Fundamenta Informaticae 126 (2013) 225–246
DOI 10.3233/FI-2013-879
IOS Press

Two-Way Finite Automata: Old and Recent Results*

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

Abstract. The notion of *two-way automata* was introduced at the very beginning of automata theory. In 1959, Rabin and Scott and, independently, Shepherdson, proved that these models, both in the deterministic and in the nondeterministic versions, have the same power of one-way automata, namely, they characterize the class of *regular languages*. In 1978, Sakoda and Sipser posed the question of the costs, in the number of the states, of the simulations of one-way and two-way nondeterministic automata by two-way deterministic automata. They conjectured that these costs are exponential. In spite of all attempts to solve it, this question is still open. In the last ten years the problem of Sakoda and Sipser was widely reconsidered and many new results related to it have been obtained. In this work we discuss some of them. In particular, we focus on the restriction to the unary case, namely the case of automata defined over the one letter input alphabet, and on the connections with open questions in space complexity.

Keywords: two-way finite automata; descriptional complexity; computational complexity; space complexity

1. Introduction

Finite state automata are usually presented as devices which are able to recognize input strings using a fixed amount of memory, implemented by a finite state control (see, e.g., [17]). The input is written on a read-only tape, which is scanned by an input head. In the basic model the head is moved only from

*A preliminary version of this work was presented at the *18th international workshop on Cellular Automata and Discrete Complex Systems* and *3rd international symposium Journes Automates Cellulaires*, La Marana, Corsica, September 19–21, 2012.

†Address for correspondence: Dipartimento di Informatica, Università degli Studi di Milano, Italia

left to right. For this reason, the model is also called *one-way finite automaton*. It can be defined in the deterministic and the nondeterministic versions (1DFA and 1NFA, respectively). It is well known that both of them share the same recognition power, i.e., they characterize the class of regular languages. However, nondeterministic finite automata can be exponentially smaller. In fact, each n-state 1NFA can be simulated by an equivalent 1DFA with 2^n states and this cost cannot be reduced [32, 36, 38].

What happens if we allow to move the input head in both directions?

In spite of this additional feature, the resulting models, which are called *two-way finite automata*, have the same computational power as one-way automata, i.e., they still characterize the class of regular languages, as independently proved by Rabin and Scott [39] and by Shepherdson [44], at the beginning of automata theory. However, from the point of view of the size (measured in terms of states) the situation is different and, at the moment, we still do not have a complete picture of the relationships between the sizes of different variants of finite automata. More precisely:

- By an analysis of the constructions given in [39, 44], it turns out that the simulations of n-state two-way nondeterministic finite automata (2NFAs, for short) and n-state two-way deterministic finite automata (2DFAs, for short) by 1DFAs can be done with a number of states exponential in a polynomial in n. Furthermore, a lower bound exponential in n follows from the simulation of 1NFAs by 1DFAs.

- The exact bound for the simulation of 2NFAs by 1NFAs has been found in [22].

- *The costs of the simulations of* 1NFA*s by* 2DFA*s and of* 2NFA*s by* 2DFA*s are still unknown.*

The problem concerning the costs at the last point, was raised in 1978 by Sakoda and Sipser [41], who conjectured that these costs are not polynomial. To support such a conjecture, Sakoda and Sipser presented a complete analogy with the P versus NP question, by introducing a notion of reducibility between families of regular languages which allows to obtain families of complete languages for these simulations. (For a detailed discussion on this approach we address the reader to the recent paper by Kapoutsis [24].) In spite of many attempts to solve it, the problem is still open.

In the last decade, several new results related to the Sakoda and Sipser question have been discovered. The aim of this paper is to present and discuss some of them (mainly with respect to the question of 2NFAs versus 2DFAs) together with some older results in this area.

We will keep the presentation at an informal level, trying to avoid, as much as possible, technical details, that can be found in the references. After presenting and discussing, in Section 2, the main features of two-way automata, in Section 3 we presents two simple examples of languages for which the possibility of moving the input head in both directions is very helpful in the recognition process. In particular, for one of those languages we discuss different recognition strategies, which turn out to be useful to illustrate, in Section 4, several restricted versions of two-way automata considered in the literature. This leads to the study of the Sakoda and Sipser question under some restrictions. In particular, in the literature three family of restrictions have been considered:

- *Restrictions on the simulating machines.*
  Exponential separations have been found for the simulations of 1NFAs and 2NFAs by *restricted versions* of 2DFAs. These results are briefly discussed in Subsection 4.1.

- *Restrictions on the class of languages.*
  The case of *unary languages*, namely languages defined over the one letter alphabet, has been considered in the literature, and it is discussed in Section 5. While the simulation of unary 1NFAs by 2DFAs can be done with a quadratic number of states, the problem of the cost of the simulation of 2NFAs by 2DFAs is still open *even in this restricted case*.

- *Restrictions on the simulated machines.*
  Recently, the results concerning the unary case have been extended to the study of the simulation of a *restricted* version of 2NFAs (over arbitrary alphabets) by (unrestricted) 2DFAs. These results will be also mentioned in Section 5.

Important relationships of the Sakoda and Sipser problem with the question of the relationships between determinism and nondeterminism for logarithmic space have been discovered. We will discuss them in Section 6. In particular, an important role in the recent developments in this aspect is given by the investigation of the unary case.

We conclude the paper with some final remarks, in Section 7.

2. Preliminaries

A formal definition of the model we are interested in, namely *two-way finite automata*, would require many technical details, which sometimes are different in the papers published in the literature, but which do not affect the essential features of the model. Hence, with the aim of keeping the presentation at an easy understandable level, we just present an informal description of the model.

We assume that the reader is familiar with standard notions concerning finite state automata, as presented for instance in [17]. We denote by Σ the *input alphabet*, by Σ^* the set of all strings over Σ, and, for each integer $n \geq 0$, by Σ^n the set of strings of length n. The length of a string $w \in \Sigma^*$ will be denoted by $|w|$.

A computation of a one-way automaton starts on the leftmost input symbol in the initial state; at each step the input head is moved one position to the right; the computation ends after reading the rightmost input symbol. For two-way automata slightly different definitions are given in the literature. We skip technical details and we emphasize the main features.

- First of all, we assume that the input string is surrounded on the input tape by two special symbols, $\vdash, \dashv \notin \Sigma$, called, respectively, the *left* and the *right end-marker*. Hence, on input $w \in \Sigma^*$, the tape contains $\vdash w \dashv$.

- To present recognition algorithms, sometimes we need to number input cells. So, we assume that on input w the cells are counted from 0 to $|w| + 1$, where cells 0 and $|w| + 1$ contain the end-markers, and the remaining cells contain "real" input symbols. The input head cannot violate the end-markers.

- The computation starts in a designated initial state with the head scanning the first "real" input symbol, i.e., on cell 1. Sometimes it is more convenient to start from cell 0. It should be clear that this does not significantly change the model.

- To reflect the acceptance condition of one-way automata, we can stipulate that a string is accepted by a two-way automaton if and only if there is a computation which reaches the right end-marker in a final state. However, this condition can be slightly modified by considering acceptance on the left end-marker or just on one of the two end-markers.

  A different possibility is to state that a string is accepted if and only if there is a computation which reaches a final state, regardless the input head position.

  Further variants are possible. It should be clear that all these variants are equivalent. Adding one or two states, we can easily convert a two-way automaton with an acceptance condition into another one with a different acceptance condition. For this reason, here we do not fix any particular acceptance condition.

- The transition function can be defined by allowing only moves to the left and to the right or even by allowing stationary moves, i.e., transitions that keep the head on the same input cell. Even this possibility does not significantly change the model and the number of states.

- When we say that a two-way automaton A has $f(n) + \text{const}$ states, we mean that A has $f(n) + c$ states, where c is a small constant (in all examples $c < 10$ is enough). This constant can slightly change depending on the choice of the initial configuration, of the acceptance condition, and of the possibility of stationary moves.

