Two-Way Finite Automata: Old and Recent Results

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

The notion of *two-way automata* was introduced at the very beginning of automata theory. In 1959, Rabin and Scott [31] and, independently, Shepherdson [35], 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 [32] posed the question of the cost, in the number of the states, of the simulation 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 and on the connections with open questions in space complexity.

1 Introduction and Preliminaries

Finite state automata are usually presented as devices which are able to recognize input strings using a fixed amount of memory, implemented using a finite state control (see, e.g., [12]). The input string is written on a read-only tape, which is scanned by an input head. In the basic model the input head is moved only from 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 [25, 28, 30].

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 [31] and by Shepherdson [35], at the beginning of automata theory. However, from the point of view of the size (measured in terms of states) the situation is different. We still do not have a complete picture of the relationships between the sizes of different variants of finite automata.

By an analysis of the constructions given in [31, 35], 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 [17].

The costs of the simulations of 1NFA*s by* 2DFA*s and of* 2NFA*s by* 2DFA*s are still unknown.* The problem of stating them was raised in 1978 by Sakoda and Sipser [32], with the conjecture that they are not polynomial. In spite of all attempts to solve it, this problem is still open.

E. Formenti (Ed.): AUTOMATA and JAC 2012 conferences
EPTCS 90, 2012, pp. 3–20, doi:10.4204/EPTCS.90.1

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

Technical Issues

We will keep the presentation at an informal level, trying to avoid, as much as possible, technical details. For this reason we do not give a formal definition of the main model we are interested in, but we just present an informal description.

We assume that the reader is familiar with standard notions concerning finite state automata, as presented for instance in [12]. We denote by Σ the *input alphabet*, by Σ^* the set of all strings over Σ, and by Σ^n the set of strings of length n, where $n \geq 0$ is an integer. 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 immediately after the execution of the move which reads 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 endmarker*. Hence, if the input is $w \in \Sigma^*$, then the input 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 numbered from 0 to $|w|+1$, where cells 0 and $|w|+1$ contain the endmarkers, and the remaining cells contain "real" input symbols. The input head cannot violate the endmarkers.

- The computation starts in a designed 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 for 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 endmarker in a final state. However, this condition can be slightly modified by considering acceptance on the left endmarker or just on one endmarker.

  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 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.

- We point out that a two-way automaton can enter into a loop. In this case the computation is rejecting.

- When we say that a two-way automaton A has $f(n)+_{...}$ 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.
- An *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 make 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 contains just one head reversal.

2 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 the optimality of the subset construction (actually, this very simple example does not achieve exactly the optimality, but it is very close to it). In particular, for each $n \geq 1$, we can prove the following:

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

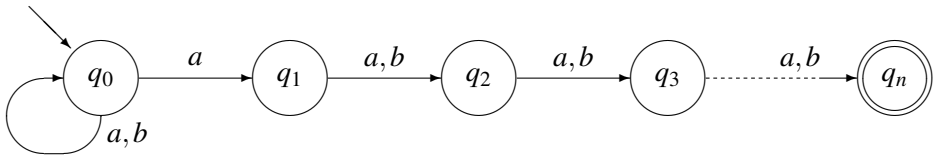

Figure 1: A 1NFA accepting the language $I_n = (a+b)^* a (a+b)^{n-1}$.

- 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 $n+_{...}$ 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 endmarker. 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 a 2DFA having approximatively the same size. So the example could suggest the possibility of replacing the nondeterminism in one-way automata by two-way motion.

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.

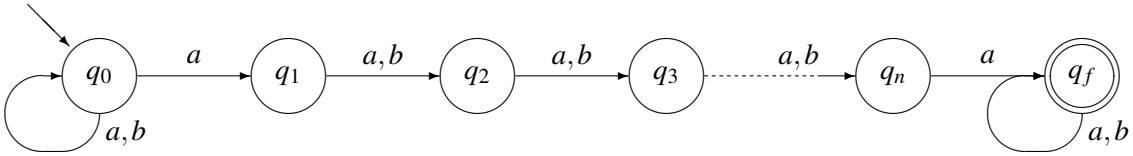

Figure 2: A 1NFA accepting the language $L_n = (a+b)^* a (a+b)^{n-1} a (a+b)^*$.

