Deterministic One-Way Simulation of Two-Way Deterministic Finite Automata over Small Alphabets

Viliam Geffert[1(✉)] and Alexander Okhotin[2(✉)]

[1] Department of Computer Science, P. J. Šafárik University, Jesenná 5, 04154 Košice, Slovakia
viliam.geffert@upjs.sk

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

Abstract. It is shown that a two-way deterministic finite automaton (2DFA) with n states over an alphabet Σ can be transformed to an equivalent one-way automaton (1DFA) with $|\Sigma| \cdot \mathcal{F}(n) + 1$ states, where $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1} \leq (n+1)^{n+1}/(\ln(n+1) \cdot e^{1-o(1)})^{n+1}$.

This reflects the fact that, by keeping the last processed symbol in memory, the simulating 1DFA needs to remember only the state from which the 2DFA leaves the prefix read so far for the first time to the right together with a function that maps some $n-k$ states moving to the left from the last processed symbol to some other k states moving to the right from this symbol. This reduces the number of functions describing the behaviour of the 2DFA on the prefix read so far.

A close lower bound of $\mathcal{F}(n)$ states is established using a 5-symbol alphabet. The complexity of transforming a sweeping or a direction-determinate 2DFA to a 1DFA is shown to be exactly $\mathcal{F}(n)$.

Keywords: Finite automata · Two-way automata · State complexity

1 Introduction

The state complexity of transforming two-way finite automata to one-way automata has received much attention in the literature. For deterministic (2DFA) and nondeterministic (2NFA) automata, the exact complexity of transforming them to deterministic and nondeterministic one-way automata (1DFA, 1NFA) was determined by Kapoutsis [5]. Similar problems for related models were investigated as well, and the state complexity of transformations involving sweeping automata [14], complements of deterministic two-way automata [15], alternating automata [4], unambiguous automata [11] was estimated. For the basic transformations of 2DFA/2NFA to 1DFA/1NFA, the case of a unary alphabet has

Supported by the Slovak grant contract VEGA 1/0177/21.

Y.-S. Han and S.-K. Ko (Eds.): DCFS 2021, LNCS 13037, pp. 26–37, 2021.
https://doi.org/10.1007/978-3-030-93489-7_3

received special attention [2,7,9], and its complexity was shown to be much less than in the general case studied by Kapoutsis [5].

Even though all four bounds by Kapoutsis [5] are precise, his lower bound arguments rely on using alphabets of exponential size, and this leaves open the state complexity of this transformation for small alphabets. As previously shown by the authors [3], the 2DFA-to-1NFA transformation can in fact be improved from $\binom{2n}{n+1}$ to $|\Sigma| \cdot \binom{n}{\lfloor n+1 \rfloor} + 1$ states, and a close lower bound of $\binom{n}{\lfloor n+1 \rfloor}$ states holds already for a 2-symbol alphabet. The purpose of this paper is to improve the 2DFA-to-1DFA transformation in a similar way.

A transformation of 2DFA to 1DFA was known since the introduction of two-way automata by Rabin and Scott [12]. Shepherdson [13] presented a transformation of an n-state 2DFA to a 1DFA with ca. $(n+1)^{n+1}$ states. The lower bound was given by Moore [10], who constructed an n-state 2DFA over a fixed alphabet, for which the every 1DFA must have at least $(\frac{n-5}{2})^{\frac{n-5}{2}}$ states. Much later, the precise succinctness tradeoff between 2DFA and 1DFA was determined by Kapoutsis [5], who showed that $\mathcal{K}(n) = n(n^n - (n-1)^n) + 1$ states are sufficient and in the worst case necessary, with the lower bound established over a growing alphabet with around n^n symbols.

This paper investigates the transformation of 2DFA to 1DFA for small alphabets. Classical transformations, which are recalled in Sect. 3, compute the outcomes of all computations of the 2DFA on the prefix read by the 1DFA. According to the new method, presented in Sect. 4, the constructed 1DFA additionally remembers *the last symbol read*, and this allows it to store less information on the computations: the resulting number of states is $|\Sigma| \cdot \mathcal{F}(n) + 1$, where $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1}$. Then, provided that the alphabet is small, the overall number of states is reduced.

As a side note, if the original 2DFAs are sweeping or direction-determinate, the complexity of transforming them to 1DFAs is shown to be exactly $\mathcal{F}(n)$, regardless of the size of the input alphabet.

The proposed construction for small alphabets is actually fairly close to optimal. As shown in Sect. 5, already for a fixed 5-symbol alphabet, there exist n-state 2DFA that require 1DFA with at least $\mathcal{F}(n)$ states.

The growth rate of the function $\mathcal{F}(n)$ is estimated in Sect. 6, and it is shown that $\mathcal{F}(n) \leq (n+1)^{n+1}/(\ln(n+1) \cdot e^{1-o(1)})^{n+1}$. Therefore, for every alphabet of size subexponential in n, with $|\Sigma| \leq \frac{e-1}{e^2} \cdot (\ln(n+1) \cdot e^{1-o(1)})^{n+1}$, the proposed transformation yields fewer states than the exact bound by Kapoutsis [5].

