Longer Shortest Strings in Two-Way Finite Automata

Stanislav Krymski and Alexander Okhotin(✉)

Department of Mathematics and Computer Science, St. Petersburg State University, 7/9 Universitetskaya nab, 199034 Saint Petersburg, Russia
krymskiy.stas@yandex.ru, alexander.okhotin@spbu.ru

Abstract. In a recent paper, Dobronravov et al. ("On the length of of shortest strings accepted by two-way finite automata", DLT 2019) prove that the shortest string in a language recognized by an n-state two-way finite automaton (2DFA) can be at least $7^{n/5} - 1$ symbols long, improved to $10^{n/5} - 1 = \Omega(1.584^n)$ in their latest contribution. The lower bound was obtained using "direction-determinate" 2DFA, which always remember their direction of motion at the last step, and used an alphabet of size $\Theta(n)$. In this paper, the method of Dobronravov et al. is extended to a new, more general class: the semi-direction-determinate 2DFA. This yields n-state 2DFA with shortest strings of length $7^{n/4} - 1 = \Omega(1.626^n)$. Furthermore, the construction is adapted to use a fixed alphabet, resulting in shortest strings of length $\Omega(1.275^n)$. It is also shown that an n-state semi-direction-determinate 2DFA can be transformed to a one-way NFA with $O(\frac{1}{\sqrt{n}}3^n)$ states.

1 Introduction

The length of the shortest string in a language is a natural descriptional complexity measure. For one-way nondeterministic finite automata (1NFA) with n states, this length is at most $n - 1$, as the length of the shortest part to an accepting state. For other models, the same question turns out to be much more interesting. The length of shortest strings *not* accepted by an n-state 1NFA was studied by Ellul et al. [5]; Alpoge et al. [1] studied shortest strings in intersections of deterministic one-way automata (1DFA); Chistikov et al. [2] investigated the same question for counter automata. The length of shortest strings in formal grammars was estimated by Pierre [11].

For deterministic two-way finite automata (2DFA) with n states, it is no surprise that the shortest string can be of length exponential in n: the well-known result by Kozen [9] on the PSPACE-completeness of their emptiness problem implicitly relies on this fact. At the same time, an exponential upper bound on this length is given by transforming a 2DFA to a one-way nondeterministic automaton (1NFA) by the method of Kapoutsis [7], which yields $\binom{2n}{n+1} = \Theta(\frac{1}{\sqrt{n}}4^n)$ states. Therefore, the length of the shortest string is less

Research supported by Russian Science Foundation, project 18-11-00100.

than 4^n. Overall, the longest length of a shortest string is of the order $\Theta(1)^n$, with the base of exponentiation bounded by 4. The question is, what is the exact base?

This question was first addressed in a recent paper by Dobronravov et al. [3], who proposed a method for constructing 2DFA with long shortest strings. Their method is based on taking a small base 2DFA with k states and a shortest string of length $\ell - 1$, and then constructing n-state 2DFA that simulate the base automaton on multiple levels for different subsets of the alphabet, ultimately obtaining a shortest string of length $\Omega((\sqrt[k]{\ell})^n)$. The base 2DFA must belong to a subclass of *direction-determinate 2DFA*; these are automata that remember the direction of the last transition in their state. Dobronravov et al. [3,4] presented a base 2DFA with 5 states and with a shortest string of length 9, leading to n-state 2DFA with shortest strings of length $\Omega((\sqrt[5]{10})^n) \geqslant \Omega(1.584^n)$.

The method of Dobronravov et al. [3] relies on finding sophisticated small automata, and the direction-determinance requirement makes it complicated. Furthermore, the n-state 2DFA constructed are also direction-determinate, and since it is known that every n-state automaton from this class can be transformed to an equivalent 1NFA with only $\binom{n}{\lfloor n/2 \rfloor} = \Theta(\frac{1}{\sqrt{n}}2^n)$ states [6], this method cannot possibly provide shortest strings of length 2^n or more.

This paper presents an improvement to the method of Dobronravov et al. [3], based on relaxing the condition of direction-determinacy. A more general class of *semi-direction-determinate two-way automata* is introduced, and it is shown that small examples from this class can be used to construct n-state 2DFA with shortest accepted strings of exponential length. An example of a 3-state semi-direction-determinate 2DFA with a shortest string of length 3 is presented in Sect. 3: to compare, a 3-state direction-determinate 2DFA cannot have shortest string longer than 2 symbols. In Sect. 4, the construction of Dobronravov et al. [3] is generalized to support semi-direction-determinate automata.

The original construction uses an alphabet of size linear in n, the new construction may use exponentially many symbols. In Sect. 5, the construction is improved to use a fixed alphabet independent of n, at the expense of obtaining shorter longest strings.

