HAL open science

Weight-reducing Turing machines

Bruno Guillon, Giovanni Pighizzini, Luca Prigioniero, Daniel Průša

To cite this version:

Bruno Guillon, Giovanni Pighizzini, Luca Prigioniero, Daniel Průša. Weight-reducing Turing machines. Information and Computation, 2023, 292, pp.105030. 10.1016/j.ic.2023.105030 . hal-04093540

HAL Id: hal-04093540

https://hal.science/hal-04093540v1

Submitted on 10 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Weight-Reducing Turing Machines*

Bruno Guillon[a], Giovanni Pighizzini[b,1,*], Luca Prigioniero[b], Daniel Průša[c,2]

[a] *LIMOS, Université Clermont-Auvergne, France*
[b] *Dipartimento di Informatica, Università degli Studi di Milano, Italy*
[c] *Faculty of Electrical Engineering, Czech Technical University, Prague*

Abstract

It is well known that one-tape Turing machines running in linear time are no more powerful than finite automata; namely they recognize exactly the class of regular languages. We prove that it is not decidable if a one-tape machine runs in linear time, even if it is deterministic and restricted to use only the portion of the tape that initially contains the input. This motivates the introduction of a constructive variant of one-tape machines, called a weight-reducing machine, and the investigation of its properties. We focus on the deterministic case. In particular, we show that, paying a polynomial size increase only, each weight-reducing machine can be turned into a halting one that runs in linear time. Furthermore each weight-reducing machine can be converted into equivalent nondeterministic and deterministic finite automata by paying an exponential and doubly-exponential increase in size,

*This work contains, in an extended form, some material and results that were previously presented in a preliminary form in conference papers [1] and [2].

*Corresponding author

Email addresses: `bruno.guillon@uca.fr` (Bruno Guillon), `pighizzini@di.unimi.it` (Giovanni Pighizzini), `prigioniero@di.unimi.it` (Luca Prigioniero), `prusapa1@cmp.felk.cvut.cz` (Daniel Průša)

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

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

respectively. These costs cannot be reduced in the worst case.
Keywords: one-tape Turing machines, descriptional complexity, Hennie machines

1. Introduction

The characterization of classes of languages in terms of recognizing devices is a classical topic in formal languages and automata theory. The bottom level of the Chomsky hierarchy, i.e., the class of type 3 or regular languages, is characterized in terms of deterministic and nondeterministic finite automata (DFAs and NFAs, respectively). The top level of the hierarchy, i.e., type 0 languages, can be characterized by Turing machines (in both deterministic and nondeterministic versions), even in the following restricted case: there is only a single (infinite or semi-infinite) tape, which initially holds only the input, and whose contents can be rewritten by the finite control.

Considering machines that make a restricted use of space or time, it is possible to characterize other classes of the hierarchy. On the one hand, if the available space is restricted only to the portion of the tape that initially contains the input and nondeterministic transitions are allowed, the resulting model, known as *linear-bounded automaton*, characterizes type 1 or context-sensitive languages [3]. On the other hand, when the length of the computations, i.e., the time, is linear in the input length, one-tape Turing machines are no more powerful than finite automata; namely they recognize only regular languages, as proved by Hennie in 1965 [4].[3]

[3]Actually, the model considered by Hennie was deterministic. Several extensions of this result, including that to the nondeterministic case and greater time lower bounds for

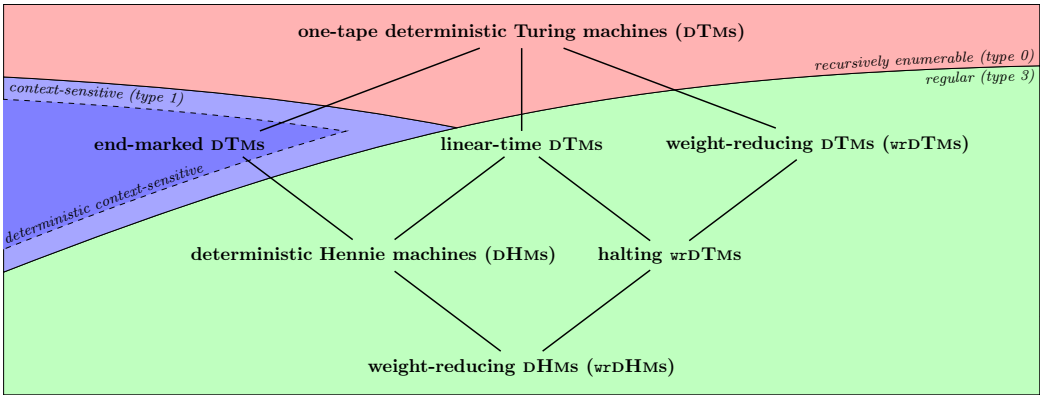

Figure 1: Variants of one-tape deterministic Turing machines and their expressive power confronted with the Chomsky hierarchy. In particular, end-marked DTMs are known as *deterministic linear bounded automata* in the literature, and recognize the so-called *deterministic context-sensitive languages*, a subclass of context-sensitive languages; see [10]. It is still unknown if this inclusion is strict. Each line, from top to bottom, connects a model with one of its special cases.

The main purpose of this paper is the investigation of some fundamental properties of several variants of one-tape Turing machines running in linear time. We now give an outline of the motivation for this investigation and of the results we present. (Figure 1 summarizes the models we are going to discuss and their relationships.)

A natural question concerning models that share the same computational power is the comparison of the sizes of their descriptions. In this respect, one could be interested in comparing one-tape Turing machines running in linear time with equivalent finite automata.

Here, we prove that there exists no recursive function bounding the size

nonregular language recognition, have been stated in the literature [5, 6, 7, 8, 9].

blowup resulting from the conversion from one-tape Turing machines running in linear time into equivalent finite automata. Hence, one-tape linear-time Turing machines can be arbitrarily more succinct than equivalent finite automata. Furthermore, the problem of deciding whether a given one-tape Turing machine runs in linear time is undecidable, in general.[4] These results remain true in the restricted case of *end-marked machines*—namely, one-tape deterministic Turing machines that do not have any extra space, other than the tape portion that initially contains the input. Deterministic end-marked machines running in linear time will be called *Hennie machines*.

To overcome the above-mentioned "negative" results, we consider a syntactical restriction on one-tape deterministic Turing machines, thus introducing *weight-reducing Turing machines*. This restriction aims to force the machine to run in linear time, by making any tape cell rewriting decreasing according to some fixed partial order on the tape alphabet. However, due to the unrestricted amount of available tape space, these devices can have non-halting computations. Nevertheless, they run in linear time as long as they are halting. Indeed, we prove that each computation either halts within a time that is linear in the input length, or is infinite. In the paper we show that it is possible to decide whether a weight-reducing Turing machine is halting. As a consequence, it is also possible to decide whether such a machine runs in linear time. Furthermore, with a polynomial size increase, any such machine can be made halting, and hence forced to run in linear time. Our main result is that the tight size cost of converting a weight-reducing

[4]For the sake of completeness, we mention that it is decidable whether a machine makes at most $cm + d$ steps on input of length m, for any *fixed* $c, d > 0$ [11].

Turing machine into a DFA is double exponential. This cost reduces to a simple exponential when the target device is an NFA.

Considering end-marked Turing machines satisfying the weight-reducing syntactic restriction, we obtain *weight-reducing Hennie machines*. These devices do not allow infinite computations, and hence always run in linear time. The above-stated double exponential blowup is easily extended to them.

