is denoted by \vdash^* . On an input string $w \in \Sigma^*$, the *initial configuration* is q_0w . An *accepting configuration* is a halting configuration zq_fu such that q_f is a final state of the machine. A *computation* is a (possibly infinite) sequence of successive configurations. It is *accepting* if it is finite, its first configuration is initial, and its last configuration is accepting. Therefore,

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid q_0w \vdash^* zq_fu, \text{ where } q_f \in F \text{ and } zq_fu \text{ is halting}\}.$$

We say that two machines \mathcal{T} and \mathcal{H} *agree* on some language L if every string in L is accepted by \mathcal{T} if and only if it is accepted by \mathcal{H} .

The notions of configurations, successors, computations, and halting configurations naturally transfer to one-way and two-way finite automata.

In the paper we consider the following restrictions of DTMs.

End-marked machines. We say that a DTM is *end-marked*, if at the beginning of the computation the input string is surrounded by two special symbols belonging to Γ , \triangleright and \triangleleft respectively, called the *left* and the *right endmarkers*, which can never be overwritten, and that prevent the head to fall out the tape portion that initially contains the input. Formally, for each transition $\delta(p, \sigma) = (q, \tau, d)$, $\sigma = \triangleright$ (resp., $\sigma = \triangleleft$) implies $\tau = \sigma$ and $d = +1$ (resp., $d = -1$). This is the deterministic restriction of the well-known *linear-bounded automata* [14]. For end-marked machines, the initial configuration on input w is $q_0\triangleright w\triangleleft$.

Weight-reducing Turing machines. A DTM is *weight-reducing* (wrDTM), if there exists a partial order $<$ on Γ such that each rewriting is decreasing, *i.e.*, $\delta(p, \sigma) = (q, \tau, d)$ implies $\tau < \sigma$. By this condition, in a wrDTM the number of visits to each tape cell is bounded by a constant. However, one wrDTM could have non-halting computations which, hence, necessarily visit infinitely many tape cells.

Linear-time Turing machines. A DTM is said to be *linear-time* if over each input w , its computation halts within $O(|w|)$ steps.

Hennie machines. A *Hennie machine* (DHM) is a linear-time DTM which is, furthermore, end-marked.

Weight-Reducing Hennie machines. By combining previous conditions, *weight-reducing Hennie machines* (wrDHM) are defined as particular DHM, for which there exists an order $<$ over $\Gamma \setminus \{\triangleright, \triangleleft\}$ such that $\delta(p, \sigma) = (q, \tau, d)$ implies $\tau < \sigma$ unless $\sigma \in \{\triangleright, \triangleleft\}$. Observe that each end-marked wrDTM can execute a number of steps which is at most linear in the length of the input. Hence, end-marked wrDTM are necessarily weight-reducing Hennie machines.

The *size* of a machine is given by the total number of symbols used to write down its description. Therefore, the size of a one-tape Turing machine is bounded by a polynomial in the number of states and of working symbols, namely, it is $\Theta(\#Q \cdot \#\Gamma \cdot \log(\#Q \cdot \#\Gamma))$. In the case of nondeterministic (resp., deterministic) finite automata, since no writings are allowed and hence the working alphabet is not provided, the size is linear in the number of instructions and states, which is bounded by a function quadratic (resp., subquadratic) in the number of states and linear in the number of input symbols, namely, it is $\Theta(\#\Sigma \cdot \#Q^2)$ (resp., $\Theta(\#\Sigma \cdot \#Q \cdot \log(\#Q))$).

We now state two preliminary results that will be used in the subsequent sections for building weight-reducing Turing machines and weight-reducing Hennie machines, respectively.

It is known that a DTM works in linear time if and only if there exists a constant k such that in no computation the head visits each tape cell more than k times [8]. The following lemma states that, whenever k is known, a weight-reducing Turing machine can be obtained, augmenting linearly the working alphabet only. Indeed, we can enforce the machine to store on each tape cell the number of further visits the head is allowed to perform on the cell. As this number decreases, the overwriting is decreasing. Using k copies of each working symbols is enough to implement this. Therefore, in order to

define a wrDTM , it is sufficient to define a DTM and to provide a constant k bounding the number of visits of each tape cell.

Lemma 1 ([6]). *Let $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ be a DTM such that, for any input, \mathcal{T} performs at most k computation steps on each tape cell. Then there is a wrDTM \mathcal{A} accepting $L(\mathcal{T})$ with the same set of states Q as \mathcal{T} and working alphabet of size $O(k \cdot \#\Gamma)$. Furthermore, on each input \mathcal{A} uses the same space as \mathcal{T} . Hence, if \mathcal{T} is linear time or end-marked then so is \mathcal{A} .*

Weight-reducing Turing machines extend weight-reducing Hennie machines by allowing the use of some extra space besides the portion that initially contains the input. Indeed, the former model can use a bi-infinite tape while the latter is end-marked. However, it has been shown that every finite computation of a wrTM uses a constant amount of this extra space [6]. We do not know whether the use of this extra space can be avoided in general, while keeping the weight-reducing property and bounding the size increase by a polynomial. Nevertheless, this can be achieved when the inputs are long enough.

