An Exponential Gap between LasVegas and Deterministic Sweeping Finite Automata

Christos Kapoutsis, Richard Královič, and Tobias Mömke

Department of Computer Science, ETH Zürich

Abstract. A two-way finite automaton is *sweeping* if its input head can change direction only on the end-markers. For each $n \geq 2$, we exhibit a problem that can be solved by a $O(n^2)$-state sweeping *Las Vegas* automaton, but needs $2^{\Omega(n)}$ states on every sweeping *deterministic* automaton.

1 Introduction

One of the major goals of the theory of computation is the comparative study of probabilistic computations, on one hand, and deterministic and nondeterministic computations, on the other. An important special case of this comparison concerns probabilistic computations of zero error (also known as "LasVegas computations"): how does ZPP compare with P and NP? Or, in informal terms: *Can every fast Las Vegas algorithm be simulated by a fast deterministic one? Can every fast nondeterministic algorithm be simulated by a fast Las Vegas one?*

Naturally, the computational model and resource for which we pose these questions are the Turing machine and time, respectively, as these give rise to the best available theoretical model for the practical problems that we care about. However, the questions have also been asked for other computational models and resources. Of particular interest to us is the case of restricted models, where the questions appear to be much more tractable. Conceivably, answering them there might also improve our understanding of the harder, more general settings.

In this direction, Hromkovič and Schnitger [1] studied the case of one-way finite automata, where efficiency is measured by size (number of states). They showed that, in this context, LasVegas computations are not more powerful than deterministic ones—intuitively, *every small one-way Las Vegas finite automaton* ($1\text{P}_0\text{FA}$) *can be simulated by a small deterministic one* (1DFA). This immediately implied that, in contrast, nondeterministic computations are more powerful than LasVegas ones: *there exist small one-way nondeterministic finite automata* (1NFAs) *that cannot be simulated by any small* $1\text{P}_0\text{FA}$.

For the case of two-way finite automata (2DFAs, $2\text{P}_0\text{FA}$s, and 2NFAs), though, the analogous questions remain open [2]: *Can every small* $2\text{P}_0\text{FA}$ *be simulated by a small* 2DFA*? Can every small* 2NFA *be simulated by a small* $2\text{P}_0\text{FA}$*?* Note that a negative answer to either question would confirm the long-standing conjecture that 2NFAs can be exponentially more succinct than 2DFAs [5].

* Work supported by the Swiss National Science Foundation grant 200021-107327/1.

In this article we provide such negative answers for the special case where the two-way automata involved are *sweeping* (SDFAs, SP$_0$FAs, SNFAs), in the sense that their input head can change direction only on the end-markers. Both answers use the crucial fact (adapted from [2, 4]) that a problem can be solved by small SP$_0$FAs iff small SNFAs can solve both that problem and its complement. Based on that, the answer to the second question is an immediate corollary of the recent result of [3]. The first question is answered by exhibiting a specific problem (inspired by *liveness* [5]) that cannot be solved by small SDFAs but is such that small SNFAs can solve both it and its complement. Our contribution is this latter theorem.

We stress that the expected running time of all probabilistic automata in this article is (required to be finite, but) allowed to be exponential in the length of the input, as our focus is on size complexity only. Our theorem should be interpreted as a first step towards the more natural (and more faithful to the analogy with ZPP, P, and NP) case where size and time must be held small simultaneously.

The next section defines the basics and presents the problem witnessing the separation. Section 3 describes a SP$_0$FA that solves this problem with $O(n^2)$ states. Section 4 proves that every SDFA solving the same problem needs at least $2^{\Omega(n)}$ states. Finally, Section 5 sketches a bigger picture that our theorem fits in.

2 Preliminaries

By $[n]$ we denote $\{1, 2, \ldots, n\}$. If Σ is an alphabet, then Σ^* is the set of all finite strings over Σ. If $z \in \Sigma^*$, then $|z|$, z_t, z^t, and z^{R} are its length, t-th symbol (if $1 \leq t \leq |z|$), t-fold concatenation with itself (if $t \geq 0$), and reverse. A *problem* (or *language*) over Σ is any $L \subseteq \Sigma^*$; then $\overline{L}$ is its complement. If $\# \notin \Sigma$, then $L^{\#}$ is the problem $\#(L\#)^*$ of all #-delimited finite concatenations of strings of L.

An automaton *solves* (or *recognizes*) a problem iff it accepts exactly the strings of that problem. A *family of automata* $M = (M_n)_{n\geq 0}$ solves a *family of problems* $\Pi = (\Pi_n)_{n\geq 0}$ iff, for all n, M_n solves Π_n. The automata of M are 'small' iff, for some polynomial p and all n, M_n has at most $p(n)$ states. Often, the generic member of a family informally denotes the family itself: e.g., "Π_n can be solved by a small 1DFA" means that some family of small 1DFAs solves Π.

If f is a function and $t \geq 1$, then f^t is the t-fold composition of f with itself.