The resulting new lower bounds on the length of shortest strings are presented in Sect. 6. The constructions are based on a provided example of a 4-state semi-direction-determinate 2DFA with a shortest string of length 6. This leads to n-state 2DFA over a growing alphabet with shortest strings of length $\Omega(1.626^n)$ and 2DFA over a fixed alphabet with shortest strings of length $\Omega(1.275^n)$.

The last result of this paper is a transformation of n-state semi-direction-determinate 2DFA to 1NFA with $O(\frac{1}{\sqrt{n}}3^n)$ states.

2 Definitions

Definition 1. *A* nondeterministic two-way finite automaton *(2NFA) is a quintuple $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$, in which:*

- *Σ is a finite alphabet, the tape is bounded by a left end-marker $\vdash \notin \Sigma$, and a right end-marker $\dashv \notin \Sigma$;*
- *Q is a finite set of states;*
- *$Q_0 \subseteq Q$ is the set of initial states;*
- *$\delta \colon Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1,+1\}}$ is the transition function, which specifies all possible transitions in a certain state while observing a certain tape symbol;*
- *$F \subseteq Q$ is the set of accepting states, effective at the right end-marker $\dashv$.*

On an input string $w \in \Sigma^*$, a 2NFA operates on a read-only tape containing this string enclosed within end-markers ($\vdash w \dashv$). It begins its computation in any initial state at the left end-marker ($\vdash$). At every step, when $\mathcal{A}$ is in a state $q \in Q$ and observes a symbol $a \in \Sigma \cup \{\vdash, \dashv\}$, the transition function $\delta(q, a)$ provides a set of pairs (q', d) of the next state q' and the direction of head's motion, $d \in \{-1, +1\}$. If any sequence of nondeterministic choices leads the automaton to an accepting state while at the right end-marker ($\dashv$), then the string is said to be accepted.

The set of all accepted strings, denoted by $L(\mathcal{A})$, is the language recognized by the 2NFA.

Other types of finite automata are obtained by restricting 2NFA. An automaton is *deterministic* (2DFA), if $|Q_0| = 1$ and $|\delta(q, a)| \leqslant 1$ for all q and a. An automaton is *direction-determinate* [10], if, for every state $q \in Q$, all transitions to q move the head in the same direction $d(q) \in \{-1, +1\}$.

In a *one-way* automaton (1DFA or 1NFA), all transitions move its head to the right, so that the automaton makes a single left-to-right pass, accepting or rejecting in the end. End-markers are of no use in one-way automata, and are usually omitted from the definition.

Theorem A (Kapoutsis *[7]***).** *For every n-state 2NFA, there exists a 1NFA with $\binom{2n}{n+1} = \Theta(\frac{1}{\sqrt{n}} 4^n)$ states, which recognizes the same language.*

Since a k-state 1NFA cannot have a shortest accepted string of length greater than $k - 1$, this has the following implication.

Corollary B *For every n-state 2NFA, the length of the shortest string it accepts is at most $\binom{2n}{n+1} - 1$.*

For direction-determinate automata, the bound in Theorem A is reduced by adapting the method of Kapoutsis [7] to produce fewer states. Accordingly, the length of their shortest strings cannot exceed the following bound.

Theorem C (Geffert and Okhotin *[6]***).** *For every n-state direction-determinate 2NFA, there is a 1NFA with $\binom{n}{\lfloor n/2 \rfloor} = \Theta(\frac{1}{\sqrt{n}} 2^n)$ states that recognizes the same language.*

As far as shortest strings are concerned, 2DFA have the same power as 2NFA.

Theorem D (Dobronravov et al. *[3]***).** *For every n-state 2NFA, there exists an n-state 2DFA with the shortest string of the same length.*

The construction increases the size of the alphabet by a factor of n^n.

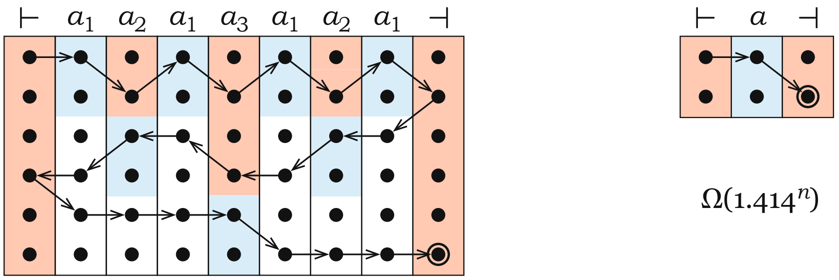


Fig. 1. (left) A 3-pass 2DFA with $2 \cdot 3$ states and with a shortest string of length $2^3 - 1$; (right) the base automaton.

3 Shortest Strings in 2DFA

Two-way automata with n states and with the shortest string of length exponential in n can be constructed by the following simple method.