- A *head reversal* is any change of the input head direction, i.e., a two-way automaton makes one head reversal when, after a sequence of transitions moving the head to the right, it makes a transition moving the head to the left or vice versa. Stationary moves are not taken into account to compute head reversals. For instance, a sequence of two moves to the right, one stationary move, one move to the right, one stationary move again, and one move to the left, presents just one head reversal.

- We point out that two-way automata can enter into infinite loops. In this case the computation is rejecting. This is a relevant difference with one-way automata that, at each step of their computations, move the head one position to the right and, hence, can only have halting computations.

3. Two Examples

Let us start by considering the following family of languages

$$I_n = (a+b)^* a (a+b)^{n-1} ,$$

namely, for each integer $n > 0$, I_n is the set of strings whose nth symbol from the right is an a. This is a classical example used to present an exponential blow-up for the conversion of 1NFAs into equivalent 1DFAs.[1] In particular, for each $n > 0$, we can prove the following:

- The language I_n is accepted by the 1NFA with $n + 1$ states in Figure 1.

[1] The fact that the classical subset construction is state optimal is shown by more sophisticated examples, leading from n-state 1NFAs to equivalent 2^n-state 1DFAs, that are proved to be minimal [32, 36, 38]. The family $(I_n)_{n>0}$ achieves a slightly smaller gap, but it is more simple and easy to understand.

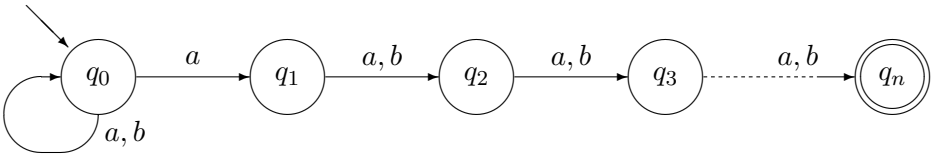

Figure 1. A 1NFA accepting the language $I_n = (a+b)^*a(a+b)^{n-1}$.

- Each 1DFA accepting I_n requires 2^n states. Intuitively, this can be proved by observing that in order to accept the language I_n, a 1DFA needs to remember the last n input symbols. It is a standard exercise to depict a 1DFA matching this lower bound.
- The language I_n is accepted by a 2DFA with $n + \text{const}$ states which reverses its input head just one time during each computation. The automaton, firstly scans the input from left to right, only to reach the right end-marker. Then it moves n positions to the left, finally checking whether or not the reached input cell contains the symbol a.

This simple example emphasizes that the possibility of moving the input head in both directions can drammatically reduce the size of deterministic automata. In particular, in this case, *one reversal is enough* to reduce an automaton of exponential size in n to an automaton of linear size.

We can also observe that the language I_n is accepted by a 1NFA and by a 2DFA having approximatively the same size. So, this example could suggest the possibility of replacing nondeterminism in one-way automata by two-way motion, without significantly change the size.

We now present a more elaborated variant of this example which will be also useful to discuss some restricted versions of two-way automata considered in the literature. For each $n > 0$, let us consider the language

$$L_n = (a+b)^*a(a+b)^{n-1}a(a+b)^* \ .$$

In this case, we ask that each string in the language contains two letters a's with $n-1$ symbols in between.

The language L_n can be easily accepted by the 1NFA with $n+2$ states in Figure 2.

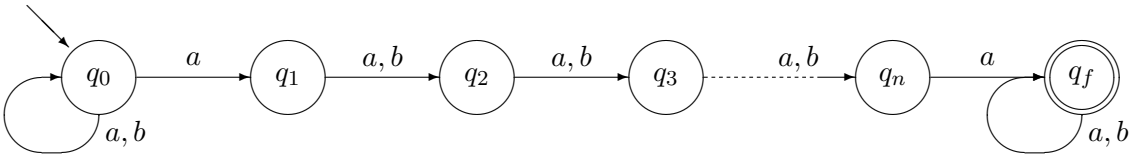

Figure 2. A 1NFA accepting the language $L_n = (a+b)^*a(a+b)^{n-1}a(a+b)^*$.

What about acceptance of L_n by one-way and two-way deterministic *automata?*

Let us start by studying acceptance in the one-way case. The idea is very similar to the one outlined for the language I_n.

We can build an automaton A_n which remembers in its finite control the last n input symbols. Hence, when in the state remembering $\sigma_1\sigma_2\ldots\sigma_n$ a new input symbol γ is read, the automaton moves to the

state remembering $\sigma_2 \dots \sigma_n \gamma$. However, in the case $\sigma_1 = \gamma = a$, the automaton moves to its only final state, where it remains when further input symbols are received. In Figure 3, the automaton A_3 accepting the language L_3 is represented. Notice that with this strategy the resulting 1DFA has $2^n + 1$ states.

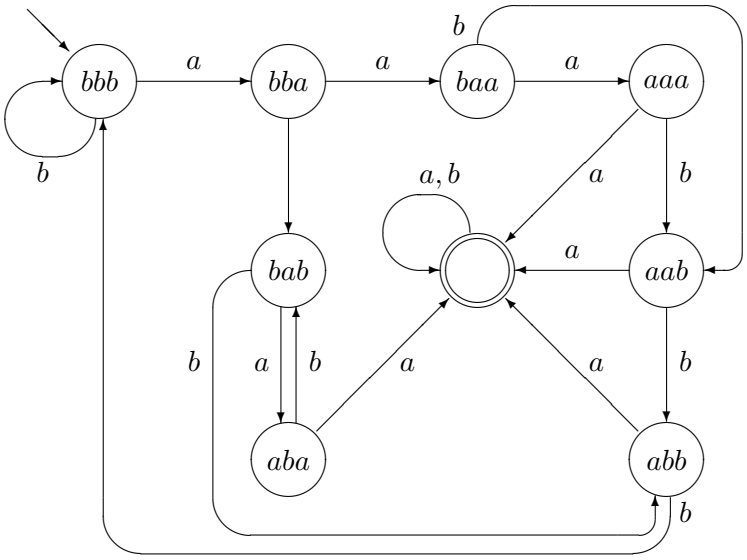

Figure 3. The 1DFA A_3 accepting the language $L_3 = (a+b)^* a (a+b)^2 a (a+b)^*$.

We can show that each automaton A_n is minimal. This can be done by using classical distinguishability arguments (see, e.g., [17]) along the following lines:

- Each two pairwise different strings x, y of length n are distinguishable. To prove this, it is enough to consider the string $b^{i-1}a$, where i, $1 \leq i \leq n$, is the index of the leftmost letter different in x and y, and to verify that exactly one string between $xb^{i-1}a$ and $yb^{i-1}a$ belongs to L_n.
- Each string of length n does not belong to L_n and, hence, it is distinguishable from a^{n+1} which belongs to L_n.
- Hence, the $2^n + 1$ strings in the set $\Sigma^n \cup \{a^{n+1}\}$ are pairwise distinguishable for L_n. As a consequence, $2^n + 1$ is a lower bound for the number of states of each 1DFA accepting L_n. This lower bound matches the number of the states of the automaton A_n above described.

Now, we discuss different strategies for accepting L_n using two-way automata. In the following let $w = w_1 w_2 \cdots w_m$, with $w_i \in \{a, b\}$, $i = 1, \dots, m$, $m \geq 0$, be an input string for which we have to check the membership to L_n.

(i) *Naïf algorithm*
To decide whether a string $w \in \Sigma^*$ belongs to L, for $i = 1, \dots, |w| - n$ we check if both symbols in cells i and $i + n$ are a's. The input is accepted if for at least one i, this condition is satisfied. This algorithm can be implemented by a 2DFA that counts n positions forward to move from cell i to cell $i + n$ and then counts $n - 1$ positions backward to reach cell $i + 1$. Furthermore, when

moving from cell i to cell $i+n$, the automaton needs to remember whether or not the symbol in cell i is a. This leads to a 2DFA with $5n+\text{const}$ states which moves the input head along a zig-zag trajectory. In particular, $2n$ states are used to count n positions forward, moving from cell i to cell $i+n$, while remembering the symbol in cell i, $n-1$ states are used to count n positions backward. Furthermore, after a pair of cells i and $i+n$ both containing the symbol a is found, this algorithm continues to scan the input in the same way (this will be useful later as an example to introduce *oblivious* automata). Hence, further n states to count positions forward and $n-1$ states to count positions backward, are used.

