ELSEVIER Theoretical Computer Science 262 (2001) 1–24

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the power of Las Vegas II: Two-way finite automata [☆]

Juraj Hromkovič[a,*], Georg Schnitger[b]

[a] *Lehrstuhl für Informatik I, RWTH Aachen, Ahornstraße 55, 52074 Aachen, Germany*

[b] *Fachbereich Informatik, Johann Wolfgang Goethe Universität Frankfurt, Robert Mayer Straße 11-15, 60054 Frankfurt am Main, Germany*

Received August 1999; revised November 1999; accepted February 2000

Abstract

The investigation of the computational power of randomized computations is one of the central tasks of complexity and algorithm theory. While for one-way finite automata the power of different computational modes was successfully determined, one does not have any nontrivial result relating the power of determinism, Las Vegas and nondeterminism for two-way finite automata. The main results of this paper are as follows.

(i) If, for a regular language L, there exist small two-way nondeterministic finite automata for both L and L^{C}, then there exists a small two-way Las Vegas finite automaton for L.

(ii) There is a quadratic gap between nondeterminism and Las Vegas for two-way finite automata.

(iii) For every $k \in \mathcal{N}$, there is a regular language S_k such that S_k can be accepted by a two-way Las Vegas finite automaton with $O(k)$ states, but every two-way deterministic finite automaton recognizing S_k has at least $\Omega(k^2/\log_2 k)$ states.

© 2001 Elsevier Science B.V. All rights reserved.

1. Introductions and definitions

The comparative study of the computational power of deterministic, randomized and nondeterministic computations is one of the central tasks of complexity and algorithm theory. This is not surprising since a problem should be called tractable if there is a randomized polynomial-time algorithm solving it. In this paper we focus on the power

☆ The work on this paper has been supported by DFG-Project HR 14/3-2. An extended abstract of this paper has been presented at ICALP '99.

* Corresponding author.

E-mail address: jh@I1.informatik.rwth-aachen.de (J. Hromkovič).

of Las Vegas computations. [1] The central questions P versus ZPP and ZPP versus NP are unresolved. But for restricted models some progress has been achieved. [2] A linear relation between determinism and Las Vegas was established for the time complexity of CREW PRAMs [5] and one-way communication complexity [4]. A polynomial relation between determinism and Las Vegas is known for the combinational complexity of nonuniform circuits (see, for instance, [19]), two-way communication complexity [11], the size of one-way finite automata [4] and the size of OBDDs [4]. The last result is in contrast to the exponential gap between Las Vegas and determinism for one-time-only branching programs [15] that are a generalization of OBDDs. An exponential gap between Las Vegas and nondeterminism is known for one-way finite automata [4], two- and one-way communication complexity (see, for instance, [1, 7, 9, 13, 20]) and OBDDs [14]. [3]

In this paper we concentrate on two-way finite automata. In contrast to the well-developed theory of one-way finite automata we do not know to answer basic questions (like whether there is a polynomial relation between determinism and nondeterminism) for two-way automata. Only when restricting this model to so-called two-way sweeping finite automata, Sipser [17] has proved an exponential gap between determinism and nondeterminism. Whether an exponential gap exists in the general case is a famous old open problem [3, 10, 16, 17]. Two-way Las Vegas finite automata have not been considered up till now.

In this paper we consider four finite automata models: two-way deterministic finite automaton (2DFA), two-way Las Vegas finite automaton (2LVFA), two-way self-verifying nondeterministic finite automaton (2SVNFA) and two-way nondeterministic finite automaton (2NFA). For 2DFAs and 2NFAs we consider the standard definition used in the literature [8, 17]. The states of these automata are considered to be divided into three disjoint sets of working, accepting and rejecting states. No action is possible from any rejecting or accepting state. For convenience we assume that the nondeterministic choice of 2NFAs is bounded by 2. For every automaton A, $L(A)$ denotes the language accepted by A. A configuration of a finite automaton A is a pair (p, i_1), where p is a state of the automaton A and i_1 is the position of the head on the input tape.

For 2LVFAs and 2SVNFAs we consider the standard definitions as introduced in [4]. A 2SVNFA M can be viewed as a 2NFA with four types of states: working, accepting, rejecting and neutral ("I do not know") states. There is no possible move from accepting, rejecting and neutral states. M is not allowed to make mistakes: If there is a computation of M on an input x finishing in an accepting (rejecting) state, then x must be in $L(M)$ (x must not be in $L(M)$, i.e. $x \in (L(M))^{\complement}$). For every input

[1] The term "Las Vegas" was introduced by Babai [2] to distinguish probabilistic algorithms that reply correctly when they reply at all from those that occasionally (with some bounded probability) make a mistake.

[2] This paper is a continuation of [4], where one-way communication protocols, finite automata and ordered binary decision diagrams have been investigated.

[3] There are several further results for Monte Carlo randomized (one-sided error and two-sided error) computations but we do not deal with this form of randomization here.

y there is at least one computation of M that finishes either in an accepting state (if $y \in L(M)$) or in a rejecting state (if $y \notin L(M)$).

A 2LVFA A may be viewed as a 2SVNFA with probabilities assigned to every nondeterministic branching. The probability of a computation of A is defined through the transition probabilities of A. We require for every $y \in L(A)$ ($y \notin L(A)$) that A reaches an accepting (rejecting) state with a probability of at least $\frac{1}{2}$.

For any regular language L we define $s(L)$, $ns(L)$, $svns(L)$, and $lvs(L)$, respectively, as the size (the number of states) of a minimal one-way deterministic, nondeterministic, self-verifying nondeterministic, and Las Vegas finite automaton for L. Analogously, we define $s_2(L)$, $ns_2(L)$, $svns_2(L)$ and $lvs_2(L)$ for two-way versions of these automata. In [4] it is proved that $lvs(L) \geqslant \sqrt{s(L)}$ for every regular language L and the optimality of this lower bound is proved by constructing a language L' with $s(L') = \Omega((lvs(L'))^2)$. Moreover, exponential gaps between $svns(L)$ and $lvs(L)$, and between $svns(L)$ and $ns(L)$ are established in [4].

The proofs of these results are based on the fact that $s(L)$ is exactly the number of messages of the best uniform one-way communication protocol for L. For every mode of computation the number of messages of the best one-way protocol is a lower bound on the number of states of the corresponding finite automata model. Since we know the relations between determinism, Las Vegas, and nondeterminism for two-way communication protocols and the number of messages of the best two-way deterministic (Las Vegas, nondeterministic) protocols for L is a trivial lower bound on $s_2(L)$ ($lvs_2(L)$, $ns_2(L)$), the first idea could be to investigate the relations between $s_2(L)$, $lvs_2(L)$, $svns_2(L)$ and $ns_2(L)$ in the same way as in the one-way case. But in the two-way case this approach does not work because there is an exponential gap between $s_2(L)$ and the numbers of messages of the optimal two-way protocol.[4]

We focus on the relation between $lvs_2(L)$ on one side and the other measures on the other side. Our main results are as follows:

1. For every regular language L,

$$lvs_2(L) \leqslant 4\, svns_2(L) + 3.$$

2. There is a sequence of regular languages $\{M_k\}_{k=1}^{\infty}$ such that
   (i) $ns_2(M_k) \leqslant 2k + O(1)$,
   (ii) $lvs_2(M_k) \geqslant k^2 - 2$.
3. There is a sequence of regular languages $\{S_k\}_{k=1}^{\infty}$ such that
   (i) $lvs_2(S_k) = O(k)$,
   (ii) $s_2(S_k) = \Omega(k^2/\log k)$, and
   (iii) $lvs(S_k) = 2^{\Omega(k)}$.
4. There is a sequence of regular languages $\{L_k\}_{k=1}^{\infty}$ such that
   (i) $ns(L_k) = O(k^3)$ and $ns_2(L_k) = O(k)$,

[4] This contrasts to the one-way case where the complexity measures are even equal.

(ii) $ns((L_k)^{\complement}) \geqslant 2^k$ and so $s(L_k) \geqslant lvs(L_k) \geqslant 2^k$,
(iii) $lvs_2(L_k) = \mathrm{O}(k)$.

By the first result, small two-way nondeterministic finite automata for L and $L^{\complement}$ imply a small 2LVFA for L. The second result shows a quadratic gap between nondeterminism and Las Vegas. The third result shows an almost quadratic gap between determinism and Las Vegas for two-way finite automata. The last result shows that two-way Las Vegas can be essentially better than one-way nondeterminism. A similar result was established by Sakoda and Sipser [18] who proved an exponential gap between 2DFAs and NFAs by presenting a language B_n with $s_2(B_n) = \mathrm{O}(n^2)$ and $ns(B_n) \geqslant 2^n$ for every $n \in \mathbb{N}$. Note that our result is stronger than the consequence of [18] for Las Vegas, because we establish $lvs_2((L_k)^{\complement}) = \mathrm{O}(k)$ and $ns((L_k)^{\complement}) \geqslant 2^k$. A consequence is an exponential gap between one-way Las Vegas finite automata and two-way ones. Moreover, the language L_k is a candidate for an exponential gap between two-way Las Vegas and two-way determinism.