2 Two-Way Automata

Definition 1. *A* two-way deterministic finite automaton *(2DFA) is a quintuple* $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$*, in which* Σ *is a finite alphabet, which does not contain two special symbols: the left endmarker* ($\vdash$) *and the right endmarker* ($\dashv$)*;* Q *is a finite set of states;* $q_0 \in Q$ *is the initial state;* $\delta : Q \times (\Sigma \times \{\vdash, \dashv\}) \to Q \times \{-1, +1\}$ *is a partially defined transition function; and* $F \subseteq Q$ *is the set of accepting states, effective at the right endmarker* $\dashv$*.*

Given an input string $w \in \Sigma^*$, a 2DFA operates on a read-only tape containing the string $\vdash w \dashv$. It begins its computation in the state q_0, with the head observing the left endmarker $\vdash$. At every step of the computation, when $\mathcal{A}$ is in a state $q \in Q$ and observes a square of the tape containing a symbol $a \in \Sigma \cup \{\vdash, \dashv\}$, the value $\delta(q, a)$ defines the next state and the direction of motion. The computation of $\mathcal{A}$ on w is defined uniquely; if $\mathcal{A}$ eventually reaches an accepting state in F while at the right endmarker $\dashv$, this computation is accepting; otherwise, it either encounters an undefined transition or gets into an infinite loop. The set of strings, on which the computation is accepting, is the language recognized by the 2DFA, denoted by $L(\mathcal{A})$.

Two-way automata of a special kind, which remember the direction of their motion and can turn only on the endmarkers, are known as sweeping [14]. There is also a larger class of direction-determinate automata [8] that remember the direction in which they came to the current state, but may turn at any point.

Definition 2. *A 2DFA is said to be* direction-determinate*, if there is a partition $Q = Q_{-1} \uplus Q_{+1}$, so that every transition $\delta(p, a) = (q, d)$ must satisfy $q \in Q_d$. The states in Q_{-1} are called* left-bound*, those in Q_{+1}* right-bound*.*

A direction-determinate 2DFA is called sweeping*, if $\delta(p, a) = (q, d)$ implies $p, q \in Q_d$, as long as $a \in \Sigma$ (that is, the symbol a is not an endmarker).*

One-way automata (1DFA) are a special case of 2DFA, which move to the right after every transition. This makes the endmarkers unnecessary.

3 The Known Simulation of 2DFA by 1DFA

The new transformation of 2DFA to 1DFA presented in this paper is based on the classical transformation by Shepherdson [13], with the refinements by Kapoutsis [6] that made it optimal.

Theorem 1 (Kapoutsis [5,6]). *For every n-state 2DFA, there exists a partial 1DFA with $\mathcal{K}(n) = n \cdot (n^n - (n-1)^n)$ states recognizing the same language.*

Proof (a sketch). The idea is to precompute all computations of the 2DFA on the processed prefix of the input string, and to remember their outcomes in the state of the constructed 1DFA. These precomputed computations can later be joined together into longer computations, and eventually the 1DFA can determine the outcome of the single computation beginning in the initial configuration.

Let $(Q, \Sigma, \delta, q_0, F)$ be a 2DFA. The 1DFA has states of the form $(\hat{q}, f)$, where $\hat{q} \in Q$ is a state of the 2DFA and $f : Q \to Q$ is a function mapping states to states. On an input string $w \in \Sigma^*$, the 1DFA reaches a state $(\hat{q}, f)$, for which

- in the computation of the 2DFA on $\vdash w$, the first time it leaves the rightmost symbol of $\vdash w$ to the right, it does so in the state $\hat{q}$;
- if the 2DFA begins its computation at the rightmost symbol of $\vdash w$ in a state q, then it eventually moves to the right of this rightmost symbol in the state $f(q)$, and if this computation of a 2DFA rejects or loops, then $f(q)$ is defined as $\hat{q}$: this represents the *behaviour* of the 2DFA on this prefix.

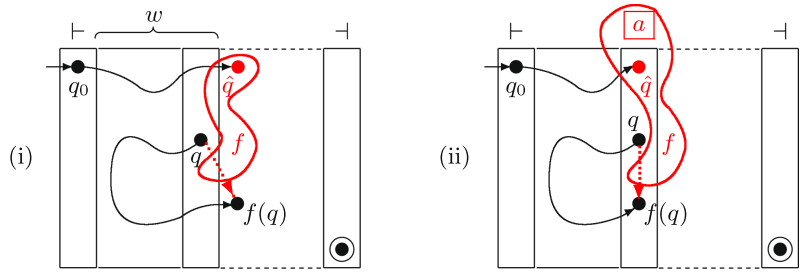

Fig. 1. The data collected by a 1DFA simulating a 2DFA: (i) in the construction by Kapoutsis using states $(\hat{q}, f)$, with $f : Q \to Q$; (ii) in the new construction using states $(a, \hat{q}, f)$, with $f : \overleftarrow{Q}_a \to \overrightarrow{Q}_a$.

These conditions defining a state $(\hat{q}, f)$ are illustrated in Fig. 1(i). The image of f must contain $\hat{q}$, because this state must be reached from some state at the previous position; this accounts for the term $-(n-1)^n$ in the expression for the number of states in the resulting 1DFA for L. The number of such pairs $(\hat{q}, f)$ is exactly $\mathcal{K}(n) = n \cdot (n^n - (n-1)^n)$. □

It is interesting to note that the construction in Theorem 1 yields fewer states on any direction-determinate 2DFA (hence, on any sweeping 2DFA).