(ii) *An improved algorithm*
As already outlined, it is immediate to observe that the naïf algorithm can be improved. First, when the symbol w_i is b, we do not need to immediately inspect the symbol w_{i+n}. Second, when a cell i is found such that both symbols w_i and w_{i+n} are a's, the automaton can accept without checking the remaining cells. This leads to an algorithm which uses no more than $2n+\text{const}$ states.

(iii) *A different strategy: head reversals only at the end-markers*
We can describe a different algorithm to recognize L_n, which is implemented by a 2DFA performing head reversal *only* when the input head is visiting the end-markers. Hence, in this algorithm a computation is a sequence of left-to-right and right-to-left traversals of the input, which are also called *sweeps*.

We give an informal description of the algorithm:

- The automaton performs at most n sweeps from left to right, interleaved with sweeps from right to left.
- In the ith sweep from left to right, $1 \leq i \leq n$, the automaton starting from the cell i, inspects the contents of cells $i,\ i+n,\ i+2\cdot n,\ i+3\cdot n,\ \ldots$, in order to check if two of them which are consecutive in this list (i.e., cells $i+j\cdot n$ and $i+(j+1)\cdot n$, for some $j \geq 0$) contain the symbol a. If this happens then the automaton stops and accepts.

  To locate the cells that must be inspected, a counter c modulo n is kept in the finite control. This counter can be implemented using n states. Furthermore, the automaton needs to remember the content of the last inspected cell. This doubles the number of the states.
- When in the ith scan from left to right, the right end-marker is reached, there are two possibilities. If $i < n$ then the automaton makes a sweep from right to left, in order to prepare the $(i+1)$th scan from left to right. If $i = n$ then the automaton stops and rejects.

This strategy can be implemented with $2n^2+n+\text{const}$ states, by using $2n$ states for each sweep from left to right, and just one state for each sweep from right to left, while keeping track in the finite control of the counter i.

We can reduce the number of states from quadratic to linear by avoiding to store the counter i for sweeps. To this aim, also during sweeps from right to left we use c to count the input length modulo n.

- The first sweep starts with the head on the first input symbol (cell 1) and the counter c containing 0.

- During each sweep from left to right, the counter c is incremented by one (modulo n) at each step. Each cell which is reached with 0 in the counter is inspected.
- When the right end-marker is reached, the automaton starts a sweep from right to left, decrementing the counter c (modulo n) at each move. Hence, in a sweep from left to right and in the next sweep from right to left, a same cell is reached with the same value of c.
- When the left end-marker is reached, the head is moved to the first input cell, *without changing the counter*, starting a new sweep from left to right. In this way, the input cells which are inspected in the ith sweep from left to right are those in positions $i,\ i+n,\ i+2\cdot n,\ i+3\cdot n,\ \ldots$.
- However:
  - when two cells which are consecutively inspected in the same sweep from left to right, i.e., cells $i+(j-1)\cdot n$ and $i+j\cdot n$, for some j, both contain the symbol a, then the automaton stops and accepts;
  - if the left end-marker is reached with the counter containing 0, then all the positions have been inspected. In this case the automaton stops and rejects.

To implement this procedure, the automaton has to remember in its finite control the counter c, which can assume n different values, and the direction of the input head. Furthermore, in each sweep from left to right the automaton has to remember whether or not the symbol in the last inspected cell was an a. Hence, the total number of states of the resulting automaton in $3n+\mathrm{const}$.

We could slightly modify this algorithm to use also sweeps from right to left to inspect input symbols. While, at least in the worst case, this decreases the recognition time, using a similar implementation the number of states becomes $4n+\mathrm{const}$.

4. Restricted Models

In the first part of this section, we briefly present and discuss some restricted variants of two-way automata that have been considered in the literature. Then, in Subsection 4.1 we summarize the most important separations that have been obtained for the simulations of 1NFAs and 2NFAs by restricted variants of 2DFAs. Let us start by presenting restricted models.

Oblivious Automata

In the previous naïf algorithm (i), we can observe that for all the inputs of the same length m the "trajectory" of the head during the computation is the same, i.e., the position of the input head at the step t does not depend on the input content, but *only on its length*. A 2DFA with this property is called *oblivious* [18].

Sweeping Automata

A two-way automaton performing head reversal *only* when the input head is visiting the end-markers is called *sweeping automaton*. This notion has been studied by Sipser [46]. In particular, for the language L_n above described, the recognition strategy (iii) defines a sweeping 2DFA.

Rotating Automata

In the previous method (iii), sweeps from right to left are only used to move back the input head on the left end-marker. This suggests another model called *rotating automata* [28], which now we briefly describe.

A computation of a rotating automaton is a sequence of left-to-right scans of the input. In particular, when the right end of the input is reached, the computation continues on the leftmost input symbol. In other words, we can imagine the input tape as circular, with a special cell containing a marker and connecting the end to the beginning of the tape. With a trivial transformation, which doubles the number of the states, each rotating automaton can be transformed into an equivalent sweeping automaton.

A simple modification of (iii) produces a rotating automaton with $2n + \mathrm{const}$ states accepting L_n. Even for the language I_n, it is possible to build a rotating automaton with $2n + \mathrm{const}$ states.

Outer Nondeterministic Automata

All the above mentioned models are defined by restricting the movement of the input head. A different kind of restriction has been recently considered in [11, 26], by introducing *outer nondeterministic automata* (2OFAs). In these models, nondeterministic choices can be taken *only* when the input head is scanning the end-markers. Hence, the transition on "real" input symbols are deterministic. This model does not have any restriction on head reversals, i.e., 2OFAs are allowed to change the direction of the input head at each position.

The deterministic algorithm (iii) for accepting L_n can be easily transformed in an algorithm for a (degenerate) outer nondeterministic automaton. At the first step the automaton guesses an integer i, with $1 \leq i \leq n$, and then it simulates the ith sweep from left to right described in algorithm (iii), rejecting if the right end-marker is reached without finding two cells $i + j \cdot n$ and $i + (j + 1) \cdot n$, both containing the symbol a. This can be implemented just choosing the initial value of the counter c in a nondeterministic way, at the beginning of the computation with the head on the left end-marker.

Few Reversal Automata

While all previous models are defined by introducing structural restrictions, the next model is defined by restricting the use of a resource, the *input head reversals* during computations.

A 2DFA is said to be a *few reversal automaton* if the number of head reversals is bounded by a function which is sublinear with respect to the input length, i.e., it is $o(m)$, where m is the length of the input. It has been recently proved that each 2DFA with $o(m)$ reversals is actually a 2DFA with $O(1)$ reversals, i.e., each few reversal 2DFA can make only a number of reversals which is ultimately bounded by a constant [25].

Notice that the algorithm (i) above described, clearly uses a number of reversals which is linear in the length of the input. Even the algorithm (ii) uses a linear number of reversals (consider, e.g., inputs of the form $a^n b^n a^n b^n \cdots a^n b^n$). On the other hand, in the algorithm (iii) the number of reversals is bounded by $2n - 1$, namely a constant with respect to the input length.