The paper is organized as follows. Section 2 verifies the first two results that compare nondeterminism and Las Vegas. In Section 3 the last two results focusing on the comparison of Las Vegas and determinism are presented. Section 4 is devoted to a discussion and the formulation of some open problems.

2. Las Vegas versus nondeterminism

In this section we show that in some sense the power of 2LVFAs is not too far from 2SVNFAs. First we present a simple observation.

Observation 1. *For any language L,*

$$\max\{ns_2(L), ns_2(L^{\complement})\} \leqslant svns_2(L) \leqslant ns_2(L) + ns_2(L^{\complement}) + 1.$$

Proof. Every 2SVNFA A can be transformed into a 2NFA B by adding the neutral states of A to the set of rejecting states of A. Obviously $L(A) = L(B)$ and $s(A) = s(B)$. If one takes the rejecting states of A as the accepting ones of an 2NFA C, and the accepting and the neutral states of A as the rejecting ones of C, then $L(C) = (L(A))^{\complement}$ and $s(C) = s(A)$. Thus, $svns(L) \geqslant \max\{ns(L), ns(L^{\complement})\}$.

Let E and F be two 2NFAs such that $L(E) = (L(F))^{\complement}$. A 2SVNFA D recognizing $L(E)$ can be constructed as follows. D connects a new initial state via ε-moves to the initial states of E and F. The set of accepting sets of D is exactly the set of accepting states of E, and the set of rejecting states of D is exactly the set of accepting states of F. All remaining nonworking states of E and F are neutral states of D. $\square$

We first establish a linear relation between $svns_2(L)$ and $lvs_2(L)$ for every regular language L, i.e. small nondeterministic finite automata for L and $L^{\complement}$ imply the existence of a small two-way Las Vegas finite automaton for L.

Theorem 1. *For every regular language L,*

$$svns_2(L) \leqslant lvs_2(L) \leqslant 4\, svns_2(L) + 3.$$

Idea of the Proof. $svns_2(L) \leqslant lvs_2(L)$ is obvious because every 2LVFA can be viewed as a 2SVNFA. To simulate an 2SVNFA by a 2LVFA it is sufficient to modify the method of Macarie and Seiferas [12] used to simulate two-way nondeterministic k-head finite automata by two-way one-sided error probabilistic k-head finite automata for $k \geqslant 2$. Since the proof is rather technical and no essentially new ideas are introduced we moved this proof to the appendix. □

Now, we show that nondeterminism can be more powerful than Las Vegas for two-way automata. Towards this goal we consider the following languages:

$$M_k = \{w \in \{0,1\}^* \mid \text{if } w = w_1 w_2 \dots w_m,\ m \geqslant 1,\ w_i \in \{0,1\}, \text{ for } i = 1,\dots,m, \\ \text{then there exists } r \in \mathbb{N} \text{ such that } w_{r(k^2-1)} = 1\}$$

for any $k \in \mathbb{N}$.

Theorem 2. *For any positive integer k,*
(i) $ns_2(M_k) \leqslant 2k + O(1)$,
(ii) $lvs_2(M_k) \geqslant k^2 - 2$.

Proof. (i) We first describe a 2NFA A_k that accepts M_k. A_k moves $r(k+1)$ steps to the right for some nondeterministically guessed $r \in \mathbb{N}$, and checks whether $w_{r(k+1)} = 1$. Then A_k moves $l(k-1)$ steps to the left for some $l \in \mathbb{N}$. If after these $l(k-1)$ steps to the left A_k reaches the left endmarker, then $r(k+1) = l(k-1)$, i.e. the index of the right most visited position is divisible by $(k^2-1) = (k+1)(k-1)$. Obviously, if $w \in L(M_k)$, then A_k can always guess the right position containing 1, and $2k + O(1)$ states are sufficient to move in cycles of lengths $k+1$ and $k-1$.

(ii) Following Observation 1 it is sufficient to prove $ns_2(M_k^{\mathsf{C}}) \geqslant k^2 - 2$. We prove it by contradiction. Let B_k be a 2NFA accepting M_k^{C} and $s(B_k) < k^2 - 2$. We consider the word $1^{k^2-2}0$ which does not belong to M_k, i.e. $1^{k^2-2}0 \in M_k^{\mathsf{C}}$. Using a standard pumping argument we show that B_k must accept $1^{k^2-2+k^2!}0$. Let D be an accepting computation of B_k on $1^{k^2-2}0$. We can write D as $C_1, C_2, \dots, C_m$ for some $m \in \mathbb{N}$, where

1. For every $i \in \{0, 1, \dots, \lceil m/4 \rceil - 1\}$, C_{1+4i} is a part of the computation D in which only the symbols of the input part $¢1^{k^2-2}$ were read and the last action in C_{1+4i} was the movement to the right after reading ¢.
2. For every $i \in \{1, 2, \dots, \lfloor m/2 \rfloor\}$, C_{2i} is a part of the computation D in which every 1 has been read at least once by M_k, no symbol different from 1 was read, and in the last step of C_{2i} the reading head was moved to a symbol different from 1.
3. For every $i \in \{0, 1, \dots, L(m-3)/4\}$, C_{3+4i} is a part of the computation D in which only the symbols of the input part $1^{k^2-2}0\$$ were read, and the last action in C_{3+4i} was the movement to the left after reading 0.

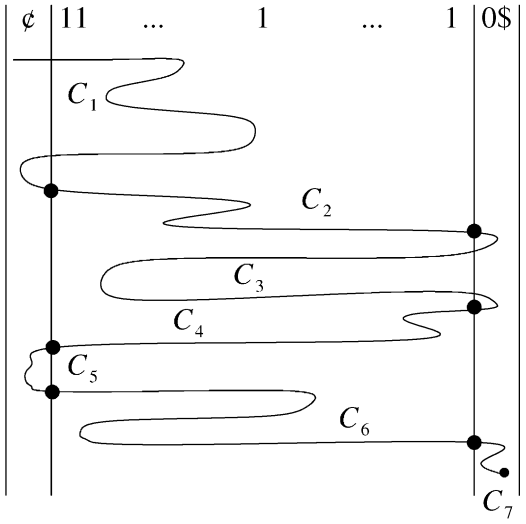

Fig. 1. A partition of a computation an the input $1^{k^2-2}0$.

The computation parts of the kind 2 are called *exhaustive walks* (on 1^{k^2-2}), and the other parts (of the kind 1 and 3) are called *local walks* (on 1^{k^2-2}). As one can see at Fig. 1 the reading head crosses the whole subword 1^{k^2-2} from one side to the other side in any exhaustive walk (parts C_2, C_4 and C_6 at Fig. 1).

Intuitively, in any local walk the number of 1's in the input cannot be measured. Since every exhaustive walk C_{2j} consists of at least k^2-2 steps, and $s(B_k) < k^2-2$, a state p_j has appeared at least twice in C_{2j}. Moreover, we may assume that B_k is in the state p_j in different configurations $\langle p_j, i_{j,1}\rangle$, and $\langle p_j, i_{j,2}\rangle$, $i_{j,1} < i_{j,2}$. So, C_{2j} may be written as

$$C_{2j} = C_{2j,1}, \langle p_j, i_{j,1}\rangle, C_{2j,2}, \langle p_j, i_{j,2}\rangle, C_{2j,3}.$$

Now, we consider the word $1^{k^2-2+k^2!}0$ that is obviously in M_k and so not in $M_k^{\complement}$. We construct an accepting computation H of B_k on $1^{k^2-2+k^2!}$.