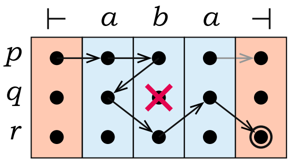
Example E ([3]). *Assume that $n = 2k$, with k odd, and let the alphabet be $\Sigma_k = \{a_1, \ldots, a_k\}$. Consider a 2DFA that makes k passes over the string. At the first pass, the automaton regards all symbols $a_2, \ldots, a_k$ as separators, and verifies that there is at least one symbol a_1 between every two separators. Similarly, at each i-th pass, the symbols $a_{i+1}, \ldots, a_k$ are regarded as separators, and the automaton checks that there is at least one symbol a_i between every two separators. Each pass uses two states, and the shortest accepted string is of length $2^k - 1$.*

For $k = 3$, the computation of the resulting 6-state automaton is illustrated in Fig. 1(left).

This yields automata with shortest strings of length $2^{n/2} - 1$. Together with Corollary B, this example shows that the longest length of the shortest string accepted by an n-state 2DFA is of the order $\Theta(1)^n$, where the base is between 1.414 and 4. The question is, what is the exact base?

The method of constructing 2DFA with longer shortest accepted strings, invented by Dobronravov et al. [3], begins with the following interpretation of Example E. At each i-th pass, counting up to two can be regarded as a simulation of a 1DFA presented in Fig. 1(right) on the symbols a_i. Any symbols $\{a_1, \ldots, a_{i-1}\}$ encountered are ignored, that is, the 2DFA continues moving without changing its state. For any separator from $\{a_{i+1}, \ldots, a_k, \vdash, \dashv\}$, the 2DFA checks that the currently simulated instance of the 1DFA is in an accepting state, and restarts the simulation in anticipation of the next substring enclosed between two separators.

Dobronravov et al. [3] extended this idea to use a direction-determinate 2DFA as a base automaton. Direction-determinance is essential for the following reason: when the constructed automaton is at an i-th pass simulating the original automaton's being in a state q, and it scans one of the symbols $a_1, \ldots, a_{i-1}$ that it is expected to skip, it knows that the state q is reachable only from the



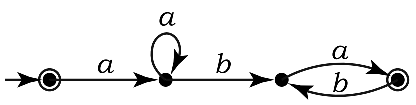


Fig. 2. (left) A 3-state 2DFA with a shortest string of length 3; (right) the equivalent 1DFA.

direction $d(q)$, and therefore can determine the direction in which to proceed: it will be $d(q)$ for i odd and $-d(q)$ for i even.

The other component of the proof of Dobronravov et al. [3] is a single example of a 5-state direction-determinate 2DFA with a shortest string of length 9. Iterating it in the same way as in Example E yields a family of n-state 2DFA with shortest accepted strings of length $10^{n/5} - 1 = \Omega(1.584^n)$ [3].

This paper extends the method of Example E beyond direction-determinate base automata. The new, less restrictive family of base automata is best illustrated by the following small specimen.

Example 1. Let $\mathcal{A}$ be a 2DFA over the alphabet $\Sigma = \{a, b\}$, with the states $Q = \{p, q, r\}$, where p is initial and r is accepting, and with the following transitions:

$$\begin{aligned} &\delta(p, \vdash) = (p, +1), \\ &\delta(p, a) = (p, +1), \quad \delta(q, a) = (r, +1), \\ &\delta(p, b) = (q, -1), \quad \delta(r, b) = (q, +1). \end{aligned}$$

The shortest string accepted by $\mathcal{A}$ is $w = aba$, as illustrated in Fig. 2(left). To see that w is indeed the shortest string accepted by $\mathcal{A}$, it is sufficient to transform this automaton to the minimal equivalent partial 1DFA, which is presented in Fig. 2(right). The shortest string is clearly visible in the figure.

The above automaton is not direction-determinate, since the state q is enterable both from the left and from the right. The new notion of *semi-direction-determinate automata* allows such states, but imposes special restrictions on each of them. This new type of automata, which generalizes direction-determinate automata, is defined as follows.

Definition 2. *A 2DFA $(\Sigma, Q, q_0, \delta, F)$ is called* semi-direction-determinate*, if there exists a partial function $d\colon Q \to \{-1, +1\}$, such that:*

1. *every transition $\delta(p, a) = (q, d)$ leading to a state q with $d(q)$ defined moves the head in the prescribed direction $d = d(q)$;*
2. *whenever a transition $\delta(p, a) = (q, d)$ leads to a state q with $d(q)$ undefined, the transition $\delta(q, a)$ may either proceed to (q, d), or be undefined.*

The 2DFA in Example 1 is semi-direction-determinate, with $d(p) = d(r) = +1$ and $d(q)$ undefined. Transitions leading to q are $\delta(p, b) = (q, -1)$, and $\delta(r, b) = (q, +1)$; since $\delta(q, b)$ is undefined, these transitions are allowed.