The paper is organized as follows. Section 2 presents the fundamental notions and definitions, including those related to the computational models we are interested in. Section 3 is devoted to prove the above-mentioned undecidability and non-recursive trade-off results concerning Hennie machines. In Section 4, after proving that it can be decided if a deterministic Turing machine is weight-reducing, we describe a procedure that, given a linear-time machine together with the coefficient of its linear bound on time, turns it into an equivalent weight-reducing machine. Furthermore, we present a simulation of weight-reducing machines by finite automata, studying its size cost. In Section 5 we show how to decide if a weight-reducing machine halts on any input and if it runs in linear time. We also prove that with a polynomial increase in size, each weight-reducing machine can be transformed into an equivalent one that always halts and runs in linear time.

2. Preliminaries

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

We assume the reader familiar with notions from formal languages and

automata theory (see, e.g., [12]). For a set S, its cardinality is denoted by $\#S$ and 2^S is the family of all its subsets. Given an alphabet Σ, the length of a string $w \in \Sigma^*$ is denoted $|w|$, the i-th symbol of w is denoted w_i, $i = 1, \ldots, |w|$, and ε denotes the empty string.

The 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 *tape alphabet* including symbols of Σ and the special *blank symbol*, denoted by $\not b$, with $\not b \notin \Sigma$, 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
- $\delta : Q \times \Gamma \to Q \times (\Gamma \setminus \{\not b\}) \times \{-1, +1\}$ is the partial *deterministic transition function*.[5]

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

[5]For technical reasons, our Turing machines cannot write the blank symbol. Hence, when the machine scans a blank, it overwrites the contents of the cell with another symbol in $\Gamma \setminus \{\not b\}$. It is possible to notice that these machines have the same computational power as machines that can write the blank symbol on the tape. In fact, each machine that can write blanks can be replaced by a machine that writes an auxiliary symbol different than blank. In this way it can be proved that the power of these machines does not change.

cell with τ and moves the head one cell to the left or to the right according to $d = -1$ or $d = +1$, respectively, 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 the *initial segment*, and the remaining infinitely many cells contain the blank symbol. The computation on input w starts in the initial state with the head scanning the leftmost symbol of w, if $w \neq \varepsilon$, or a blank tape cell, otherwise. The input is *accepted* if the machine eventually halts in a final state. The language accepted by a DTM $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

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 q, 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 $q\not{b}z$ 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 the device may enter a configuration $z'q'u'$ from a configuration zqu in one step, 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 $q_0 w$. An *accepting configuration* is a halting configuration $zq_f u$ 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,

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid q_0 w \vdash^* z q_f u, \text{ where } q_f \in F \text{ and } z q_f u \text{ is halting}\}.$$

In the paper we consider the following restrictions of DTMs (see Figure 1).

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 leave the portion of the tape that initially contains the input. Moreover, $\triangleright$ and $\triangleleft$ can never be written by the machine. Formally, for each transition $\delta(p, \sigma) = (q, \tau, d)$, $\tau = \triangleright$ (resp., $\tau = \triangleleft$) if and only if $\sigma = \triangleright$ (resp., $\sigma = \triangleleft$); furthermore, this implies $d = +1$ (resp., $d = -1$). For end-marked machines, the initial configuration on input w is $q_0 \triangleright w \triangleleft$. We point out that end-marked machines are the deterministic restriction of the well-known *linear-bounded automata* [3]. Notice that any machine without endmarkers can simulate an end-marked machine by overwriting the first blank to the left and to the right of the input word with special symbols representing the left and the right endmarker, respectively. Hence, end-marked machines can be seen as particular cases of machines without endmarkers.

Weight-reducing Turing machines. A DTM is *weight-reducing* (*wr*DTM), if there exists a partial order $<$ on Γ such that each rewriting is decreasing, i.e., $\delta(p, \sigma) = (q, \tau, d)$ implies $\tau < \sigma$, for $\sigma, \tau \notin \{\triangleright, \triangleleft\}$. By this condition, in a wrDTM the number of visits to each tape cell is

bounded by a constant. However, a wrDTM could have non-halting computations that therefore must visit infinitely many tape cells.

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

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

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

We also consider finite automata. We briefly recall their definition. A *nondeterministic finite automaton* (NFA) is a computational device equipped with a finite control and a finite read-only tape that is scanned by an input head in a one-way fashion. Formally, it 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

- $\delta : Q \times \Sigma \to 2^Q$ 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 one position to the right. 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 with the head to the right of the rightmost input symbol. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. An NFA $\mathcal{A}$ is said to be *deterministic* (DFA) whenever $\#\delta(q, \sigma) \leq 1$, for any $q \in Q$ and $\sigma \in \Sigma$.[6]

The notions of configurations, successors, computations, and halting configurations, previously introduced in the context of DTMs, naturally transfer to NFAs.

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 tape symbols. More precisely, the device is fully represented by its transition function. Hence, having a suitable encoding of the sets Q and Γ, since each element of a set A can be represented in size $\Theta(\log \#A)$, to specify the value $\delta(q, \sigma)$, for $q \in Q$, $\sigma \in \Gamma$, size $\Theta(\log(\#Q \cdot \#\Gamma))$ can be used. So, since we have to specify the image of each element of the domain $Q \times \Gamma$, the size of the machine is $\Theta(\#Q \cdot \#\Gamma \cdot \log(\#Q \cdot \#\Gamma))$. With the same argument, the size of a DFA is $\Theta(\#\Sigma \cdot \#Q \cdot \log \#Q)$.

[6]Note that we allow DFAs to have partial transition functions.

3. Hennie Machines: Undecidability and Non-Recursive Trade-Offs

In this section we investigate some basic properties of DTMs. First of all, we prove that it cannot be decided whether an end-marked DTM runs in linear time. As a consequence, it cannot be decided if a DTM is a Hennie machine. Since linear-time DTMs accept only regular languages [4], it is natural to investigate the size cost of their conversion into equivalent finite automata. Even in the restricted case of deterministic Hennie machines we obtain a "negative" result, by proving a non-recursive trade-off between the size of Hennie machines and that of the equivalent finite automata.

Let us start by proving the following undecidability result.

Theorem 1. *It is undecidable whether an end-marked* DTM *runs in linear time.*

Proof. We show that the problem of deciding if a DTM $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ with a semi-infinite tape halts on the empty word ε reduces to this problem. From $\mathcal{T}$ we can construct an end-marked Turing machine $\mathcal{H}$ with input alphabet $\{a\}$ as follows. Given an input $v \in a^*$, the device $\mathcal{H}$ starts to simulate $\mathcal{T}$ over ε; namely, the transitions of $\mathcal{T}$ on the blank symbol are simulated by moves of $\mathcal{H}$ on the symbol a. The portion of the tape storing the word v corresponds to the maximum amount of space that $\mathcal{H}$ is allowed to use for the simulation of $\mathcal{T}$. If, during the simulation, $\mathcal{H}$ reaches the right endmarker, then it stops the simulation and performs additional $\Theta(|v|^2)$ computation steps.[7] Otherwise, $\mathcal{H}$ continues the simulation of $\mathcal{T}$ and halts if $\mathcal{T}$ halts.

[7] This can be achieved, for example, by a sequence of steps that overwrites every tape cell

One can verify that the construction yields the following properties:

- If $\mathcal{T}$ halts on ε in t steps visiting s tape cells, then $\mathcal{H}$ performs $O(t)$ computation steps on any input of length greater than s, while it performs $O(t^2)$ steps on shorter inputs. In both cases, the time is bounded by a constant in the input length.

- If $\mathcal{T}$ does not halt on ε, then for any input v either the simulation reaches the right endmarker and then $\mathcal{H}$ performs further $\Theta(|v|^2)$ computation steps, or it does not halt because $\mathcal{T}$ enters an infinite loop, without reaching such a tape cell. In both cases $\mathcal{H}$ is not a linear-time DTM.

This allows to conclude that $\mathcal{H}$ is a linear-time DTM if and only if $\mathcal{T}$ halts on input ε, which is known to be undecidable. $\square$

We now show that the size trade-off from linear-time DTM to finite automata is not recursive. More precisely, we obtain a non-recursive trade-off between the sizes of Hennie machines and finite automata.