Let, for any configuration C, $\tilde{C}$ denote the state of C. Analogously, for any computation $F = F_1, \ldots, F_m$, $\tilde{F} = \tilde{F}_1, \ldots, \tilde{F}_m$ is the sequence of states of F. So, the computation H on $1^{k^2-2+k^2!}$ can be determined by $\tilde{H}$ as follows:

$$\begin{aligned}\tilde{H} = {} & \tilde{C}_1, \tilde{C}_{2,1}, (p_1, \tilde{C}_{2,2})^{k^2!/(i_{1,2}-i_{1,1})} p_1, \tilde{C}_{2,3},\\ & \tilde{C}_3, \tilde{C}_{4,1}, (p_2, \tilde{C}_{4,2})^{k^2!/(i_{2,2}-i_{2,1})} p_2, \tilde{C}_{4,3},\\ & \tilde{C}_5, \ldots, \tilde{C}_m.\end{aligned}$$

This construction works because $0 < i_{j,2}, -i_{j,1} \leqslant k^2-2$ for every i, and so $i_{j,2} - i_{j,1}$ divides $k^2!$ for every j. $\square$

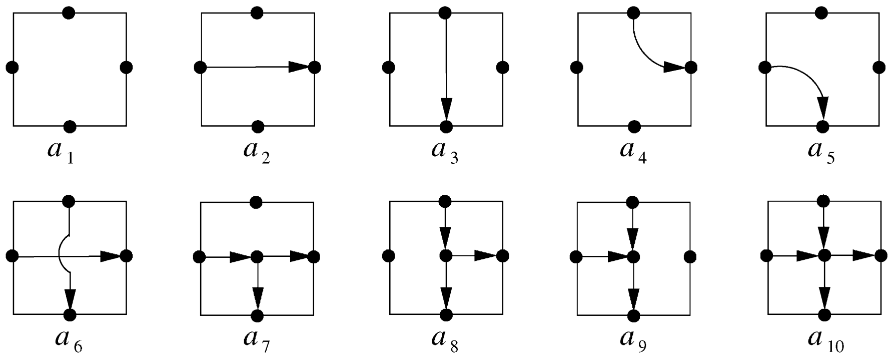

Fig. 2. The geometrical interpretation of the symbols of Σ.

3. Las Vegas versus determinism

The goal of this section is to find a family of languages $L_1, L_2, L_3, \ldots$, such that L_k can be recognized with $O(k)$ states by 2LVFAs but not within a linear number of states by 2DFAs. Following the result of Section 2 we see that we have to search for languages L_k such that both L_k and L_k^{C} are easy for two-way nondeterministic finite automata but not for deterministic automata. To achieve this goal we shall consider languages whose words define directed, acyclic graphs embedded in the two-dimensional grid. The idea is to define the membership of a word in L_k in such a way that one requires (for the acceptance) the existence of a directed path leading from the upper boundary to the lower boundary of the layout of the corresponding graph. Usually, to find a path nondeterministically is easy. But the main point is that we restrict the class of graphs considered (and their layouts, too) in such a way that it is also easy to nondeterministically verify the nonexistence of such paths. On the other hand, the class of graphs considered is sufficiently complicated for a superlinear lower bound on the number of states of 2DFAs.

In what follows, we consider the alphabets $\Sigma = \{\#, a_1, a_2, a_3, \ldots, a_{10}\}$ and $\Delta = \Sigma - \{a_6\}$, where the a_i's have the geometrical interpretations as depicted in Fig. 2. Each a_i represents a square of the layout of a graph in the two-dimensional grid. Each such square contains either four or five vertices. Each of the four boundaries of the square contains exactly one vertex. The fifth vertex, if any, lies in the middle of the square (see the interpretation of symbols a_7, a_8, a_9, a_{10}). Some directed edges between the vertices of the squares are allowed. Following Fig. 2, we see that the edges lead only from the left to the right or downwards. Note that this is a crucial fact in our construction. If we would allow edges in other directions, then we would not be able to find any simple 2LVFA for the language constructed. On the other hand if one would take still a harder restriction like the prohibition[5] of the use of a_6, then we would

[5] This corresponds to the requirement for the planarity of the layout.

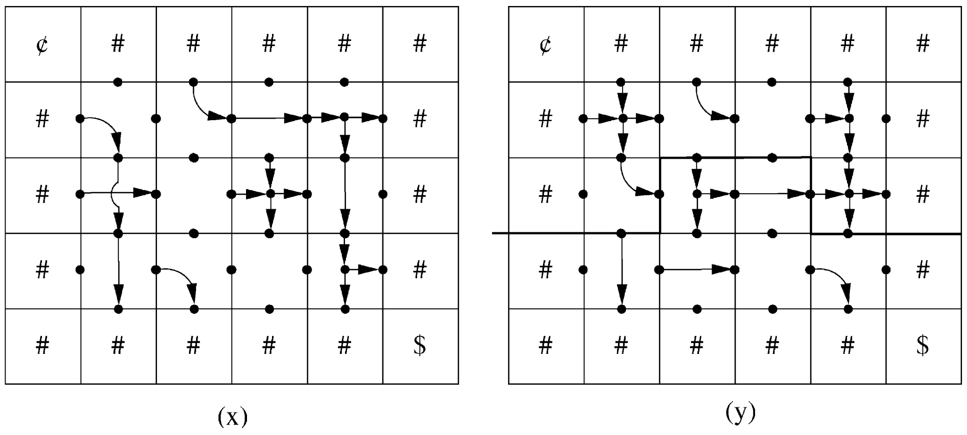

Fig. 3. The geometrical interpretation of the input words $x = \#^6 a_5a_4a_2a_7\ \#\#a_6a_1a_{10}a_3\ \#\#a_3a_5a_1a_8\#^6$ and $y = \#^6 a_{10}a_4a_1a_9\ \#\#a_4a_8a_2a_{10}\ \#\#a_3a_2a_1a_5\#^6$.

not be able to prove any superlinear lower bound on the size of two-way deterministic finite automata.

In what follows, we consider only words from

$$U = \bigcup_{i=1}^{\infty} U_i, \quad U_i = \#^{i+1}(\#\Sigma^i\#)^*\#^{i+1}.$$

Fig. 3 shows the geometrical interpretations for two words x and y from U_4. The symbol # defines the boundary of the layout and every subword from $\#\Sigma^i\#$ corresponds to one row of the picture. So, the words from U_i correspond to layouts with i columns and an arbitrary number of rows. For any $x \in L$, we denote by $G(x)$ the directed, acyclic graph determined by x, and $Pic(x)$ denotes the grid layout of $G(x)$ as depicted in Fig. 3. In what follows, we will speak about rows and columns of pictures (graph layout). The rows are ordered in the order of their occurrence in the input word. For instance, $a_5a_4a_2a_7$ is the first row of $Pic(x)$. The columns are ordered from the left to the right. For instance, the second column of $Pic(y)$ corresponds to $a_4a_8a_2$. For any row R of a $Pic(z)$ for a $z \in U_i$, we distinguish, with the obvious meaning, three classes of vertices of $G(x)$:

- the *upper vertices* of R (exactly i vertices),
- the *middle vertices* of R (j vertices with $i + 1 \leqslant j \leqslant 2i + 1$),
- the *lower vertices* of R (exactly i vertices).

So, the lower vertices of the mth row R_m are the upper vertices of the $(m + 1)$th row R_{m+1}. The only own vertices of every row are its middle vertices. Similarly, we speak about *left*, *middle* and *right* vertices for every column of a picture. The upper vertices of the first row of a picture $Pic(z)$, $z \in U$, are called the *sources of the graph* $G(z)$. The lower vertices of the last row of a picture $Pic(z)$, $z \in U$, are called the *destinations* *of the graph* $G(z)$. For every $i \in \mathbb{N}$ and every $z \in U_i, s_1(z), s_2(z), \ldots, s_i(z)$ $(d_1(z), d_2(z), \ldots, d_i(z))$ denote of the sources (the destinations) of $G(z)$. Note, that $s_i(z)$ and $d_i(z)$ are the middle vertices of the ith column of $Pic(z)$.

Before defining some special languages we give an obvious observation.

Observation 2. *For every* $i \in \mathbb{N}$, *the language* U_i *can be accepted by a one-way deterministic finite automaton with* $3(i+2)$ *states.*

Observe, that one can interpret a two-way finite automaton A as moving within $Pic(x)$ instead on x. Obviously, each horizontal movement in the picture corresponds to one step of A on x. To realize a vertical movement A needs to move $k+2$ steps using $k+2$ special states.