Lemma 2. *Let \mathcal{T} be a weight-reducing Turing machine which uses at most C initially-blank cells in every halting computation. Then, there exists a weight-reducing Hennie machine \mathcal{H} of size polynomial in the size of \mathcal{T} , which agrees with \mathcal{T} on every input of length at least C .*

Proof. The idea of the proof is the same as the folkloric simulation of Turing machines working on a bi-infinite tape by Turing machines working on a semi-infinite tape. Indeed, we fold the portion of the tape occurring to the left of the initial segment onto the complementary portion of the tape, thus creating a second track. Similarly, we can fold the portion of the tape occurring to the right of the initial segment onto the complementary portion of the tape. Next, as \mathcal{T} uses at most C initially-blank cells in total, and providing the input has length at least C , we observe that the additional tracks do not overlap hence only one additional track is sufficient for the simulation. On shorter inputs, the simulation could fail giving an outcome different from those of \mathcal{T} . Doing so, we obtain a DHM \mathcal{H} that uses twice the number of states of \mathcal{T} (two copies of each state of \mathcal{T} for indicating which track should be read), and the working alphabet $\Gamma_{\mathcal{H}} = \Gamma \cup \Gamma^2$ where Γ is the working alphabet of \mathcal{T} . In particular, the size of \mathcal{H} is polynomial in the size of \mathcal{T} . Finally, we can extend the order $<_{\mathcal{T}}$ on Γ witnessing that \mathcal{T} is weight-reducing, to an order $<_{\mathcal{H}}$ on $\Gamma_{\mathcal{H}}$ witnessing that \mathcal{H} is weight-reducing. \square

3 Simulating Two-way Automata by Weight-reducing Machines

This section is devoted to present our main simulation: we show that every 2NFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ can be transformed into an equivalent wrDTM of size polynomial in the size of \mathcal{A} . Our construction is based on the classical simulation of 2NFAs by 1DFAs, inspired from Shepherdson's construction [24]. The main idea is to perform forward moves, while updating a table of size $n = \#Q^2$ that describes parts of computations which may occur to the left of the current position. In parallel, an adaptation of the classical powerset construction for converting 1NFAs into 1DFAs is done, in such a way that the set of states that are accessible from the initial configuration when visiting for the first time the current head position is updated at each move. In the simulation by 1DFAs, the table and the set are stored on the finite state control. In our simulation by wrDTMs they will be written, under a suitable encoding, in $O(n)$ many tape cells.

To describe computation paths that occur on some restricted part of the tape, we define *partial configurations*, by relaxing the definition of configurations, as strings xqy where q is a state and $xy \in \{\triangleright, \varepsilon\} \cdot \Sigma^* \cdot \{\triangleleft, \varepsilon\}$ is a factor of the tape content. The successor relation \vdash on configurations extends onto partial configurations. In particular, $zpX \vdash^* zXq$ with $|X| = 1$ means that there exists a computation path

- starting from the rightmost position of zX (with the head scanning the symbol X) in state p ,
- ending while entering the cell to the right of this position in state q , and
- which visits only cells from the part of the tape containing zX in the meantime.

By storing in a set τ_{zX} the pairs of states (p, q) such that such a computation path exists, we save the possible behaviors of \mathcal{A} when moving backward from the current position. Indeed, since acceptance is made after violating the right endmarker, from such a point the device should eventually turn back forward from the current position, in order to reach an accepting configuration. We are going to give the formal definition of the set τ_{zX} for a prefix zX of the tape content, together with the definition of the set γ_{zX} of states that are reachable from the initial configuration, when visiting for the first time the position to the right of the part containing zX . Formally, for

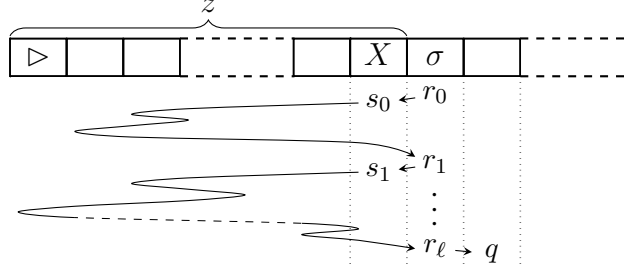


Figure 2: A computation path from $p = r_0$ to q giving $(p, q) \in \tau_{z\sigma}$. For each i , $(s_i, r_{i+1}) \in \tau_z$ and $(s_i, -1) \in \delta(r_i, \sigma)$, while $(q, +1) \in \delta(r_\ell, \sigma)$.

a prefix $zX \in \{\triangleright\} \cdot \Sigma^* \cdot \{\varepsilon, \triangleleft\}$ of the tape content with $|X| = 1$:

$$\begin{aligned} \tau_{zX} &= \{(p, q) \in Q \times Q \mid zpX \vdash^* zXq\}, \text{ and} \\ \gamma_{zX} &= \{r \in Q \mid q_0zX \vdash^* zXr\}. \end{aligned}$$

Observe that a word $w \in \Sigma^*$ is accepted by \mathcal{A} if and only if $F \cap \gamma_{\triangleright w \triangleleft} \neq \emptyset$. In order to simulate \mathcal{A} on input w , it is thus sufficient to incrementally compute γ_z for each prefix z of $\triangleright w \triangleleft$. To do so, we will keep updated the table τ_z as well. Indeed, given γ_z , τ_z and a symbol σ , it is possible to compute $\gamma_{z\sigma}$ and $\tau_{z\sigma}$. This is achieved by observing that (see Figure 2):

1. $(p, q) \in \tau_{z\sigma}$ if and only if there exists a sequence $r_0, s_0, r_1, s_1, \dots, r_\ell \in Q$, with $\ell \geq 0$, satisfying:
 - $r_0 = p$,
 - $(q, +1) \in \delta(r_\ell, \sigma)$, and
 - $(s_i, -1) \in \delta(r_i, \sigma)$ and $(s_i, r_{i+1}) \in \tau_z$, for $i = 0, \dots, \ell - 1$.
2. $q \in \gamma_{z\sigma}$ if and only if there exists $p \in \gamma_z$ such that $(p, q) \in \tau_{z\sigma}$.