In the nondeterministic case, we can have several computations for a same input string. For this reason we can measure head reversals in different ways. For example, we can consider reversals in *all computations*, or only in *all accepting computations*, or just in *one accepting computation*. This can lead to different notions of few reversal 2NFAs (something similar is well known in space complexity,

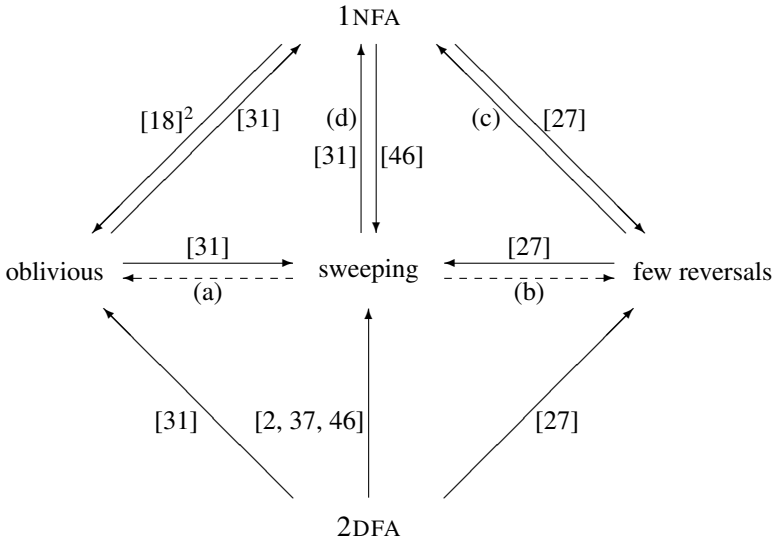

Figure 4. An arrow from a class A of machines to a class B indicates an exponential separation, i.e., the state cost of the simulation of machines in the class A by machines in the class B can be exponential. A dashed arrow indicates the existence of a polynomial simulation. The conversions corresponding to arrows marked (a) and (b) can be easily obtained by squaring the number of the states. (c) derives from (b) and (d). The (trivial) dashed arrow from oblivious, sweeping, and few reversal automata to 2DFAs are not depicted.

where different space notions have been considered, see, e.g., [34]). In our knowledge, at the moment the resource input head reversals for 2NFAs has not been investigated. However, the same notion has been considered for other computational models as *counter machines* [20], *two-way pushdown automata* [19, 33], and *Turing machines* [6, 4, 12].

Unambiguous Automata

This is a well known classical notion: a nondeterministic automaton is *unambiguous* if and only if for each input string there is at most one accepting computation. While the 1NFA above described to recognize I_n is unambiguous, it can be easily seen that the 1NFA A_n accepting L_n can have many accepting computations for a same input, i.e., it is ambiguous.

4.1. Simulations by Restricted Versions of 2DFAs

As already mentioned in the introduction, the Sakoda and Sipser question asks the costs, in states, of the simulations of 1NFAs and 2NFAs by 2DFAs. Separations have been obtained by considering restrictions on the target machines. They are summarized, together with other separations, in Figure 4. Their proofs use rather involved arguments.

In particular, the simulations of n-state 1NFAs (and hence also 2NFAs) by sweeping, oblivious, and few reversal automata require exponentially many states.[2]

[2] A stronger separation can be given by considering the degree of non-obliviousness, that counts the number of different trajectories of the head on inputs of the same length. Hence, a 2DFA has a sublinear degree of non-obliviousness if and only if the number of different trajectories on inputs of length n is $o(n)$. In [18] it was proven that the simulation of 1NFAs by 2DFAs with a sublinear degree of non-obliviousness requires exponentially many states.

Note that all above restrictions are related to the movement of the input head.

However, these results do not solve the general problem. In fact, it has been also proved that the simulations of (unrestricted) 2DFAs by these restricted models require exponentially many states.

Concerning few reversals 2DFAs, we already mentioned that a $o(n)$ upper bound on reversals implies a $O(1)$ upper bound [25]. We can also compare the size of 2DFAs making a fixed numbers of reversals. For example, we observed that the language I_n is accepted by a 2DFA with $n + \mathrm{const}$ states that makes only one reversal, while each 1DFA (i.e., each 2DFA making 0 reversals) needs 2^n states to accept it. Hence, 2DFAs making 0 reversals can be exponentially larger than 2DFAs making 1 reversal.

What about 2DFA*s making* k *reversals versus* 2DFA*s making* $k + 1$ *reversals, for* $k > 0$*?*

In the case $k = 1$, this question has been solved by Balcerzak and Niviński [1], by proving an exponential separation. Recently Kapoutsis and Pighizzini extended this separation to each integer k, providing an infinite reversal hierarchy of 2DFAs [25]. It should be interesting to investigate similar questions in the nondeterministic case.

5. The Case of Unary Languages

Unary languages are defined over a one letter alphabet Σ. In the following we stipulate $\Sigma = \{a\}$.

The state costs of the optimal simulations between different variant of unary automata have been obtained by Chrobak [7] and by Mereghetti and Pighizzini [35]. They are summarized in Figure 5.

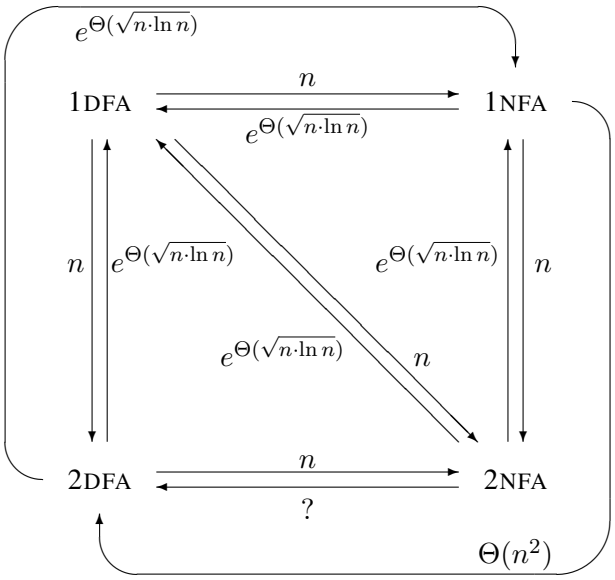

Figure 5. Costs of the *optimal* simulations between different kinds of *unary* automata. An arc labeled $f(n)$ from a vertex x to a vertex y means that a unary n-state automaton in the class x can be simulated by an $f(n)$-state automaton in the class y. The $e^{\Theta(\sqrt{n\cdot\ln n})}$ costs for the simulations of 1NFAs and 2DFAs by 1DFAs as well the cost $\Theta(n^2)$ for the simulation of 1NFAs by 2DFAs have been proved in [7]. The $e^{\Theta(\sqrt{n\cdot\ln n})}$ cost for the simulation of 2NFAs and 1DFAs has been proved in [35]. The other $e^{\Theta(\sqrt{n\cdot\ln n})}$ costs are easy consequences. All the n costs are trivial. The arc labeled "?" represents the open question of Sakoda and Sipser.

From the picture we can observe that the cost of the optimal simulations in the unary case can be smaller than in the general case. For example, the cost of the simulation of n-state 1NFAs by 1DFAs reduces from 2^n to $e^{\Theta(\sqrt{n \cdot \ln n})}$. Quite surprisingly, eliminating at the same time both nondeterminism and two-way motion costs as eliminating only one of them.

The question 1NFAs versus 2DFAs has been solved in the unary case in [7] by showing that the tight cost is polynomial, more precisely $\Theta(n^2)$. This gives also the best known lower bound for the general case.

In spite the unary case looks simpler than the general one, the question of 2NFAs versus 2DFAs not only is still open even in this case, but it seems also to be difficult and, at the same time, very challenging, even for its connections, presented in Section 6, with the open question of the relationship between deterministic and nondeterministic logarithmic space.

Normal Forms for Unary Nondeterministic Automata

The "simplicity" of automata over a unary alphabet, with respect to automata over general alphabets, allows to give normal forms for unary 1NFAs and 2NFAs. These forms, at the price of a small increasing in the number of the states, strongly restrict the use of nondeterminism and head reversals.