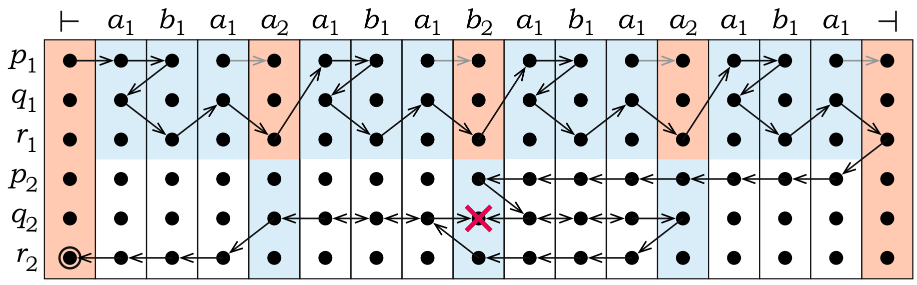


Fig. 3. A 2NFA obtained by iterating the semi-direction-determinate 2DFA in Example 1 twice, as it accepts its shortest string.

It turns out that the condition of semi-direction-determinance is sufficient to iterate the example in generally the same way as in Fig. 1, and thus to obtain longer shortest strings than presented by Dobronravov et al. [3].

4 Iterating Semi-direction-determinate Automata

The proposed new construction of two-way finite automata, given below, actually produces a nondeterministic automaton, which is then processed by Theorem D. A 2NFA obtained in this way, along with its shortest string, is illustrated in Fig. 3.

Lemma 1. *Let $\mathcal{A} = (\Sigma, Q, q^{init}, \delta, F)$ be a semi-direction-determinate 2DFA with k states, which satisfies a further technical condition: for every state $q \in Q$, if $\delta(q, \vdash)$ is defined, then $d(q) = -1$, and if $\delta(q, \dashv)$ is defined or $q \in F$, then $d(q) = +1$. Let $\ell - 1$ be the length of the shortest string accepted by $\mathcal{A}$. Then, for every odd number $m \geqslant 3$, there exists a km-state 2NFA $\mathcal{B}_m$, defined over an alphabet of size $m \cdot |\Sigma|$, with the shortest accepted string of length $\ell^m - 1$.*

Proof. Let $Q = Q_{+1} \cup Q_{-1} \cup Q_?$ be $\mathcal{A}$'s set of states, where $Q_{+1} = \{\, q \mid d(q) = +1 \,\}$, $Q_{-1} = \{\, q \mid d(q) = -1 \,\}$ and $Q_? = \{\, q \mid d(q) \text{ is not defined} \,\}$. The new 2NFA $\mathcal{B}_m$ is defined over the alphabet $\Omega = \bigcup_{i=1}^{m} \Sigma_i$, where $\Sigma_i = \{\, a_i \mid a \in \Sigma \,\}$. It makes m passes over the input. At an i-th pass, with $i \in \{1, \ldots, m\}$, it sees its tape as $\vdash u_0 \#_1 u_1 \#_2 \ldots \#_n u_n \dashv$, where $\#_1, \ldots, \#_n \in \{a_{i+1}, \ldots, a_m\}$ are separators and $u_0, \ldots, u_n \in (\Sigma_1 \cup \ldots \cup \Sigma_i)^*$ are the substrings they separate.

The substrings are processed one by one, from left to right for odd i, and from right to left for even i. For each string u_j, the automaton $\mathcal{B}_m$ simulates the computation of $\mathcal{A}$ on that string (if i is odd) or on its reverse (if i is even), taking into account only symbols a_i, with $a \in \Sigma$. All other symbols a_j, with $j < i$ and $a \in \Sigma$, are ignored by passing over them without changing the state: for states q with $d(q)$ defined, the new automaton knows in which direction to proceed; and if $d(q)$ is undefined, then the automaton moves nondeterministically in either direction.

Each symbol $\#_t$ separates u_{t-1} from u_t, and when the automaton $\mathcal{B}_m$ reaches this symbol in a state q_i, it is expected to simulate the computation of $\mathcal{A}$ on an end-marker. By the technical assumption, $\mathcal{A}$ may have a transition or acceptance there only if $d(q)$ is defined, and $\mathcal{B}$ knows from $d(q)$, whether it currently simulates $\mathcal{A}$ on u_{i-1} or on u_i.

The automaton $\mathcal{B}_m$ uses the set of states $Q_m = \{\, q_i \mid q \in Q,\ i \in \{1, \ldots, m\} \,\}$. A state q_i means simulating $\mathcal{A}$ in the state $q \in Q$ at the i-th pass. At odd-numbered passes, the substrings $u_0, \ldots, u_n$ are processed from left to right, and from right to left at even-numbered passes. Let $d(i)$ be the general direction of traversal at the i-th pass: $d(i) = +1$ for odd i and $d(i) = -1$ for even i.