Sweeping automata. A *sweeping deterministic finite automaton* (SDFA) [6] over an alphabet Σ and a set of states Q is any triple $M = (q_s, \delta, q_a)$ of a *start* state $q_s \in Q$, an *accept* state $q_a \in Q$, and a *transition function* δ which partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to Q, for some *end-markers* $\vdash, \dashv \notin \Sigma$. An input $z \in \Sigma^*$ is presented to M surrounded by the end-markers, as $\vdash z \dashv$. The computation starts at q_s and on $\vdash$. The next state is always derived from δ and the current state and symbol. The next position is always the adjacent one in the direction of motion; except when the current symbol is $\vdash$ or when the current symbol is $\dashv$ and the next state is not q_a, in which cases the next position is the adjacent one towards the other end-marker. Note that the computation can either loop, or hang, or fall off $\dashv$ into q_a. In this last case we call it *accepting* and say that M *accepts* z.

More generally, for any input string $z \in \Sigma^*$ and state p, the *left computation of M from p on z* is the unique sequence $\text{LCOMP}_{M,p}(z) := (q_t)_{1\leq t\leq m}$, where $q_1 := p$; every next state is $q_{t+1} := \delta(q_t, z_t)$, provided that $t \leq |z|$ and the value of δ is defined; and m is the first t for which this provision fails. If $m = |z| + 1$, we say that the computation *exits z into q_m*; otherwise, $1 \leq m \leq |z|$ and the computation *hangs at q_m*. The *right computation of M from p on z* is denoted by $\text{RCOMP}_{M,p}(z)$ and defined symmetrically, with $q_{t+1} := \delta(q_t, z_{|z|+1-t})$.

The *traversals of M on z* are the members of the unique sequence $(c_t)_{1\leq t<m}$ where $c_1 := \text{LCOMP}_{M,p_1}(z)$ for $p_1 := \delta(q_s, \vdash)$; every next traversal c_{t+1} is either $\text{RCOMP}_{M,p_{t+1}}(z)$, if t is odd and c_t exits into a state q_t such that $\delta(q_t, \dashv) = p_{t+1} \neq q_a$, or $\text{LCOMP}_{M,p_{t+1}}(z)$, if t is even and c_t exits into a state q_t such that $\delta(q_t, \vdash) = p_{t+1}$; and m is either the first t for which c_t cannot be defined or ∞, if c_t exists for all t. Then, the *computation of M on z*, denoted by $\text{COMP}_M(z)$, is the concatenation of $(q_s), c_1, c_2, \ldots$ and possibly also (q_a), if m is finite and even and c_{m-1} exits into a state q_{m-1} such that $\delta(q_{m-1}, \dashv) = q_a$.

If M is allowed more than one next move at each step, we say it is *nondeterministic* (a SNFA). Formally, this means that δ partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to the set of *all non-empty subsets* of Q. Hence, on any $z \in \Sigma^*$, $\text{COMP}_M(z)$ is a *set* of computations. If at least one of them is accepting, we say that *M accepts z*.

If M follows exactly one of its nondeterministic choices at each step according to some rational distribution, we say it is *probabilistic* (a SPFA). Formally, this means that δ partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to the set of *all rational distributions* over Q—i.e., all total functions from Q to the rational numbers that obey the axioms of probability. Hence, on any $z \in \Sigma^*$, $\text{COMP}_M(z)$ is a *rational distribution* of computations. The expected length of a computation drawn from this distribution is called the *expected running time* of M on z.

For M to be a *Las Vegas* SPFA (a SP_0FA), a few extra conditions should hold. First, a special *reject* state $q_r \in Q$ must be specified—so that $M = (q_s, \delta, q_a, q_r)$. Second, whenever the current symbol is $\dashv$ and the next state is q_r, the next position is the adjacent one in the direction of motion—so that a computation may also fall off $\dashv$ into q_r, in which case we call it *rejecting*. Last, on any $z \in \Sigma^*$, a computation drawn from $\text{COMP}_M(z)$ must be either accepting with probability 1 or rejecting with probability 1. In the former case, we say that *M accepts z*. The concept of Las Vegas randomness is closely related to the self-verifying nondeterminism (see [2]).

Finally, a sweeping automaton is called *one-way* (1DFA, 1NFA, 1PFA, $1\text{P}_0\text{FA}$) if it halts immediately after reading the right end-marker. Formally, this means that the value of the transition function on any state and on $\dashv$ is always either undefined or q_a (for 1DFAs); or $\{q_a\}$ (for 1NFAs); or the unique distribution over $\{q_a\}$ (for 1PFAs); or some distribution over $\{q_a, q_r\}$ (for $1\text{P}_0\text{FA}$s).

The witness. In this section we define the family of problems Π that witnesses the separation between small SP_0FAs and small SDFAs. Let $n \geq 2$ be arbitrary.

Problem Π_n consists of all #-delimited concatenations of the strings of another problem, Π'_n. That is, $\Pi_n := (\Pi'_n)^{\#} = \#(\Pi'_n\#)^*$. So, we need to present Π'_n.