For the one-way case, we mention the *Chrobak normal form* [7]. In this form, the transition graph of an automaton consists of a deterministic path from the initial state to a state q, together with $k \geq 0$ deterministic loops. From the state q there are k outgoing edges, each one of them connecting q to exactly one state in each of the k loops. Hence, in each computation, a 1NFA in this form is allowed to make at most one nondeterministic choice, when it has to leave the state q. A degenerate case of 1NFA in Chrobak normal form is an automaton whose transition graph consists exactly of one deterministic loop, without the initial path. The important fact is that each n-state unary 1NFA can be converted into an equivalent one in Chrobak normal form with no more than n^2 states in the initial path and n states in the loops. Hence, the conversion does not significantly increase the number of the states.[3]

A generalization of the Chrobak normal form to the two-way case has been obtained by Geffert, Mereghetti, and Pighizzini [13], and it is given in the next Theorem 5.1. In order to present it, it is useful to relax the notion of equivalence between automata, by allowing a finite number of "errors". More precisely, two finite automata are said to be *almost equivalent* if the symmetric difference of their accepted languages is finite, i.e., the languages accepted by the two automata coincide, with the possible exception of a finite number of strings.

Theorem 5.1. Each n-state unary 2NFA A can be transformed into an almost equivalent 2NFA M such that:

- M is *quasi-sweeping*, namely, head reversals and nondeterministic choices are possible only when the head is scanning the end-markers.[4]

[3] Besides [7], we refer the reader to [9, 10, 43] . All these papers present different algorithms and techniques for the conversion of unary 1NFAs into Chrobak normal form.

[4] In [46], the term *sweeping* was introduced for *deterministic* automata making head reversals only on the end-markers. In *quasi-sweeping automata* the restrictions are relaxed, by allowing nondeterministic choices, but *only* when the input head is scanning the end-markers, not on "real" input symbols. A further natural extension can be obtained by allowing nondeterministic choices on any input position, still keeping the possibility of reversing the head direction only when the input head is visiting the end-markers, thus obtaining the natural nondeterministic counterpart of sweeping automata.

- M has at most $2n + 2$ states,
- the languages accepted by A and M can differ only on strings of length at most $5n^2$.

An inspection to the proof of Theorem 5.1 [13, 15] shows that M and its computations have a very simple structure. In particular, in each sweep M uses a deterministic loop to count the input length modulo one integer.

The 2NFA M can be easily turned into an automaton "fully" equivalent to the original 2NFA A, by adding $5n^2 + \mathrm{const}$ states, used to fix the "errors", in a preliminary scan of the input.

We point out that in the deterministic case, i.e., for unary 2DFAs, a similar normal form has been obtained in [30].

The normal form in Theorem 5.1 gives a strong simplification of the structure of unary 2NFAs. This has been an important tool to prove several results on unary 2NFAs. First of all, it has been used in [13] to prove a subexponential, but still superpolynomial upper bound for the conversion of unary 2NFAs into equivalent 2DFAs:

Theorem 5.2. Each unary n-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states.

It is interesting to discuss the main idea in the proof of this result. Suppose the given n-state 2NFA A is already in the normal form of Theorem 5.1. We can observe that if an accepting computation C visits the left end-marker more than n times, then there must exist a shorter accepting computation C' on the same input. In fact, in C at least one state q should be visited twice with the head on the left end-marker. So, the computation C' can be obtained by cutting the part of C between these two repetitions of q. Hence, if we assume acceptance on the left end-marker, in order to detect if an input is accepted, it is enough to check the existence of a computation starting in the initial state with the head on the left end-marker, ending in a final state with the head on the same end-marker, and visiting the left end-marker at most n times.

To this aim, we define a predicate $reachable(p, q, k)$ which holds true exactly when there is a computation path which starts in the state p on the left end-marker, ends in the state q on the same end-marker, and visits such end-marker at most k times. This predicate can be recursively verified by using a divide-and-conquere technique.[5] The implementation of the resulting procedure, described in [13], leads to a 2DFA with $e^{O(\ln^2 n)}$ states.

In the case the given automaton is not in normal form, we first convert it into an almost equivalent 2NFA in normal form and then we apply the above procedure to the resulting automaton. Finally, with a small modification which does not increase the state upper bound, we fix the "errors", i.e., we manage strings of length $\leq 5n^2$, in order to obtain a 2DFA fully equivalent to the original 2NFA.

The upper bound in Theorem 5.2 is subexponential, in the sense that it grows less than the exponential function e^n, but it is superpolynomial, by growing faster than any polynomial. The natural question is that of investigating whether or not it is tight. At the moment we do not have an answer to it. However, the question is related to the relationship between deterministic and nondeterministic logarithmic space. The discussion of this point is postponed to the next section.

The normal form in Theorem 5.1 has been used to prove other interesting properties of unary 2NFAs. Among them:

[5] The procedure looks very similar to the famous one introduced by Savitch in [42] to remove nondeterminism in space bounded machines.

(i) Each unary n-state 2NFA accepting a language L can be transformed into a 2NFA with $O(n^8)$ states accepting the complement of L [14].

(ii) Each unary n-state 2NFA can be transformed into an equivalent *unambiguous* 2NFA with a number of states polynomial in n [15].

The proof of (i) is given by using an *inductive counting* argument. This technique was originally and independently introduced in space complexity by Immerman [21] and Szelepcsényi [47]. Essentially, given a 2NFA M accepting a language L, the 2NFA M_c accepting the complement of L implements an algorithm which counts, for $t = 0, \ldots, n$, the number of states that are reachable on the left end-marker by all computation paths of M on the given input, starting from the initial configuration and visiting the left end-marker exactly t times. Furthermore, as a side effect of this counting, the algorithm generates all the states that are reachable on the left end-marker. We already observed that an input is accepted by a 2NFA in normal form if and only if there is an accepting computation path which visits the left end-marker at most n times. Hence, assuming acceptance on the left end-marker, using the above-outlined procedure, M_c can detect whether or not M accepts the input. Thus, M_c accept if and only if M rejects.

The result (ii) was obtained by adapting one of constructions discussed in the next section (in particular, the construction used to prove Lemma 6.1) and using another result in space complexity, proved by Reinhardt and Allender [40].

Recently [11], the results we just discussed, concerning the unary case, have been extended to the case of *outer nondeterministic automata* (without any restriction on the alphabet size). In particular, concerning the simulation by 2DFAs, the following result has been proved:

Theorem 5.3. Each n-state outer nondeterministic automaton can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states.

While the proof of Theorem 5.2 strongly relies on the normal form for unary 2NFAs presented in Theorem 5.1, the construction obtained to prove Theorem 5.3 uses a procedure which, essentially, is able to detect infinite loops in outer nondeterministic automata. This procedure is derived from an algorithm, presented in [14], to make two-way *deterministic* automata halting, which, in turns, is derived by a complementation technique for space bounded deterministic Turing machines discovered by Sipser [45].

6. Relationships with the L versus NL Question

Interesting connections between the question of Sakoda and Sipser and the open question of the relationship between the classes of languages accepted in logarithmic space by deterministic and nondeterministic Turing machines (denoted by L and NL, respectively) have been obtained. In this section we will discuss them.

(i) First of all, Berman and Lingas [3] proved that if $\mathsf{L} = \mathsf{NL}$ then for each n-state 2NFA A with an input alphabet of σ symbols there exists a 2NFA B with a number of states polynomial in n and σ which agrees with A on strings of length at most n. Hence $\mathsf{L} = \mathsf{NL}$ would imply a polynomial simulation of 2NFAs by 2DFAs on "short" inputs.

This result was recently improved along the following lines.

(ii) Geffert and Pighizzini [15] considered the unary case. They proved that $\mathsf{L} = \mathsf{NL}$ would imply a polynomial simulation of unary 2NFAs by 2DFAs.[6] Compared with condition (i), we can observe that while only devices with a unary input alphabet are considered here, the restriction on the length of the inputs is removed.

This result shows the relevance of the unary case. In fact, proving the optimality of the bound in Theorem 5.2 or even proving a smaller, but still superpolynomial lower bound for the simulation of unary 2NFAs by 2DFAs, would imply the separation of L and NL.