We represent a pair (γ_z, τ_z) as a word uv in $\{0, 1\}^*$ with $|u| = n$ and $|v| = n^2$. Each bit of u (resp., v) indicates the membership of some state p (resp., some pair (p, q) of states) to the set γ_z (resp., τ_z) through an implicitly fixed bijection from Q to $\{1, \dots, n\}$ (resp., from Q^2 to $\{1, \dots, n^2\}$). For each input symbol σ , there exists a halting DHM \mathcal{T}_σ which computes $(\gamma_{z\sigma}, \tau_{z\sigma})$ from (γ_z, τ_z) in the following sense. On input $uv \in \{0, 1\}^{n+n^2}$ encoding (γ_z, τ_z) , \mathcal{T}_σ ends the computation with the tape containing the encoding $u'v' \in \{0, 1\}^{n+n^2}$ of $(\gamma_{z\sigma}, \tau_{z\sigma})$. Notice that this computation does not depend on the entire z , which, indeed, is not given to \mathcal{T}_σ , but only on the information on z stored in γ_z and τ_z which are given in input.

Lemma 3. *For each $\sigma \in \Sigma \cup \{\triangleleft\}$, there exists a halting DHM \mathcal{T}_σ with $O(n^6)$ states and $O(1)$ working symbols that on input (γ_z, τ_z) halts with the tape containing $(\gamma_{z\sigma}, \tau_{z\sigma})$ after $O(n^5)$ visits to each cell. The input and the output are represented on the tape as strings in $\{0, 1\}^{n+n^2}$.*

Proof. Fixed σ , let uv be the input string encoding the pair of tables (γ_z, τ_z) , of size n and n^2 respectively. In order to update them, \mathcal{T}_σ uses a second track on the tape, on which it will progressively build the updated tables. At the end of the computation, namely when the updated tables have been determined and written down over the second track, the device performs a projection of the tape on its second track, in order to produce the correct output, and halts.

We fix the working alphabet $\Gamma = \{0, 1\} \cup \{0, 1\}^2$. The “simple” symbols from $\{0, 1\}$ are used only for the input and the output of \mathcal{T}_σ . From now on, we suppose that the tape contains only symbols from the 2-track alphabet part, *i.e.*, the right side of the union. Moreover, since the length of the input is fixed and it is $n + n^2$, we can suppose that \mathcal{T}_σ keeps updated a state component of size $n + n^2$ which always stores the position of its head on the tape. This allows it to navigate over the tables.

We divide the tape into two parts: a prefix \bar{u} of length n (thus covering the factor u which encodes γ_z on its first track) and a suffix \bar{v} of length n^2 (thus covering the factor v which encodes τ_z on its first track). As previously explained, the updated table $\gamma_{z\sigma}$ can easily be obtained once the updated table $\tau_{z\sigma}$ has been computed. Hence, we first show how to write the table $\tau_{z\sigma}$ on the second track of \bar{v} . This is achieved using the space n available on the second track of \bar{u} as temporary memory, to which we refer as *temporary table*.

As observed above (see Figure 2), a computation path on the segment containing $z\sigma$ starting from the rightmost position of the segment and exiting the segment to the right at its last step, *i.e.*, a computation of the form $zp\sigma \vdash^* z\sigma q$, can be decomposed into an alternation of computation paths on the segment containing z (described by the table τ_z) and of backward computation steps on σ connecting these paths, followed by a last forward computation step on σ that exits the segment. For each state p , in order to decide which pairs (p, q) belong to $\tau_{z\sigma}$, the machine \mathcal{T}_σ first computes the set Z_p of states that are reachable at the rightmost position of the segment containing $z\sigma$, from the state p at the same position, by visiting only cells from the segment, *i.e.*,

$$Z_p = \{r \mid zp\sigma \vdash^* zr\sigma\}.$$

Thus, a pair (p, q) belongs to $\tau_{z\sigma}$ if and only if for some $r \in Z_p$, we have

$(q, +1) \in \delta(r, \sigma)$. For a fixed p , \mathcal{T}_σ can incrementally construct Z_p on the temporary table as follows. Initially, all the cells from the table are unmarked (*i.e.*, contain 0) except the one corresponding to state p which contains 1. The update process behaves as follows: for each state r corresponding to a marked cell, each state s such that $(s, -1) \in \delta(r, \sigma)$, and each state r' such that $(s, r') \in \tau_z$, the machine marks the cell corresponding to r' with 1 in the temporary table. Since Z_p has size bounded by n , after at most n passes, the temporary table is not modified anymore and contains exactly an encoding of Z_p .