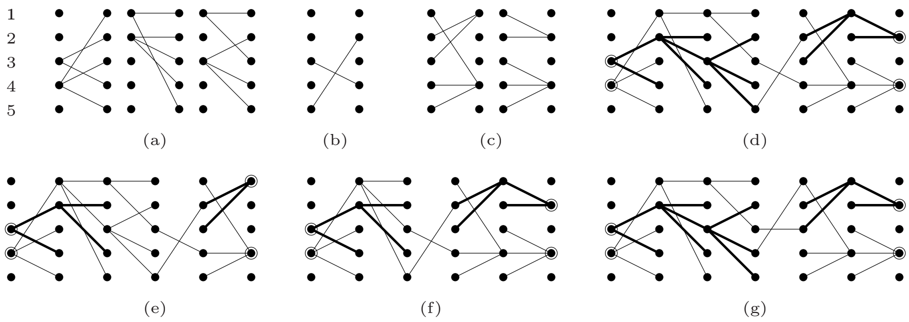

Fig. 1. (a) Three symbols of $\varGamma_5$; e.g., the leftmost one is $(3, 4, \{2, 4\})$. (b) The symbol $\{(3, 4), (5, 2)\}$ of X_5. (c) Two symbols of $\varDelta_5$. (d) The string defined by the six symbols of (a)-(c); in circles: the roots of the four trees; in bold: the two upper trees; the string is in $\varPi'_5$. (e) The upper left tree vanishes. (f) No tree vanishes, but the middle edges miss the upper left tree. (g) A well-formed string that does not respect the tree order.

Problem $\varPi'_n$ is defined over the alphabet $\varSigma'_n := \varGamma_n \cup X_n \cup \varDelta_n$, where:

$$\begin{aligned} \varGamma_n &:= \{\ (i, j, \alpha)\ \mid\ i, j \in [n] \ \text{ and } \ i < j \ \text{ and } \ \emptyset \neq \alpha \subsetneq [n]\ \}, \\ X_n &:= \{\ \{(i, r), (j, s)\}\ \mid\ i, j, r, s \in [n] \ \text{ and } \ i \neq j \ \text{ and } \ r \neq s\ \}, \\ \varDelta_n &:= \{\ (\alpha, j, i)\ \mid\ i, j \in [n] \ \text{ and } \ i < j \ \text{ and } \ \emptyset \neq \alpha \subsetneq [n]\ \}. \end{aligned}$$

Intuitively, each $(i, j, \alpha) \in \varGamma_n$ represents a two-column graph (Fig. 1a) that has n nodes per column and contains exactly the edges that connect the ith left node to all right nodes inside α and the jth left node to all right nodes outside α. Symmetrically, each $(\alpha, j, i) \in \varDelta_n$ represents a similar graph (Fig. 1c) containing exactly the edges that connect the ith and jth right nodes to the left nodes inside α and outside α, respectively. Finally, each $\{(i, r), (j, s)\} \in X_n$ represents a graph (Fig. 1b) containing only the edges connecting the ith and jth left nodes to the rth and sth right nodes, respectively. In all cases, we say that i and j (and r and s, in the last case) are the *roots* of the given symbol.

Of all strings over $\varSigma'_n$, consider those following the pattern $\varGamma_n^* X_n \varDelta_n^*$. Each of them represents the multi-column graph (Fig. 1d) that we get from the corresponding sequence of two-column graphs when we identify adjacent columns. The symbol of X_n is called 'the middle symbol'—although it may very well not be in the middle position. If we momentarily hide the edges of that symbol, we easily see that the graph consists of exactly four disjoint trees, stemming out of the roots of the leftmost and rightmost columns. The tree out of the upper root of the leftmost column is naturally referred to as "the upper left tree". Similarly, the other trees are called "lower left", "upper right", and "lower right". Notice that, starting from the leftmost column, the two left trees may or may not both reach the left column of the middle symbol, as one of them may at some point 'cover all nodes' (Fig. 1e). Similarly, at least one of the two right trees reaches the right column of the middle symbol, but not necessarily both. Also observe that, in the case where all four trees make it to the middle symbol, the two edges of that symbol may or may not collectively 'touch' all trees (Fig. 1f). A string over Σ'_n is called *well-formed* if it belongs to $\Gamma_n^* X_n \Delta_n^*$ and is such that each of the four trees contains exactly one of the roots of the middle symbol (Fig. 1dg).

Of all well-formed strings over Σ'_n, problem Π'_n consists of those that 'respect the tree order', in the sense that the two edges of the middle symbol do not connect an upper tree to a lower one (Fig. 1d). In other words, this is the set

$$\Pi'_n := \{z \in (\Sigma'_n)^* \mid z \text{ is well-formed and respects the tree order}\}.$$

Hence, to solve $\Pi_n = \texttt{\#}(\Pi'_n\texttt{\#})^*$ means to check that the input string (over $\Sigma_n := \Sigma'_n \cup \{\texttt{\#}\}$) starts and ends with # and is such that *every* infix between two successive copies of # is well-formed and respects the tree order.

3 The upper bound

In this section we prove that Π_n can be solved by a SP_0FA with $O(n^2)$ states.

One-way nondeterministic finite automata. The next two simple lemmata reduce solving Π_n with a small SP_0FA to solving Π'_n and $\overline{\Pi'_n}$ with small 1NFAs.

Lemma 1 (adapted from [2, 4])**.** *If each of L and $\overline{L}$ can be solved by a* 1NFA *with m states, then L can be solved by a* SP_0FA *with $1 + 2m$ states.*

Proof. Suppose M and $\overline{M}$ are two m-state 1NFAs solving L and $\overline{L}$, respectively. Then, on any input z, exactly one of the computation trees of M and $\overline{M}$ on z contains accepting computations. We construct a SP_0FA M' for L that navigates probabilistically through these trees, trying to discover such a computation. If it succeeds, then it accepts or rejects, depending on which tree the computation was found in. If it fails, it sweeps back to the left end-marker and tries again.