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 final control the last n input symbols. Hence, when in the state corresponding to $\sigma_1\sigma_2\ldots\sigma_n$ a new input symbol γ is read, the automaton moves to the state corresponding to $\sigma_2\ldots\sigma_n\gamma$. However, in the case $\sigma_1 = \gamma = a$ the automaton moves to its only final state, where it loops on each input symbol. 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.

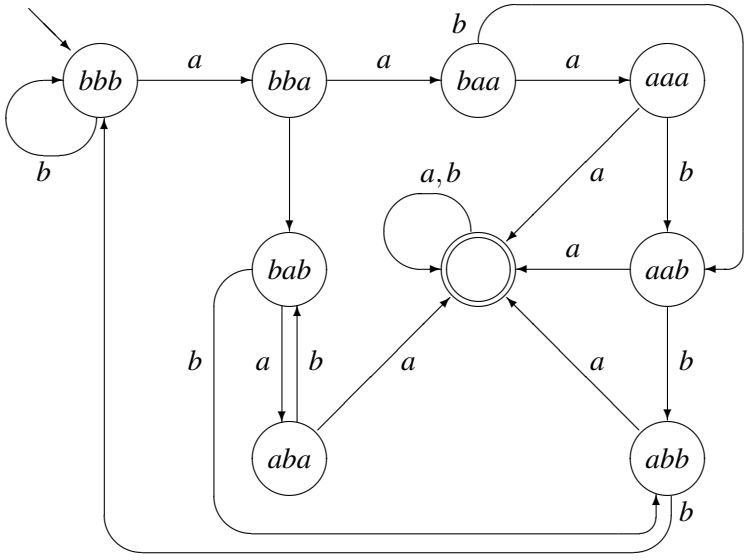

Figure 3: The 1DFA A_3 accepting the language $L_3 = (a+b)^* a (a+b)^2 a (a+b)^*$.

We can show that each automaton A_n is minimal. This can be done by using classical distinguishability arguments (see, e.g., [12]) 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 a different strategy to accept L_n using a two-way automaton. In the following let $w = w_1 w_2 \cdots w_m$, with $w_i \in \{a,b\}$, $i = 1, \ldots, m$, $m \geq 0$, be the input string for which we want to check the membership to L_n.

(i) *Naïf algorithm*
To decide whether a string $w \in \Sigma^*$ belongs to L, for $i = 1, \ldots, |w| - n$ we check if both symbols in positions i and $i+n$ are a's. The input is accepted if for at least one i the condition is satisfied. This algorithm can be implemented by a 2DFA that to move from position i to position $i+n$ counts n positions forward, and then counts $n-1$ positions backward to reach position $i+1$. Furthermore, when moving from position i to position $i+n$, the automaton needs to remember whether or not the symbol in position i is a. This leads to a 2DFA with $O(n)$ states which moves the input head along a zig-zag trajectory.

(ii) *An improved algorithm*
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 inspect the symbol w_{i+n}. Second, when a position i is found such that both symbols w_i and w_{i+n} are a's, the automaton can accept without checking the remaining positions. This leads to an algorithm which uses no more than $2n+_{\ldots}$ states.

(iii) *A different strategy: head reversals only at the endmarkers*
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 endmarkers. Hence, in this algorithm a computation is a sequence of left-to-right and right-to-left traversals of the input string, 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 $c_{\rightarrow}$ modulo n is kept in the finite control. This counter can be implemented using n states. However, 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 endmarker 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 $O(n^2)$ states, by keeping track in the finite control of the counter i, and by using $2n$ states for each sweep from left to right, and just one state for each sweep from right to left.

We can reduce the number of states to $O(n)$ by avoiding to store the counter i for sweeps. To this aim, also during sweeps from right to left we count the input length modulo n, by introducing another counter $c_{\leftarrow}$. After the ith sweep from left to right, the sweep from right to left starts by assigning to the counter $c_{\leftarrow}$ a value which depends on the current value of $c_{\rightarrow}$. In this way, at the end of the traversal from right to left, when the left endmarker is reached again, from the value of $c_{\leftarrow}$ it is possible to reconstruct the value of i, in order to prepare the next sweep.