So done, computing the set Z_p uses only a polynomial number of states in n . This is however not sufficient to get a weight-reducing machine of polynomial size. To this end, using Lemma 1, we should indeed prove that the number of visits to each cell is bounded by some polynomial in n . To update Z_p , three nested loops on states, namely on r , s , and r' , are used. Once such a triple is fixed, the machine navigates on the tape in order to check that r is currently marked in the temporary table, and $(s, r') \in \tau_z$ (notice that the condition $(s, -1) \in \delta(r, \sigma)$ is verified in constant time, since σ is fixed). These two conditions require to read the corresponding cells in the temporary table (on \bar{u}) and in the table τ_z (on \bar{v}), respectively. This can be performed by visiting each tape cell at most twice. Marking the cell corresponding to r' also implies to scan the tape part \bar{u} twice. As the operation is repeated for each triple, we obtain that the number of visits to each cell in a pass for updating Z_p is $O(n^3)$. Since the number of passes is at most n , the total number of visits to each cell is $O(n^4)$. Moreover, this operation can be implemented by using $O(n^3)$ states because we just need to remember the values of r , s , and r' , but not the number of the pass: it is sufficient to use a bit to remember if during the last pass at least one state has been added to Z_p . This is because if no states are added during a pass, then no states will be added executing further iterations.

Once Z_p has been computed, for each state r corresponding to a marked cell in the temporary table, and each state q such that $(q, +1) \in \delta(r, \sigma)$, \mathcal{T}_σ adds the pair (p, q) to the table $\tau_{z\sigma}$ represented on the second track of \bar{v} . This requires to visit $O(n)$ times each tape cell and can be performed using only a quadratic number of states in n .

By repeating this for each state p , we manage to update the table from τ_z to $\tau_{z\sigma}$. Finally, we can update the table γ_z . As observed before, it is sufficient to consider for each state q , whether $(p, q) \in \tau_{z\sigma}$ for some $p \in \gamma_z$. This last step requires only a quadratic number of states and a linear number of visits to each cell. Combining the above-described subroutines, and taking into account that the state component which stores the head position has size $O(n^2)$, we obtain a DHM with $O(n^6)$ states and $O(1)$ working symbols,

whose number of visits to each tape cell is in $O(n^5)$. \square

We are now ready to state our main simulation.

Theorem 1. *Every n -state 2NFA can be transformed into an equivalent halting wrDTM of size polynomial in n .*

Proof. Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a 2NFA. We build a deterministic Turing machine that mimics the simulation of \mathcal{A} by a 1DFA: after reading any prefix z of an input w , the machine stores the tables $\gamma_{\triangleright z}$ and $\tau_{\triangleright z}$ and finally checks the existence of a final state in $\gamma_{\triangleright w \triangleleft}$. The tables are stored on a suitable tape track and updated each time a further input symbol is read, using the method presented in Lemma 3. This can be achieved by switching between two tape tracks at each update of the tables. However, as the number of updates is linear in the length of the input, storing and updating the tables on a fixed part of length $n + n^2$ of the tape would lead to a non-weight-reducing Turing machine. To handle this issue, for each prefix z of w , we store the tables $\gamma_{\triangleright z}$ and $\tau_{\triangleright z}$ on the $n + n^2$ cells that precede the last position of z . (Remember that, in a wrDTM , some initially-blank cells to the left of the initial segment are available.) Thus, at each update of the tables made according to Lemma 3, the tables are shifted one cell to the right. Hence, since a fixed cell may occur in $n + n^2$ successive stored tables, the number of visits to each cell is in $O(n^7)$, and the number of states is in $O(n^6)$. We thus obtain a halting weight-reducing Turing machine equivalent to \mathcal{A} whose size is polynomial in the size of \mathcal{A} by Lemma 1. Furthermore, the machine uses only $n + n^2$ initially-blank cells, that are all to the left of the initial segment. \square

4 Simulating Two-way Automata by Hennie Machines

In Section 3 we provided a polynomial size conversion from 2NFAs to wrDTMs . The resulting machines use further tape cells, besides the initial segment. In this section we study how to make such a simulation when the use of such extra space is not allowed, namely when we want to obtain a deterministic Hennie machine. We show that a polynomial conversion still exists, but we are not able to guarantee that the resulting machine is weight reducing. Actually this issue is related to “short” inputs, namely to strings of length less than n^2 , where n is the number of states of the given 2NFA. For such inputs we do not have enough tape space to perform the simulation in Theorem 1. We will deal with them, by using a different technique.

Let us start by considering acceptance of “long” inputs.

Theorem 2. *For each n -state 2NFA \mathcal{A} , there exists a wrDHM \mathcal{H} of size polynomial in n which agrees with \mathcal{A} on strings of length at least n^2 .*

Proof. The technique used in the proof of Theorem 1 can be exploited, with slight modifications. Indeed, when recovering the tables corresponding to the “short” prefixes z ’s of the tape content, the wrDTM machine resulting from the above construction uses up to $n + n^2$ initially-blank cells to the left of the initial segment, that are not any longer available with a wrDHM. By folding n of these cells on an additional track, as in the proof of Lemma 2, we can reduce this space amount to n^2 cells by a polynomial size increasing. Hence, applying Lemma 2, we can obtain a wrDHM which agrees with the original machine on inputs of length at least n^2 . \square

In the case of *unary* 2NFAs, namely working on a single-letter input alphabet, the number of short inputs that are not handled by Theorem 2 is n^2 . They can be managed in a read-only preliminary phase which uses $O(n^2)$ states.