To get a quadratic gap between 2LVFAs and 2DFAs we consider very special graph layouts. Let, for every even $k \in \mathbb{N}$,

$$\begin{aligned}S_k = \{&x \in U_k \mid x = \#^{k+1}\#x_1\#x_2\#\ldots\#x_{k/2+4}\#\#^{k+1}, |x_i| = k \text{ for } i = 1,\ldots,k/2+4,\\ &x_1, x_{k/2+4} \in a_1^* a_3 a_1^*, x_2 = (a_3a_1)^{k/2}, x_{k/2+3} = (a_1a_3)^{k/2},\\ &\text{for } i = 3,4,\ldots,k/2+2, x_i \in (a_0a_3)^{i-3}a_4b(a_6b)^*, b \in \{a_6, a_{10}\},\\ &G(x) \text{ contains such a directed path } P \text{ from a source to a destination,}\\ &\text{that } P \text{ contains exactly two direction changes (vertical } \to \text{ horizontal}\\ &\to \text{ vertical) and the first change of the direction is given by } a_4\}.\end{aligned}$$

Observe, that for every $x \in S_k$, $G(x)$ contains exactly one path from a source to a destination, because there is only one source s_{2i+1} with outdegree 1 (given by x_1) and only one destination d_{2j} with indegree 1 (given by $x_{k/2+4}$). Moreover, if a consistent path exists, then it must go vertically from s_{2i+1} to a_4 in the $(i+3)$th row, and it must again change the direction in the square $(i+3, 2j)$ in order to reach d_{2j}. So, if one knows the indices $2i+1$ and $2j$, then it is sufficient to look at the unambiguously given sequence of squares starting in $(1, 2i+1)$ and finishing in $(k/2+4, 2j)$ in order to check the existence of the special path defined by S_k. Observe, that the middle part $x_2\#x_3\#\ldots\#x_{k/2+3}$ of x may connect any upper vertex of the second row laying in an odd column $2j-1$ $(j=1,\ldots,k/2)$ with any lower vertex of the $(k/2+3)$th row that lies in one of the columns $2j, 2j+2, \ldots, k$. An example of such a middle part is given in Fig. 4. For instance, there is the possibility to connect s_1 with d_2 and d_6, but not with d_4 and d_8, by a consistent, directed path.

Theorem 3. *For any positive, even integer* k,
(i) $svns(S_k) = O(k^3)$ *and* $svns_2(S_k) = O(k)$,
(ii) $lvs_2(S_k) = O(k)$,
(iii) $s_2(S_k) = \Omega(k^2/\log k)$,
(iv) $s(S_k) \geqslant 2^{k/2}$ *and* $lvs(S_k) \geqslant 2^{k/4}$.

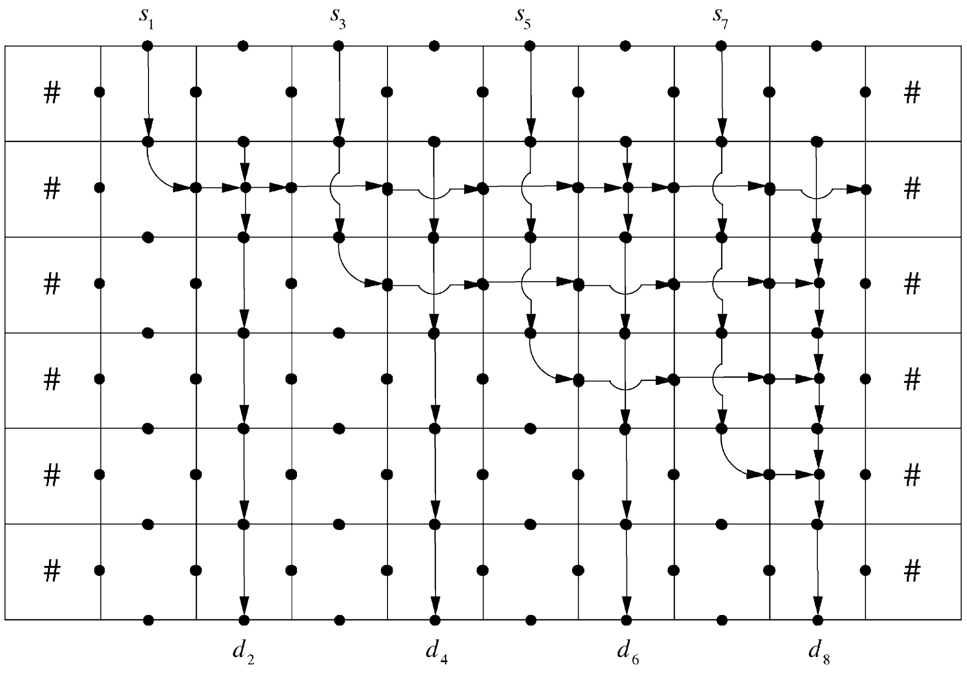

Fig. 4.

Proof. (i) Two runs with O(k) states from the left to the right on an input x are sufficient to check (deterministically) whether x has the form of words in S_k. The search for a consistent path in $G(x)$ starts in the only upper vertex of the first row with outdegree 1. Let this vertex be s_i. The path unambiguously continues downwards via the ith column until a_4 is reached in the $(i+3)$th row. Then the self-verifying nondeterministic automaton A accepting M_k moves to the right. In every even column A nondeterministically decides whether to continue to the right or downwards. If A decides to continue downwards, then A stores the contents of this position (a_6 or a_{10}) in its state. If the decision was correct, then A reaches the only destination. A accepts if the stored symbol was a_{10}, and A rejects if the stored symbol was a_6. So, a self-verifying nondeterministic automaton can determine whether $G(x)$ contains a path from a source to a destination with O(k) states.

Using O(k^3) states, a one-way self-verifying nondeterministic FA accepting S_k can simulate the three runs of the 2SVNFA in one run over the input.

(ii) $lvs_2(S_k) = \mathrm{O}(k)$ is a direct consequence of (i) and Theorem 1.

(iii) Let D be a 2DFA accepting L_k with a set Q of states. For every input subword from $F_k = (\#\Sigma^k\#)^*$ we define the pattern of y, $Pat(y)$, as follows. For every state $q \in Q$ and every of the two positions in which D can move to y and start to read it, we determine the state in which D leaves and the side on which B leaves y. Obviously, there are at most $(2|Q|)^{2\cdot|Q|}$ different patterns. If one considers the relation xR_Dy on F_k if and only if x and y have the same pattern, then F_k consists of equivalence classes corresponding to the patterns. It can be easily proved that if $x, y \in F_k$ have the same pattern, then for every $z_1, z_2 \in \Sigma^*$

$$z_1 x z_2 \in L \Leftrightarrow z_1 y z_2 \in L_k.$$

Now, we consider a large set S_k of words from F_k so that for every two $x_1, x_2 \in S_k$, $x_1 \neq x_2$ implies $Pat(x_1) \neq Pat(x_2)$. For simplicity we assume k is even. To every symmetric $k/2 \times k/2$ Boolean matrix $[a_{ij}]_{k/2 \times k/2}$ we assign words $x \in F_k$ that have the following properties:

(1) if $a_{ij} = a_{ji} = 1$, $i \leqslant j$, there is a directed path from the $(2i-1)$th source of $Pic(x)$ to the $2j$th destination of $Pic(x)$, and
(2) there are no directed paths from the sources of $Pic(x)$ to the destinations of $Pic(x)$ besides the paths determined by (1).

For every symmetric $k/2 \times k/2$ Boolean matrix M there exists at least one word from F_k corresponding to M. Fig. 4 describes the idea of constructing a corresponding word for a given matrix M. For instance $Pic(x)$ in Fig. 4 corresponds to the matrix

$$M_x = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

We show, for all $x, y \in F_k$, that $M_x \neq M_y$ implies $Pat(x) \neq Pat(y)$. Let $M_x \neq M_y$, $M_x = [v_{ij}]_{k/2 \times k/2}$ and $M_y = [w_{ij}]_{k/2 \times k/2}$. Without loss of generality, we may assume that there exist $i, j \in \{1, \ldots, k/2\}$, $i \leqslant j$, such that $1 = v_{ij} \neq w_{ij} = 0$. So, there is a directed path from $(2i-1)$th source of $Pic(x)$ to the $(2j)$th destination of $Pic(x)$, but there is no path from the $(2i-1)$th source of $Pic(y)$ to the $(2j)$th destination of $Pic(y)$. Now, we choose

$$z_1 = \#^{k+1}\#(a_1)^{2j-1}a_3(a_1)^{k-2j}\#\#^{k+1}$$

and

$$z_2 = \#(a_1)^{2j-1}a_3(a_1)^{k-2j}\#\#^{k+1}.$$

One observes, that $z_1 x z_2 \in S_k$ but $z_1 y z_2 \notin S_k$.

So, the number $(2|Q|)^{2\cdot|Q|}$ of distinct patterns has to be at least as large as the number $2^{\binom{k/2}{2}}$ of different symmetric $k/2 \times k/2$ Boolean matrices. Thus, $|Q| = \Omega(k^2/\log_2 k)$.

(iv) If one takes the prefix $\#^{k+1}\#a_3(a_1)^{k-1}\#$, then there are $2^{k/2}$ possible choices for the middle part $\#x_2\#x_3\#\ldots\#x_{k/2+3}$ resulting in different sets of paths from s_1 to the lower vertices of the $(k/2-3)$th row on the $k/2$ even positions. So, $s(S_k) \geqslant 2^{k/2}$. In [4] it is proved that $lvs(L) \geqslant \sqrt{s(L)}$ for every regular language L and so $lvs(S_k) \geqslant 2^{k/4}$. $\square$

Sakoda and Sipser [18] proved an exponential gap between 2DFAs and NFAs by constructing a language B_n with $s_2(B_n) = O(n^2)$ and $ns(B_n) \geqslant 2^n$ for every $n \in \mathbb{N}$. This directly implies an exponential gap between $ls_2(B_n)$ and $ns(B_n)$. We improve this result for Las Vegas automata by proving $ls_2(L_k^{\complement}) = \mathrm{O}(k)$ and $ns(L_k^{\complement}) \geqslant 2^k$ for the following languages L_k, $k \in \mathbb{N}$.