More specifically, on input z, M' performs a series of sweeps. Each left-to-right sweep is an attempt to find an accepting computation of either M or $\overline{M}$ on z, while right-to-left sweeps are just rewinds. A left-to-right sweep starts with M' selecting one of M and $\overline{M}$ uniformly at random. Then, the selected 1NFA is simulated on z: at each step, M' either follows one of the possible next states uniformly at random or—if there are no such states (i.e., the 1NFA would hang at that point)—simply stops the simulation and sweeps blindly to $\dashv$. If the simulation ever reaches a situation where the 1NFA would be about to fall off $\dashv$ into its accepting state, then M' has discovered the desired accepting computation and therefore falls off $\dashv$, too, into its own accepting or rejecting state (depending on whether it had been simulating M or $\overline{M}$, respectively). Otherwise, the simulation stops somewhere before or at $\dashv$, in which case M' finishes the left-to-right sweep, sweeps back to $\vdash$, and starts a new attempt.

It is not hard to see that M' can be constructed out of a copy of M, a copy of $\overline{M}$, and 1 extra state.[(1)] Also, M' halts only after finding an accepting computation, which happens with probability 1, and then decides correctly. Finally, since each attempt uses at most $2|z| + 2$ steps and succeeds with probability at least $\frac{1}{2}(\frac{1}{m})^{|z|+1}$, the average running time is at most $(2|z| + 2) \cdot 2m^{|z|+1} = 2^{O(|z|)}$. $\square$

Lemma 2. *If L can be solved by a* 1NFA *with m states, then $L^{\#}$ can be solved by an* 1NFA *with $2 + m$ states. Similarly, if $\overline{L}$ can be solved by a* 1NFA *with m states, then $\overline{L^{\#}}$ can be solved by an* 1NFA *with $4 + m$ states.*

Proof. Suppose M is an m-state 1NFA solving L. A 1NFA M' for $L^{\#}$ can simply simulate M successively on every #-delimited infix of its input, until the input is exhausted or one of these simulations produces no accepting computation. Easily, M' can be constructed out of one copy of M and two new states.(2)

Similarly, if M is an m-state 1NFA for $\overline{L}$, then a 1NFA M' for $\overline{L^{\#}}$ can simply simulate M on a nondeterministically chosen #-delimited infix of its input, and accept if the simulation accepts; at the same time, additional nondeterministic threads accept if the input fails to be a #-delimited concatenation of infixes. Easily, M' can be constructed out of one copy of M and four new states.(3) □

Two upper bounds for Π'_n. It is now enough to prove that each of Π'_n and $\overline{\Pi'_n}$ can be solved by a 1NFA with $O(n^2)$ states. To see how, let us first suppose that the input is promised to be of the form $\Gamma_n^* X_n \Delta_n^*$.

It is easy to see that such an input is in Π'_n iff it contains two disjoint paths that run from the leftmost to the rightmost column and have their right endpoints in the same order as their left endpoints. *To verify this condition*, a 1NFA M can simply guess the two paths (at each step remembering only the most recent node in each of them) and accept iff their last nodes are in the order in which the paths started. This can be done easily with $2\binom{n}{2}$ states.(4) *To disprove this condition*, a 1NFA $\overline{M}$ can look for one of the following 'flaws': (i) in some $a \in \Gamma_n$, one of the roots touches two roots of the following symbol, (ii) in some $a \in \Delta_n$, one of the roots touches two roots of the preceding symbol, or (iii) the input (is well-formed, but) does not respect the tree order. The last flaw can be detected easily, with a slightly modified copy of M; detecting (ii) is then possible with one additional state; a final modification—requiring $\binom{n}{2}$ new states—ensures that (i) is also detected. Overall, $1 + 3\binom{n}{2}$ states are enough.(5)

Now, if the input is not promised to be of the form $\Gamma_n^* X_n \Delta_n^*$, we can simply augment M and $\overline{M}$ to also check this additional condition. Specifically, given that $\Gamma_n^* X_n \Delta_n^*$ can be recognized by a 1DFA M' with only two states, Π'_n can be solved by the (standard) Cartesian product of M and M' that accepts iff both of them accept (and is twice as big as M); similarly, $\overline{\Pi'_n}$ can be solved by an augmented version of $\overline{M}$ that includes M' as an additional nondeterministic thread (and has two more states than $\overline{M}$).

4 The lower bound

Much like what we did in Section 3, we first reduce the task of proving a lower bound for SDFAs solving Π_n to the task of proving a lower bound for a simpler class of automata (the *parallel intersection automata*, see below) solving Π'_n. Essential in this reduction is the notion of *generic strings* (adapted from [6]). So, we start with the definition and properties of these strings, continue with the reduction, and conclude with the lower bound for the simpler setting.

Generic strings. Let M be a SDFA over an alphabet Σ and state set Q. For any $y \in \Sigma^*$, consider the set of all states that can be produced on the rightmost boundary of y by left computations of M:

$$\text{LVIEWS}_M(y) := \{q \in Q \mid (\exists p \in Q)[\text{LCOMP}_{M,p}(y) \text{ exits into } q]\}.$$

How does this set change if we replace y with some right-extension yz of it? In other words, how do the sets $\text{LVIEWS}_M(y)$ and $\text{LVIEWS}_M(yz)$ compare?