Corollary 1. *For every n-state direction-determinate 2DFA with $|Q_{+1}| = k$, there exists a partial 1DFA with k^{n-k+1} states recognizing the same language.*

Proof. The idea is simple: in the automaton in Theorem 1, in a pair $(\hat{q}, f)$, the function f needs to be defined only on arguments in Q_{-1} (its values on Q_{+1} are irrelevant for the construction), and all its values are in Q_{+1} by definition. Furthermore, the state $\hat{q}$ is also in Q_{+1} by definition. Then there are exactly $k^{n-k} \cdot k = k^{n-k+1}$ such pairs $(\hat{q}, f)$, where $k = |Q_{+1}|$. □

The number of states thus depends on the value of k, and the maximum number is $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1}$. It will be shown later in Sect. 6 that the maximum is reached for $k = (1 + o(1)) \cdot \frac{n+1}{\ln(n+1)}$, and that the number $\mathcal{F}(n)$ is of the order $(n+1)^{n+1}/(\ln(n+1) \cdot e^{1-o(1)})^{n+1}$.

This improvement over Theorem 1 is obtained by reducing the domain and the range of functions f. The new transformation for 2DFA of the general form presented in the next section achieves a similar reduction by additionally remembering one input symbol.

4 Efficient Transformation for Small Alphabets

The behaviour function $f : Q \to Q$ used in Theorem 1 maps states at the last symbol read to states in the next position. The new construction is different from the classical one in two respects. First, the 1DFA shall remember a different

behaviour function, which traces *computations beginning and ending at the last symbol read*, and $\hat{q}$ shall also be *positioned on the last symbol read*. Second, the 1DFA additionally remembers *the last symbol read*. Let a be this symbol. Knowing it, the 1DFA also knows the transitions by a, and in particular, in which direction they move the head. Let $\overleftarrow{Q}_a$ be the set of all states q with left-moving transitions $\delta(q,a) = (q', -1)$, and let $\overrightarrow{Q}_a$ consist of all states with right-moving transitions $\delta(q,a) = (q', +1)$. Then, the new behaviour function can map states in $\overleftarrow{Q}_a$ to states in $\overrightarrow{Q}_a$, as illustrated in Fig. 1(ii). And there are fewer such functions than functions $f : Q \to Q$, used in Theorem 1.

Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be a 2DFA. For each symbol $a \in \Sigma$, let $\overleftarrow{Q}_a = \{q : \delta(q,a) \in Q \times \{-1\}\}$ be the set of states in which $\mathcal{A}$ moves to the left on a, and similarly define $\overrightarrow{Q}_a = \{q : \delta(q,a) \in Q \times \{+1\}\}$.

Theorem 2. *For every 2DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, there exists a partial 1DFA $\mathcal{B} = (\Sigma, Q', q_0', \delta', F')$ that recognizes the same language and uses the following set of states.*

$$Q' = \{(a, \hat{q}, f) : a \in \Sigma,\ \hat{q} \in \overrightarrow{Q}_a,\ f : \overleftarrow{Q}_a \to \overrightarrow{Q}_a\} \cup \{(\vdash, q_0, f_0)\},$$

where f_0 is a trivial function with an empty domain, that is, $f_0 : \varnothing \to \varnothing$.

Proof. Each state $(a, \hat{q}, f)$ of $\mathcal{B}$ consists of three components: the last read input symbol $a \in \Sigma$; the state $\hat{q} \in \overrightarrow{Q}_a$, from which $\mathcal{A}$ first leaves the prefix read so far to the right; and the function $f : \overleftarrow{Q}_a \to \overrightarrow{Q}_a$, describing the behaviour of $\mathcal{A}$ on the prefix read so far, which maps states at the last symbol read to states at the last symbol read, as in Fig. 1(ii).

Note that $\overleftarrow{Q}_\vdash = \varnothing$, as $\mathcal{A}$ cannot move beyond the left endmarker. For this reason, there is only one function mapping $\overleftarrow{Q}_\vdash$ to $\overrightarrow{Q}_\vdash$, namely, $f_0 : \varnothing \to \varnothing$, which is a trivial function with an empty domain. This leads to the following initial state of $\mathcal{B}$:

$$q_0' = (\vdash, q_0, f_0)\,.$$

This is the only state of $\mathcal{B}$ with an endmarker in the first component.

The transition in a state $(a, \hat{q}, f)$ by a symbol $b \in \Sigma$ leads to a triple $(b, \hat{r}, g)$, defined as follows. For every state $q \in \overleftarrow{Q}_b$, consider the uniquely defined sequence of states $s_0, \ldots, s_\ell, s_{\ell+1} \in Q$ entered by the automaton at the symbol b, where $s_0 = q$, $\ell \geq 0$, $s_0, \ldots, s_\ell \in \overleftarrow{Q}_b$, $s_{\ell+1} \in \overrightarrow{Q}_b$, and every two consecutive states s_i, s_{i+1} in this sequence are connected in one of the following two ways. Let $\delta(s_i, b) = (t, -1)$. Then,

- either $t \in \overrightarrow{Q}_a$, and then $\delta(t, a) = (s_{i+1}, +1)$,
- or $t \in \overleftarrow{Q}_a$, in which case $\delta(f(t), a) = (s_{i+1}, +1)$.