Let, for every $k \in \mathbb{N}$,

$$L_k = \{x \in \Delta^* \mid x \in U_k \text{ and } G(x) \text{ contains a directed path starting in a source of } G(x) \text{ and finishing in a destination of } G(x)\}.$$

Let $L = \bigcup_{k=1}^{\infty} L_k$ and $\tilde{U} = U \cap (\Delta \cup \{\#\})^*$.

There are two reasons to consider the languages L_k. The first reason is to show an exponential gap between 2LVFAs and NFAs ($ns(L_k^{\complement}) = 2^{\Omega(lvs_2(L_k))}$). The second and major reason is that we conjecture that L_k is a candidate for an exponential gap between 2DFAs and 2LVFAs. If one is interested in the exponential gap between NFAs and 2LVFAs only, then it is sufficient to take the following languages:

$$D_k = L_k \cap \{a_1, a_{10}, \#\}^*$$

for any $k \in \mathbb{N}$, and several of the following technicalities may be essentially reduced.

Observation 3. *If $z \in L_k$, $k \in \mathbb{N}$, then there exist l, m, $1 \leqslant l \leqslant m \leqslant k$ such that $G(x)$ has a path from $s_l(z)$ to $d_m(z)$.*

Proof. The proof is obvious, because we can move only downwards or from the left to the right in $Pic(x)$. □

Observation 4. *For every $z \in L$, $G(z)$ is acyclic.*

Before formulating the next theorem we shall show that the nonexistence of any path from a source of a $G(z)$ to a destination of a $G(z)$, $z \in \tilde{U}$, implies the existence of a special cut of $Pic(z)$. In this way, we reduce a task defined by the universal quantifier (every path starting in a source does not reach any destination) to a task defined by the existential quantifier.[6]

Definition 1. Let $z \in \tilde{U}$. A *cut C of $Pic(z)$* is a path consisting of the undirected grid-edges of $Pic(z)$ having the following properties:

(i) C divides the vertices of $G(z)$ into three disjoint groups Up_C, Low_C, and Mid_C by removing the vertices of Mid_C lying on the grid-edges of C,

(ii) $Up_C \cup Mid_C$ contains all sources of $G(z)$,

(iii) Low_C contains all destinations of $G(z)$, and

(iv) for every node u in Mid_C, if there is an edge (v, u) in $G(x)$ with $v \in Up_C$ or u is a source, then u has outdegree 0.

An example of a cut is indicated in Fig. 3 (y) by a thick horizontal line. The picture $Pic(x)$ at Fig. 3 (x) does not contain any cut.

[6] Which is very convenient for a nondeterministic automaton.

Lemma 1. *For every $k \in \mathbb{N}$ and every $x \in L_k$, $Pic(x)$ does not contain any cut.*

Proof. The proof directly follows from the definitions of $G(x)$, $Pic(x)$ and Definition 1. □

In what follows, we shall show that if $x \in (U_k \cap \tilde{U}) - L_k$ (i.e., if $G(x)$ does not contain any path from a source of $G(x)$ to a destination of $G(x)$), then there exists a special, nice cut of $Pic(x)$ that can be found by a simple strategy. This will be useful for the proof of our next theorem. Note, that we can always consider a cut as a path of grid-edges, that starts from the left border of the picture and ends on the right border of the picture.

Definition 2. Let $z \in \tilde{U}$. A vertex u of $G(z)$ is called a 1-*vertex* if there exists a directed path from a source of $G(z)$ to u. A vertex v of $G(z)$ is called a 0-*vertex*, if there exists no directed path from any source of $G(z)$ to v.

Observation 5. *A word z belongs to $(U_k \cap \tilde{U}) - L_k$ for some $k \in \mathbb{N}$ if and only if all destinations of $G(z)$ are* 0*-vertices.*

Observation 6. *For any cut C of $Pic(z)$, with $z \in \tilde{U}$, every vertex in Low_C is a* 0*-vertex.*

Definition 3. Let $z \in \tilde{U}$. A horizontal edge e of the grid of $Pic(z)$ is called the 0-*maximal* edge of a column H if
(i) e lies in the column H,
(ii) all middle and right vertices of H lying below e are 0-vertices (the vertex involved in e may be a 1-vertex) and
(iii) any grid-edge of H lying above e does not have property (ii).

Let $y \in \tilde{U}$. A cut of $Pic(y)$ is called 0-*maximal* if it contains the 0-*maximal* edge of every column of $Pic(y)$.

The cut C of $Pic(y)$ in Fig. 3 is not 0-maximal. This is because the cut in the third column does not contain the 0-maximal edge of that column. The 0-maximal edge of the third column is the upper edge containing the source $s_3(y)$ of this column. All other columns of $Pic(y)$ have their 0-maximal edges in C.

Lemma 2. *For every $k \in \mathbb{N}$ and every $x \in (U_k \cap \tilde{U}) - L_k, Pic(x)$ contains a* 0*-maximal cut.*

Proof. A more important fact than the almost obvious assertion of Lemma 2 is that one can construct a 0-maximal cut C from given (precomputed) 0-maximal edges by a simple deterministic strategy.

Let e_i be the 0-maximal edge of the ith column of $Pic(x)$ for $i = 1, \ldots, k$. C starts with e_1 and finishes with e_k. So, it is sufficient to explain how to connect e_i and e_{i+1} via grid-edges for $i = 1, \ldots, k-1$. Since $x \notin L_k$ no destination of $G(x)$ lies on e_i's.

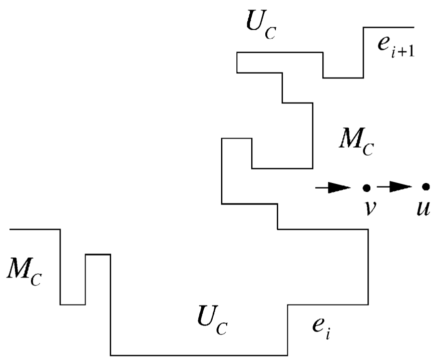

Fig. 5. A possible path around 0-vertices connected to the 0-vertex v in the $(i+1)$th column.

If e_i and e_{i+1} are on the same horizontal level, then we are done. If the horizontal level (common part of two rows) of e_{i+1} lies below the horizontal level of e_i then the situation is again simple. One moves from the right end of e_i down via grid-edges between the ith column and the $(i+1)$th column until e_{i+1} is reached. Moving down we do not need to take care of the contents of adjacent squares because the vertices of Low_C are on the left side of the vertical grid-edges used in the connection and $G(x)$ does not have any edge directed from the right to the left.

If the horizontal level of e_{i+1} lies above the horizontal level of e_i, then we try to go upwards via grid-edges between the ith column and the $(i+1)$th column. Now, we have to be careful because the area of Low_C is lying on the right side. So, we move upwards via a vertical grid-edge g only if the vertex v of g has indegree zero or outdegree zero. If there are two edges (w,v) and (v,u) adjacent to v we cannot go up (see property (iv) of a cut in Definition 1). Since u is a middle vertex or a right vertex of the $(i+1)$th column and u lies below e_{i+1}, u is a 0-vertex. Because of the existence of the path w,v,u in $G(x)$, the vertices w and v must be 0-vertices.

Now, we draw a path going around all 0-vertices having a directed path to v in $G(x)$, and reaching again the left border of the $(i+1)$th column. We start this drawing procedure called Go Around by moving to the left. Note, that the movement to the left is allowed now without any restriction, because Low_C is considered to be above the horizontal grid-edges we use (see Fig. 5).

One can simply determine the set $Ver(v)$ of all 0-vertices having a directed path to v. No source is among these vertices and all these vertices lay in the rectangle of $Pic(z)$ with v as the left-down corner of $R(v)$. So, one can easily draw the boundary (consisting of grid-edges) around these special 0-vertices by following the next rules:

(i) if the last step was the movement to the left, you never continue downwards,
(ii) if the last step was the movement downwards, you never continue to the left, and
(iii) if you move to the right or upwards via a grid-edge h, then the vertex laying on h does not have both indegree and outdegree 1.