Consider the partial function $\text{LMAP}_M(y,z) : \text{LVIEWS}_M(y) \to Q$ which, for every $q \in \text{LVIEWS}_M(y)$, is defined only if $\text{LCOMP}_{M,q}(z)$ does not hang and, if so, returns the state that this computation exits into. Easily, the values of this function: (i) are all in $\text{LVIEWS}_M(yz)$,(6) and (ii) cover the entire $\text{LVIEWS}_M(yz)$.(7) So, $\text{LMAP}_M(y,z)$ is a *partial surjection* from $\text{LVIEWS}_M(y)$ to $\text{LVIEWS}_M(yz)$. This immediately implies Fact 1. Fact 2 is equally simple.(8)

Fact 1 *For all* y,z: $|\text{LVIEWS}_M(y)| \geq |\text{LVIEWS}_M(yz)|$.

Fact 2 *For all* y,z: $\text{LVIEWS}_M(yz) \subseteq \text{LVIEWS}_M(z)$.

Now consider any property $\emptyset \neq P \subseteq \Sigma^*$ which is *infinitely extensible to the right*, in the sense that every string that has the property can be right-extended into a longer one that also has it. Fact 1 implies the following about the behavior of M on P: if we start with any $y \in P$ and keep right-extending it ad infinitum into $yz, yzz', yzz'z'', \dots \in P$, then from some point on the corresponding sequence of the sizes of the sets $|\text{LVIEWS}_M(\cdot)|$ will become constant. Any of the extensions after that point is called L-generic (for M) over P. Summarizing:

Definition 1. *A string y is* L-generic *over P if $y \in P$ and, for all $yz \in P$,* $|\text{LVIEWS}_M(y)| = |\text{LVIEWS}_M(yz)|$.

Fact 3 *Suppose $P \subseteq \Sigma^*$ is* non-empty *and* infinitely extensible to the right. *Then* L*-generic strings over P exist.*

Note that a symmetric argument works in the other direction, too: working with right computations and left-extensions, we can define $\text{RVIEWS}_M(y)$ and $\text{RMAP}_M(z,y)$; conclude Facts 1 and 2 for $\text{RVIEWS}_M(y)$ and $\text{RVIEWS}_M(zy)$; define R-generic strings; and conclude Fact 3 for them, too. In fact, we can often construct strings, called simply *generic*, that are simultaneously L- and R-generic:

Fact 4 *Suppose that y_{L} and y_{R} are* L*-generic and* R*-generic over P, respectively. Then every string in P of the form $y_{\text{L}} z y_{\text{R}}$ is generic over P.*

Proof. For any L-generic string over P, all right-extensions of it in P are clearly also L-generic. In the other direction, the symmetric statement is true. $\square$

The next lemma is the key for the reduction presented in Lemma 4.

Lemma 3. *Suppose a* SDFA *M solves $L^\#$ and y is generic for it over $L^\#$. Then a string x belongs to L iff $\text{LMAP}_M(y, xy)$ and $\text{RMAP}_M(yx, y)$ are total and injective.*

Proof. Suppose $x \in L$. Since $y \in L^{\#}$ (because y is generic over $L^{\#}$), we know yxy is also in $L^{\#}$. Hence, yxy is a right-extension of y in $L^{\#}$. Since y is L-generic, this implies that $|\text{LVIEWS}_M(y)| = |\text{LVIEWS}_M(yxy)|$.

Now consider $\text{LMAP}_M(y, xy)$. By the discussion before Fact 1, we already know this is a partial surjection from $\text{LVIEWS}_M(y)$ to $\text{LVIEWS}_M(yxy)$. Since the two sets are of equal size, the function must be total. For the same reason, it must also be injective. The argument for $\text{RMAP}_M(yx, y)$ is symmetric.

Conversely, suppose $\text{LMAP}_M(y, xy)$ is total and injective. Since we already know that it partially surjects $\text{LVIEWS}_M(y)$ to $\text{LVIEWS}_M(yxy)$, we can conclude that it is actually a bijection between the two sets. Now, by Fact 2, we also know that $\text{LVIEWS}_M(yxy) \subseteq \text{LVIEWS}_M(y)$. Hence, $\text{LMAP}_M(y, xy)$ bijects $\text{LVIEWS}_M(y)$ into one of its subsets. Clearly, this is possible only if this subset is the set itself. So, $\text{LMAP}(y, xy)$ is a permutation π of $\text{LVIEWS}_M(y)$. Symmetrically, if $\text{RMAP}_M(yx, y)$ is total and injective, then it is a permutation ρ of $\text{RVIEWS}_M(y)$.

Now pick any $k \geq 1$ such that each of π^k and ρ^k is the identity on its domain, and consider the string $z := y(xy)^k = (yx)^k y$. It is easy to verify that $\text{LMAP}_M\big(y, (xy)^k\big)$ equals $\text{LMAP}_M(y, xy)^k = \pi^k$, and is therefore the identity on $\text{LVIEWS}_M(y)$. Similarly, $\text{RMAP}_M\big((yx)^k, y\big)$ equals ρ^k, and is therefore the identity on $\text{RVIEWS}_M(y)$. Intuitively, this means that, computing through z, the left-to-right computations of M do not notice the presence of $(xy)^k$ to the right of the prefix y; similarly, the right-to-left computations do not notice the presence of $(yx)^k$ to the left of the suffix y. Consequently, M does not distinguish between y and z: it either accepts both of them or rejects both of them.(9) Since M solves $L^{\#}$ and $y \in L^{\#}$, we know M accepts y. Therefore, M accepts z as well. Hence, every #-delimited infix of z is in L. In particular, $x \in L$. $\square$