3 Restricted Models

We now briefly present and discuss some restricted variants of two-way automata that have been considered in the literature.

Oblivious Automata

In the naïf algorithm (i) we described to recognize language L_n, 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 time t does not depend on the input content, but only on its length. A 2DFA with this property is called *oblivious*.

Sweeping Automata

A two-way automaton performing head reversal *only* when the input head is visiting the endmarkers is called *sweeping automaton*. This notion has been studied by Sipser [36]. In particular, for the language L_n above described, the recognition strategy (iii) is based on a sweeping 2DFA.

Rotating Automata

The method (iii) suggests another model, called *rotating automata* [21], which now we briefly mention. 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 with 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.

The reader can verify that languages I_n and L_n can be accepted by rotating automata with $O(n)$ 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 [8, 16], by introducing *outer nondeterministic automata* (2OFAs). In these models nondeterministic choices can be taken *only* when the input head is scanning the endmarkers. Hence, the transition on "real" input symbols are deterministic. This model does not have any restriction on head reversals, i.e., 2OFAs can 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 endmarker 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 $c_{\rightarrow}$ in a nondeterministic way, at the beginning of the computation with the head on the left endmarker.

Few Reversal Automata

All the models above discussed are defined by introducing structural restrictions. In the next model the restriction is of a different kind. On each computation we count the number of reversals of the input head during the computation. A 2DFA is said to be *few reversals* if the number of head reversals 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 a 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 [15].

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 $a^n b^n a^n b^n \ldots a^n b^n$). On the other hand, in the algorithm (iii) the number of reversals is bounded by $2n-1$, which is 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, where different space notions have been considered, see, e.g., [26]).

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 computation for a same input string, i.e., it is ambiguous.

4 Restrictions on the Simulating Machines

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

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. See Figure 4 for a summary of these and other separations. Their proofs use rather involved arguments.

Concerning few reversals 2DFAs, we already mentioned that a $o(n)$ upper bound on reversals implies a $O(1)$ upper bound [15]. 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 $n+_{\ldots}$ states that makes only

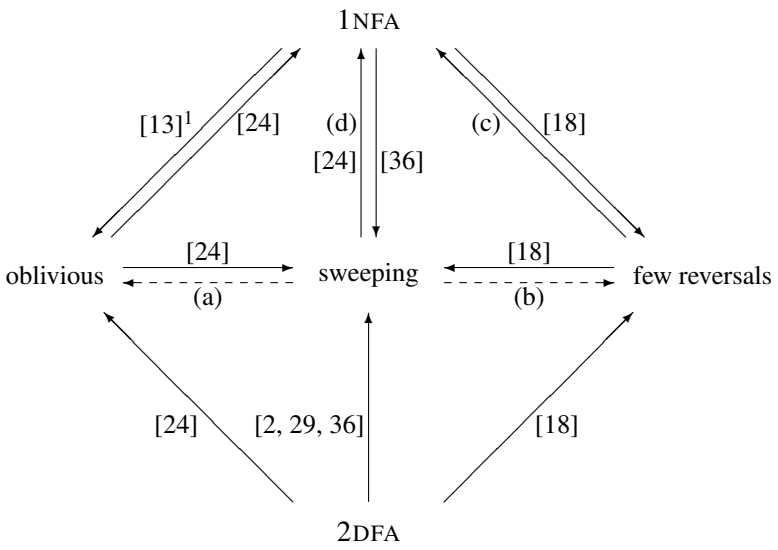

Figure 4: An arrow from a class A of machines to a class B denotes an exponential separation, i.e., the state cost of the simulation of machines in the class A by machines in the class B can be exponential. A dashed arrow indicates the existence of a polynomial simulation. The conversions corresponding 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.