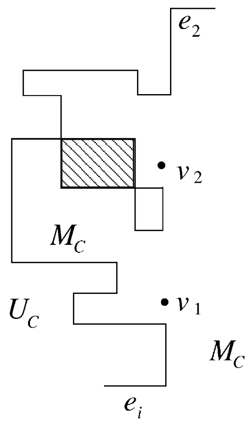

Fig. 6. Two different boundaries built by two applications of Go Around may cross each other.

We can draw this boundary with rules (i) and (ii), because all edges of $G(x)$ are directed from the left to the right or downwards. Because of property (iii) we are sure that the bounded area $Area(v)$ containing $Ver(v)$ does not contain any 1-vertex[7] as an inner vertex.

So the boundaries built by the procedure Go Around is a consistent candidate for a part of the cut C.

After finishing Go Around we are again on the left side of the $(i+1)$th column above v. If we have finished above e_{i+1} then we simply go down to e_{i+1} and finish the attempt to connect e_i and e_{i+1}. Otherwise, we try to continue upwards to e_{i+1}. If it is impossible we again use the drawing procedure Go Around for a new left vertex of the $(i+1)$th column. It can happen that some areas $Area(v_1)$ and $Area(v_2)$ overlap (see Fig. 6). But in this case we may take the common boundary bounding the union of these two areas. So, in this way we find a part of the cut C that connects e_i with e_{i+1}.

Let, for every $i \in \{1,\ldots,k-1\}$, P_i be the path on grid-edges that connects e_i and e_{i+1}. To complete the proof it remains to show that P_i and P_j do not have any common part if $i \neq j$. Let us assume the opposite as depicted at Fig. 7. Making a cycle we obtain a closed area H bounded by a part of P_i, the whole $P_{i+1},\ldots,P_{j-2}$, a part of P_{j-1} and by the 0-maximal edges $e_{i+1},\ldots,e_{j-1}$. We claim that all vertices of $G(x)$ laying in H are 0-vertices. This is true because a vertex of H can be reached from a source only from the left side or from the upper side. But on the left side we have the part of the ith column whose middle and right vertices are 0-vertices. Following the construction of P_{j-1} we know that all nodes laying on the horizontal grid-edges used for the movement to the left are 0-vertices. So, there is no possibility to reach any vertex of H from a

[7] Observe at Fig. 5 that moving upwards Low_C is always on the right side of the grid-edge used and moving to the right Low_C is always below the grid-edge used.

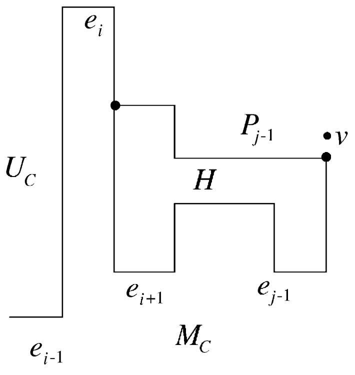

Fig. 7. An impossible situation.

1-vertex. But this is a contradiction with the 0-maximality of e_{i+1} because the whole intersection of H and the $(i+1)$th column lies above e_{i+1} and it contains 0-vertices only. □

Now, we are ready to formulate our last result.

Theorem 4. *For every* $k \in \mathbb{N}$,

(i) $ns_2(L_k) = O(k)$ *and* $ns(L_k) = O(k^2)$,

(ii) $ns((L_k)^{\complement}) \geqslant 2^k$ *and so* $s(L_k) \geqslant lvs(L_k) \geqslant 2^k$,

(iii) $lvs_2(L_k) = O(k)$.

Proof. (i) To decide whether $x \in U_k$ or not it is sufficient to read once the input from the left to the right. This can be done deterministically with $3(k+2)$ states (see Observation 1). If one knows that $x \in U_k$ then a NFA A can follow any path of $G(x)$. If the path is going to the right, A moves its head to the right. If the path is going downwards, then A moves $k+1$ steps to the right in order to achieve the corresponding square of the grid. So, if there is a path from a source to a destination, then A can nondeterministically find it.

(ii) It is sufficient to show that $ns((L_k)^{\complement}) \geqslant 2^k$. Let us consider 2^k words from $\#^{k+1}\#\{a_1, a_3\}^k\#$ as possible input prefixes. Clearly, these 2^k words $w_1, w_2, \ldots, w_{2^k}$ correspond to the 2^k different assignments of 1- and 0-vertices to the lower vertices of the first row. For every $w_i = \#^{k+2} w_{i1} w_{i2} \ldots w_{ik}\#$ we consider $\overline{w_i} = \#\overline{w_{i1}}\,\overline{w_{i2}} \ldots \overline{w_{ik}}\#^{k+2}$, where $\overline{w_{ij}} = a_1$ if $w_{ij} = a_3$ and $\overline{w_{ij}} = a_3$ if $w_{ij} = a_1$ for $j = 1, \ldots, k$. It is sufficient to observe that $w_i\overline{w_i} \notin L_k$ for any $i \in \{1, 2, \ldots, k\}$, but $w_i\overline{w_j} \in L_k$ for every $i \neq j$, $i, j \in \{1, 2, \ldots, k\}$. (The $2^k \times 2^k$ communication submatrix M for $(L_k)^{\complement}$ with rows corresponding to $w_1, \ldots, w_{2^k}$ and columns corresponding to $\overline{w_1}, \ldots, \overline{w_{2^k}}$ is the diagonal matrix [7, 9]. Every one-way nondeterministic protocol needs 2^k different messages to recognize M and the number of states of every NFA is at least as large as the number of messages of the optimal one-way nondeterministic protocol [7, 4].)

(iii) In (i) we have already shown that there is a 2NFA accepting L_k with $O(k)$ states. Following Observation 1 and Theorem 1 it is sufficient to show that there is a 2NFA B accepting $(L_k)^{\mathrm{C}}$ with $O(k)$ states. Obviously, to recognize that $x \notin U_k$, $O(k)$ states are sufficient. It remains to find a "proof" that $x \notin L_k$ if $x \in U_k \cap \tilde{U}$. Clearly, the existence of a cut of $Pic(x)$ is the proof of the fact $x \notin L_k$. To search for a cut B proceeds by the following 10 rules:

The start rule:

1. Moving downwards B determines the 0-maximal edge of the first column and moves to the right on this edge.

The movement rules:

2. B always remembers in its states the direction of the last movement on the grid-edges.
3. B never traverses the last grid-edge in the reverse direction.
4. B never moves to the left when the preceding movement was downwards.
5. B never moves downwards when the preceding movement was to the left.
6. B never moves to the right or upwards via a grid-edge h, when the vertex laying on h has both indegree and outdegree 1.
7. In every point of the grid B guesses an allowed direction to continue with.

The acceptance rules:

8. If B reaches a grid-edge containing a destination, then B halts and rejects. If B reaches a grid-edge containing a source with outdegree 1, then B halts and rejects.
9. If B reaches the right boundary of $Pic(x)$, then B halts and accepts.
10. If there is no possibility to continue in an inner point of the grid, then B halts and rejects.

Following the proof of Lemma 2 and the strategy of B described above we can be sure that if $x \in (U_k \cap \tilde{U}) - L_k$, then B will find a cut[8] of $Pic(x)$.

It remains to prove that if B reaches the right boundary of $Pic(x)$ by a path (crossing possibly several times itself), then $x \notin L_k$. This is obvious for any simple path found by B, because of rules 4, 5 and 6. However the path B may cross itself. Let v_1 be the first crossing point of the trajectory of B (see Fig. 8). The trouble is when visiting v_1 the second time B has another view on the layout of M_U than when visiting v_1 for the first time. We distinguish two possibilities according to the direction of the movement in which the point v_1 was repeatedly visited. Because of rules 4 and 5, this cannot happen by moving upwards or to the right:

1. Consider that B reaches v_1 for the second time after moving to the left (Figs. 8 and 9). Then the next part of the trajectory of B lies in Low_C. This means that if B wants to reach the right side of $Pic(x)$ then B must cross its trajectory again in a point v_2. Note, that this is due to rules 4 and 5 forbidding to move downwards

[8] Following the above rules of B one can be sure that B will find a 0-maximal cut of $Pic(x)$.

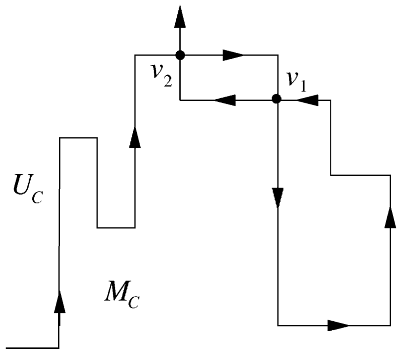

Fig. 8. A trajectory of the movement of B that crosses itself, after some movement to the left.