Parallel intersection automata. A *parallel intersection automaton* over Σ is any pair $M = (\mathcal{L}, \mathcal{R})$ of families of 1DFAs over Σ. To run M on an input x means to run each of its component 1DFAs on x, but with a twist: each $D \in \mathcal{L}$ reads x from left to right, while each $D \in \mathcal{R}$ reads x from right to left. We say M *accepts* x iff all these computations are accepting—i.e., iff all $D \in \mathcal{L}$ accept x and all $D \in \mathcal{R}$ accept x^{R}. The next lemma presents a non-trivial connection with SDFAs—implicitly present already in the argument of [6].

Lemma 4. *If $L^{\#}$ can be solved by a* SDFA *of size m, then L can be solved by a parallel intersection automaton with at most $2\binom{m}{2}$ components, each of size $\binom{m}{2}$.*

Proof. Suppose a SDFA M over a set Q of m states solves $L^{\#}$. We will construct a parallel intersection automaton $M' = (\mathcal{L}, \mathcal{R})$ that solves L, as follows.

First, we fix y to be any generic string for M over $L^{\#}$ (we know such y exist, by Facts 3,4 and easy properties of $L^{\#}$). Then (Lemma 3) an arbitrary x is in L iff $\text{LMAP}_M(y, xy)$ and $\text{RMAP}_M(yx, y)$ are both total and injective, namely iff:

- for all distinct $p, q \in \text{LVIEWS}_M(y)$: both $\text{LCOMP}_{M,p}(xy)$ and $\text{LCOMP}_{M,q}(xy)$ exit xy, and they do so into different states, and
- for all distinct $p, q \in \text{RVIEWS}_M(y)$: both $\text{RCOMP}_{M,p}(yx)$ and $\text{RCOMP}_{M,q}(yx)$ exit yx, and they do so into different states.

Letting $m_{\mathrm{L}} := |\textsc{lviews}_M(y)|$ and $m_{\mathrm{R}} := |\textsc{rviews}_M(y)|$, we see that checking $x \in L$ reduces to checking $\binom{m_{\mathrm{L}}}{2}+\binom{m_{\mathrm{R}}}{2}$ separate conditions, one for each unordered pair of distinct states from $\textsc{lviews}_M(y)$ or from $\textsc{rviews}_M(y)$. The components of M' are designed to check exactly these conditions.

Before describing these components, let us rewrite the above conditions a bit more nicely. First, we need a concise way of saying whether two left computations on y exit into different states or not, and similarly for right computations. To this end, we define the following relations on Q:

- $p \asymp_{\mathrm{L}} q$ iff both $\textsc{lcomp}_{M,p}(y)$ and $\textsc{lcomp}_{M,q}(y)$ exit y, and they do so into different states.
- $p \asymp_{\mathrm{R}} q$ iff both $\textsc{rcomp}_{M,p}(y)$ and $\textsc{rcomp}_{M,q}(y)$ exit y, and they do so into different states.

Now, the conditions from above can be rephrased as follows:

- for all distinct $p, q \in \textsc{lviews}_M(y)$: both $\textsc{lcomp}_{M,p}(x)$ and $\textsc{lcomp}_{M,q}(x)$ exit x, and they do so into states that are $\asymp_{\mathrm{L}}$-related, and
- for all distinct $p, q \in \textsc{rviews}_M(y)$: both $\textsc{rcomp}_{M,p}(x)$ and $\textsc{rcomp}_{M,q}(x)$ exit x, and they do so into states that are $\asymp_{\mathrm{R}}$-related,

and it is now straightforward to build 1DFAs that check each of them.

For example, the 1DFA checking the condition for the pair $p, q \in \textsc{lviews}_M(y)$ has 1 state for each unordered pair of distinct states from Q, with $\{p, q\}$ being both the start and the accept state. On $\vdash$, $\{p, q\}$ simply goes to itself. At every step after that, the automaton tries to compute the next pair by applying the transition function of M on the current symbol and each of the two states of the current pair. If either application returns no value or both return the same value, the automaton simply hangs; else, it moves to the corresponding pair. On $\dashv$, the pairs leading to $\{p, q\}$ (and thus to acceptance) are exactly the $\asymp_{\mathrm{L}}$-related ones.

Overall, we need $\binom{m_{\mathrm{L}}}{2} + \binom{m_{\mathrm{R}}}{2} \leq 2\binom{m}{2}$ automata, each of size $\binom{m}{2}$. □

A lower bound for $\boldsymbol{\Pi'_n}$. By Lemma 4, it is now enough to prove that no parallel intersection automaton can solve Π'_n with a *small number* of *small components*. The next lemma proves something much stronger: no parallel intersection automaton can solve Π'_n with small components, irrespective of their number. The argument is similar to that of [5, Theorem 4.2.3].

Lemma 5. *In any parallel intersection automaton solving Π'_n, at least one of the components has size strictly greater than $(2^n - 2)/n$.*

Proof. Towards a contradiction, suppose $M = (\mathcal{L}, \mathcal{R})$ solves Π'_n with at most $(2^n - 2)/n$ states in each one of its components. We can then prove the following.