Theorem 3. *Every n -state unary 2NFA is equivalent to a wrDHM of size polynomial in n .*

Proof. We simulate a given unary 2NFA \mathcal{A} as follows. Let X be the finite set $\{i < n^2 \mid i = |w| \text{ for some } w \in L\}$. First, the head of our simulating wrDHM is moved rightward to test whether the input is shorter than n^2 , using a counter from 0 to $n^2 - 1$. In this case, the machine accepts if and only if the counter value belongs to X . Otherwise, the head is moved back to the left endmarker and the simulation from Theorem 2 is performed. With respect to the wrDTM obtained from Theorem 2, our device uses $O(n^2)$ additional states and $O(1)$ extra working symbols. Hence, our construction yields a wrDHM equivalent to \mathcal{A} of size polynomial in n . \square

In the nonunary case, since the number of short strings is exponential in n , we cannot apply the same technique as in Theorem 3. However, we are able to obtain a polynomial size Hennie machine (not necessarily weight reducing), using a different technique, which is based on the analysis of the computation graph of the simulated 2NFA.

Theorem 4. *Each n -state 2NFA is equivalent to a DHM of size polynomial in n .*

Proof. Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a 2NFA, with $\#Q = n$. Without loss of generality we suppose $F = \{q_f\}$. Let $w \in \Sigma^*$ be an input word, with $m = |w|$. We distinguish three cases, depending on m . Observe that the simulating DHM \mathcal{H} can decide the case by performing a reading traversal of the input using a polynomial number of states.

If $m \geq n^2$, then \mathcal{H} simulates \mathcal{A} as in Theorem 2.

If $m \leq \log n$, then \mathcal{H} simulates a 1DFA with a polynomial number of states in n (and in the number of input symbol which is assumed to be a fixed constant), which agrees with \mathcal{A} on all strings of length at most $\log n$.

Finally, if $\log n < m < n^2$, then \mathcal{H} checks whether there is an accepting computation of \mathcal{A} on w by analyzing the *computation graph* $G = \langle V, E \rangle$ of \mathcal{A} on w , defined as follows. The set of vertices of G is $V = Q \times \{0, \dots, m+1\} \cup \{(q_f, m+2)\}$, where the pair $(q, i) \in V$ corresponds to the configuration on input w in which \mathcal{A} is in the state q while scanning the i -th symbol of the input tape. The edges in E represent single moves, *i.e.*, there exists an edge from (q, i) to (p, j) if and only if $(p, j-i) \in \delta(q, \tilde{w}_i)$, where $\tilde{w} = \triangleright w \triangleleft$.

The simulating Hennie machine \mathcal{H} should check the existence of a computation of \mathcal{A} starting from the initial state q_0 with the head on the left endmarker (*i.e.*, at position 0) and ending in the unique final state q_f after violating the right endmarker (*i.e.*, at position $m+2$). This is equivalent to check the existence of a path from the node $(q_0, 0)$ to the node $(q_f, m+2)$ in G . Let $K = n(m+2) + 1$ be the number of nodes in G . If such a path exists, then there should exist one of length at most K . Hence, checking the existence of an accepting computation reduces to checking the existence of a path of length at most K in G . The recursive function **reachable** is used to perform this checking by calling **reachable** $(q_0, 0, q_f, m+2, K)$.

Let us describe how a call of **reachable** (p, i, q, j, T) works. The function has to check if there exists a computation from (p, i) to (q, j) of length at most T . This is done by using a *divide-and-conquer* technique as in the famous proof of the Savitch's Theorem [23]. If $(p, i) = (q, j)$, *i.e.*, there is a path of length 0 from (p, i) to (q, j) , then the function returns **true** independently of T (Line 1). Otherwise, if $T = 0$ but $(p, i) \neq (q, j)$, then the function returns **false** (Line 2), while, if $T = 1$, the function returns **true** if there is a suitable edge in G (Line 4). In order to verify that, \mathcal{H} saves (q, i) and (p, j) in its internal state and then, if the distance between i and j is 1, it moves its head to position i , reads the symbol \tilde{w}_i , and checks $(q, j-i) \in \delta(p, \tilde{w}_i)$. Notice that this read-only process uses only a number of states polynomial in n .

In the recursive case, for checking if there exists a path in the graph from (p, i) to (q, j) of length at most $T > 1$, \mathcal{H} verifies whether there exists a node $(r, \ell) \in Q \times \{0, \dots, m+1\}$ such that there is a path from (p, i) to (r, ℓ)

Function `reachable`(p, i, q, j, T): boolean

Checks the existence of a path from (p, i) to (q, j) of length less than or equal to T in the graph of the configurations of a given 2NFA on input $w = w_1 \cdots w_m$

```

1 if  $(p, i) = (q, j)$  then return true
2 if  $T = 0$  then return false
3 if  $T = 1$  then
4   | if  $(q, j - i) \in \delta(p, \tilde{w}_i)$  then return true
5 else
6   | foreach  $r, \ell \in Q \times \{0, \dots, m + 1\}$  do
7     |   | if reachable( $p, i, r, \ell, \lfloor T/2 \rfloor$ ) then
8       |   |   | if reachable( $r, \ell, q, j, \lceil T/2 \rceil$ ) then return true
9 return false

```

of length at most $\lfloor \frac{T}{2} \rfloor$ and a path from (r, ℓ) to (q, j) of length at most $\lceil \frac{T}{2} \rceil$ (Lines 6 to 8). This is done by trying all possible nodes (r, ℓ) until finding one satisfying these conditions. If it does not exist, then the procedure returns **false** (Line 9).