The automaton $\mathcal{B}_m$ is constructed as semi-direction-determinate, with $d(q_i) = d(q) \cdot d(i)$ if $d(q)$ is defined, and $d(q_i)$ undefined if so is $d(q)$.

Let the initial transition of $\mathcal{A}$ be $\delta(q^{init}, \vdash) = r$. Then, the initial state of $\mathcal{B}_m$ is q_1^{init}, with the following initial transition.

$$\delta'(q_1^{init}, \vdash) = (r_1, +1) \tag{1a}$$

At every i-th pass, with $i \in \{1, \ldots, m\}$, each $\mathcal{A}$'s transition $\delta(q, a) = (r, d)$, with $a \in \Sigma$ and $q, r \in Q$, is implemented in $\mathcal{B}_m$ by the following transition on the symbol a_i with a matching subscript.

$$\delta'(q_i, a_i) = (r_i, d \cdot d(i)) \tag{1b}$$

Each lesser symbol a_j, with $j < i$ and $a \in \Sigma$, is ignored by continuing in the same direction. Whenever $\mathcal{B}_m$ simulates $\mathcal{A}$ in a state q with the direction $d(q)$ defined, it knows in which direction to proceed.

$$\delta'(q_i, a_j) = (q_i, d(q) \cdot d(i)), \qquad \text{for } q \in Q,\ d(q) \text{ is defined}, j < i,\ a \in \Sigma \tag{1c}$$

If $d(q)$ is undefined, the automaton $\mathcal{B}_m$ can move in either direction nondeterministically (and this is the only place where $\mathcal{B}_m$ uses its nondeterminism).

$$\delta'(q_i, a_j) = \big\{(q_i, +1), (q_i, -1)\big\}, \quad \text{for } q \in Q,\ d(q) \text{ undefined}, j < i,\ a \in \Sigma \tag{1d}$$

Following these transitions, the automaton can freely move over a substring $x \in (\Sigma_1 \cup \ldots \cup \Sigma_{i-1})^*$ in a state q_i. If $\mathcal{B}_m$ ever crosses this substring, then it correctly simulates one transition of $\mathcal{A}$. If it returns to the symbol a_i from which it entered x, then, by the definition of semi-direction-determinancy, it cannot proceed anywhere else except back into x in the same state q_i. For this reason, regardless of the nondeterministic choices it makes, $\mathcal{B}_m$ can either continue simulating $\mathcal{A}$, or loop.

Next, let $¢_i$ and $\$_i$ be the end-markers at which the i-th pass begins and ends, respectively ($¢_i = \vdash$ and $\$_i = \dashv$ for i odd, and vice versa for i even). For each $\mathcal{A}$'s transition $\delta(q, \vdash) = (r, +1)$ for turning at the left end-marker, with $q \neq q^{init}$, the new automaton executes the same turn on any separator symbols.

$$\delta'(q_i, s) = (r_i, d(i)), \qquad \text{for } s \in \{¢_i\} \cup \Sigma_{i+1} \cup \ldots \cup \Sigma_m \tag{1e}$$

Each turn at the right end-marker, $\delta(q, \dashv) = (r, -1)$, is implemented similarly.

$$\delta'(q_i, s) = (r_i, -d(i)), \qquad \text{for } s \in \{\$_i\} \cup \Sigma_{i+1} \cup \ldots \cup \Sigma_m \tag{1f}$$

If $q \in F$ is an accepting state of $\mathcal{A}$, effective at the right end-marker ($\dashv$), then $\mathcal{B}_m$ moves on to the next block through a separator symbol: if the initial transition is $\delta(q^{init}, \vdash) = (r, +1)$, it goes through the separator in the state r, thus simulating the end of one computation and the beginning of another.

$$\delta'(q_i, s) = (r_i, d(i)), \qquad \text{for } s \in \Sigma_{i+1} \cup \ldots \cup \Sigma_m \tag{1g}$$

When $\mathcal{B}_m$ finishes processing the last block at its i-th pass, it proceeds to the next pass.

$$\delta'(q_i, \$_i) = (r_{i+1}, d(i+1)) \qquad (i < m) \tag{1h}$$

For $i = m$, the automaton $\mathcal{B}_m$ accepts; accordingly, its set of accepting states is $F' = \{q_m\}$.

Note that, by the technical assumption, case (1e) is possible only for $d(q) = -1$, whereas cases (1f–1h) require $d(q) = +1$ and are mutually exclusive. Hence, at most one of these transitions may be defined.