Theorem 2. *For any recursive function f and arbitrarily large integers n,*

---

of the initial segment with a special marker $\sharp \notin \Gamma$, moving the head to the right endmarker after each rewriting. More precisely, from the cell containing the right endmarker, $\mathcal{H}$ moves its head to the preceding cell and overwrites the contents with the special marker. After that, it moves its head to the right endmarker, then moves it backward to the rightmost cell not containing the special marker and writes $\sharp$, thus repeating the procedure until all tape cells but those containing the endmarkers have been overwritten with the special marker.

there is a DHM *with $O(n)$ states and a fixed tape alphabet whose equivalent finite automata require at least $f(n)$ states.*

Proof. We recall that a busy beaver is, among all possible n-state deterministic Turing machines with a two-symbol tape alphabet that start the computation over a blank tape, one that halts after writing on the tape the maximum possible number $\Sigma(n)$ of (not necessarily consecutive) 1's. The function $\Sigma(n)$ is known to be not bounded by any recursive function, i.e., it grows asymptotically faster than any computable function [13].

Here we consider a modification of the busy beaver that operates on a semi-infinite tape (instead of bi-infinite) and starts the computation on the leftmost cell, according to [14]. This variant defines a different function $\Sigma(n)$ that is also not bounded by any recursive function. As a consequence, the amount of the space $s(n)$ used by this modified version of an n-state busy beaver is also not bounded by any recursive function.

For each $n > 0$, let $w_n = a^{s(n)}$ and let $L_n = \{w_n\}$. This language is accepted by an end-marked DTM $\mathcal{H}_n$ with $O(n)$ states and $O(1)$ tape symbols that simulates a given n-state busy beaver (n-BB) and accepts an input $w \in a^*$ if and only if the space used by n-BB equals $|w|$. The DTM $\mathcal{H}_n$ simulates the moves of n-BB on $\not\!\!b$ with moves on a. When n-BB uses more than $|w|$ space, at some point during the simulation the right endmarker is reached. At that point the simulation is aborted and the machine rejects. Furthermore, the simulation of n-BB does not depend on the input. Hence, it is made in constant time. This allows to conclude that, with respect to the input length, $\mathcal{H}_n$ runs in linear time, so it is a Hennie machine.

By summarizing, L_n is accepted by a DHM with $O(n)$ states, so it is of

size $O(n \log n)$. On the other hand, since the only string in L_n is $a^{s(n)}$, the minimum DFA accepting it requires $s(n)+1$ states, so its size is $\Theta(s(n) \log s(n))$. Since $s(n)$ is not bounded by any recursive function, this completes the proof.

□

4. Weight-Reducing Machines: Decidability, Expressiveness and Descriptional Complexity

In Section 3 we proved that it cannot be decided whether an end-marked DTM runs in linear time. In this section we show that it is possible to decide whether DTMs are weight reducing. Furthermore, every linear-time DTM $\mathcal{T}$ with the length of each computation bounded by $Kn + C$, where K, C are constants and n denotes the input length, can be transformed into an equivalent weight-reducing machine whose size is bounded by a recursive function of K and the size of $\mathcal{T}$.

We also present a simulation of weight-reducing machines by finite automata, thus concluding that weight-reducing machines express exactly the class of regular languages. From such a simulation, we will obtain the size trade-off between weight-reducing machines and finite automata that, hence, is recursive. This contrasts with the non-recursive trade-off from Hennie machines to finite automata, proved in Section 3.

Proposition 1. *It is decidable whether a* DTM *is weight-reducing.*

Proof. Let $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ be a DTM. To decide if there is any order $<$ on Γ proving that $\mathcal{T}$ is weight-reducing, it suffices to check whether

the directed graph $G = \langle \Gamma, E \rangle$, with

$$E = \{(\tau, \sigma) \mid \exists p, q \in Q \ \exists d \in \{-1, +1\} : \delta(p, \sigma) = (q, \tau, d)\},$$

is acyclic (each topological ordering of G acts as the required order $<$). $\square$

We now study how linear-time DTMs can be made weight-reducing. To this end, we use the fact that each DTM running in linear time makes a constant number of visits to each tape cell. This property is stated in the following lemma, which derives from [4, Proof of Theorem 3].

Lemma 1. *Let $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ be a DTM. If there exist two constants K and C such that every computation of $\mathcal{T}$ has length bounded by $Kn + C$, where n denotes the input length, then $\mathcal{T}$ never visits a tape cell more than $2K \cdot (\#Q)^K + K$ times.*

The following lemma, which will be used in this section to study trade-offs between the computational models we are investigating and finite automata, presents a transformation from linear-time Turing and Hennie machines into equivalent weight-reducing ones.

Lemma 2. *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* wr*DTM $\mathcal{A}$ accepting $L(\mathcal{T})$ with the same set of states Q as $\mathcal{T}$ and tape alphabet of size $O(k \cdot \#\Gamma)$. Furthermore, on each input $\mathcal{A}$ uses the same space as $\mathcal{T}$, and if $\mathcal{T}$ is linear time or end-marked then so is $\mathcal{A}$.*

Proof. To obtain $\mathcal{A}$, we incorporate a counter into the tape alphabet of $\mathcal{T}$. For each scanned cell, the counter records the maximum number of visits $\mathcal{A}$ can

perform during the remaining computation steps over the cell. More formally, letting $\Sigma_{\not b}$ denote the alphabet $\Sigma \cup \{\not b\}$, we define $\mathcal{A} = \langle Q, \Sigma, \Gamma', \delta', q_0, F\rangle$ with $\Gamma' = \Sigma_{\not b} \cup ((\Gamma \setminus \Sigma_{\not b}) \times \{0, \ldots, k-1\})$ and, for all $q, q' \in Q$, $a, a' \in \Gamma$, $d \in \{-1, +1\}$ where $\delta(q, a) = (q', a', d)$, δ' fulfils the conditions

$$\begin{aligned} \delta'(q, a) &= (q', (a', k-1), d), && \text{if } a \in \Sigma_{\not b}, \\ \delta'(q, (a, i)) &= (q', (a', i-1), d), && \text{otherwise, for } i = 1, \ldots, k-1. \end{aligned}$$

Using an ordering $<$ on Γ' such that

$$\begin{aligned} (a, i) &< b && \text{for } a \in \Gamma \setminus \Sigma_{\not b}, b \in \Sigma_{\not b}, i = 0, \ldots, k-1, \text{ and} \\ (a, i) &< (b, j) && \text{for } a, b \in \Gamma \setminus \Sigma_{\not b}, i, j = 0, \ldots, k-1, i < j, \end{aligned}$$

it is easy to see that $\mathcal{A}$ is a wrDTM equivalent to $\mathcal{T}$. Furthermore, there is a natural bijection between computations of $\mathcal{T}$ and those of $\mathcal{A}$, which preserves time (length of computations) and space (cells visited during the computation). Thus, if $\mathcal{T}$ is linear time then so is $\mathcal{A}$.