Recursive calls to the function `reachable` can be naturally saved on a pushdown store. More precisely, at the beginning of the simulation the store is empty. When a call to `reachable`(p, i, q, j, T) is performed, the activation record, consisting of the parameters p, i, q, j , and T , is pushed on the top of the pushdown. Similarly, when `reachable` returns, the activation record of the last call is popped off. The function `reachable` uses seven variables, five of them being arguments saved on the pushdown store, and two being the local variables r and ℓ . As these two local variables are arguments of inner recursive calls, their values can be recovered when popping off the inner activation record (after the corresponding call has returned). Hence, the state components saving r and ℓ are freed at each recursive call. Therefore, all the checks performed by `reachable` can be done with a number of states that is polynomial in n , and using the pushdown alphabet $(Q \times \{0, \dots, n^2\})^2 \times \{0, \dots, \lceil \log K \rceil\}$ of size polynomial in n .

Finally, notice that the maximum recursion depth is $\lceil \log K \rceil = O(\log n)$. The stack of recursion calls can be stored in a separated track on $\log n$ tape cells, by using standard space compression techniques, that only induce a polynomial increase of the working alphabet. Since the input length m is larger than $\log n$, \mathcal{H} has enough space in its initial segment. The number of visits to each cell is super-polynomial in n , hence the machine is not weight-reducing. However, because the input lengths are bounded by n^2 , the number of visits to each cell is bounded by a number which only depends on n .

Considering also how the machine works on inputs of length at least n^2 , this allows us to conclude that the working time of the whole machine \mathcal{H} is linear in the input length. \square

It is natural to ask if Theorem 4 can be improved in order to obtain from a given 2NFA \mathcal{A} an equivalent wrDHM of polynomial size. In the light of Theorem 2, to do that it will be enough to obtain a wrDHM of polynomial size which agrees with the 2NFA on “short” inputs. With this respect, we point out that the problem of Sakoda and Sipser seems to be hard even when restricted to strings of length polynomial in the number of states of \mathcal{A} [12].

5 The One-Way Case

In this section we restrict our attention to one-way automata simulations. A natural question is to ask if in the case of 1NFAs results stronger than those presented in Section 4 can be achieved. A simulation of 1NFAs by wrDHMs was studied in [20, Theorem 11], claiming that each n -state 1NFA \mathcal{A} has an equivalent wrDHM \mathcal{H} of size polynomial in n . Unfortunately, the presented proof is incorrect as it casts the problem of \mathcal{A} acceptance as the problem of reachability in an undirected computation graph. Existence of a path connecting the initial and an accepting configuration in such a graph does not guarantee the existence of an accepting computation of \mathcal{A} since the path can include “back” edges.

By revising [20, Theorem 11], we prove two weakened variants of this result. In the first one, the simulation holds only for long enough inputs. The improvement with respect to Theorem 2 is that, in this case, short inputs are the strings of length less than n rather than n^2 . In the second variant, we show that each 1NFA can be simulated by a deterministic Hennie machine which, however, is not weight-reducing.

Let us start by presenting the weight-reducing simulation for “long” inputs.

Theorem 5. *For each n -state 1NFA \mathcal{A} , there exists a wrDHM \mathcal{H} of size polynomial in n which agrees with \mathcal{A} on strings of length at least n .*

Proof. In order to obtain \mathcal{H} , we build a dTM \mathcal{T} equivalent to \mathcal{A} with the following properties:

- (P1) the size of \mathcal{T} is bounded by a polynomial in the size of \mathcal{A} ;
- (P2) the number of visits to each cell in any computation is bounded by a polynomial in n ;
- (P3) only n tape cells beside the initial segment are used, in any computation.

Then, using (P2) and Lemma 1, \mathcal{T} can be turned into an equivalent **wrDTM** preserving (P1) and (P3). Finally, using (P3) and Lemma 2, the resulting **wrDTM** is converted to the wanted **wrDHM** \mathcal{H} whose size is polynomial in the size of \mathcal{A} .

We build \mathcal{T} by adapting the classical powerset construction for converting 1NFAs into 1DFAs. In this standard construction, the simulating 1DFA scans the input word, while keeping updated, at each step, the subset of states that can be reached by \mathcal{A} from the initial configuration while reading input symbols. Our construction uses this technique but, instead of storing the subset in the finite control of the simulating machine, \mathcal{T} stores the successive subsets on the tape, exploiting the writing capability of **wrDTMs** in order to avoid an exponential size blowup.

Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ with $Q = \{q_0, q_1, \dots, q_{n-1}\}$. Following the notations of Section 3, for any input prefix z , we consider the set $\gamma_z \subseteq Q$ of states that are reachable from the initial configuration after having read z . In particular, γ_ε is the singleton $\{q_0\}$ and an input w is accepted by \mathcal{A} if and only if $\gamma_w \cap F$ is nonempty. In order to store any subset γ_z of Q on the tape, we represent it as a word $u \in \{0, 1\}^n$, in which u_i is 1 if and only if $q_i \in \gamma_z$. Ranging over input prefixes, \mathcal{T} iteratively computes $\gamma_{z\sigma}$ from γ_z , where $\sigma \in \Sigma$ is the $(|z| + 1)$ -th input symbol. To avoid erasing γ_z when writing down $\gamma_{z\sigma}$, \mathcal{T} uses two distinct tape tracks: γ_z is read from the first track and $\gamma_{z\sigma}$ is written on the second one (shifted by one cell to the right with respect to γ_z). The role of the two tracks is inverted after each such update.

At the beginning of the computation, \mathcal{T} writes γ_ε on the n cells preceding the initial segment (on one track, suppose the first one for ease of exposition). During the simulation, when \mathcal{T} reaches a position for the first time, the n tape cells preceding that position will contain the encoding of γ_z (on the first track) where z is the input prefix read so far. From such a point, \mathcal{T} is able to compute $\gamma_{z\sigma}$, working on the tape portion of length $n + 1$ that contains both γ_z and the current position containing σ , and to which we refer as the *current working segment*. Indeed, for each $q_i \in Q$, we have $q_i \in \gamma_{z\sigma}$ if and only if $q_i \in \delta(q_j, \sigma)$ for some $q_j \in \gamma_z$. Since \mathcal{T} can test for each $q_i, q_j \in Q$ whether, on the one hand, q_j belongs to γ_z (by visiting the j -th cell of the current working segment which stores the j -th bit of γ_z on the first track), and, on the other hand, $q_i \in \delta(q_j, \sigma)$ (by inspecting the transition table of \mathcal{A}), it can write $\gamma_{z\sigma}$ on the tape. More precisely, when the two above conditions are satisfied, \mathcal{T} writes 1 on the $(i + 1)$ -th cell of the current working segment (on the second track), for storing the i -th bit of $\gamma_{z\sigma}$. Notice that $\gamma_{z\sigma}$ is shifted one cell to the right with respect to γ_z , so that its last bit is written on the rightmost cell of the current working segment (which initially contains σ).

Furthermore, during the update procedure, each cell of the current working segment is visited $O(n)$ times. Finally, when reaching the first cell to the right of the initial segment (containing the blank symbol), \mathcal{T} accepts if and only if γ_w contains a final state. By slightly modifying the above update procedure, we may assume that this information has been prepared in the finite control. Indeed, when computing $\gamma_{z\sigma}$, \mathcal{T} can store in a state component whether $\gamma_{z\sigma}$ intersects F .

The described approach implies the use of n extra tape cells to the left of the initial segment, on which \mathcal{T} has initially written γ_ε . No extra space is necessary, thus (P3) is satisfied. Moreover, the simulating machine uses $O(1)$ working symbols, and $O(\#\Sigma \cdot n^3)$ states for simultaneously storing the variables σ , q_i , q_j , and the head position relative to the current working segment. This implies (P1). Next, each tape cell is used for storing at most n successive subsets of states. Hence, its total number of visits is in $O(n^2)$, yielding (P2). \square

By Theorem 4, every n -state 2NFA can be transformed into an equivalent DHM of size polynomial in n . So, using the same result, we can also transform 1NFAs. For this particular case, we now present a more direct simulation which uses a technique similar to that of proof of Theorem 4. Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be an n -state 1NFA. At the cost of one extra state, we suppose $F = \{q_f\}$. Let $w \in \Sigma^*$ be an input to \mathcal{A} . Distinguish two cases by $m = |w|$.

If $m \leq n$, the machine \mathcal{H} checks whether there is an accepting computation of \mathcal{A} on w by calling `reachableOneWay`($q_0, 0, q_f, m$), a slightly modified version of the function `reachable` presented in the proof of Theorem 4, adapted to deal with 1NFAs.

Function `reachableOneWay`(p, i, q, j): boolean

Checks the existence of a path from (p, i) to (q, j) of length $j-i$ in the graph of the configurations of a given 1NFA on input $w = w_1 \cdots w_m$

```

10 if  $(p, i) = (q, j)$  then return true
11 if  $j - i = 1$  and  $q \in \delta(p, w_j)$  then return true
12 if  $j - i > 1$  then
13   foreach  $r \in Q$  do
14     if reachableOneWay( $p, i, r, \lfloor (i+j)/2 \rfloor$ ) then
15       if reachableOneWay( $r, \lceil (i+j)/2 \rceil, q, j$ ) then return true
16 return false

```

The recursion depth is $O(\log m)$. At each level of the recursion, the function needs to store states p, q, r and indices i, j . This can be done in a

tape cell if $O(n^5)$ working tape symbols are provided for this purpose. The function runs in $n^{O(\log m)}$ time.

If $m > n$, the Hennie machine \mathcal{H} executes the computation described in the proof of Theorem 5.

In conclusion, the constructed machine \mathcal{H} has the number of states and working tape symbols polynomial in n , hence it is of size polynomial in n . Note that the number of transitions performed by \mathcal{H} over any tape field is $n^{O(\log n)}$.