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

In the case $k = 1$ this question has been solved by Balcerzak and Niviński [1], by proving an exponential separation. Recently Kapoutisis and Pighizzini extended this separation to each integer k, providing an infinite reversal hierarchy of 2DFAs [15]. 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 [4] and by Mereghetti and Pighizzini [27] and are summarized in Figure 5.

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 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

[1] 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 [13] it was proven that the simulation of 1NFAs by 2DFAs with a sublinear degree of non-obliviousness requires exponentially many states.

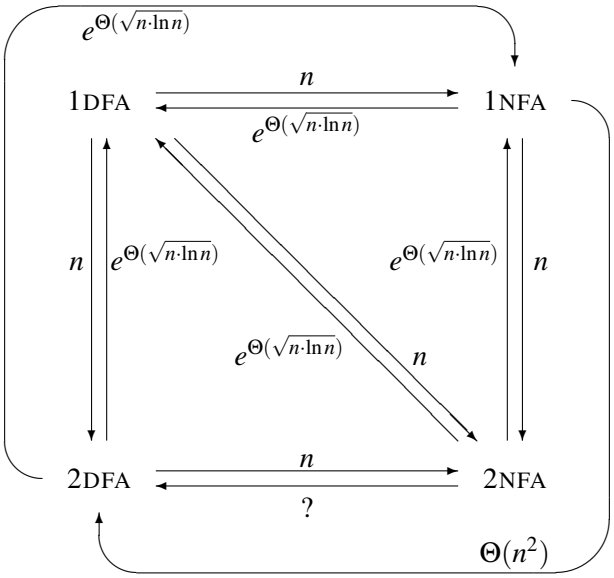

Figure 5: Costs of the *optimal* simulations between different kinds of *unary* automata. An arc labeled $f(n)$ from a vertex x to a vertex y means that a unary n-state automaton in the class x can be simulated by an $f(n)$-state automaton in the class y. The $e^{\Theta(\sqrt{n\cdot\ln n})}$ costs for 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 [4]. The $e^{\Theta(\sqrt{n\cdot\ln n})}$ cost for the simulation of 2NFAs and 1DFAs has been proved in [27]. 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.

costs as eliminating only one of them.

The question 1NFAs versus 2DFAs has been solved in the unary case in [4] 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. We will now discuss its status.

Normal Forms for Unary Nondeterministic Automata

The "simplicity" of automata over a unary alphabet, with respect to automata over a general alphabet, 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 [4]. In this form the transition graph of the 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 connects q to exactly one state in each of the k loops. Hence, a 1NFA in this form is allowed to make in its computation at most one nondeterministic choice, when it is in 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. 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.[2]

A generalization of the Chrobak normal form to the two-way case has been obtained by Geffert, Mereghetti, and Pighizzini [9]. 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 expect for a finite number of strings.

Theorem 5.1 ([9]) *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 endmarkers.*[3]
- *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 shows that M and its computations have a very simple structure (see also [11]). In particular, in each traversal of the input 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 $5n^2+_{\ldots}$ states, used to fix, in a preliminary scan of the input, the "errors".

We point out that for unary 2DFAs a similar normal form has been obtained in [23].

The normal form in Theorem 5.1 gives a strong simplification of unary 2NFAs which has been an important tool to prove several results on unary 2NFAs. First of all, it has been used in [9] to prove a subexponential, but still superpolynomial upper bound for the conversion of unary 2NFAs into equivalent 2DFAs:

Theorem 5.2 ([9]) *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 endmarker more than n times, then there exists a shorter accepting computation C' on the same input. In fact, in C at least a same state q must be visited twice with the head at the left endmarker and so the computation C' can be obtained by cutting the part of C between the two repetitions. Hence, if we assume acceptance on the left endmarker, to detect if an input string is accepted it is enough to check the existence of a computation starting in the initial state with the head on the left endmarker, ending in a final state with the head on the same endmarker, and visiting the left endmarker at most n times.

To this aim we can introduce a predicate $reachable(p,q,k)$ which holds true exactly when there is a path starting in the state p on the left endmarker, ending in the state q on the same endmarker and visiting it at most k times. This predicate can be recursively computed using a divide-and-conquere technique. The implementation of the resulting procedure leads to a 2DFA with $e^{O(\ln^2 n)}$ states.