The language recognized by this automaton is expressed as follows. For each $i \in \{0, 1, \ldots, m\}$, let $L_i \subseteq (\Sigma_1 \cup \ldots \cup \Sigma_i)^*$ be the language representing all substrings, on which the computation of $\mathcal{A}$ is successfully simulated at the i-th pass. Then, $L_0 = \{\varepsilon\}$, $L_i = \bigcup_{a^{(1)} \ldots a^{(n)} \in L(\mathcal{A})} L_{i-1} a_i^{(1)} L_{i-1} a_i^{(2)} L_{i-1} \ldots a_i^{(n)} L_{i-1}$ for odd i, and symmetrically for even i. The automaton $\mathcal{B}_m$ recognizes exactly L_m, and the length of the shortest string therein is $\ell^m - 1$. The proof is omitted due to space constraints. □

It is also worth note that, once the transformation in Theorem D is applied to the 2NFA produced by Lemma 1, the resulting 2DFA is semi-direction-determinate.

For the 3-state base automaton in Example 1, with a shortest string of length 3, Lemma 1 yields a lower bound $\Omega(4^{n/3}) = \Omega(1.587^n)$ for a growing alphabet. This already improves over the previously known result.

5 Encoding in a Fixed Alphabet

Example E, as well as all other constructions of DFA with exponentially long shortest accepted strings known to date, essentially rely on using an alphabet that grows with n. The construction by Dobronravov et al. [3] uses an alphabet of size $\Theta(n)$; the new construction in Lemma 1 provides a 2NFA using $\Theta(n)$ symbols, which is then turned to a 2DFA with exponentially many symbols.

For a fixed alphabet, no results on the length of shortest accepted strings are known yet. The first such result shall now be presented. This is an adaptation of the construction in Lemma 1, in which every symbol a_j is replaced by an encoding over a fixed alphabet; a new 2DFA carries out a computation simular to the one in Lemma 1. It uses twice as many states as in the original version, resulting in a weaker lower bound.

Lemma 2. *Let $\mathcal{A} = (\Sigma, Q, q^{init}, \delta, F)$ be a k-state semi-direction-determinate 2DFA that satisfies the conditions of Lemma 1. Let $\ell - 1$ be the length of its shortest string. Then, for every even number $m \geqslant 2$, there exists a $(2(m-1)k + \frac{3}{2}m - 1)$-state 2NFA $\mathcal{C}_m$, defined over an alphabet with $2|\Sigma| + 1$ symbols, with the shortest accepted string of length at least ℓ^m.*

Proof (a sketch). Given a base semi-direction-determinate 2DFA over an alphabet Σ, the new 2NFA uses the alphabet $\Omega = \Sigma_{\pm 1} \cup \{s\}$, where $\Sigma_{\pm 1} = \{\, a_d \mid a \in \Sigma,\ d \in \{-1, +1\} \,\}$. The strings in the original construction shall be encoded by a homomorphism h, with $h(a_{2j+1}) = s^j a_{+1} s^{m-2-j}$ for odd-numbered symbols, and $h(a_{2j+2}) = s^j a_{-1} s^{m-2-j}$ for even-numbered symbols.

The new 2DFA shall first check that the input string is a well-formed image of some string, and then proceed with an m-pass simulation, using $2(m-1)|Q|$ states, cf. $m|Q|$ states in Lemma 1. After the last pass, one more state is used to move the head to the right end-marker.

Checking that an input string is an image under h takes a partial DFA with $\frac{3}{2}m - 2$ states; the construction is easy. For the simulation, the 2DFA shall use states of the form $q_{i,d}$, where $q \in Q$, $-(\frac{m}{2}-1) \leqslant i \leqslant \frac{m}{2}-1$ and $d \in \{-1, +1\}$. The subscript d indicates the general direction of the current pass, that is, $d = +1$ for the first pass, $d = -1$ for the second pass, etc. The subscript i reflects the number of the current pass whenever the automaton is at the first symbol of the image of some symbol; as the automaton moves over the image, the subscript i is in constant rotation: the automaton increments i whenever it moves to the left, and decrements it when it moves to the right. Once i exceeds $\frac{m}{2} - 1$, the counting is wrapped to $-(\frac{m}{2} - 1)$, and the other way around. This allows the automaton to compare the number of the current pass to the number of the encoded symbol.

The computation involves keeping track of several directions, and it order to explain it more clearly, it shall be described for the case of a left-to-right pass in a right-moving state. Having entered an image $h(a_{2j+1}) = s^j a_{+1} s^{m-2-j}$ in a state $q_{-i,+1}$ with $d(q) = +1$, the automaton moves to the right while inrementing i at every step, and arrives to the symbol a_{+1} in the state $q_{j-i,+1}$. If $j - i < 0$, this means that this symbol must be ignored, and the automaton proceeds further to the right while incrementing i, entering the next image in the same state $q_{-i,+1}$.

If $j - i = 0$, the automaton simulates the transition on a. If the original automaton's transition was $\delta(q, a) = (r, d(r))$, then the new automaton moves in the direction $d(r)$ in the state $r_{0+d(r),+1}$, and eventually moves out of the image in the direction $d(r)$ in the state $r_{-i,+1}$.