In the case $\mathcal{T}$ is end-marked, we can slightly modify the above construction of $\mathcal{A}$, taking into account that, by definition of end-marked devices, the DTM $\mathcal{T}$ does not use the blank symbol and does not overwrite the end-markers. The tape alphabet of $\mathcal{A}$ is $\Gamma' = \Sigma \cup \{\triangleright, \triangleleft\} \cup ((\Gamma \setminus (\Sigma \cup \{\triangleright, \triangleleft\})) \times \{0, \ldots, k-1\})$ and, for all $q, q' \in Q$, $a, a' \in \Gamma$, $d \in \{-1, +1\}$ where $\delta(q, a) = (q', a', d)$, δ' fulfils the conditions

$$\begin{aligned} \delta'(q, a) &= (q', (a', k-1), d), && \text{if } a \in \Sigma, \\ \delta'(q, a) &= (q', a, d), && \text{if } a \in \{\triangleright, \triangleleft\}, \\ \delta'(q, (a, i)) &= (q', (a', i-1), d), && \text{otherwise, for } i = 1, \ldots, k-1. \end{aligned}$$

The ordering $<$ on Γ' is defined as

$$\begin{aligned} (a,i) &< b && \text{for } a \in \Gamma \setminus (\Sigma \cup \{\triangleright, \triangleleft\}), b \in \Sigma, i = 0, \ldots, k-1, \text{ and} \\ (a,i) &< (b,j) && \text{for } a, b \in \Gamma \setminus (\Sigma \cup \{\triangleright, \triangleleft\}), i, j = 0, \ldots, k-1, i < j. \end{aligned}$$

□

By combining the above lemmas, we obtain a procedure to convert linear-time Turing machines into equivalent linear-time weight-reducing machines, as soon as a linear-time bound for the simulated device is explicitly given.

Theorem 3. *Let $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ be a* DTM*. If there exist two constants K and C such that every computation of $\mathcal{T}$ has length bounded by $Kn + C$, where n denotes the input length, then there is an equivalent linear-time* wr*DTM with the same set of states Q as $\mathcal{T}$ and tape alphabet of size $O(k \cdot \#\Gamma)$, where $k = 2K \cdot (\#Q)^K + K$.*

Proof. Direct consequence of Lemmas 1 and 2. □

We now investigate the transformation of weight-reducing machines into equivalent finite automata and its cost.

Theorem 4. *For every* wr*DTM $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ there exist an* NFA *and a* DFA *accepting $L(\mathcal{T})$ with $2^{O(\#\Gamma \cdot \log(\#Q))}$ and $2^{2^{O(\#\Gamma \cdot \log(\#Q))}}$ states, respectively.*

Proof. Assume $\mathcal{T}$ always ends each accepting computation with the head scanning a blank cell to the right of the initial segment. This can be obtained, at the cost of introducing one extra symbol in the tape alphabet,

by modifying the transition function in such a way that when $\mathcal{T}$ enters a final state it starts to move its head to the right, ending when a blank cell is reached. Let $n = \#Q$ and $m = \#\Gamma$.

We describe an NFA $\mathcal{A} = \langle Q', \Sigma, \delta', q_{\mathrm{I}}, F' \rangle$ that accepts $L(\mathcal{T})$, working on the principle of guessing the time-ordered sequences of states in which, in accepting computations, $\mathcal{T}$ scans each of the tape cells storing the input. This is a variant of the classical crossing sequence argument. In this case, for a tape cell C, we consider the sequence of states in which the cell is scanned during a computation, while a crossing sequence is defined as the sequence of states of the machine when the border between two adjacent tape cells is crossed by the head. In addition, in our simulation, together with the sequence of states, the input symbol is also guessed for the next cell.

Suppose that the time-ordered sequence of states in which a cell C is scanned in a computation ρ is $(q_1, \ldots, q_k)$. Due to the weight-reducing property, there are k or $k+1$ different contents of C in ρ, depending on whether the computation stops in q_k. Since the tape alphabet consists of m symbols, we can conclude that $k \leq m$.

The set of states Q' thus consists of a special initial state q_{I}, a special final state q_{F}, and all sequences of the form $(a, q_1, \ldots, q_k)$ where $a \in \Sigma \cup \{\not{b}\}$, $1 \leq k \leq m$, and $q_i \in Q$, for $i = 1, \ldots, k$.

Let $w \in L(\mathcal{T})$ be a non-empty input accepted by $\mathcal{T}$. Let τ_l, τ_{in} and τ_r denote the portion of $\mathcal{T}$'s tape that initially stores the blank symbols preceding w, the input w, and the blank symbols to the right of w, respectively. Let $\rho = (\mathcal{C}_0, \mathcal{C}_1, \ldots)$ be the accepting computation of $\mathcal{T}$ over w. Let $q(j)$ denote the state of $\mathcal{T}$ in configuration $\mathcal{C}_j$. Similarly, let $a(j)$ denote the

symbol scanned by $\mathcal{T}$ in $\mathcal{C}_j$. For a tape cell C in τ_{in}, let $\mathcal{C}_{j_1}, \ldots, \mathcal{C}_{j_k}$, where $j_1 < \cdots < j_k$, be the sequence of all configurations in which $\mathcal{T}$ scans C. Observe that $a(j_1)$ and $q(j_1), \ldots, q(j_{i-1})$ determine $a(j_i)$ for all $i = 2, \ldots, k$. For each configuration $\mathcal{C}_{j_i}$, it is also clear from which direction the head entered C and in which direction it moves out of it ($\mathcal{C}_{j_1}$ is always entered from the left neighboring cell, unless it is the initial configuration; for $i > 1$, $\mathcal{C}_{j_i}$ is entered from the same direction as $\mathcal{C}_{j_{i-1}}$ was left, namely, from the opposite direction than $\mathcal{C}_{j_{i-1}+1}$ was entered). For this reason, we can determine for two neighboring cells C_1 and C_2 of τ_{in} whether two sequences of states assigned to them are consistent with $\mathcal{T}$ in the sense that the rightward head movements outgoing from C_1 have corresponding incoming leftward movements to C_2 and vice versa. Similarly, we can determine whether a sequence of states assigned to the first and last cell of τ_{in} is consistent with the computation of $\mathcal{T}$ performed over the cells of τ_l and τ_r, respectively.

We now formalize these ideas.

Given $(a, q_1, \ldots, q_k), (b, p_1, \ldots, p_\ell) \in Q'$, with $a, b \in \Sigma$, let

- $a_0 = a$ and, for $i = 1, \ldots, k$, $\delta(q_i, a_{i-1}) = (q'_i, a_i, d_i)$, $q'_i \in Q$, $a_i \in \Gamma$, $d_i \in \{-1, +1\}$;
- $b_0 = b$ and, for $i = 1, \ldots, \ell$, $\delta(p_i, b_{i-1}) = (p'_i, b_i, e_i)$, $p'_i \in Q$, $b_i \in \Gamma$, $e_i \in \{-1, +1\}$.

We say that $(b, p_1, \ldots, p_\ell)$ is *consistent* with $(a, q_1, \ldots, q_k)$ when there are indices $i_1, i_2, \ldots, i_t$, $h_1, h_2, \ldots, h_t$, for some odd integer $t \geq 1$, with $1 \leq i_1 \leq i_2 \leq \cdots \leq i_t = k$, $1 = h_1 \leq h_2 \leq \cdots \leq h_t \leq \ell$, such that for $j = 1, \ldots, t$ the following holds:

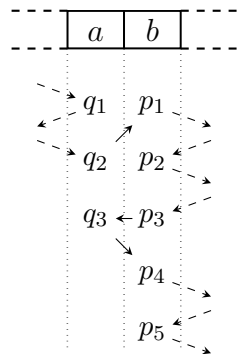

Figure 2: An example where $(b, p_1, \ldots, p_5)$ is consistent with (a, q_1, q_2, q_3). Notice that $t = 3$, $i_1 = 2$, $i_2 = i_3 = 3$, $h_1 = 1$, $h_2 = 3$, $h_3 = 4$.

- if j is odd then $q'_{i_j} = p_{h_j}$, $d_{i_j} = +1$, and, when $j < t$, $i_j < i_{j+1}$,
- if j is even then $p'_{h_j} = q_{i_j}$, $e_{h_j} = -1$, and $h_j < h_{j+1}$,

while $d_i = -1$ for $i \notin \{i_1, \ldots, i_t\}$, and $e_h = +1$ for $h \notin \{h_1, \ldots, h_t\}$.