[2] Besides [4], we refer the reader to [6, 7, 34] . All these papers present different algorithms and techniques for the conversion of unary 1NFAs into Chrobak normal form.

[3] In [36] the term *sweeping* was introduced for *deterministic* automata making head reversals only at the endmarkers. It is natural to extend this notion to the nondeterministic case, to denote 2NFAs making head reversals also at the endmarkers. In this case we have a further restriction: even nondeterministic decisions can taken only when the input head is scanning the endmarkers, not on "real" input symbols.

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 can fix the "errors", i.e., we can 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, in fact it grows faster than any polyomial.

The natural question is 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:

- 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 [10].
- Each unary n-state 2NFA can be transformed into an equivalent *unambiguous* 2NFA with a number of states polynomial in n [11].

The proof of the first result is given by using an *inductive counting* technique. The second result was obtained adapting one of constructions discussed in the next section (in particular, the construction used to prove Lemma 6.1).

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 briefly 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 $\mathsf{L} = \mathsf{NL}$ implies a polynomial simulation of 2NFAs by 2DFAs on "short" inputs.

This result was recently improved along the following lines.

(ii) Geffert and Pighizzini [11] considered the unary case. They proved that $\mathsf{L} = \mathsf{NL}$ would imply a polynomial simulation of unary 2NFAs by 2DFAs.[4] 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 [19] generalized the condition (i) by proving that $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$ 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 which agrees with A on strings of length at most n, where L/poly denotes the class

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

of languages accepted by deterministic logspace bounded machines that can access a *polynomial advice* [22].[5] Hence $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$ is *equivalent* to the existence of a state polynomial simulation of 2NFAs by 2DFAs on "short" inputs. Since $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{L}$ and $\mathsf{L} \subseteq \mathsf{NL}$, the *only-if* condition is stronger than the condition (i). Furthermore, in this case the converse also holds.

(iv) Quite recently, Kapoutsis and Pighizzini [16] proved the equivalence between $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$ and several other propositions. In particular, they show that $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$ is *equivalent* to the existence of 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 holds.

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

The Graph Accessibility Problem

A central role in the above mentioned investigations of the relationships between the L versus NL and 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 given directed graph $G = (V,E)$ and two fixed vertices $s,t \in V$, whether or not there exists a path from s to t.[6]

It is well known that GAP is an NL-complete problem [33]. 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 an 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.

First of all, in [11] 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.

If $\mathsf{L} = \mathsf{NL}$ then there is a logspace bounded deterministic machine that solves GAP. By restricting this machine to inputs encoding 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 is in 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 [11] 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 [8]. 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 [8, 11].[7]

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 [11] 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 works even under the weaker hypothesis $\mathsf{L}/\mathsf{poly} \supseteq \mathsf{NL}$, i.e.:

[5] A *polynomial advice* is a sequence of strings $(\alpha_n)_n \geq 0$, such that the length of $\alpha(n)$ 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 $\alpha(|x|)$.

[6] 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.

[7] These simulations do not require the assumption $\mathsf{L} = \mathsf{NL}$.

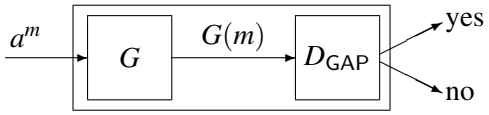

Figure 6: Simulating a unary 2NFA with a 2DFA of polynomial size, under the hypothesis $\mathsf{L} = \mathsf{NL}$

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

In [16], 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.
- 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 in 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, together with an encoding of the appropriate 2DFA B_n, where n is the number of vertices of G. 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. Such encoding is *the polynomial advice for M*. 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.

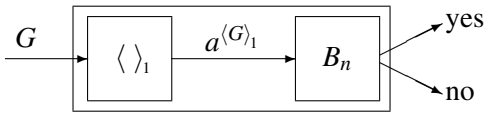

Figure 7: The machine M solving GAP using B_n as advice

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 V_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 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 that $K_n(\langle G\rangle_1) = G$.