If $j-i > 0$, the automaton treats this symbol as an end-marker and takes the appropriate action. Since the automaton is now in a state $q_{j-i,+1}$ with $d(q) = +1$, this must be a right end-marker. If the original automaton had a transition $\delta(q, \dashv) = (r, -1)$, then the new automaton moves in the direction -1 in the state $r_{j-i-1,+1}$, and later reaches the first symbol of the image in the state $r_{-i,+1}$, leaving the image to the left. If the original automaton accepts in q, the new automaton should proceed to the next substring to the right; accordingly, if the initial transition is $\delta(q^{init}, \dashv) = (r, +1)$, the new automaton moves to the right in the state $r_{j-i+1,+1}$ and eventually leaves the image in the direction $+1$ in the state $r_{-i,+1}$.

The full list of transitions is omitted due to space constraints. $\square$

An immediate application of Lemma 2 yields the following result.

Theorem 1. *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be a k-state semi-direction-determinate 2DFA with a shortest string of length $\ell - 1$, which satisfies the conditions of Lemma 1. Then, there exists a fixed alphabet Γ, such that for every n, there exists an n-state semi-direction-determinate 2DFA over Γ with a shortest string of length at leas $\Omega((\sqrt[2k+\frac{3}{2}]{\ell})^n)$.*

For instance, the automaton in Example 1 has $k = 3$ and $\ell = 4$, and Lemma 2 provides automata with $\frac{15}{2}m - 7$ states and with a shortest string of length ℓ^m. For an n-state base automaton, the length of the shortest string is then of the order $\Omega((4^{2/15})^n) = \Omega(1.203^n)$.

This can be improved by first iterating the base automaton using Lemma 1, obtaining a larger base automaton, and only then applying Lemma 2.

Theorem 2. *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be a k-state semi-direction-determinate 2DFA with a shortest string of length $\ell - 1$, which satisfies the conditions of Lemma 1. Then, for every $\varepsilon > 0$, there exists an alphabet Γ, such that for every n, there exists an n-state semi-direction-determinate 2DFA over Γ with a shortest string of length at least $\Omega((\sqrt[k]{\ell} - \varepsilon)^n)$.*

With this improvement, the base automaton in Example 1 provides shortest strings of length $\Omega((\sqrt[6]{4} - \varepsilon)^n) = \Omega(1.259^n)$ over a fixed alphabet.

6 Automata with Longer Shortest Strings

The efficiency of the proposed method relies on finding small examples of semi-direction-determinate 2DFA with long shortest accepting strings. Using a better base example given below leads to further improvement.

Example 2. Let $\mathcal{A}$ be a 2DFA over the alphabet $\Sigma = \{a, b, c, d\}$, with the states $Q = \{p, q, r, s\}$, where p is initial and s is accepting, and with transitions illustrated in Fig. 4. It is semi-direction-determinate with $d(p) = d(q) = d(s) = +1$ and $d(r)$ undefined. The shortest string accepted by $\mathcal{A}$ is $w = abcdbc$, this can be verified by transforming it to a 1DFA.

Corollary 1. *For every n, there exists a semi-direction-determinate 2DFA over an alphabet of size exponential in n, with a shortest string of length at least $\Omega(7^{n/4}) = \Omega(1.626^n)$.*

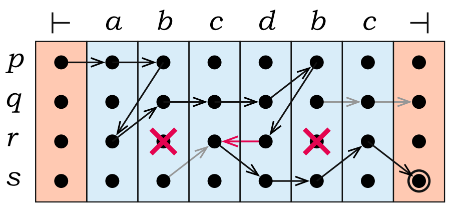


Fig. 4. A 4-state 2DFA with a shortest string of length 6.

Corollary 2. *For every* n*, there exists a semi-direction-determinate 2DFA over a fixed alphabet, with a shortest string of length at least* $\Omega((\sqrt[8]{7}-\varepsilon)^n) = \Omega(1.275^n)$*.*

7 Transforming Semi-direction-determinate to One-Way

The method of Donbronravov et al. [3] can potentially provide automata with shortest strings of length up to at most $O(\frac{1}{\sqrt{n}}2^n)$, since direction-determinate 2DFA can be transformed to 1NFA with this number of states. Although the proposed new method is not subject to this limitation, it has its own limitations, revealed by the following result.