Notice that, for any odd j, in the transition from q_{i_j} to p_{h_j} the head crosses the border between the two adjacent cells by moving from left to right, while for any even j in the transition from p_{h_i} to q_{j_i} the head crosses the same border by moving in the opposite direction. Furthermore, the conditions t odd, $i_t = k$, $h_1 = 1$, derive from the fact that, in each accepting computation of $\mathcal{T}$, each cell of the initial segment is entered from the left (with the exception of the leftmost one) and is finally exited by moving the head to the right. (See Figure 2 for an example).

In a similar way, we are going to identify the sequences from Q' that can occur on the rightmost cell of the initial segment in an accepting computation (remember that we suppose that when τ reaches a final state it starts to move its head to the right, ending when a blank cell is reached).

Using the above notation, we say that *the blank tape segment is consistent* with $(a, q_1, \ldots, q_k) \in Q'$ when there are indices $1 \leq i_1 < i_2 < \cdots < i_t = k$, and strings $\gamma_0 = \varepsilon, \gamma_1, \ldots, \gamma_t \in \Gamma^*$, such that:

- $d_{i_j} = +1$, for $j = 1, \ldots, t$, while $d_i = -1$ for $i \notin \{i_1, \ldots, i_t\}$; namely, the indices i_j correspond to transitions moving the head to the right.

- For $j = 1, \ldots, t-1$, $\mathcal{T}$ in the state q_{i_j} with the head scanning a tape cell C containing a_{i_j-1} and the string γ_{j-1} written on the non-blank cells to the right of C, moves to the right and makes a finite sequence of moves, which ends when the cell C is re-entered. At this point the state q_{i_j+1} and the string written on the non-blank cells to the right of C is γ_j.

- From $q_{i_t} = q_k$ the machine $\mathcal{T}$ moves its head to the right and, at some point, reaches a blank cell in a final state, without re-entering the cell C in between.

Let Q'_R denote the set of states $(a, q_1, \ldots, q_k) \in Q'$ such that the blank tape segment is consistent with $(a, q_1, \ldots, q_k)$.

Finally, we now identify the sequences from Q' that are consistent with τ_l, namely sequences that could occur, in accepting computations, on the tape cell that initially contains the leftmost input symbol.

Given $(a, q_1, \ldots, q_k) \in Q'$, with $a \in \Sigma$ and, as before, $a_0 = a$ and, for $i = 1, \ldots, k$, $\delta(q_i, a_{i-1}) = (q'_i, a_i, d_i)$, $q'_i \in Q$, $a_i \in \Gamma$, $d_i \in \{-1, +1\}$, let $1 \leq i_1 < i_2 < \cdots < i_t < k$ be the indices corresponding to transitions moving the head to the left, i.e., $\{i_1, i_2, \ldots, i_t\} = \{i \mid d_i = -1\}$. We say

that $(a, q_1, \ldots, q_k)$ is *consistent with the blank tape segment* when there are strings $\gamma_0 = \varepsilon, \gamma_1, \ldots, \gamma_t \in \Gamma^*$, such that:

- q_1 coincides with the initial state of $\mathcal{T}$,
- for $j = 1, \ldots, t$, $\mathcal{T}$ in the state q_{i_j} with the head scanning a tape cell C containing a_{i_j-1} and the string γ_{j-1} written on the non-blank cells to the left of C, moves to the left and makes a finite sequence of moves, up to re-enter C. At that point the state is q_{i_j+1} and the string written on the non-blank cells to the left of C is γ_j.

Let Q'_L denote the set of states from Q' that are consistent with the blank tape segment.

At this point we have developed all the tools to define the automaton $\mathcal{A} = \langle Q', \Sigma, \delta', q_{\mathrm{I}}, F' \rangle$.

- The set of states is

  $$Q' = \{q_{\mathrm{I}}, q_{\mathrm{F}}\} \cup \{(a, q_1, \ldots, q_k) \mid a \in \Sigma \cup \{\not b\}, 1 \leq k \leq m+1, q_i \in Q, i = 1, \ldots, k\}$$

  as already mentioned.

- The initial state is q_{I}.
- The transition function is defined, for $(a, q_1, \ldots, q_k) \in Q'$, $c \in \Sigma$, as

follows:

$$\delta'((a, q_1, \ldots, q_k), c) = \begin{cases} \{(b, p_1, \ldots, p_\ell) \mid (b, p_1, \ldots, p_\ell) \text{ is} \\ \quad \text{consistent with } (a, q_1, \ldots, q_k)\}, & \text{if } c = a \text{ and } (a, q_1, \ldots, q_k) \notin Q'_R \\ \{(b, p_1, \ldots, p_\ell) \mid (b, p_1, \ldots, p_\ell) \text{ is} \\ \quad \text{consistent with } (a, q_1, \ldots, q_k)\} \cup \{q_{\mathrm{F}}\}, & \text{if } c = a \text{ and } (a, q_1, \ldots, q_k) \in Q'_R \\ \emptyset, & \text{otherwise.} \end{cases}$$

The transitions from the initial state q_{I} are defined, for $a \in \Sigma$, as follows:

$$\delta'(q_{\mathrm{I}}, a) = \bigcup_{(a, q_1, \ldots, q_k) \in Q'_L} \delta'((a, q_1, \ldots, q_k), a)\,,$$

while there are no transitions from q_{F}; namely $\delta'(q_{\mathrm{F}}, a) = \emptyset$, for $a \in \Sigma$.

- The set of final states is

$$F' = \begin{cases} \{q_{\mathrm{F}}\} & \text{if } \varepsilon \notin L(\mathcal{T}) \\ \{q_{\mathrm{I}}, q_{\mathrm{F}}\}, & \text{otherwise.} \end{cases}$$

By summarizing, the NFA $\mathcal{A}$ simulates $\mathcal{T}$ as follows:

- In the initial state q_{I}, reading an input symbol a, the automaton $\mathcal{A}$ implicitly guesses a sequence $(a, q_1, \ldots, q_k) \in Q'_L$, i.e., a sequence consistent with the left tape segment, and a sequence $(b, p_1, \ldots, p_\ell)$ consistent with $(a, q_1, \ldots, q_k)$, where b is supposed to be the symbol in the cell immediately to the right. If $(a, q_1, \ldots, q_k) \in Q'_R$ is consistent with the right blank segment, then the one-symbol string a is accepted by $\mathcal{T}$. Hence in this case the final state q_{F} can be also guessed.

- When scanning an input cell containing a symbol a, in a state $(a, q_1, \ldots, q_k) \in Q'$, the machine $\mathcal{A}$ guesses a sequence $(b, p_1, \ldots, p_\ell)$ consistent with it. If the next input symbol is b, then the simulation can continue in the same way, otherwise it stops because of an undefined transition.

  Furthermore, when $(a, q_1, \ldots, q_k) \in Q'_R$, the device $\mathcal{A}$ can also guess to have reached the last input symbol, so entering the final state q_{F}. If the end of the input is effectively reached then $\mathcal{A}$ accepts.

The number of states of $\mathcal{A}$ is $2 + (\#\Sigma + 1)\sum_{i=1}^{m} n^i = 2^{O(m \log n)}$. If $\mathcal{A}$ is in turn transformed to an equivalent DFA, using the classical powerset construction, the resulting automaton has $2^{2^{O(m \log n)}}$ states. □

As a direct consequence of Theorem 4, we get that `wr`DTMs recognize exactly the class of regular languages.

Corollary 1. *A language is regular if and only if it is accepted by some* `wr`DTM.

Theorem 4 gives a double exponential upper bound for the size cost of the simulation of `wr`DTMs by DFAs. We now also prove a double exponential lower bound.