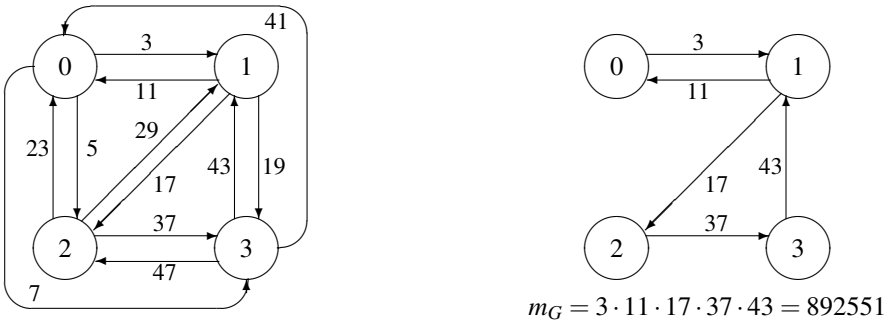

Figure 8: The complete graph K_4 with a subgraph G. The number $p_{(i,j)}$ associated with the edge (i, j) is the $(i \cdot n + j + 1)th$ prime number. In K_4 the edges (i, i) are not depicted.

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. 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 continues the same simulation after making the assignment $i \leftarrow j$, up to reach $i = n-1$. However, if in a step a pair $(i, j) \notin E$ is reached, 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 endmarkers. Furthermore, in each traversal A_n counts the input length modulo a prime number.
- On the endmarkers each state is interpreted either as a copy of a vertex in V_n or as an *hang* state.
- The automaton can traverse an input a^m from one endmarker in a copy of vertex i to the opposite endmarker in some copy of vertex j, without visiting the endmarkers in between, if and only if the number $p_{(i,j)}$ divides m. In particular, when the automaton is visiting one endmarker 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 endmarker, the automaton continues the simulation or hangs and rejects depending on the reached state.
- The computation starts on the left endmarker in a state representing the vertex 0.
- When a state representing the vertex $n-1$ is reached with the head on one of the endmarkers, the automaton A_n moves to an accepting state and stops the computation.

Using the properties related to the distribution of prime numbers, it can be proved that *the number of states of A_n is polynomial in n.*

Finally, we have to show that the machine M works in logarithmic space. Actually, we can observe that 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 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.[8]
- Due to a structural property of 2DFAs (see [23]), it is possible to modify the automaton B_n, still keeping polynomial its number of states, by replacing its unary input tape, with a tape containing a prime encoding of the unary input. 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 6.1 $\mathsf{L/poly} \supseteq \mathsf{NL}$ *if and only if the state cost of the simulation of unary* 2NFA*s by* 2DFA*s is polynomial.*

We address the reader to [16] 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, 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. In fact, in more than one case, techniques from space complexity turn to be useful to study two-way automata. For instance, the divide-and-conquere technique used to prove Theorem 5.2 derives from the proof of the famous Savitch Theorem [33]. The inductive counting tecnique used in [10] to obtain the polynomial complementation of 2NFAs derives from the argument used to prove the closure under complementation of nondeterministic space, the famous result independently proved in 1988 by Immerman [14] and Szelepcsényi [37].

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. This approach was suggested in the original paper by Sakoda and Sipser [32]. We recommend the recent paper by Kapoutisis [20] to the interested reader, 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`.

[8] More in general, a *prime encoding* of a unary string a^m is a sequence of the form $z_1\#z_2\#\cdots\#z_k$ where $z_1, z_2, \ldots, z_k$ are strings encoding in an *arbitrary order* the prime powers in the factorization of m.

References

[1] Marcin Balcerzak & Damian Niwinski (2010): *Two-way deterministic automata with two reversals are exponentially more succinct than with one reversal*. *Inf. Process. Lett.* 110(10), pp. 396–398, doi:10.1016/j.ipl.2010.03.008.

[2] Piotr Berman (1980): *A note on sweeping automata*. In J. W. de Bakker & Jan van Leeuwen, editors: *ICALP*, *Lecture Notes in Computer Science* 85, Springer, pp. 91–97, doi:10.1007/3-540-10003-2_62.