(iii) Kapoutsis [23] generalized the condition (i) by proving that $\mathsf{L/poly} \supseteq \mathsf{NL}$ if and only if for each n-state 2NFA A with an input alphabet of σ symbols there exists a 2NFA B with a number of states polynomial in n (but not dependending on σ) which agrees with A on strings of length at most n, where $\mathsf{L/poly}$ denotes the class of languages accepted by deterministic logspace bounded machines that can access a *polynomial advice*.[7] Hence $\mathsf{L/poly} \supseteq \mathsf{NL}$ is *equivalent* to the existence of a state polynomial simulation of 2NFAs by 2DFAs on "short" inputs. Since $\mathsf{L/poly} \supseteq \mathsf{L}$ and $\mathsf{L} \subseteq \mathsf{NL}$, the *only-if* condition is stronger than the condition (i). Furthermore, in this case also the converse implication holds.

(iv) Quite recently, Kapoutsis and Pighizzini [26] proved the equivalence between $\mathsf{L/poly} \supseteq \mathsf{NL}$ and several other propositions. In particular, they show that $\mathsf{L/poly} \supseteq \mathsf{NL}$ *if and only if* there is a state polynomial simulation of *unary* 2NFAs by 2DFAs. As for (iii), we can observe that the *only-if* condition is stronger than the condition in (ii) and, furthermore, in this case also the converse implication holds.

We are now going to discussing (ii) and (iv) more into details.

The Graph Accessibility Problem

A central role in the above mentioned investigations of the relationships between the L versus NL and $\mathsf{L/poly}$ versus NL questions and the problem of Sakoda and Sipser in the unary case is played by the *Graph Accessibility Problem* (GAP), which is the problem of deciding whether or not a given directed graph $G = (V, E)$, with two fixed vertices $s, t \in V$, contains a path from s to t.[8]

It is well known that GAP is an NL-complete problem [42]. Hence, $\mathsf{GAP} \in \mathsf{NL}$ and, moreover, $\mathsf{GAP} \in \mathsf{L}$ if and only if $\mathsf{L} = \mathsf{NL}$. In other words, this means that GAP is a hardest problem in NL. As we discuss below, the restriction of GAP to a fixed set of vertices represents in some sense (and under a suitable encoding) an hardest language for unary 2NFAs.

In [15], it was shown how to reduce the language accepted by a unary n-state 2NFA A to a graph with $N = O(n)$ vertices. In other words, given an integer m it is possible to obtain a graph $G(m)$ with N vertices such that the unary string a^m is accepted by A if and only if $G(m) \in \mathsf{GAP}$. Furthermore, the reduction can be computed by a finite state transducer of size polynomial in N.

[6] The restriction to the unary case concerns only two-way automata, not the classes L and NL.

[7] A *polynomial advice* [29] is a sequence of strings $(\alpha_n)_{n \geq 0}$, such that the length of α_n is bounded by a polynomial in n. Together with an input string x, the machine receives the advice corresponding to the length of x, namely the string $\alpha_{|x|}$.

[8] As customary, we use GAP also to denote the set of positive instances of the graph accessibility problem. Hence, we write $G \in \mathsf{GAP}$ if and only if the given directed graph G contains a path connecting two (implicitly) fixed vertices s and t.

If $\mathsf{L} = \mathsf{NL}$ then there is a logspace bounded deterministic machine that solves GAP. By restricting this machine to inputs which encode graphs with N vertices, we obtain a finite state automaton D_{GAP} which can decide whether or not the graph $G(m)$, resulting from the above reduction, belongs to GAP. By a suitable composition of the transducer with D_{GAP}, we get a 2DFA B equivalent to the original 2NFA A, with a number of states polynomial in n, the number of states of A (see Figure 6). We address the reader to [15] for details. In particular, we point out that the reduction uses the normal form for unary 2NFAs presented in Theorem 5.1. This construction has been extended to outer nondeterministic automata in [11]. Furthermore, with a similar technique, it is possible to show that unary 2NFAs and 2OFAs over any input alphabet can be simulated by equivalent *unambiguous* 2NFAs with polynomially many states [11, 15].[9]

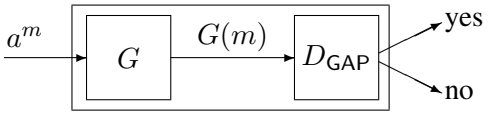

Figure 6. Simulating a unary 2NFA with a 2DFA of polynomial size, under the hypothesis $\mathsf{L} = \mathsf{NL}$.

It is quite natural to ask if the converse also holds, i.e., if a state polynomial simulation of unary 2NFAs by 2DFAs would imply $\mathsf{L} = \mathsf{NL}$. The main problem in trying to prove such a result is related to the uniformity. In particular, in [15] it is proved even a stronger result, however using the additional hypothesis that the conversion from unary 2NFAs to 2DFAs is computed by a logspace bounded transducer.

On the other hand, it is not difficult to observe that the above described construction does not use the uniformity of L and, hence, it works even under the weaker hypothesis $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$:

Lemma 6.1. If $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$ then the state cost of the simulation of unary 2NFAs by 2DFAs is polynomial.

In [26], also the converse of Lemma 6.1 has been proved. The main idea is to exhibit, under the hypothesis that the state cost of the simulation of unary 2NFAs by 2DFAs is polynomial, a logspace bounded deterministic machine M which, making use of a polynomial advice, solves the graph accessibility problem. This is done by the following steps:

- A function $\langle\ \rangle_1$ mapping instances of GAP to unary strings is provided. For each integer n, the function $\langle\ \rangle_1$ is a reduction from GAP restricted to graphs with n vertices to a unary language UGAP_n.
- A unary 2NFA A_n recognizing UGAP_n with a number of states polynomial in n is described.

[9] The simulations by *unambiguous* 2NFAs do not require the assumption $\mathsf{L} = \mathsf{NL}$. A few words about them are suitable. In the context of nonuniform space complexity, Reinhardt and Allender proved that nondeterministic computations in logarithmic space can be made unambiguous, with the additional help of a polynomially long advice [40], i.e., $\mathsf{NL} \subseteq \mathsf{UL}/\mathsf{poly}$, where $\mathsf{UL}/\mathsf{poly}$ denotes the class of languages accepted by logspace bounded *unambiguous* machines equipped with a polynomial advice. Hence, there is a machine U_{GAP} of this kind which is able to solve the problem GAP. For the simulation of unary 2NFAs, the same construction as in Figure 6 can be used, after replacing the machine D_{GAP} by U_{GAP} and the advice. The advice depends on the size of the graph $G(m)$, namely on the number of states of the given 2NFA A, while it does not depend on the input string a^m. Hence, given A, the advice it is fixed. This allows to encode it in the "hardware" of the automaton resulting from this construction.

- The automaton A_n is replaced by an equivalent 2DFA B_n.
- An instance of GAP can be solved by combining the machine computing the reduction with the 2DFA B_n, where n is the number of vertices of the instance under consideration (hence n depends only on the input length), see Figure 7. In particular, the resulting machine M receives the input string, which represents a graph G with h vertices, together with an encoding of the 2DFA B_n, *which is the advice* for M. If the state cost of the simulation of unary 2NFAs by 2DFAs is polynomial, then B_n can be encoded by a string of polynomial length in n. Furthermore, using a suitable encoding for UGAP_n (we sketch some ideas below), the workspace used by M can be bounded by a logarithmic function in n.

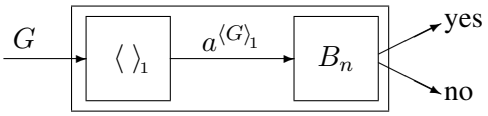

Figure 7. The machine M solving GAP using B_n as advice.

We are going to describe the encoding $\langle\ \rangle_1$ and the languages UGAP_n.

For each integer n, let $V_n = \{0, 1, \ldots, n-1\}$ and K_n be the complete graph with vertex set V_n. With each edge (i, j) of K_n we associate a different prime $p_{(i,j)}$. To this aim we choose the first n^2 prime numbers.