To this end, for each integer $n \geq 0$, we consider the language B_n over $\{0, 1, \$\}$ consisting of strings $v_1\$v_2\$\cdots\$v_k$, where $k > 2$, $v_1, v_2, \ldots, v_k \in \{0,1\}^*$, $|v_k| \leq n$, $|v_i| \geq |v_k|$ for $i = 1, \ldots, k-1$, and there exists $j < k$ such that $v_j = v_k$. Informally, every string in B_n is a sequence of binary blocks separated by the symbol \$, where the last block is of length at most n and it is a copy of one of the preceding blocks, which all are at least as long as the last one. For example,

$$v_1\$v_2\$v_3\$v_4\$v_5\$v_6 = 0011\$0101110\$011\$0011\$001\$011 \in B_4$$

since $|v_6| \leq 4$, $|v_i| \geq |v_6|$ for $i = 1, \ldots, 5$, and $v_3 = v_6$.

Lemma 3. *For every integer $n \geq 0$, the language B_n is accepted by a* `wr`DHM *with $O(1)$ states and $O(n)$ tape symbols.*

Proof. Let $\Sigma = \{0, 1, \$\}$. We first describe an end-marked DTM $\mathcal{T}$ accepting the union of all B_i's, for $i \geq 0$; then we show how $\mathcal{T}$ can be modified in order to recognize B_n, for any fixed integer n, by bounding the number of visits to each cell, thus obtaining a `wr`DHM with the desired properties. Let us define the tape alphabet of $\mathcal{T}$ as $\Gamma = \{0, 1, \$, x, f, \not{b}\}$.

Let $w \in \Sigma^*$ be an input string of the form $w = v_1\$v_2\$\cdots\$v_k$, where $v_1, \ldots, v_k \in \{0, 1\}^*$, and $v_k = a_1 \cdots a_\ell$, with $a_i \in \{0, 1\}$ for $i = 1, \ldots, \ell$. The machine $\mathcal{T}$ performs ℓ iterations. In each iteration it moves the head from the left endmarker to the right endmarker and back, thus visiting each input cell twice. It also rewrites some of the tape cells during this movement. The aim of the i-th iteration is comparing the i-th rightmost symbol of the last block with the i-th rightmost symbol of any other block. This is implemented as follows. Within the first iteration, $\mathcal{T}$ memorizes a_ℓ in the states, rewrites it by the symbol x, and moves the head leftwards. Whenever it encounters the symbol \$ and enters the right end of a block v_j, it checks if its last symbol equals a_ℓ. If so, $\mathcal{T}$ overwrites the cell contents with x; otherwise it writes f. During the i-th iteration, $\mathcal{T}$ memorizes $a_{\ell+1-i}$ (which is in the rightmost input cell not containing the symbol x) in its finite control, overwrites the cell containing it by x and checks whether the i-th rightmost symbol of each v_j, with $j < k$, matches $a_{\ell+1-i}$ (if so, it overwrites the symbol with x; if not it writes f). Notice that, at the beginning of the i-th iteration, $i > 1$, the i-th

rightmost symbol of a block is located immediately to the left of a nonempty factor consisting only of the symbols x and f. However, it could happen that for some $j < k$ there is no i-th rightmost symbol in the factor v_j—namely the block v_j is shorter than v_k. In this case the machine halts and rejects. The input w is accepted by $\mathcal{T}$ if and only if, after some iteration, all symbols of v_k have been overwritten with x and there is some v_j with all symbols also rewritten to x (this ensures $v_j = v_k$).

It can be noticed that, for any fixed integer n, a word belongs to B_n if and only if it is accepted by $\mathcal{T}$ within the first n iterations. Hence, as each iteration yields exactly two visits to each input cell, by bounding the number of visits to each cell by $2n$, we can restrict $\mathcal{T}$ to accept words from B_n only. This can be obtained by using a construction similar as those used for proving Lemma 2. We thus obtain a halting `wr`DHM $\mathcal{H}$ accepting B_n that has the same number of states as $\mathcal{T}$, hence a number that does not depend on n, and $O(n)$ tape symbols. □

Lemma 4. *Each* DFA *accepting B_n has at least 2^{2^n} states.*

Proof. Let $\mathcal{S}$ be the family of all subsets of $\{0,1\}^n$. Given a subset $S = \{w_1, \ldots, w_k\} \in \mathcal{S}$, where $w_1 < \cdots < w_k$ in the lexicographical order, consider the string $w(S) = w_1\$w_2\$\cdots\$w_k$. Let S_1 and S_2 be two different elements of $\mathcal{S}$ and let $u \in \{0,1\}^n$ be a string that is in S_1 but not in S_2 (or vice versa). Then, $w(S_1)\$u \in B_n$ and $w(S_2)\$u \notin B_n$ (or vice versa), hence $\$u$ is a distinguishing extension. Therefore, the set $w(S)$, such that $S \in \mathcal{S}$, is a set of pairwise distinguishingable strings. By the Myhill-Nerode Theorem, each

DFA accepting B_n has at least as many states as the cardinality of such a set. Hence, it is $\#\mathcal{S} = 2^{2^n}$. □

From Theorem 4 and Lemmas 3 and 4, we obtain that the size trade-offs from wrDTMs and wrDHMs to DFAs are double exponential:

Corollary 2. *For every* wr DTM *(resp.,* wr DHM*)* $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ *there exists a* DFA *accepting* $L(\mathcal{T})$ *with* $2^{2^{O(\#\Gamma\cdot\log(\#Q))}}$ *states. In the worst case, this double exponential gap is also necessary.*

As shown in Theorem 2, by dropping the weight-reducing assumption for machines, the size trade-offs in Corollary 2 become not recursive. However, provided an explicit linear bound on computation lengths, we obtain the following result. Note that Corollary 3 is stated for Turing machines, and since this is the most general model among the ones we studied (*cf.* Figure 1), the same holds for Hennie machines as well.

Corollary 3. *Let* $\mathcal{T} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ *be a* DTM*. If there exist two constants* K *and* C *such that every computation of* $\mathcal{T}$ *has length bounded by* $Kn + C$*, where* n *denotes the input length, then there exist an* NFA *and a* DFA *accepting* $L(\mathcal{T})$ *with* $2^{O(k\cdot\#\Gamma\cdot\log(\#Q))}$ *and* $2^{2^{O(k\cdot\#\Gamma\cdot\log(\#Q))}}$ *states, respectively, where* $k = 2K\cdot(\#Q)^K + K$*.*

Proof. Consequence of Theorems 3 and 4. □

5. Weight-Reducing Machines: Space and Time Usage, Haltingness

Weight-reducing Turing machines generalize weight-reducing end-marked Turing machines (that are necessarily Hennie machines) by allowing the use

of additional tape cells that initially do not contain the input and to which we refer as *initially-blank cells*. In particular, this extension allows infinite computations. For instance, a `wr`DTM can perform forward moves forever, rewriting each blank cell with some symbol. We now show that, however, due to the weight-reducing property, the number of initially-blank cells that are really useful, i.e., that is visited in halting computations, is bounded by some constant that can be computed from the size of the `wr`DTM and does not depend on the input string. This allows us to transform any `wr`DTM into an equivalent halting machine of polynomial size, which therefore operates in linear time. Notice that Theorem 4 already gave a simulation of `wr`DTMs by a halting and linear-time computational model.

Lemma 5. *Let be $\mathcal{T}$ a* `wr`DTM*. If in a computation of $\mathcal{T}$ two initially-blank tape cells, both located to the right of the initial segment, are visited in the same sequence of states, then the computation is infinite. The same result holds if the two cells are both located to the left of the initial segment.*

Proof. For ease of exposition, we index the cell positions by integers, starting with the leftmost cell of the initial segment, whose index is 1, and we identify each cell with its position. Let us fix the computation of $\mathcal{T}$ on a given input.