Claim. There exists a string $u \in \Gamma_n^*$ that admits well-formed right-extensions and has all of them accepted by every $D \in \mathcal{L}$. Symmetrically, some $v \in \Delta_n^*$ admits well-formed left-extensions and has all of them accepted by every $D \in \mathcal{R}$.

Intuitively, u is a string that manages to 'confuse' every left component of M: each of them accepts every well-formed right-extension of u (no matter whether

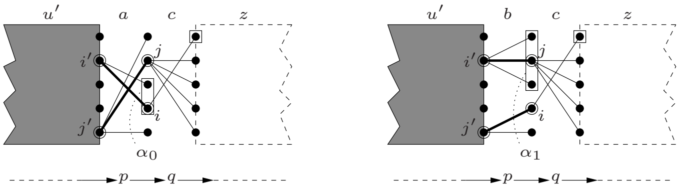

Fig. 2. Confusing D in the proof of Lemma 5.

it respects the tree order or not), exactly because it has failed to correctly keep track of the tree order inside u. Similarly for v and the right components of M.

We will prove only the first half of the claim, as the argument for the other half is symmetric. Before that, though, let us see how the claim implies a contradiction. First, since u has well-formed right-extensions, we can find nodes $i, j \in [n]$ on its rightmost column that belong to different trees. Similarly, the leftmost column of v contains nodes $r, s \in [n]$ that belong to different trees of v. Now, consider the two symbols of X_n that have i, j, r, s as their roots, namely $x := \{(i, r), (j, s)\}$ and $x' := \{(i, s), (j, r)\}$, and the strings uxv and $ux'v$. Clearly, each string is well-formed, right-extends u, and left-extends v. So, by the claim, each of them is accepted by all components of M. Hence, M accepts both strings. However, by the selection of x and x', we know that one of the strings does not respect the tree order. So, after all, M does not solve Π'_n—a contradiction.

To prove the first half of the claim, we work by induction on the size of $\mathcal{L}$.

If $\mathcal{L}$ is empty, then the claim holds vacuously for, say, the empty u.

If $\mathcal{L}$ is non-empty, we pick any D in it and let $\mathcal{L}' := \mathcal{L} - \{D\}$. Then $\mathcal{L}'$ is smaller than $\mathcal{L}$, so (by the inductive hypothesis) some $u' \in \Gamma_n^*$ admits well-formed right-extensions and has all of them accepted by all $D' \in \mathcal{L}'$. Our goal is to find two symbols $a, c \in \Gamma_n$ such that the string $u := u'ac$ admits well-formed right-extensions and has all of them accepted by all members of $\mathcal{L}$. (Fig. 2.)

We start by noting (as above) that, since u' has well-formed right-extensions, there exist nodes i' and j' in its rightmost column that belong to different trees.

Moreover, some of the well-formed right-extensions of u' respect the tree order (because, for each extension that does not, there is one that does: the one that differs only in the pairing of the roots of the middle symbol) and are therefore accepted by M. In particular, they are accepted by D. Thus, the left computation of D on each of them exits to the right. Hence, the left computation of D on u' exits to the right, too. Let p be the corresponding exit state.

Based on D, i', j', and p, we can now find the symbols a, c that we are after.

Consider all symbols of Γ_n that have i' and j' as roots. Each of them is of the form (i', j', α) and takes p to some next state. Since there are $2^n - 2$ such symbols (one for each $\emptyset \neq \alpha \subsetneq [n]$) and D has at most $(2^n-2)/n$ states, we know some next state attracts at least $(2^n - 2)/\big((2^n - 2)/n\big) = n$ symbols. Call this state q. Among the α's that correspond to the symbols taking p to q, two must be incomparable (otherwise, they would form a chain of n or more non-trivial subsets of $[n]$—a contradiction). Call these subsets α_0 and α_1. Then symbol a

is one of the two corresponding symbols, say $a := (i', j', \alpha_0)$. We also name the other symbol, say $b := (i', j', \alpha_1)$, and a node in each side of the symmetric difference of the two sets, say $i \in \alpha_0 \setminus \alpha_1$ and $j \in \alpha_1 \setminus \alpha_0$ (both sides are non-empty, by the incomparability of α_0, α_1). It is important to note that a connects i' and j' to i and j, respectively, whereas in b this connection is reversed. Finally, c is selected to be any symbol with i and j as roots, say $c := (i, j, \{1\})$.

Let us see why $u = u'ac$ is the string that we want (ubc would also do).

First, by the choice of i' and j', we know that a extends both trees of u': one to α_0, the other one to $\overline{\alpha_0}$. Similarly, c extends both trees of $u'a$, since $i \in \alpha_0$ and $j \in \overline{\alpha_0}$. Hence, $u = u'ac$ can indeed be right-extended into well-formed strings.

Second, every such extension of u is obviously a well-formed right-extension of u', and is thus accepted by all $D' \in \mathcal{L}'$ (recall the inductive hypothesis).