A graph $G = (V_n, E)$ with n vertices is encoded as the product of all prime powers corresponding to the edges in E (see Figure 8), i.e., by the number

$$\langle G \rangle_1 = \prod_{(i,j)\in E} p_{(i,j)}$$

Conversely, with each integer m we can associate the graph $K_n(m) = (V_n, E(m))$ such that $(i, j) \in E(m)$ if and only if $p_{(i,j)}$ divides m. It should be clear to the reader that $K_n(\langle G \rangle_1) = G$.

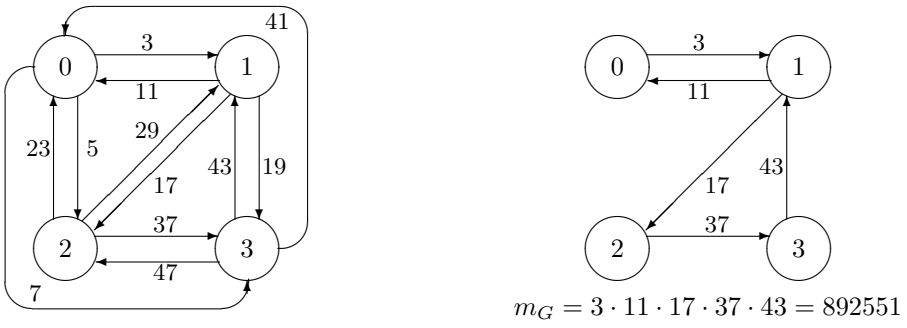

Figure 8. The complete graph K_4 with a subgraph G. The number $p_{(i,j)}$ associated with the edge (i, j) is the $(i \cdot n + j + 1)th$ prime number. In K_4 the edges (i, i) are not depicted.

$i \leftarrow 0$
while $i \neq n-1$ **do** `// input head on the left end-marker, simulating vertex` i
  guess $j \neq i$ `// try the edge` (i,j)
  scan the input from left to right counting its length modulo $p_{(i,j)}$
  if remainder $\neq 0$ **then reject** `// if` $(i,j) \notin E$ `then reject`
  move the input head to the left end-marker `// else continue the simulation from` j
  $i \leftarrow j$
endwhile
accept

Figure 9. The nondeterministic algorithm implemented by A_n to recognize UGAP_n.

We can now define the *unary encoding* of GAP, restricted to graphs with n vertices, as the following language:

$$\mathsf{UGAP}_n = \{a^m \mid K_n(m) \text{ has a path from 0 to } n-1\}$$

We now describe a 2NFA A_n recognizing UGAP_n. Roughly speaking, A_n implements the standard nondeterministic algorithm solving GAP (see Figure 9). From a vertex i (starting from $i \leftarrow 0$ at the beginning of the computation), A_n guesses another vertex j and then it verifies whether $(i,j) \in E$. If this is the case, then A_n repeats the same simulation from the vertex j, up to reach the vertex $n-1$. However, in the case a pair $(i,j) \notin E$ is guessed, then A_n hangs and rejects. To check the condition $(i,j) \in E$, A_n computes the length of its input modulo $p_{(i,j)}$.
More into details:

- A_n is outer nondeterministic and sweeping, i.e., it can reverse the input head direction and make nondeterministic choices *only* when the head is scanning one of the end-markers. Furthermore, in each traversal A_n counts the input length modulo a prime number.

- On the end-markers, each state is interpreted either as a copy of a vertex in V_n or as an *hang* state.

- The automaton A_n can traverse an input a^m from one end-marker in a copy of vertex i to the opposite end-marker in some copy of vertex j, without visiting the end-markers in between, if and only if the number $p_{(i,j)}$ divides m. In particular, when the automaton is visiting one end-marker in a state representing the vertex i, it guesses another vertex j, by entering an appropriate loop where it traverses and counts the input modulo $p_{(i,j)}$. The state in this loop which corresponds to the remainder 0 is interpreted as the vertex j of the graph, the other states are interpreted as *hang* states. Hence, when the input head reaches the opposite end-marker, the automaton continues the simulation or hangs and rejects depending on the reached state.

Using some properties related to the distribution of prime numbers (see, e.g, [16]), it can be proved that *the number of states of A_n is polynomial in n.*

Finally, we have to show how the machine M, obtained in this way, can work in logarithmic space. Actually, this is not true if we directly implement M as in Figure 7. In fact, the length of the unary encoding of a graph with n vertices can be exponential in n. For instance, $\langle K_n \rangle_1$, the unary encoding of the complete graph of n vertices, is the product of the first n prime numbers, which is exponential in n.

This problem is solved as follows:

- The unary encoding is replaced by a "prime encoding" that, in this case, is a list of all primes associated with the edges in the input graph. Hence, the output of the reduction is this list.
- Due to a structural property of 2DFAs (see [13, 30]), it is possible to modify the automaton B_n, still keeping polynomial its number of states, by replacing its unary input tape, by a tape containing a prime encoding of the input length. Hence, after these modifications, the machine M still solves GAP.
- To be stored, the prime encoding would require polynomial space, which is still too much for our purposes. To avoid this problem, the prime encoding is not kept in the internal memory of M, but it is computed and recomputed "on fly", each time B_n needs to access it. This is done by restarting the machine that from the input graph G computes the prime encoding.

Along these lines the converse of Lemma 6.1 is proved. This allows to obtain the following:

Theorem 6.2. $\mathsf{L/poly} \supseteq \mathsf{NL}$ if and only if the state cost of the simulation of unary 2NFAs by 2DFAs is polynomial.

We address the reader to [26] for the details and for the equivalence of $\mathsf{L/poly} \supseteq \mathsf{NL}$ with several other statements.

7. Concluding Remarks

We strongly believe that the Sakoda and Sipser question is a very challenging problem which deserves further investigation. Several interesting models have been considered and many deep results have been obtained in the researches related to this question.

As pointed out in the paper, several connections with space complexity have been discovered. This is not limited to the relationships with the question of the power of nondeterminism in logspace bounded computations, which has been discussed in Section 6. In fact, as we emphasized (mainly in Section 5), in more than one case, techniques from space complexity turn out to be useful to study two-way automata. For instance, the divide-and-conquere technique used to prove Theorem 5.2 derives from a similar argument used by Savitch to prove his famous theorem concerning the removal of nondeterminism in space bounded Turing machines [42]. The inductive counting technique used in [14] to obtain the polynomial complementation of 2NFAs, was introduced independently by Immerman [21] and Szelepcsényi [47] to prove the closure under complementation of nondeterministic space. The state polynomial conversion of unary 2NFAs into equivalent unambiguous 2NFAs, uses a result by Reinhardt and Allender [40], which, in the context of nonuniform complexity, shows that nondeterministic logspace bounded machines can be made unambiguous keeping the space bound. Furthermore, as mentioned at the end of Section 5, the subexponential simulation of outer nondeterministic automata by 2DFAs (besides other results on outer nondeterministic automata), uses a technique which derives from the proof a space complexity result by Sisper [45], namely the possibility of removing non-halting computations from deterministic Turing machines, without increasing the space.

As we mention in the introduction, in [41] Sakoda and Sipser formulated their question by presenting a perfect analogy with the open question of the relationship between deterministic and nondeterministic polynomial time. Actually, the complexity theory for finite automata can be developed as a part of standard complexity theory for Turing machines, with classes, reductions, complete problems and so on. For a detailed discussion on this approach, we recommend to the interested reader the recent paper by Kapoutisis [24], where the name *minicomplexity* is suggested for this theory. The same author is working to collect and organize in a website all the material and the results in this area, see `www.minicomplexity.org`.

References

[1] Balcerzak, M., Niwinski, D.: Two-way deterministic automata with two reversals are exponentially more succinct than with one reversal, *Inf. Process. Lett.*, **110**(10), 2010, 396–398.

[2] Berman, P.: A note on sweeping automata, in: *Automata, Languages and Programming* (J. de Bakker, J. van Leeuwen, Eds.), vol. 85 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1980, 91–97.

[3] Berman, P., Lingas, A.: *On the complexity of regular languages in terms of finite automata*, Technical Report 304, Polish Academy of Sciences, 1977.