The rough idea is that, due to the fact that $\mathcal{T}$ is deterministic, given a tape cell c to the right of the initial segment, the parts of computation that visit the cells to the right of c are completely determined by the sequence of states $q_0, q_1, \ldots, q_{k-1}$ in which c is visited. Hence, for each $h > 0$, the sequence of states in which any cell $c+h$ is visited also only depends on $q_0, q_1, \ldots, q_{k-1}$, and h. As a consequence, if for some h the cell $c + h$ is visited in the same

sequence of states $q_0, q_1, \ldots, q_{k-1}$ in which c is visited, then even the cell $c+2h$ is visited in the same sequence of states, and so on, thus implying that the computation is infinite.

Going more into details, suppose that the computation visits the tape cell c in configurations $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{k-1}$, $k \geq 1$. Let q_i and γ_i be the state and the non-blank content of the right part of the tape that starts from cell c, respectively, in configuration $\mathcal{C}_i$, $i = 0, \ldots, k-1$. We observe that, since c is located to the right of the initial segment, $\gamma_0 = \varepsilon$ and, for $i = 1, \ldots, k-1$, due to the fact that $\mathcal{T}$ is deterministic, q_i and γ_i are completely determined by q_{i-1} and γ_{i-1}, thus implying that, being γ_0 fixed, q_i and γ_i depend only on $q_0, q_1, \ldots, q_{i-1}$.[8]

Let us now consider a cell $c+h$ for $h > 0$. Let P be the sequence of states in which this cell is visited in the computation under consideration. Due to the fact that $\mathcal{T}$ is weight reducing, P is finite. Our aim is to prove that P only depends on $q_0, q_1, \ldots, q_{k-1}$, and h.

Since the cell $c + h$ cannot be visited before the cell c, namely before configuration $\mathcal{C}_0$, the sequence P can be decomposed as $P = P_1 P_2 \cdots P_k$ where, for $i = 1, \ldots, k$, P_i is the sequence of states that are reached when the head is visiting the cell $c + h$ after the configuration $\mathcal{C}_{i-1}$ and, for $i < k$, before the configuration $\mathcal{C}_i$.

Again using the fact that $\mathcal{T}$ is deterministic, we observe that P_i is completely determined by q_{i-1}, γ_{i-1}, and h. Since, in turn, γ_{i-1} is completely determined by $q_0, q_1, \ldots, q_{i-1}$, we conclude that P_i only depends on $q_0, q_1, \ldots, q_{i-1}$,

[8]Notice that if the part of computation between $\mathcal{C}_{i-1}$ and $\mathcal{C}_i$ only visits cells to the left of c, then γ_{i-1} and γ_i differ only in the first symbol, namely the symbol in the cell c.

and h. This allows us to conclude that the sequence $P = P_1 P_2 \cdots P_k$ of states reached at the cell $c+h$ depends only on the sequence of states $q_0, q_1, \ldots, q_{k-1}$ reached at the cell c and on h, as we claimed.

Suppose now that P coincides with the sequence $q_0, q_1, \ldots, q_{k-1}$ of states visited at the cell c. By iterating the previous argument, the sequence of states that are reached in any cell $c + hj$, $j > 0$, is $q_0, q_1, \ldots, q_{k-1}$, thus implying that the computation is infinite. This completes the proof when there are two cells that are visited in the same sequence of states and are both located to the right of the initial segment. With a similar argument, the result can be proved if the two cells are both located to the left of the initial segment. □

Lemma 6. *Let $\mathcal{T}$ be an n-state* wr*DTM that uses g tape symbols. A computation of $\mathcal{T}$ is infinite if and only if it visits at least $(n+1)^g$ consecutive initially-blank cells, i.e., tape cells to the left or to the right of the initial segment.*

Proof. Since $\mathcal{T}$ is weight reducing, the number of visits to each tape cell is bounded by a constant that, in turn, is bounded by g. Thus, each infinite computation must visit infinitely many tape cells, hence at least $(n+1)^g$ consecutive initially-blank cells.

To prove the converse, let us consider a halting computation ρ of $\mathcal{T}$ over an input word of length ℓ. By Lemma 5, ρ cannot visit two tape cells, lying at the same side of the initial segment, in the same sequence of states. Since there are less than $(n+1)^g$ nonempty distinct sequences of states of length

at most g, we conclude that the number of consecutive initially-blank cells visited during the computation is less than $(n+1)^g$. $\square$

As a consequence of the above result, we obtain the following dichotomy of computations of wrDTMs.

Proposition 2. *Let $\mathcal{T}$ be an n-state wrDTM that uses g tape symbols and let ρ be a computation of $\mathcal{T}$. Then, either ρ consists of a number of steps that is linear in the input length and it visits less than $(n+1)^g$ initially-blank cells to the left and less than $(n+1)^g$ initially-blank cells to the right of the initial segment, or ρ is infinite and visits infinitely many tape cells.*

Proof. If the amount of tape cells visited by ρ is k, for some finite k, then, as a cell cannot be visited more than g times by the weight-reducing property, ρ is finite and has length bounded by gk. Furthemore, by Lemma 6, ρ visits less than $(n+1)^g$ initially-blank cells to the left (resp., to the right) of the initial segment. Thus, $k < 2(n+1)^g - 1 + |w|$, where w is the input. Hence, the total number of steps of ρ is bounded by $gk < g(2(n+1)^g - 1 + |w|) \in O(|w|)$, i.e., it is linear in the length of the input. Conversely, if ρ visits infinitely many tape cells then it is necessarily infinite. $\square$

Proposition 3. *By a polynomial size increase, each wrDTM can be transformed into an equivalent linear-time wrDTM.*

Proof. From an n-state wrDTM $\mathcal{T}$ with a tape alphabet of cardinality g, we can build an equivalent halting wrDTM $\mathcal{T}'$ that works as follows. After an initial phase during which $\mathcal{T}'$ marks $(n+1)^g$ initially-blank cells to the left

and to the right of the initial segment, it performs a direct simulation of $\mathcal{T}$ while ensuring that no further cells than those initially marked and those of the initial segment are used.

The initial phase is implemented using a counter in basis $(n+1)$, stored on g consecutive tape cells, which is incremented up to $(n+1)^g$ and shifted along the tape. We shall describe a procedure marking the $(n+1)^g$ cells to the left of the initial segment. A similar procedure is repeated at the right of the initial segment.

At each step, the counter contains the number of marked cells minus one. Hence, at the beginning, it is initialized to value $g-1$ (in basis $n+1$) by writing the corresponding digits onto the g cells immediately to the left of the initial segment, the least significant digit being on the rightmost of these cells. Then the counter is incremented and shifted leftward by updating each digit, from right to left, namely starting from the least significant one. Let d be the digit scanned by the head. The value $d' = d + 1 \mod (n+1)$ is stored in the state control, along with a boolean variable c, for carry propagation. Then the head is moved one cell to the left, d' is written on the tape and its value updated according to the previous contents of the cell just overwritten and the value of c. In the case the previous symbol on such a cell was the blank, then all the g digits have been updated. Hence, the head is moved g positions to the right (on the least significant digit) and the counter is incremented again. This procedure stops when, moving rightward to reach the least significant digit, all the g scanned cells contain the symbol n. We now estimate the number of states used to implement this procedure. First, we notice that $O(g)$ states are sufficient to initialize

the g cells to the left of the initial segment and to move the head from the most to the least significant digit of the counter, before each increment. Also checking if all the g cells of the counter contain the symbol n, i.e., if the counter has reached the maximum value, can be done within the same state bound. During the phase that increments and shifts the counter, it is enough to remember the above-mentioned values d' and c; it is not necessary to count the positions in the counter because, as explained above, the end of this phase is recognized when a cell containing $\not{b}$ is overwritten. Hence, this phase can be implemented using $O(n)$ states. By summing up, this entire procedure can be implemented using $O(g+n)$ states. We also observe that the number of visits to each initially-blank cell is bounded by $2g$ because every cell is visited at most twice when containing the i-th digit of the current counter value, for $i = 1, \ldots, g$.