This exchange between a and b is illustrated in Fig. 2. Define $g(q) := s_{\ell+1}$. However, if the construction of the above sequence reaches an undefined transition, if the sequence is infinite, or if it reaches $\hat{q}$, then let the value $g(q)$ be temporarily undefined—to be specified later.

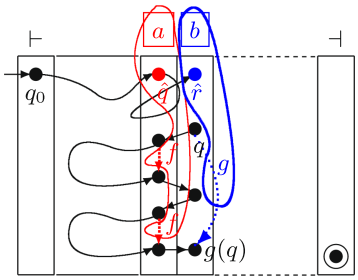

Fig. 2. New simulation of a 2DFA by a 1DFA: transition from a state $(a, \hat{q}, f)$ by a symbol b to a state $(b, \hat{r}, g)$.

The state $\hat{r}$ is set by first considering the transition $\delta(\hat{q}, a) = (q', +1)$: if $q' \in \overrightarrow{Q}_b$, then $\hat{r} := q'$, and if $q' \in \overleftarrow{Q}_b$, then $\hat{r} := g(q')$, provided that the value $g(q')$ has been defined already. With $\hat{r}$ specified, all values of g not defined yet can be set equal to $\hat{r}$.

Finally, let

$$\delta'((a, \hat{q}, f), b) := (b, \hat{r}, g)\,.$$

However, if the above construction has left $\hat{r}$ undefined, then let the whole transition $\delta'((a, \hat{q}, f), b)$ be undefined.

The set F' of accepting states of $\mathcal{B}$ is defined similarly to transitions. A state $(a, \hat{q}, f)$ is marked as accepting, if the following condition holds. Consider the uniquely defined sequence of states $s_0, \ldots, s_\ell, \ldots \in Q$, finite or infinite, where $\delta(\hat{q}, a) = (s_0, +1)$, and every two consecutive states s_i, s_{i+1} in this sequence are connected in one of the following two ways. Let $\delta(s_i, \dashv) = (t, -1)$. Then,

- either $t \in \overrightarrow{Q}_a$, and then $\delta(t, a) = (s_{i+1}, +1)$,
- or $t \in \overleftarrow{Q}_a$, in which case $\delta(f(t), a) = (s_{i+1}, +1)$.

If this sequence eventually reaches an accepting state $s_\ell \in F$ at the right endmarker, then let the state $(a, \hat{q}, f)$ be accepting for $\mathcal{B}$. In all other cases, i.e., if the above sequence reaches an undefined transition or it is infinite, $(a, \hat{q}, f) \notin F'$.

The correctness statement for the construction reads as follows.

Claim 1. Let $(a, \hat{q}, f)$ be the state reached by $\mathcal{B}$ after reading $w \in \Sigma^*$. Then:

- *the last symbol of w is a (if $w = \varepsilon$, then $a = \vdash$);*
- *the automaton $\mathcal{A}$, having begun its computation on the tape $\vdash w$ at the left endmarker in the state q_0, eventually reaches the last symbol of w in the state $\hat{q}$;*
- *for each state $q \in \overleftarrow{Q}_a$, the automaton $\mathcal{A}$, having begun its computation on the tape $\vdash w$ at the last symbol of w in the state q, eventually reaches the last symbol of w in the state $f(q) \in \overrightarrow{Q}_a$; and if the latter computation rejects or loops, then $f(q) = \hat{q}$.*

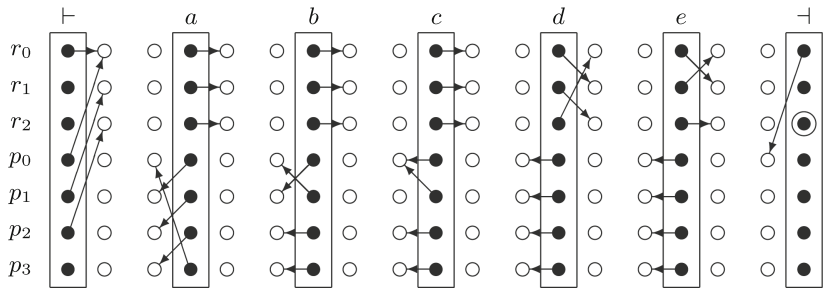

Fig. 3. Transitions of the 2DFA $\mathcal{A}$, defined in Lemma 1, for $k = 3$ and $\ell = 4$.

This can be proved by induction on the length of w; the proof is omitted due to space constraints. □

How many states are there in such automaton? For each symbol a, the construction produces $|\overrightarrow{Q}_a|^{|\overleftarrow{Q}_a|+1}$ states, which is at most $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1}$, and Theorem 2 produces a 1DFA with at most $|\Sigma| \cdot \mathcal{F}(n)$ states.

5 Lower Bound

The first lower bounds on the 2DFA to 1DFA transformation were given by Barnes [1] and by Moore [10], the latter uses a sweeping 2DFA with half of the states right-bound and the other half left-bound, so that the simulating 1DFA has to store an arbitrary function from right-bound states to left-bound states, and therefore must have at least $(\frac{n-5}{2})^{\frac{n-5}{2}}$ states. This paper establishes a stronger lower bound by using an optimal distribution between left-bound and right-bound states. First, the lower bound is presented for a sweeping 2DFA with k right-bound states and ℓ left-bound states.