[3] Piotr Berman & Andrei Lingas (1977): *On the complexity of regular languages in terms of finite automata*. Technical Report 304, Polish Academy of Sciences.

[4] Marek Chrobak (1986): *Finite automata and unary languages*. *Theor. Comput. Sci.* 47(3), pp. 149–158, doi:10.1016/0304-3975(86)90142-8. Errata: [5].

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

[6] Pawel Gawrychowski (2011): *Chrobak normal form revisited, with applications*. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud & Denis Maurel, editors: *CIAA*, *Lecture Notes in Computer Science* 6807, Springer, pp. 142–153, doi:10.1007/978-3-642-22256-6_14.

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

[8] Viliam Geffert, Bruno Guillon & Giovanni Pighizzini (2012): *Two-way automata making choices only at the endmarkers*. In Adrian Horia Dediu & Carlos Martín-Vide, editors: *LATA*, *Lecture Notes in Computer Science* 7183, Springer, pp. 264–276, doi:10.1007/978-3-642-28332-1_23. Available at `http://arxiv.org/abs/1110.1263`.

[9] Viliam Geffert, Carlo Mereghetti & Giovanni Pighizzini (2003): *Converting two-way nondeterministic unary automata into simpler automata*. *Theor. Comput. Sci.* 295, pp. 189–203, doi:10.1016/S0304-3975(02)00403-6.

[10] Viliam Geffert, Carlo Mereghetti & Giovanni Pighizzini (2007): *Complementing two-way finite automata*. *Inf. Comput.* 205(8), pp. 1173–1187, doi:10.1016/j.ic.2007.01.008.

[11] Viliam Geffert & Giovanni Pighizzini (2011): *Two-way unary automata versus logarithmic space*. *Inf. Comput.* 209(7), pp. 1016–1025, doi:10.1016/j.ic.2011.03.003.

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

[13] Juraj Hromkovič & Georg Schnitger (2003): *Nondeterminism versus determinism for two-way finite automata: Generalizations of Sipser's separation*. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow & Gerhard J. Woeginger, editors: *ICALP*, *Lecture Notes in Computer Science* 2719, Springer, pp. 439–451, doi:10.1007/3-540-45061-0_36.

[14] Neil Immerman (1988): *Nondeterministic space is closed under complementation*. *SIAM J. Comput.* 17(5), pp. 935–938, doi:10.1137/0217058.

[15] Christos Kapoutsis & Giovanni Pighizzini (2012): *Reversal hierarchies for small 2DFAs*. In: *MFCS 2012*, Lecture Notes in Computer Science, Springer. To appear.

[16] Christos Kapoutsis & Giovanni Pighizzini (2012): *Two-way automata characterizations of L/poly versus NL*. In Edward A. Hirsch, Juhani Karhumäki, Arto Lepistö & Michail Prilutskii, editors: *CSR*, *Lecture Notes in Computer Science* 7353, Springer, pp. 217–228.

[17] Christos A. Kapoutsis (2005): *Removing bidirectionality from nondeterministic finite automata*. In Joanna Jedrzejowicz & Andrzej Szepietowski, editors: *MFCS*, *Lecture Notes in Computer Science* 3618, Springer, pp. 544–555, doi:10.1007/11549345_47.

[18] Christos A. Kapoutsis (2011): *Nondeterminism is essential in small 2FAs with few reversals*. In Luca Aceto, Monika Henzinger & Jiri Sgall, editors: *ICALP (2)*, *Lecture Notes in Computer Science* 6756, Springer, pp. 198–209, doi:10.1007/978-3-642-22012-8_15.

[19] Christos A. Kapoutsis (2011): *Two-way automata versus logarithmic space*. In Alexander S. Kulikov & Nikolay K. Vereshchagin, editors: *CSR*, *Lecture Notes in Computer Science* 6651, Springer, pp. 359–372, doi:10.1007/978-3-642-20712-9_28.