Theorem 3. *For every* n*-state semi-direction-determinate 2DFA there exists a 1NFA with* $\sum_{k=0}^{n-1}\binom{n}{k}\binom{n-k}{k+1} = \Theta(\frac{1}{\sqrt{n}}3^n)$ *states, which recognizes the same language.*

The construction is based on the known transformation of an arbitrary 2DFA to an 1NFA by Kapoutsis [7], Upon reading a prefix u of an input string, the 1NFA remembers the states in which the 2DFA crosses the border between the last symbol of u and the next symbol to the right. This is represented by a pair (P, R), with $P, R \subseteq Q$ and $|P| = |R| + 1$. It turns out that for a semi-direction-determinate automaton, pairs (P, R) with $P \cap R \neq \varnothing$ are useless and can be omitted. The number of remaining pairs is $\binom{n}{k}\binom{n-k}{k+1}$ for a fixed cardinality $|P| = k$, and summation over all k yields the desired formula. This is a known integer sequence, OEIS A005717 [12], and it is of the order $\Theta(\frac{1}{\sqrt{n}}3^n)$.

8 Conclusion

This paper has made a new addition to the zoo of different variants of two-way finite automata, such as sweeping, direction-determinate, halting, reversible, nondeterministic, alternating, pebble, etc. Understanding the difference between these variants is an important research subject; in particular, the relative size of 2NFA and 2DFA appears to be the key to solving the *L vs. NL* problem in the complexity theory [8]. The length of the shortest string is a natural complexity measure that may be useful to compare some of these models.

Table 1. The known bounds on the length of shortest strings for subfamilies of 2DFA.

Family	Lower bound	Upper bound
Sweeping	$\Omega((\sqrt[3]{3})^n) = \Omega(1.442^n)$	$\binom{n}{\lfloor n/2 \rfloor} = \Theta(\frac{1}{\sqrt{n}}2^n)$
Direction-determinate	$\Omega((\sqrt[5]{10})^n) = \Omega(1.584^n)$	$\binom{n}{\lfloor n/2 \rfloor} = \Theta(\frac{1}{\sqrt{n}}2^n)$
Semi-direction-determinate	$\boldsymbol{\Omega((\sqrt[4]{7})^n) = \Omega(1.626^n)}$	$\boldsymbol{O(\frac{1}{\sqrt{n}}3^n)}$
All 2DFA, and also 2NFA	$\boldsymbol{\Omega((\sqrt[4]{7})^n) = \Omega(1.626^n)}$	$\binom{2n}{n+1} = \Theta(\frac{1}{\sqrt{n}}4^n)$

The known lower and upper bounds on the length of shortest accepted strings in n-state 2DFA from different subclasses are presented in Table 1. There is still a long way to go to any precise answers.

Acknowledgement. The authors are grateful to the anonymous reviewers for careful reading and for pertinent remarks.

References

1. Alpoge, L., Ang, T., Schaeffer, L., Shallit, J.: Decidability and shortest strings in formal languages. In: Holzer, M., Kutrib, M., Pighizzini, G. (eds.) DCFS 2011. LNCS, vol. 6808, pp. 55–67. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22600-7_5
2. Chistikov, D., Czerwiński, W., Hofman, P., Pilipczuk, M., Wehar, M.: Shortest paths in one-counter systems. In: Jacobs, B., Löding, C. (eds.) FoSSaCS 2016. LNCS, vol. 9634, pp. 462–478. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49630-5_27
3. Dobronravov, E., Dobronravov, N., Okhotin, A.: On the length of shortest strings accepted by two-way finite automata. In: Hofman, P., Skrzypczak, M. (eds.) DLT 2019. LNCS, vol. 11647, pp. 88–99. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24886-4_6
4. Dobronravov, E., Dobronravov, N., Okhotin, A.: On the length of shortest strings accepted by two-way finite automata, revised full version, submitted
5. Ellul, K., Krawetz, B., Shallit, J., Wang, M.-W.: Regular expressions: new results and open problems. J. Automata Lang. Comb. **10**(4), 407–437 (2005)
6. Geffert, V., Okhotin, A.: One-way simulation of two-way finite automata over small alphabets. In: NCMA 2013 (Umeå, vol. 13–14, Sweden, August 2013
7. Kapoutsis, C.: Removing bidirectionality from nondeterministic finite automata. In: Jędrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 544–555. Springer, Heidelberg (2005). https://doi.org/10.1007/11549345_47
8. Kapoutsis, C.A.: Two-way automata versus logarithmic space. Theory Comput. Syst. **55**(2), 421–447 (2014)
9. Kozen, D.: Lower bounds for natural proof systems. In: FOCS 1977, pp. 254–266 (1977). http://dx.doi.org/10.1109/SFCS.1977.16
10. Kunc, M., Okhotin, A.: Reversibility of computations in graph-walking automata. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 595–606. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40313-2_53

11. Pierre, L.: Rational indexes of generators of the cone of context-free languages. Theor. Comput. Sci. **95**(2), 279–305 (1992). https://doi.org/10.1016/0304-3975(92)90269-L
12. Sloane, N.J.A. (ed.): The On-Line Encyclopedia of Integer Sequences, published electronically at https://oeis.org