Lemma 1. *For all $\ell \geq k \geq 3$, there exists a sweeping 2DFA with $k + \ell$ states over the alphabet $\Sigma = \{a, b, c, d, e\}$, such that every 1DFA recognizing the same language must use at least $k^{\ell+1}$ states.*

Proof. Define a 2DFA $\mathcal{A} = (\Sigma, Q, r_0, \delta, F)$ with $\Sigma = \{a, b, c, d, e\}$ and with the set of states $Q = \{r_0, \ldots, r_{k-1}, p_0, \ldots, p_{\ell-1}\}$. In the states $r_0, \ldots, r_{k-1}$, the automaton moves to the right until it reaches the right endmarker, and the states $p_0, \ldots, p_{\ell-1}$ are used for moving to the left without changing the direction.

The automaton begins its computation by the transition (see also Fig. 3)

$$\delta(r_0, \vdash) = (r_0, +1),$$

and then changes its direction of motion only at the endmarkers, using the following transitions:

$$\begin{aligned} &\delta(p_i, \vdash) = (r_i, +1), \quad \text{for } i \in \{0, \ldots, k-1\}, \\ &\delta(r_0, \dashv) = (p_0, +1). \end{aligned}$$

At the right endmarker $\dashv$, the automaton accepts in the state r_{k-1}, that is, $F = \{r_{k-1}\}$. The transitions at the left endmarker $\vdash$ in the states p_k, p_{k+1}, $\ldots$, $p_{\ell-1}$, as well as at the right endmarker $\dashv$ in the states r_1, r_2, $\ldots$, r_{k-2} are undefined, and some inputs are rejected in these configurations. The remaining transitions at the endmarkers are undefined as well, but the automaton shall never get into the corresponding configurations.

The symbols a and b do not affect the computation on the way from left to right, and apply permutations to the states $p_0, \ldots, p_{\ell-1}$ on the way back. Specifically, a implements a circular permutation.

$$\begin{aligned} &\delta(r_i, a) = (r_i, +1), && \text{for } i \in \{0, \ldots, k-1\}, \\ &\delta(p_i, a) = (p_{(i+1) \bmod \ell}, +1), && \text{for } i \in \{0, \ldots, \ell-1\}. \end{aligned}$$

The symbol b swaps p_0 with p_1.

$$\begin{aligned} \delta(r_i, b) &= (r_i, +1), && \text{for } i \in \{0, \ldots, k-1\}, \\ \delta(p_0, b) &= (p_1, -1), \\ \delta(p_1, b) &= (p_0, -1), \\ \delta(p_i, b) &= (p_i, -1), && \text{for } i \in \{2, \ldots, \ell-1\}. \end{aligned}$$

Using a and b, one can generate an arbitrary permutation of the states $p_0, \ldots, p_{\ell-1}$. The next symbol c merges both states p_0 and p_1 into p_0.

$$\begin{aligned} \delta(r_i, c) &= (r_i, +1), && \text{for } i \in \{0, \ldots, k-1\}, \\ \delta(p_0, c) &= (p_0, -1), \\ \delta(p_1, c) &= (p_0, -1), \\ \delta(p_i, c) &= (p_i, -1), && \text{for } i \in \{2, \ldots, \ell-1\}. \end{aligned}$$

This allows to implement all completely defined functions from $\{p_0, \ldots, p_{\ell-1}\}$ to $\{p_0, \ldots, p_{\ell-1}\}$, injective and non-injective.

The last two symbols d and e are used to permute the states $r_0, \ldots, r_{k-1}$, in the same way as a and b permute the second group of states. The symbol d defines a circular permutation, and e exchanges the first two states:

$$\begin{aligned} &\delta(r_i, d) = (r_{(i+1) \bmod k}, +1), && \text{for } i \in \{0, \ldots, k-1\}, \\ &\delta(p_i, d) = (p_i, -1), && \text{for } i \in \{0, \ldots, \ell-1\}, \\ &\delta(r_0, e) = (r_1, +1), \\ &\delta(r_1, e) = (r_0, +1), \\ &\delta(r_i, e) = (r_i, +1), && \text{for } i \in \{2, \ldots, k-1\}, \\ &\delta(p_i, e) = (p_i, -1), && \text{for } i \in \{0, \ldots, \ell-1\}. \end{aligned}$$

Claim 2. For every complete function $g : \{p_0, \ldots, p_{\ell-1}\} \to \{p_0, \ldots, p_{\ell-1}\}$, there exists a string $w_g \in \{a, b, c\}^*$, on which the automaton, having begun its computation at the last symbol of w_g in a state p_i, eventually exits the string to the left of its leftmost symbol in the state $g(p_i)$.

It is well-known that every complete function can be expressed as a composition of three generators: circular permutation g_a, swapping of two elements g_b, and merging of two elements g_c. Let $g = g_{\sigma_1} \circ \ldots \circ g_{\sigma_m}$, where $\sigma_1, \ldots, \sigma_m \in \{a, b, c\}$. Then $w_g := \sigma_1 \ldots \sigma_m$ is the desired string.