[4] Bertoni, A., Mereghetti, C., Pighizzini, G.: An Optimal Lower Bound for Nonregular Languages, *Inf. Process. Lett.*, **50**(6), 1994, 289–292, Errata: [5].

[5] Bertoni, A., Mereghetti, C., Pighizzini, G.: Corrigendum: An Optimal Lower Bound for Nonregular Languages, *Inf. Process. Lett.*, **52**(6), 1994, 339.

[6] Chen, J., Yap, C.-K.: Reversal Complexity, *SIAM J. Comput.*, **20**(4), 1991, 622–638.

[7] Chrobak, M.: Finite automata and unary languages, *Theor. Comput. Sci.*, **47**(3), 1986, 149–158, Errata: [8].

[8] Chrobak, M.: Errata to: Finite automata and unary languages: [Theoret. Comput. Sci. 47 (1986) 149-158], *Theor. Comput. Sci.*, **302**(1-3), 2003, 497 – 498.

[9] Gawrychowski, P.: Chrobak Normal Form Revisited, with Applications, in: *Implementation and Application of Automata* (B. Bouchou-Markhoff, P. Caron, J.-M. Champarnaud, D. Maurel, Eds.), vol. 6807 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2011, 142–153.

[10] Geffert, V.: Magic numbers in the state hierarchy of finite automata, *Inf. Comput.*, **205**(11), 2007, 1652–1670.

[11] Geffert, V., Guillon, B., Pighizzini, G.: Two-Way Automata Making Choices Only at the Endmarkers, in: *Language and Automata Theory and Applications* (A.-H. Dediu, C. Martín-Vide, Eds.), vol. 7183 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2012, 264–276.

[12] Geffert, V., Mereghetti, C., Pighizzini, G.: Sublogarithmic Bounds on Space and Reversals, *SIAM J. Comput.*, **28**(1), 1998, 325–340.

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

[14] Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata, *Inf. Comput.*, **205**(8), 2007, 1173–1187.

[15] Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space, *Inf. Comput.*, **209**(7), 2011, 1016–1025.

[16] Hardy, G., Wright, E.: *An Introduction to the Theory of Numbers*, Oxford Science Pub., 1979.

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

[18] Hromkovič, J., Schnitger, G.: Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipsers Separation, in: *Automata, Languages and Programming* (J. Baeten, J. Lenstra, J. Parrow, G. Woeginger, Eds.), vol. 2719 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2003, 193–193.

[19] Ibarra, O. H.: A Note on Semilinear Sets and Bounded-Reversal Multihead Pushdown Automata, *Inf. Process. Lett.*, **3**(1), 1974, 25–28.

[20] Ibarra, O. H.: Reversal-Bounded Multicounter Machines and Their Decision Problems, *J. ACM*, **25**(1), 1978, 116–133.

[21] Immerman, N.: Nondeterministic space is closed under complementation, *SIAM J. Comput.*, **17**(5), 1988, 935–938.

[22] Kapoutsis, C.: Removing Bidirectionality from Nondeterministic Finite Automata, in: *Mathematical Foundations of Computer Science 2005* (J. Jedrzejowicz, A. Szepietowski, Eds.), vol. 3618 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2005, 544–555.

[23] Kapoutsis, C.: Two-Way Automata versus Logarithmic Space, in: *Computer Science Theory and Applications* (A. Kulikov, N. Vereshchagin, Eds.), vol. 6651 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2011, 359–372.

[24] Kapoutsis, C.: Minicomplexity, in: *Descriptional Complexity of Formal Systems* (M. Kutrib, N. Moreira, R. Reis, Eds.), vol. 7386 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2012, 20–42.

[25] Kapoutsis, C., Pighizzini, G.: Reversal Hierarchies for Small 2DFAs, in: *Mathematical Foundations of Computer Science 2012* (B. Rovan, V. Sassone, P. Widmayer, Eds.), vol. 7464 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2012, 554–565.

[26] Kapoutsis, C., Pighizzini, G.: Two-Way Automata Characterizations of L/poly versus NL, in: *Computer Science, Theory and Applications* (E. Hirsch, J. Karhumäki, A. Lepistö, M. Prilutskii, Eds.), vol. 7353 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2012, 217–228.

[27] Kapoutsis, C. A.: Nondeterminism is essential in small two-way finite automata with few reversals, *Information and Computation*, **222**, 2013, 208 – 227.

[28] Kapoutsis, C. A., Královic, R., Mömke, T.: Size complexity of rotating and sweeping automata, *J. Comput. Syst. Sci.*, **78**(2), 2012, 537–558.

[29] Karp, R., Lipton, R.: Turing machines that take advice, in: *Logic and Algorithmic* (E. Engeler et al, Ed.), L'Enseignement Mathématique, Genève, 1982, 191–209.

[30] Kunc, M., Okhotin, A.: Describing Periodicity in Two-Way Deterministic Finite Automata Using Transformation Semigroups, in: *Developments in Language Theory* (G. Mauri, A. Leporati, Eds.), vol. 6795 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2011, 324–336.

[31] Kutrib, M., Malcher, A., Pighizzini, G.: Oblivious Two-Way Finite Automata: Decidability and Complexity, in: *LATIN 2012: Theoretical Informatics* (D. Fernández-Baca, Ed.), vol. 7256 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2012, 518–529.

[32] Lupanov, O.: A comparison of two types of finite automata, *Problemy Kibernet*, **9**, 1963, 321–326, (in Russian). German translation: Über den Vergleich zweier Typen endlicher Quellen, Probleme der Kybernetik 6, 329–335 (1966).

[33] Malcher, A., Mereghetti, C., Palano, B.: Descriptional complexity of two-way pushdown automata with restricted head reversals, *Theor. Comput. Sci.*, **449**, 2012, 119–133.

[34] Mereghetti, C.: Testing the descriptional power of small Turing machines on nonregular language acceptance, *Int. J. Found. Comput. Sci.*, **19**(4), 2008, 827–843.

[35] Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata, *SIAM J. Comput.*, **30**(6), 2001, 1976–1992.

[36] Meyer, A. R., Fischer, M. J.: Economy of description by automata, grammars, and formal systems, *SWAT '71: Proceedings of the 12th Annual Symposium on Switching and Automata Theory*, IEEE Computer Society, Washington, DC, USA, 1971.

[37] Micali, S.: Two-way deterministic finite automata are exponentially more succinct than sweeping automata, *Inf. Process. Lett.*, **12**(2), 1981, 103–105.

[38] Moore, F.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata, *Computers, IEEE Transactions on*, **C-20**(10), 1971, 1211 – 1214.

[39] Rabin, M. O., Scott, D.: Finite automata and their decision problems, *IBM J. Res. Dev.*, **3**(2), 1959, 114–125.

[40] Reinhardt, K., Allender, E.: Making Nondeterminism Unambiguous, *SIAM Journal on Computing*, **29**(4), 2000, 1118–1131.

[41] Sakoda, W. J., Sipser, M.: Nondeterminism and the size of two-way finite automata, *STOC* (R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, A. V. Aho, Eds.), ACM, 1978.

[42] Savitch, W. J.: Relationships between nondeterministic and deterministic tape complexities, *J. Comput. Syst. Sci.*, **4**(2), 1970, 177–192.

[43] Sawa, Z.: Efficient Construction of Semilinear Representations of Languages Accepted by Unary Nondeterministic Finite Automata, *Fundamenta Informaticae*, **123**(1), 2013, 97–106.

[44] Shepherdson, J. C.: The reduction of two-way automata to one-way automata, *IBM J. Res. Dev.*, **3**(2), 1959, 198 –200.

[45] Sipser, M.: Halting Space-Bounded Computations, *Theor. Comput. Sci.*, **10**, 1980, 335–338.

[46] Sipser, M.: Lower bounds on the size of sweeping automata, *J. Comput. Syst. Sci.*, **21**(2), 1980, 195–202.

[47] Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata, *Acta Inf.*, **26**(3), 1988, 279–284.