Finally, every such extension of u, say uz, is also accepted by D. To see why, consider the computations of D on $u'a$ and $u'b$. Both exit into q (by the selection of a, b, q). So, the computation of D on $uz = u'acz$ has the same suffix as the computation of D on $u'bcz$. Hence, D either accepts both strings or rejects both strings. In the latter case, M would also reject both strings, contradicting the fact that one of them respects the tree order (the strings differ only at a and b, which connect i' and j' to i and j differently). Hence, D must be accepting both strings. In particular, it accepts $u'acz = uz$. $\square$

5 A bigger picture

Our theorem is only a piece in the puzzle defined by the study of size complexity in finite automata. An elegant theoretical framework for describing this puzzle is due to Sakoda and Sipser [5]. Analogous to the framework built on other computational models and resources (e.g., Turing machines and time), it is based on the notions of a *reduction* and of a *complexity class*. However, a member of a class in this framework is always a *family of problems* and each class contains exactly every family that is solvable by a family of small automata of a corresponding type. For example, 1D *contains exactly every family of problems that can be solved by some family of small* 1DFAs. Similarly, the classes 1N, 2D, and 2N were defined for 1NFAs, 2DFAs, and 2NFAs, respectively, while co1D, co1N, co2D, and co2N were defined to consist of the corresponding families of complements.

Replacing 1DFAs with SDFAs, SP$_0$FAs, or SNFAs in the above definition, we can naturally define the classes SD, SP$_0$X, and SN, respectively, for sweeping and/or LasVegas automata.[1] Then, SD $\subseteq$ SP$_0$X $\subseteq$ SN (trivially), $\Pi \in$ 1N $\cap$ co1N $\subseteq$ SP$_0$X (by Sect. 3), $\Pi \notin$ SD (by Sect. 4), and therefore SD $\not\supseteq$ SP$_0$X (our theorem; note that we have actually proved a stronger fact: SD $\not\supseteq$ 1N $\cap$ co1N). At the same time, we also have SP$_0$X $\subseteq$ SN $\cap$ coSN (trivially) and coSN $\not\supseteq$ SN (by [3]), so that SP$_0$X $\not\supseteq$ SN. Overall, the trivial chain SD $\subseteq$ SP$_0$X $\subseteq$ SN is actually SD $\subsetneq$ SP$_0$X $\subsetneq$ SN.

Figure 3 shows in more detail the relations between the several classes, including those for *Monte-Carlo* automata ("P$_1$" and "P$_2$"—for one-sided and two-sided

[1] Note the "X" in "SP$_0$X". The name "SP$_0$" is reserved for the more natural class where the SP$_0$FAs must run in *polynomial* expected time. Similarly for 2P$_0$X, RP$_0$X, SP$_1$X, etc.

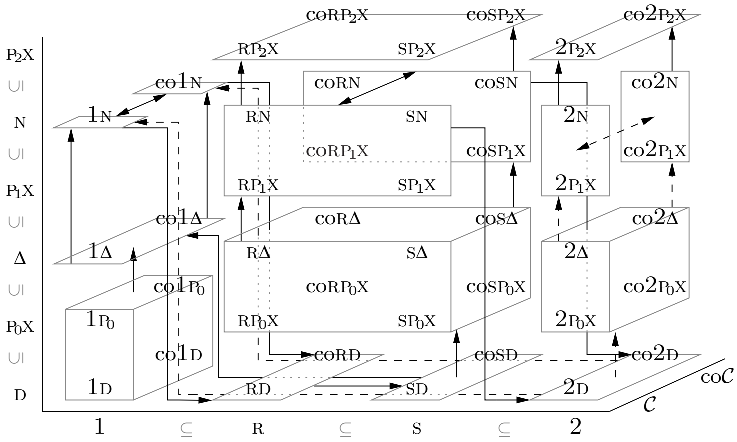

Fig. 3. A map of classes: boxes mean equality; the axes show the easy inclusions; a solid arrow $A \to B$ means $A \not\supseteq B$; a dashed arrow $A \to B$ means we conjecture $A \not\supseteq B$.

error), *self-verifying* automata ("Δ"—these capture the intersection of nondeterminism and co-nondeterminism; e.g., $1\Delta = 1\mathrm{N} \cap \mathrm{co}1\mathrm{N}$), and *rotating* automata ("R"—these are sweeping automata capable of only left-to-right sweeps).

Most facts on this map are trivial, or easy, or modifications/consequences of known results [1–6] and of our main theorem. Exceptions include the ability of small nondeterministic and probabilistic rotating automata to simulate their sweeping counterparts: $\mathrm{RN} = \mathrm{SN}$, $\mathrm{RP_0X} = \mathrm{SP_0X}$, $\mathrm{RP_1X} = \mathrm{SP_1X}$, and $\mathrm{RP_2X} = \mathrm{SP_2X}$. A more detailed presentation will appear in the full version of this article.

Acknowledgement We would like to thank Juraj Hromkovič for suggesting the problem and for giving us many helpful comments.

References

1. J. Hromkovič and G. Schnitger. On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Information and Computation*, 169:284–296, 2001.
2. J. Hromkovič and G. Schnitger. On the power of Las Vegas II: two-way finite automata. *Theoretical Computer Science*, 262(1–2):1–24, 2001.
3. C. A. Kapoutsis. Small sweeping 2NFAs are not closed under complement. In *Proceedings of the ICALP*, pages 144–156, 2006.
4. I. I. Macarie and J. I. Seiferas. Strong equivalence of nondeterministic and randomized space-bounded computations. Manuscript, 1997.
5. W. J. Sakoda and M. Sipser. Nondeterminism and the size of two way finite automata. In *Proceedings of the STOC*, pages 275–286, 1978.
6. M. Sipser. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 21(2):195–202, 1980.