Claim 3. For every state $r_{i_0} \in \{r_0, \ldots, r_{k-1}\}$ and for every complete function $f : \{p_0, \ldots, p_{\ell-1}\} \to \{r_0, \ldots, r_{k-1}\}$, there exists a string $u_{r_{i_0},f} \in \{a, b, c, d\}^*$, such that the automaton, operating on the tape $\vdash u_{r_{i_0},f}$,

- *having begun its computation at the left endmarker $\vdash$ in the state r_0, eventually exits the string to the right of its rightmost symbol in the state r_{i_0};*
- *having begun its computation at the last symbol of $u_{r_{i_0},f}$ in a state p_j, eventually reaches the left endmarker $\vdash$, and then moves from there to the right in the state $f(p_j)$.*

Consider the function $g : \{p_0, \ldots, p_{\ell-1}\} \to \{p_0, \ldots, p_{k-1}\}$ defined as follows: for every state p_j, first take $f(p_j) = r_i$. Then define $g(p_j) := p_{(i-i_0) \bmod k}$. Now, let w_g be the string defined for g in Claim 2, and define the string $u_{r_{i_0},f}$ as

$$u_{r_{i_0},f} := w_g d^{i_0}.$$

Then, having started on $\vdash w_g d^{i_0}$ in the initial configuration, the automaton moves all the way to the right: it remains in the state r_0 while reading w_g, and then finishes reading d^{i_0} in the state r_{i_0}, as desired. If the automaton begins its computation at the last symbol of $\vdash w_g d^{i_0}$ in a state p_j, then it first moves all the way to the left, arriving to the left endmarker in the state $g(p_j) = p_{(i-i_0) \bmod k}$. After that, it leaves this endmarker in the state $r_{(i-i_0) \bmod k}$ and starts sweeping back to the right. It remains in $r_{(i-i_0) \bmod k}$ while reading w_g, and then leaves d^{i_0} in the state $r_i = f(p_j)$. This proves Claim 3.

Claim 4. For every permutation $\pi : \{r_0, \ldots, r_{k-1}\} \to \{r_0, \ldots, r_{k-1}\}$, there exists a string $x_\pi \in \{d, e\}^*$, on which the automaton, beginning its computation at the first symbol of x_π in a state r_i, eventually exits the string to the right of its rightmost symbol in the state $\pi(r_i)$.

This is a standard generation of a permutation using two generators.

Claim 5. For every two distinct states $r_i, r_j \in \{r_0, \ldots, r_{k-1}\}$ and for every two complete functions $f, g : \{p_0, \ldots, p_{\ell-1}\} \to \{r_0, \ldots, r_{k-1}\}$ (not necessarily distinct), there exists a string $v \in \{d, e\}^*$, for which the string $u_{r_i,f}v$ is accepted but the string $u_{r_j,g}v$ is rejected.

Define a permutation $\pi : \{r_0, \ldots, r_{k-1}\} \to \{r_0, \ldots, r_{k-1}\}$ by setting $\pi(r_i) = r_{k-1}$ and $\pi(r_j) = r_1$, while the rest of the values can be set arbitrarily. Set $v := x_\pi$, where $x_\pi \in \{d, e\}^*$ is the string defined for π in Claim 4. On the input $u_{r_i,f}x_\pi$, the automaton accepts the string in one left-to-right traversal, entering the state r_i after reading $u_{r_i,f}$, and then reaching the state $\pi(r_i) = r_{k-1}$ upon reading x_π. On the other hand, on the input $u_{r_j,g}x_\pi$, the automaton is in the state r_j after $u_{r_j,g}$, and then it enters the state $\pi(r_j) = r_1$ by x_π, in which the transition by the right endmarker is undefined, and thus the string is rejected.

Claim 6. For every state $r_i \in \{r_0, \ldots, r_{k-1}\}$ and for every two distinct complete functions $f, g : \{p_0, \ldots, p_{\ell-1}\} \to \{r_0, \ldots, r_{k-1}\}$, there exists a string $v \in \{a, d, e\}^*$, for which exactly one of the strings $u_{r_i,f}v$ and $u_{r_i,g}v$ is accepted by the automaton.

Let p_j be any argument on which f and g assume different values. Then at least one of $f(p_j)$ and $g(p_j)$ is different from r_i; assume, without loss of generality, that $f(p_j) = r_t \neq r_i$, and $g(p_j) \neq r_t$. Define a permutation $\pi : \{r_0, \ldots, r_{k-1}\} \to \{r_0, \ldots, r_{k-1}\}$ by $\pi(r_i) = r_0$ and $\pi(r_t) = r_{k-1}$, the rest of the values can be anything. Let $x_\pi \in \{d, e\}^*$ be the string defined for π in Claim 4. Then the promised string is defined by

$$v := x_\pi a^j.$$

The 2DFA accepts the string $u_{r_i,f} x_\pi a^j$ in two stages. In the first left-to-right sweep, it enters the state r_i after $u_{r_i,f}$, then comes to the state $r_0 = \pi(r_i)$ after x_π, and stays in this state until it reaches the right endmarker $\dashv$. Then it makes a right-to-left sweep, first coming to the state p_j after reading a^j, then maintaining this state while reading x_π, so that it enters the prefix $\vdash u_{r_i,f}$ from the right in the state p_j. By Claim 3, the automaton eventually moves from the last symbol of this prefix to the right in the state $f(p_j) = r_t$. The automaton continues by moving to the right through the substring x_π, and finishes reading it in the state $\pi(r_t) = r_{k-1}$. In this state, the automaton passes through a^j and reaches the right endmarker for the second time, accepting this time.