6 Conclusion

Sakoda and Sipser raised the question of the cost of the elimination of non-determinism from finite automata exploiting the possibility of moving the head in both directions. Though they conjectured that this cost is exponential, we proved that a determinization of polynomial size cost is possible for some simulating machines which are deterministic and have some extra capabilities than 2DFAs. The extensions of 2DFAs we considered are some special cases of one-tape Turing machines which are not more powerful than 2DFAs, namely they recognize the class of regular languages.

In Theorem 4 we showed such a polynomial determinization using Hennie machines. However, this result is not fully satisfying because of known drawbacks of DHMs: on the one hand, it is not decidable whether a Turing machine is actually a Hennie machine; on the other hand, no recursive function bounds the size blowup of the conversion of DHMs into 1DFAs. These drawbacks are avoided when moving to weight-reducing devices [6].

We do not know if the simulations of Theorem 4 can be changed in such a way that the resulting DHM is always weight-reducing while keeping the size cost polynomial, and we leave it as an open problem. However, we solve this question in some particular cases. Indeed, when equivalence is required only over long enough inputs, or when the input alphabet is unary, polynomial size determinizations using wrDHMs are possible (Theorems 2, 3 and 5). Moreover, our main result states that it is always possible to eliminate non-determinism from finite automata by using weight-reducing linear-time Turing machines as long as they are not required to be end-marked (Theorem 1), namely when the simulating machine is allowed to use extra tape beside the part that initially contains the input.

References

- [1] Viliam Geffert, Bruno Guillon, and Giovanni Pighizzini. Two-way automata making choices only at the endmarkers. *Inf. Comput.*, 239:71–86, December 2014.
- [2] Viliam Geffert, Carlo Mereghetti, and Beatrice Palano. More concise representation of regular languages by automata and regular expressions. *Inf. Comput.*, 208(4):385–394, 2010.
- [3] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. *Theor. Comput. Sci.*, 295(1–3):189 – 203, 2003.
- [4] Bruno Guillon, Giovanni Pighizzini, and Luca Prigioniero. Non-self-embedding grammars, constant-height pushdown automata, and limited automata. *Int. J. Found. Comput. Sci.*, 31(8):1133–1157, 2020.
- [5] Bruno Guillon, Giovanni Pighizzini, Luca Prigioniero, and Daniel Průša. Two-way automata and one-tape machines - read only versus linear time. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 366–378. Springer, 2018.
- [6] Bruno Guillon, Giovanni Pighizzini, Luca Prigioniero, and Daniel Průša. Weight-reducing Turing machines. *CoRR*, 2021.
- [7] Juris Hartmanis. Computational complexity of one-tape Turing machine computations. *J. ACM*, 15(2):325–339, 1968.
- [8] F. C. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, 8(6):553–578, 1965.
- [9] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
- [10] Juraj Hromkovic and Georg Schnitger. Nondeterminism versus determinism for two-way finite automata: Generalizations of Sipser’s separation. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 439–451. Springer, 2003.

- [11] Christos A. Kapoutsis. Nondeterminism is essential in small two-way finite automata with few reversals. *Inf. Comput.*, 222:208–227, 2013.
- [12] Christos A. Kapoutsis. Two-way automata versus logarithmic space. *Theory Comput. Syst.*, 55(2):421–447, 2014.
- [13] Christos A. Kapoutsis and Giovanni Pighizzini. Two-way automata characterizations of L/poly versus NL. *Theory Comput. Syst.*, 56(4):662–685, 2015.
- [14] S.-Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, 1964.
- [15] Pascal Michel. An NP-complete language accepted in linear time by a one-tape Turing machine. *Theor. Comput. Sci.*, 85(1):205–212, 1991.
- [16] Giovanni Pighizzini. Nondeterministic one-tape off-line Turing machines. *Journal of Automata, Languages and Combinatorics*, 14(1):107–124, 2009.
- [17] Giovanni Pighizzini. Two-way finite automata: Old and recent results. *Fundam. Inform.*, 126(2-3):225–246, 2013.
- [18] Giovanni Pighizzini and Andrea Pisoni. Limited Automata and Regular Languages. *IJFCS*, 25(07):897–916, November 2014.
- [19] Giovanni Pighizzini and Luca Prigioniero. Limited automata and unary languages. *Inf. Comput.*, 266:60–74, 2019.
- [20] Daniel Průša. Weight-reducing Hennie machines and their descriptive complexity. In Adrian-Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 553–564. Springer, 2014.
- [21] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.
- [22] William J. Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 275–286. ACM, 1978.

- [23] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.
- [24] J. C. Shepherdson. The Reduction of Two-Way Automata to One-Way Automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.
- [25] Michael Sipser. Lower Bounds on the Size of Sweeping Automata. *J. Comput. Syst. Sci.*, 21(2):195–202, 1980.
- [26] Kohtaro Tadaki, Tomoyuki Yamakami, and Jack C. H. Lin. Theory of one-tape linear-time Turing machines. *Theor. Comput. Sci.*, 411(1):22–43, 2010.
- [27] B. A. Trakhtenbrot. Turing machine computations with logarithmic delay, (in Russian). *Algebra I Logica*, 3:33–48, 1964.
- [28] Klaus W. Wagner and Gerd Wechsung. *Computational Complexity*. D. Reidel Publishing Company, Dordrecht, 1986.
- [29] Daniel A. Walters. Deterministic context-sensitive languages: Part II. *Inf. Control.*, 17(1):41–61, 1970.