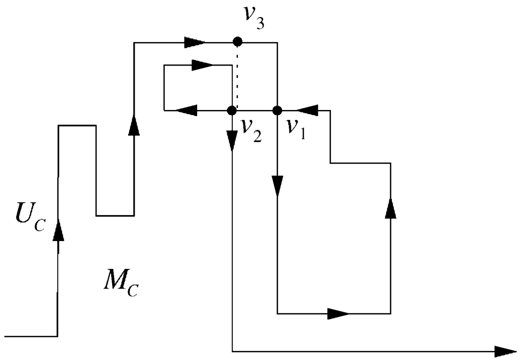

Fig. 9.

after moving upwards or to the left. If v_2 was visited before v_1 was visited for the first time (v_2 lies before v_1 on the trajectory, Fig. 8), then reaching v_2 again B has the same orientation according to the layout of Low_C and Up_C as when visiting v_2 for the first time. So, removing the cycle from v_2 to v_2 we obtain a new trajectory that has two crossing points less than the original one. The only remaining possibility is that both visits of v_2 happen after the second visit of v_1 (see Fig. 9). In this case we fix the point v_3 laying immediately above v_2 on the trajectory. Then we remove the part of the trajectory from v_3 to the second visit of v_2 and add a new part from v_3 to v_2. This part can be added without looking on the contents of the adjacent squares because the trajectory goes downwards with Low_C on the left side. Again, we get a new trajectory that has two crossing points less than the original one.

2. Let B reach v_1 for the second time after moving downwards (Figs. 10 and 11). The situation at Fig. 10 can be solved in the same way as the situation at Fig. 8 by removing the trajectory part from v_2 to v_2. For the situation at Fig. 11 we first remove the part of the trajectory from the first visit of v_1 to the second visit of v_2.

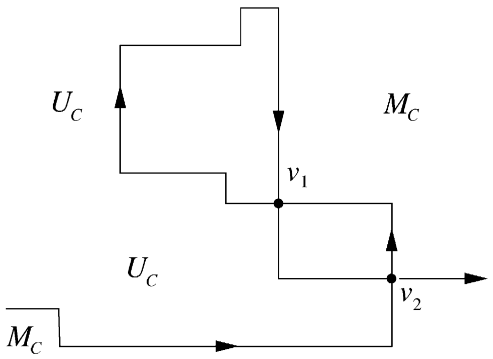

Fig. 10. A trajectory of the movement of B that crosses itself, after some movement downwards.

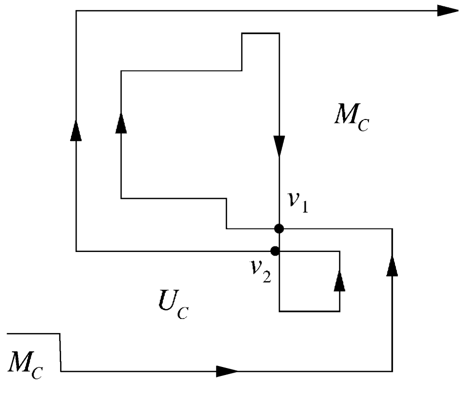

Fig. 11.

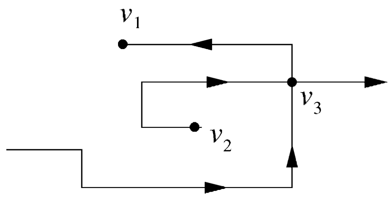

Fig. 12.

If the initial part of the trajectory to v_1 and the final part of the trajectory from v_2 do not cross each other, then these two trajectory parts together surely separate the sources from the destinations. If these two trajectories cross each other in a point v_3 (Fig. 12), then they have the same orientation in v_3. So, removing the trajectory part from v_3 to v_1 and from v_2 to v_3 we obtain a new trajectory having 4 crossing points less than the original one.[9] □

[9] Note that the situation at Fig. 11 can be also solved in an analogous way as the situation at Fig. 9 was solved. So, one does not need to break the trajectory.

4. Conclusions and open problems

In this paper we have proved that two-way Las Vegas finite automata are very powerful because

(i) if one has small nondeterministic automata for L and $L^{\complement}$, then one has a small 2LVFA for L,

(ii) 2LVFAs may be essentially more powerful than NFAs, i.e. $ns((L_k)^{\complement}) = 2^{\Omega(lvs_2((L_k)^{\complement}))}$,

(iii) there is an exponential gap between LVFAs and 2LVFAs, and

(iv) there is at least an almost quadratic gap between 2DFAs and 2LVFAs.

The main remaining open problems are

1. Is there an exponential gap between 2DFAs and 2LVFAs?
2. Is there an exponential gap between 2LVFAs and 2NFAs?

Observe, that a positive answer to any of these two questions would solve the famous open problem of [17] asking for the relation between 2DFAs and 2NFAs. We conjecture that $s_2(L_k)$ for the language L_k of Theorem 4 is not polynomial in k. Since $lvs_2(L_k) = O(k)$ the proof of our conjecture would yield the solution of the first open problem.

Note, that the simulation of 2SVNFAs by 2LVFAs in Theorem 1 results in a Las Vegas automaton that works in an expected exponential time according to the input length. This contrasts to deterministic automata which always work in linear time.[10] Also, the length of the shortest accepting (rejecting) computation of a 2NFA on any input x is of linear length in $|x|$. Exponential time seems to be inevitable because one needs to increase the probability of success by numerous repetitions of simulation attempts. But one can consider the comparison between 2LVFAs and 2DFAs to be unfair, since this can be viewed as a comparison of exponential time and linear time. So, a research problem is to consider polynomial-time 2LVFAs and compare them with 2DFAs, 2NFAs, as well as with unrestricted 2LVFAs.

These comparisons are of importance especially if one considers Las Vegas automata as a data structure for the representation of regular languages. But this can be practical only if the Las Vegas automaton works in a reasonable time. This question is strongly connected with the question whether there is a reasonable general upper bound $f(n)$ on the number of random bits used by 2LVFAs on inputs of length n.

If one wants to use Las Vegas automata to represent regular languages, the following questions may be of interest:

(i) Given a probabilistic automaton A, how hard is it to decide whether A is a Las Vegas automaton?

(ii) One-way Las Vegas finite automata work in linear time and so they are a reasonable data structure for regular languages if one wants to decide the membership

[10] More precisely, in time at most $s \cdot n$, where s is the number of states and n is the length of the tape.

problem. But the efficiency of the use of LVFAs for regular languages representation depends on the kind of manipulations one would like to realize with the representations. How hard is it to find a minimal LVFA? How hard is it to construct the minimal DFA for a given LVFA? The straightforward approach uses the construction of an equivalent NFA for a given DFA, and so the space complexity of this algorithm is exponential. Does there exist a more efficient way? How hard is it to decide the equivalence problem for LVFAs?

(iii) All known examples of one-way Las Vegas finite automata use a constant number of random decisions only, and all these random decisions can be moved to the very beginning of the computation (before starting to read the input) [4]. Is there a possibility to convert any LVFA into a LVFA using a constant number of random decisions? Can all these random decisions be moved to the very beginning?

Acknowledgements

We would like to thank Pavol Ďuriš for his comments. We are indebted to Tomasz Jurdzinski pointing out an error in a previous version of this paper.

Appendix. The Proof of Theorem 1

Let A be a 2SVNFA. We shall construct a 2LVFA B with $L(B)=L(A)$ and $s(B)\leqslant 4s(A)+3$. The construction is based on the method of Macarie and Seiferas [10] who generalized the method for the simulation of nondeterministic space by one-sided error Monte Carlo probabilistic space [6] for small (sublogarithmic) spaces. In fact, we have only to consider "one-head" finite automata instead of multihead finite automata or Turing machines and we have to exchange one-sided-error Monte Carlo computations for Las Vegas computations. But this exchange causes no trouble because we additionally know that $(L(A))^{\complement}$ is easy for nondeterministic finite automata, too.

First, we describe the work of B and then we prove that B is really a 2LVFA accepting $L(A)$.

The states of B consist of the states of A plus some additional states. For each nondeterministic transition of A to two states[11] we have the same two transitions, each with probability $\frac{1}{2}$, in B. The accepting (rejecting) states of B are exactly the accepting (rejecting) states of A. B has only one neutral state q_{neutral}, where q_{neutral} is a new state. Every neutral state of A is a working state of B. If B working on an input w reaches a neutral state of A, then B starts the following procedure:

1. Move the head on the left endmarker ¢ in a new special state.

[11] Note that we agreed to assume here that each nondeterministic branching is bounded by 2.

2. B tosses $r_w = (2s(A) + 1)(|w| + 2)$ times a coin, where $|w|$ is the length of the input w. Obviously, this can be realized by using $2s(A) + 1$ new working states and by running $|w| + 2$ steps (via the input tape) in each one of these states. If the outcome is "all heads" (this happens with the probability of $1/2^{r_w}$), then B moves to the state q_{neutral} and halts. Otherwise B immediately enters a new special state in which the head is moved to the left endmarker ¢. After this B restarts a new simulation on the input w from the initial state.