The computation on the string $u_{r_i,g} x_\pi a^j$ begins in the same way. Eventually, the automaton enters the prefix $\vdash u_{r_i,g}$ from the right in the state p_j, and later emerges from this prefix to the right in the state $g(p_j) \neq r_t$. Next, the automaton reads the substring x_π from left to right, and finishes reading it in a state that is *not* r_{k-1}. If the state is r_0, then the automaton loops, and if it is neither r_0 nor r_{k-1}, it rejects at the right endmarker.

Now Lemma 1 is proved as follows. Assume that $\mathcal{B}$ is a 1DFA recognizing the same language as the 2DFA $\mathcal{A}$. Then, the states reached by $\mathcal{B}$ upon reading different strings of the form $u_{r_i,f}$ must be pairwise distinct, for otherwise it would not be able to accept one of them and reject the other upon reading a string constructed in Claims 5 and 6. □

Theorem 3. *For every $n \geq 6$, there exists a language over a 5-symbol alphabet recognized by an n-state sweeping 2DFA, such that every 1DFA recognizing that language needs to have at least $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1}$ states.*

Therefore, the state complexity of transforming a sweeping or a direction-determinate 2DFA to a 1DFA is exactly $\mathcal{F}(n)$, whereas for 2DFA of the general form it is between $\mathcal{F}(n)$ and $|\Sigma| \cdot \mathcal{F}(n)$.

6 Estimation

Both the upper bound and the lower bound on the 2DFA to 1DFA tradeoff have been expressed in terms of the function $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1}$. We are now going to estimate the growth rate of this function.

Lemma 2. *For each $a \geq 16$, the maximum of the real function $\mathcal{G}_a(x) = x^{a-x}$ is reached for $x_a = (1+o(1)) \cdot \frac{a}{\ln a}$, and this maximum accordingly is of order $\mathcal{G}_a(x_a) \leq \frac{a^a}{(\ln a)^a \cdot (e^{1-o(1)})^a}$.*

Proof. First, it is quite easy to derive the following inequality, for $a \geq 3$:

$$a \leq \left(\tfrac{a}{\ln a}\right)^{1+r_1(a)}, \quad \text{where } r_1(a) = \tfrac{\ln\ln a}{\ln a - \ln\ln a}. \tag{1}$$

This follows from $\frac{\ln\ln a}{\ln a - \ln\ln a} \leq r_1(a)$. It should also be easily seen that $r_1(a) > 0$ for $a \geq 3$, and that $\lim_{a\to\infty} r_1(a) = 0$.

Our next task is to find the maximum for $\mathcal{G}_a(x) = x^{a-x} = e^{\ln x\cdot(a-x)}$. By differentiating $\mathcal{G}_a(x)$, we get $\mathcal{G}'_a(x) = e^{\ln x\cdot(a-x)}\cdot(\frac{a-x}{x} - \ln x)$ and, by setting $\mathcal{G}'_a(x_a) = 0$, we see that $\frac{a-x_a}{x_a} - \ln x_a = 0$, which gives

$$x_a \cdot \ln x_a = a - x_a. \tag{2}$$

To evaluate x_a, we first derive the following inequalities, using (2) and (1), under assumption that $a \geq 16$:

$$\begin{aligned}
\tfrac{a}{\ln a}\cdot\ln\tfrac{a}{\ln a} + \tfrac{a}{\ln a} &= \tfrac{a}{\ln a}\cdot(\ln\tfrac{a}{\ln a} + 1) < \tfrac{a}{\ln a}\cdot(\ln\tfrac{a}{\ln a} + \ln\ln a)\\
&= \tfrac{a}{\ln a}\cdot\ln(\tfrac{a}{\ln a}\cdot\ln a) = \tfrac{a}{\ln a}\cdot\ln a = a = x_a\cdot\ln x_a + x_a\,,\\
x_a\cdot\ln x_a &= a - x_a < a = \tfrac{a}{\ln a}\cdot\ln a \leq \tfrac{a}{\ln a}\cdot\ln(\tfrac{a}{\ln a})^{1+r_1(a)}\\
&= (1+r_1(a))\cdot\tfrac{a}{\ln a}\cdot\ln\tfrac{a}{\ln a} \leq (1+r_1(a))\cdot\tfrac{a}{\ln a}\cdot\ln((1+r_1(a))\cdot\tfrac{a}{\ln a})\,.
\end{aligned}$$

Now, since both $t\cdot\ln t + t$ and $t\cdot\ln t$ are real functions monotone increasing in t, we see that $\frac{a}{\ln a} < x_a$ and $x_a < (1+r_1(a))\cdot\frac{a}{\ln a}$. Thus, the exact value x_a can be expressed in the form

$$x_a = (1+r_2(a))\cdot\tfrac{a}{\ln a}, \quad \text{with } 0 < r_2(a) < r_1(a). \tag{3}$$

Since $\lim_{a\to\infty} r_1(a) = 0$ by (1), we see that $\lim_{a\to\infty} r_2(a) = 0$ as well.