Once the space is marked, $\mathcal{T}'$ simulates $\mathcal{T}$, stopping and rejecting if the simulation reaches a blank cell.

From Lemma 6, we can easily conclude that each halting computation of $\mathcal{T}$ is simulated by an equivalent halting computation of $\mathcal{T}'$, while each infinite computation of $\mathcal{T}$ is replaced in $\mathcal{T}'$ by a computation that reaches a blank cell and then stops and rejects.

To complete the proof we have to show how to convert $\mathcal{T}'$ into a weight-reducing equivalent machine of size polynomial in the size of $\mathcal{T}$. We remind the reader that $\mathcal{T}$ has n states and a tape alphabet of g symbols. Since it is weight-reducing, it can visit each tape cell no more than g times. To mark the cells in the initial phase $\mathcal{T}'$ uses $O(g+n)$ states and an alphabet of $n+1$ symbols; hence the total number of states of $\mathcal{T}'$ is $O(g+n)$ and the

total number of symbols is $O(g+n)$. By the previous analysis, we notice that during the initial phase marking the space, each marked cell is visited $2g$ times. Furthemore, since after marking cells to the left of the input, the head is moved to the right until the first blank to mark the portion to the right of the input, and then back to first cell of the input to start the simulation of $\mathcal{T}$, $\mathcal{T}'$ can visit each cell two more times. By summing up, we conclude that in each computation of $\mathcal{T}'$ each tape cell is visited $O(g)$ times. Hence, applying Lemma 2 we can convert $\mathcal{T}'$ into a weight-reducing machine $\mathcal{T}''$, with $O((g+n)g)$ states and a tape alphabet of $O((g+n)g)$ symbols. This completes the proof.

For the sake of completeness, we mention that, with a finer analysis, it can be shown that the size of the tape alphabet of the final machine can be reduced to $O(gn)$ symbols. To this aim, instead of applying Lemma 2 to the whole machine $\mathcal{T}'$, we can directly build $\mathcal{T}'$ introducing only transitions satisfying the weight-reducing property. For the transitions used in the preliminary phase, marking initially-blank cells to the left and to the right of the input, we use an alphabet of $O(gn)$ symbols, encoding suitable counters for visits, together with the $n+1$ digits. Furthermore, the original alphabet of $\mathcal{T}$ is extended with $2g$ symbols that are used for the two extra visits that $\mathcal{T}'$ can make to each tape cell before simulating $\mathcal{T}$. □

Using Lemma 6 and Proposition 3, we prove the following property.

Theorem 5. *It is decidable whether a* `wr`DTM *halts on each input string.*

Proof. From any given `wr`DTM $\mathcal{T}$, we construct a halting `wr`DTM $\mathcal{T}'$ that, besides all the strings accepted by $\mathcal{T}$, accepts all the strings on which $\mathcal{T}$ does

not halt. To this end, we can slightly modify the construction used to prove Proposition 3, in such a way that when the head reaches a blank cell outside the initial segment and the initially marked space, the machine stops and accepts. Hence, the given wrDTM $\mathcal{T}$ halts on each input string if and only if the finite automata that are obtained from $\mathcal{T}$ and $\mathcal{T}'$ according to Theorem 4 are equivalent. □

By Proposition 2, the number of steps of any halting computation of a wrDTM is linear in the input length. Hence, from Theorem 5 we obtain:

Corollary 4. *It is decidable whether a wr*DTM *runs in linear time.*

6. Conclusion

In this work, we investigated deterministic one-tape Turing machines running in linear time. Although these devices are known to be equivalent to finite automata [4], one cannot decide whether a given Turing machine is linear-time even in the case of end-marked devices, as we showed in Theorem 1. Furthermore, there is no recursive function bounding the size blowup of the conversion of DHMs into DFAs (Theorem 2). To avoid these negative results, we introduced a weight-reducing restriction, which forces one-tape Turing machines to run in linear time as long as they halt. Indeed, we proved that each computation of a wrDTM either is infinite or halts within a linear number of steps in the input length (Proposition 2). The weight-reducing restriction is syntactic and can be checked (Proposition 1). Furthermore, we proved in Theorem 4 that each wrDTM can be converted into an NFA (resp., DFA) whose size is exponential (resp., doubly-exponential) with respect to the size of the converted device. These costs are tight (Corollary 2).

Weight-reducing Turing machines are not restrictions of linear-time machines. Indeed, they allow infinite computations. However, whether a wrDTM halts on all inputs can be decided (Theorem 5). Furthermore, with a polynomial increase in size, each weight reducing machine can be made halting and linear-time (Proposition 3). Still, *halting* wrDTMs are not particular DHMs as they allow the use of extra space besides the initial segment contrary to DHMs that are end-marked. We do not know at the time of writing whether this extra space usage is useful for representing regular languages concisely. In other words, we leave open the question of the size cost of turning wrDTMs into equivalent wrDHMs.

In a related paper [15], we investigate the computational models considered here by focusing on the Sakoda and Sipser question about the size cost of the determinization of two-way finite automata [16]. We indeed propose a new approach to this famous open problem, which consists in converting two-way nondeterministic automata into equivalent deterministic extensions of two-way finite automata, paying a polynomial increase in size only. The considered extensions are variants of linear-time deterministic Turing machines, including wrDHMs and DHMs.

References

[1] D. Průša, Weight-reducing Hennie machines and their descriptional complexity, in: A. Dediu, C. Martín-Vide, J. L. Sierra-Rodríguez, B. Truthe (Eds.), Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014.

Proceedings, Vol. 8370 of Lecture Notes in Computer Science, Springer, 2014, pp. 553–564.

[2] B. Guillon, G. Pighizzini, L. Prigioniero, D. Průša, Two-way automata and one-tape machines - read only versus linear time, in: M. Hoshi, S. Seki (Eds.), Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings, Vol. 11088 of Lecture Notes in Computer Science, Springer, 2018, pp. 366–378.

[3] S. Kuroda, Classes of languages and linear-bounded automata, Information and Control 7 (2) (1964) 207–223.

[4] F. C. Hennie, One-tape, off-line Turing machine computations, Information and Control 8 (6) (1965) 553–578.

[5] B. A. Trakhtenbrot, Turing machine computations with logarithmic delay, (in Russian), Algebra I Logica 3 (1964) 33–48.

[6] J. Hartmanis, Computational complexity of one-tape Turing machine computations, J. ACM 15 (2) (1968) 325–339.

[7] P. Michel, An NP-complete language accepted in linear time by a one-tape Turing machine, Theor. Comput. Sci. 85 (1) (1991) 205–212.

[8] G. Pighizzini, Nondeterministic one-tape off-line Turing machines, Journal of Automata, Languages and Combinatorics 14 (1) (2009) 107–124.

[9] K. Tadaki, T. Yamakami, J. C. H. Lin, Theory of one-tape linear-time Turing machines, Theor. Comput. Sci. 411 (1) (2010) 22–43.

[10] D. A. Walters, Deterministic context-sensitive languages: Part II, Inf. Control. 17 (1) (1970) 41–61.

[11] D. Gajser, Verifying time complexity of Turing machines, Theor. Comput. Sci. 600 (2015) 86–97.

[12] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[13] T. Radó, On non-computable functions, Bell System Technical Journal 41 (3) (1962) 877–884.

[14] T. R. S. Walsh, The busy beaver on a one-way infinite tape, SIGACT News 14 (1) (1982) 38–43.

[15] B. Guillon, G. Pighizzini, L. Prigioniero, D. Průša, Converting nondeterministic two-way automata into small deterministic linear-time machines, Information and Computation (2022) 104938.

[16] W. J. Sakoda, M. Sipser, Nondeterminism and the size of two way finite automata, in: R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, A. V. Aho (Eds.), Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA, ACM, 1978, pp. 275–286.