Note, that the goal of the use of the above procedure is to essentially decrease the probability of finishing in the state q_{neutral} and so to increase the probability of reaching an accepting (rejecting) state if $w \in L(A)$ ($w \notin L(A)$).

Besides this, B has still $s(A)$ new states, each one as a counterpart for a state of A. These states are used after every simulation step of B in order to randomly decide (with equal probabilities) whether to continue in the simulation of the computation of A or to restart a new simulation. As one can see later, this additional coin tossing is important in order to decrease the probability of simulating a very long computation of A that contains repetitions of the same configurations.

We observe that B has $4s(A)+3$ states, and that B can reach an accepting (rejecting) state on an input w if and only if $w \in L(A)$ ($w \notin L(A)$). So, it remains to show that for every input $w \in L(A)$ ($w \notin L(A)$) the probability of reaching an accepting (rejecting) state is at least $\frac{1}{2}$.

For any input w, there exist at most $\delta_w = s(A)(|w| + 2)$ different configurations of A on w. So, if $w \in L(A)$ ($w \notin L(A)$), then there exists at least one computation of A on w finishing in an accepting (rejecting) state after at most δ_w computation steps. So, simulating δ_w steps of A, B reaches an accepting (rejecting) state with probability at least $\left(\frac{1}{2}\right)^{\delta_w}$.

In what follows, we show that Prob(B accepts $w \mid w \in L(A)) \geq \frac{1}{2}$. The case $w \notin L(A)$ is treated analogously. Let *Step*(w) be the random variable[12] saying how many steps of A will be simulated by B on w in *one* attempt. Obviously,

$$\text{Prob}(\mathit{Step}(w) < \delta_w) = \sum_{i=1}^{\delta_w - 1} \frac{1}{2^i} < 1 - \frac{1}{2^{\delta_w}}. \tag{1}$$

Let *ACCEPT*(w) be the event that B accepts w during one simulation attempt of B and let *Neg-ACCEPT*(w) be the complementary event.

Due to $\text{Prob}(\mathit{ACCEPT}(w) \mid \mathit{Step}(w) \geq \delta_w) \geq \left(\frac{1}{2}\right)^{\delta_w}$ and (1) we obtain

$$\begin{aligned}
&\text{Prob}(\mathit{Neg\text{-}ACCEPT}(w)) \\
&\quad = \text{Prob}(\mathit{Step}(w) < \delta_w)\text{Prob}(\mathit{Neg\text{-}ACCEPT}(w) \mid \mathit{Step}(w) < \delta(w)) \\
&\qquad + \text{Prob}(\mathit{Step}(w) \geq \delta_w)\text{Prob}(\mathit{Neg\text{-}ACCEPT}(w) \mid \mathit{Step}(w) \geq \delta_w)
\end{aligned}$$

[12] Obviously, *Step*(w) depends on $|w|$, but not on the word w itself.

$$
\begin{aligned}
&\leqslant \mathrm{Prob}(\mathit{Step}(w) < \delta_w)1 + (1 - \mathrm{Prob}(\mathit{Step}(w) < \delta_w))\left(1 - \frac{1}{2^{\delta_w}}\right)\\
&\leqslant \frac{1}{2^{\delta_w}}\mathrm{Prob}(\mathit{Step}(w) < \delta_w) + 1 - \frac{1}{2^{\delta_w}}\\
&< \frac{1}{2^{\delta_w}}\left(1 - \frac{1}{2^{\delta_w}}\right) + \left(1 - \frac{1}{2^{\delta_w}}\right)\\
&= 1 - \frac{1}{2^{2\delta_w}}.
\end{aligned}
$$

Finally, we bound the probability that B does not accept w in any simulation attempt. We assume as before that $w \in L(A)$.

$$
\begin{aligned}
&\mathrm{Prob}(B \text{ does not accept } w)\\
&\leqslant \mathrm{Prob}(\mathit{Neg\text{-}ACCEPT})\frac{1}{2^{r_w}}\sum_{i=0}^{\infty}(\mathrm{Prob}(\mathit{Neg\text{-}ACCEPT}))^i\left(1 - \frac{1}{2^{r_w}}\right)^i\\
&\leqslant \left(1 - \frac{1}{2^{2\delta_w}}\right)\frac{1}{2^{r_w}}\sum_{i=0}^{\infty}\left(1 - \frac{1}{2^{2\delta_w}}\right)^i\left(1 - \frac{1}{2^{r_w}}\right)^i\\
&\leqslant \left(1 - \frac{1}{2^{2\delta_w}}\right)\frac{1}{2^{r_w}}\frac{1}{1 - (1 - \frac{1}{2^{2\delta_w}})(1 - \frac{1}{2^{r_w}})}.
\end{aligned}
$$

Since,

$$
1 - \left(1 - \frac{1}{2^{2\delta_w}}\right)\left(1 - \frac{1}{2^{r_w}}\right) > 1 - \left(1 - \frac{1}{2^{2\delta_w}}\right) = \frac{1}{2^{2\delta_w}},
$$

we obtain

$$
\begin{aligned}
\mathrm{Prob}(B \text{ does not accept } w) &< 1\frac{1}{2^{r_w}}\frac{1}{1/2^{2\delta_w}}\\
&= 2^{2\delta_w - r_w}\\
&= 2^{2s(A)(|w|+2) - 2(s(A)+1)(|w|+2)}\\
&= 2^{-2(|w|+2)} \leqslant \tfrac{1}{16}. \qquad \square
\end{aligned}
$$

References

[1] A.V. Aho, J.E. Hopcroft, M. Yannakakis, On notions of information transfer in VLSI circuits, Proc. 15th Ann. ACM STOC, ACM, 1983, pp. 133–139.

[2] L. Babai, Monte Carlo algorithms in graph isomorphism techniques, Research Report No. 79-10, Département de mathématiques et statistique, Université de Montréal, 1979.

[3] P. Berman, A note on sweeping automata, in: J.W. de Bakker, Jan van Leeuwen (Eds.), Proc. 7th ICALP '80, Lecture Notes in Computer Science, vol. 85, Springer, Berlin, 1980, pp. 91–97.

[4] M. Dietzelfelbinger, M. Kutylowski, R. Reischuk, Exact lower bounds for computing Boolean functions on CREW PRAMs, J. Comput. System Sci. 48 (1994) 231–254.

[5] P. Ďuriš , J. Hromkovič, J.D.P. Rolim, G. Schnitger, Las Vegas versus determinism for one-way communication complexity, finite automata and polynomial-time computations, in: Proc. STACS '97, Lecture Notes in Computer Science, vol. 1200, Springer, Berlin, 1997, pp. 117–128.

[6] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comput. 6 (1977) 675–695.

[7] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computations, Addison-Wesley, Reading, MA, 1979.

[8] J. Hromkovič, Communication Complexity and Parallel Computing, Springer, Berlin, 1997.

[9] E. Kushilevitz, N. Nisan, Communication Complexity, Cambridge University Press, Cambridge, 1997.

[10] I.I. Macarie, J.I. Seiferas, Strong equivalence of nondeterministic and randomized space-bounded computations, manuscript, 1997.

[11] K. Mehlhorn, E. Schmidt, Las Vegas is better than determinism in VLSI and distributed computing, Proc. 14th ACM STOC'82, ACM, 1982, pp. 330–337.

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

[13] Ch. Papadimitriou, M. Sipser, Communication complexity, J. Comput. System Sci. 28 (1984) 260–269.

[14] M. Sauerhoff, On nondeterminism versus randomness for read-once branching programs. Electron. Coll. on Computational Complexity, TR 97-030.

[15] M. Sauerhoff, On the size of randomized OBDDs and read-once branching programs for k-stable functions, in: Proc. STACS '99, Lecture Notes in Computer Science, 1563, Springer 1999, pp. 488–499.

[16] W.J. Sakoda, M. Sipser, Nondeterminism and the size of two way finite automata, Proc. 10th ACM STOC, ACM, 1978, pp. 275–286.

[17] M. Sipser, Lower bounds on the size of sweeping automata, Proc. 11th ACM STOC, 1979, pp. 360–364.

[18] M. Sipser, Lower bounds on the size of sweeping automata, J. Computer System Sci. 21 (1980) 195–202.

[19] I. Wegener, The Complexity of Boolean Functions, Wiley-Teubner Series in Computer Science, Wiley and Teubner, Stuttgart, 1987.

[20] A.C. Yao, Some complexity questions related to distributed computing, Proc. 11th ACM STOC, ACM, 1979, pp. 209–213.