It only remains to evaluate $\mathcal{G}_a(x_a)$. Using (2) and (3), we get:

$$\begin{aligned}
\mathcal{G}_a(x_a) = x_a^{a-x_a} &= e^{\ln x_a\cdot(a-x_a)} = e^{a\cdot\ln x_a - x_a\cdot\ln x_a} = e^{a\cdot\ln x_a - a + x_a}\\
&= e^{a\cdot(\ln(1+r_2(a))+\ln a-\ln\ln a)-a+(1+r_2(a))\cdot a/\ln a}\\
&= e^{a\cdot\ln(1+r_2(a))+a\cdot\ln a-a\cdot\ln\ln a-a+a/\ln a+r_2(a)\cdot a/\ln a}\\
&= e^{a\cdot\ln a-a\cdot\ln\ln a-a}\ \cdot\ e^{a\cdot(\ln(1+r_2(a))+1/\ln a+r_2(a)/\ln a)}\\
&= \tfrac{a^a}{(\ln a)^a\cdot e^a}\cdot e^{a\cdot r_3(a)} = \tfrac{a^a}{(\ln a)^a\cdot(e^{1-r_3(a)})^a}, \quad \text{where}\\
r_3(a) &= \ln(1+r_2(a)) + \tfrac{1}{\ln a} + \tfrac{r_2(a)}{\ln a}\,.
\end{aligned}$$

Since $r_2(a) > 0$ and $\lim_{a\to\infty} r_2(a) = 0$, also $r_3(a) > 0$ for $a > 1$ and, moreover, $\lim_{a\to\infty} r_3(a) = 0$. □

Using this lemma, the following estimation of $\mathcal{F}(n)$ can be obtained:

Theorem 4. $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1} \leq \frac{(n+1)^{n+1}}{(\ln(n+1))^{n+1}\cdot(e^{1-o(1)})^{n+1}}$.

Proof. $\mathcal{F}(n) = \max_{k=0}^{n} k^{n-k+1} = \max_{k=0}^{n} k^{a-k} \leq \mathcal{G}_a(x_a) = \frac{a^a}{(\ln a)^a\cdot(e^{1-o(1)})^a}$ $= \frac{(n+1)^{n+1}}{(\ln(n+1))^{n+1}\cdot(e^{1-o(1)})^{n+1}}$, using substitution $a = n+1$. □

We are now ready to compare $\mathcal{F}(n)$ with $\mathcal{K}(n) = n\cdot(n^n - (n-1)^n)$, the standard tradeoff for 2DFA to 1DFA transformation, derived by Kapoutsis [5]:

Theorem 5. $\sup_{n\to\infty} \frac{\mathcal{F}(n)}{\mathcal{K}(n)} \cdot \frac{e-1}{e^2} \cdot (\ln(n+1) \cdot e^{1-o(1)})^{n+1} \leq 1$.

This is based on the fact that $\lim_{n\to\infty} \frac{(n+1)^{n+1}}{n\cdot(n^n-(n-1)^n)} = \frac{e^2}{e-1}$ (omitted due to space constraints). Accordingly, the proposed 2DFA to 1DFA transformation given in Theorem 2 improves over the construction by Kapoutsis [5] as long as $|\Sigma| < \frac{e-1}{e^2} \cdot (\ln(n+1) \cdot e^{1-o(1)})^{n+1}$.

References

1. Barnes, B.: A two-way automaton with fewer states than any equivalent one-way automaton. IEEE Trans. Comput. **C-20**, 474–475 (1971)
2. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoret. Comput. Sci. **295**, 189–203 (2003)
3. Geffert, V., Okhotin, A.: One-way simulation of two-way finite automata over small alphabets. In: Proceedings of Non-Classical Models of Automata & Application, pp. 151–162. Österreichische Comput. Gesellschaft (2013)
4. Geffert, V., Okhotin, A.: Transforming two-way alternating finite automata to one-way nondeterministic automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014. LNCS, vol. 8634, pp. 291–302. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44522-8_25
5. 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
6. Kapoutsis, C.: Algorithms and lower bounds in finite automata size complexity. Ph.D. thesis, Massachusetts Institute of Technology (2006)
7. Kunc, M., Okhotin, A.: Describing periodicity in two-way deterministic finite automata using transformation semigroups. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 324–336. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22321-1_28
8. Kunc, M., Okhotin, A.: Reversibility of computations in graph-walking automata. Inform. Comput. **275** (2020). Art. 104631
9. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. **30**, 1976–1992 (2001)
10. Moore, F.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata by deterministic automata. IEEE Trans. Comput. **C-20**, 1211–1214 (1971)
11. Petrov, S., Okhotin, A.: On the transformation of two-way deterministic finite automata to unambiguous finite automata. In: Leporati, A., Martín-Vide, C., Shapira, D., Zandron, C. (eds.) LATA 2021. LNCS, vol. 12638, pp. 81–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68195-1_7
12. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. **3**, 114–125 (1959)
13. Shepherdson, J.: The reduction of two-way automata to one-way automata. IBM J. Res. Develop. **3**, 198–200 (1959)
14. Sipser, M.: Lower bounds on the size of sweeping automata. In: Proceedings of the ACM Symposium on Theory of Computing, pp. 360–364 (1979)
15. Vardi, M.: A note on the reduction of two-way automata to one-way automata. Inform. Process. Lett. **30**, 261–264 (1989)