[20] Christos A. Kapoutsis (2012): *Minicomplexity*. In Martin Kutrib, Nelma Moreira & Rogério Reis, editors: *DCFS*, *Lecture Notes in Computer Science* 7386, Springer, pp. 20–42, doi:10.1007/978-3-642-31623-4_2.

[21] Christos A. Kapoutsis, Richard Královic & Tobias Mömke (2012): *Size complexity of rotating and sweeping automata*. *J. Comput. Syst. Sci.* 78(2), pp. 537–558, doi:10.1016/j.jcss.2011.06.004.

[22] R.M. Karp & R.J. Lipton (1982): *Turing machines that take advice*. In E. Engeler et al, editor: *Logic and Algorithmic*, L'Enseignement Mathématique, Genève, pp. 191–209.

[23] Michal Kunc & Alexander Okhotin (2011): *Describing periodicity in two-way deterministic finite automata using transformation semigroups*. In Giancarlo Mauri & Alberto Leporati, editors: *Developments in Language Theory*, *Lecture Notes in Computer Science* 6795, Springer, pp. 324–336, doi:10.1007/978-3-642-22321-1_28.

[24] Martin Kutrib, Andreas Malcher & Giovanni Pighizzini (2012): *Oblivious two-way finite automata: decidability and complexity*. In David Fernández-Baca, editor: *LATIN*, *Lecture Notes in Computer Science* 7256, Springer, pp. 518–529, doi:10.1007/978-3-642-29344-3_44.

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

[26] Carlo Mereghetti (2008): *Testing the descriptional power of small Turing machines on nonregular language acceptance*. *Int. J. Found. Comput. Sci.* 19(4), pp. 827–843, doi:10.1142/S012905410800598X.

[27] Carlo Mereghetti & Giovanni Pighizzini (2001): *Optimal simulations between unary automata*. *SIAM J. Comput.* 30(6), pp. 1976–1992, doi:10.1137/S009753979935431X.

[28] A. R. Meyer & M. J. Fischer (1971): *Economy of description by automata, grammars, and formal systems*. In: *SWAT '71: Proceedings of the 12th Annual Symposium on Switching and Automata Theory (swat 1971)*, IEEE Computer Society, Washington, DC, USA, pp. 188–191.

[29] Silvio Micali (1981): *Two-way deterministic finite automata are exponentially more succinct than sweeping automata*. *Inf. Process. Lett.* 12(2), pp. 103–105, doi:10.1016/0020-0190(81)90012-0.

[30] F.R. Moore (1971): *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), pp. 1211 – 1214, doi:10.1109/T-C.1971.223108.

[31] M. O. Rabin & D. Scott (1959): *Finite automata and their decision problems*. *IBM J. Res. Dev.* 3(2), pp. 114–125, doi:10.1147/rd.32.0114.

[32] William J. Sakoda & Michael Sipser (1978): *Nondeterminism and the size of two-way finite automata*. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman & Alfred V. Aho, editors: *STOC*, ACM, pp. 275–286, doi:10.1145/800133.804357.

[33] Walter J. Savitch (1970): *Relationships between nondeterministic and deterministic tape complexities*. *J. Comput. Syst. Sci.* 4(2), pp. 177–192, doi:10.1016/S0022-0000(70)80006-X.

[34] Zdenek Sawa (2010): *Efficient construction of semilinear representations of languages accepted by unary NFA*. In Antonín Kucera & Igor Potapov, editors: *RP*, *Lecture Notes in Computer Science* 6227, Springer, pp. 176–182, doi:10.1007/978-3-642-15349-5_12.

[35] J. C. Shepherdson (1959): *The reduction of two-way automata to one-way automata*. *IBM J. Res. Dev.* 3(2), pp. 198 –200, doi:10.1147/rd.32.0198.

[36] Michael Sipser (1980): *Lower bounds on the size of sweeping automata*. *J. Comput. Syst. Sci.* 21(2), pp. 195–202, doi:10.1016/0022-0000(80)90034-3.

[37] Róbert Szelepcsényi (1988): *The method of forced enumeration for nondeterministic automata*. *Acta Inf.* 26(3), pp. 279–284, doi:10.1007/BF00299636.