On the Size Complexity of Two-Way Finite Automata with Drop-Once Pebbles

Georgy Kipriyanov
Department of Mathematics and Computer Science, Saint Petersburg State University, Russia

Alexander Okhotin
Department of Mathematics and Computer Science, Saint Petersburg State University, Russia

Abstract

A two-way finite automaton with drop-once pebbles (O. Martynova, A. Okhotin, "A time to cast away stones: On a family of pebble automata", IJFCS, 37 (2026)) may drop its pebbles at any squares of the tape, but a pebble once dropped cannot be moved anymore. In this paper, it is proved that transforming an n-state deterministic automaton with k drop-once pebbles to a standard two-way deterministic finite automaton (2DFA) requires $\Theta(n^{k+1})$ states in the worst case. For nondeterministic two-way automata with one drop-once pebble, it is proved that transforming them to a 2DFA requires at least $2^{\frac{n}{3}-o(n)}$ states, transforming to a two-way nondeterministic automaton (2NFA) takes at least $2^{\frac{n}{6}-o(n)}$ states, and, finally, determinizing them to a deterministic two-way automaton with one drop-once pebble requires at least $2^{\frac{n}{6}-o(n)}$ states.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Models of computation

Keywords and phrases Finite automata, two-way automata, pebble automata, determinization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.68

Funding This work was supported by the Russian Science Foundation, project 23-11-00133.

1 Introduction

Two-way finite automata are a classical model of computation, which continues to attract general attention, in particular, due to its connections to unsolved problems in the complexity theory. The most important and long-standing open question in the area, raised by Sakoda and Sipser [17], is whether two-way nondeterministic finite automata (2NFA) can be determinized to two-way deterministic automata (2DFA) with only polynomial increase in the number of states. This question has defied all efforts so far: the only known lower bound on the complexity of this transformation is $\Omega(n^2)$, where n is the number of states in the 2NFA, whereas the best guaranteed transformation uses almost 2^{n^2} states [9]. The importance of this problem is supported by its connection to the *L vs. NL* question in the complexity theory discovered by Berman and Lingas [1], and by the more recent result on the connection to the *NL vs. L/poly* question by Kapoutsis [11]. However, for several variants of two-way automata, lower bounds on their determinization complexity have been obtained: Sipser [19] proved that determinizing sweeping automata requires exponentially many states, whereas Kapoutsis [10] established an exponential lower bound for two-way automata with few reversals.

This paper investigates the size complexity and specifically the complexity of determinization for *two-way automata with drop-once pebbles*, recently investigated by Martynova and Okhotin [15] in the more general context of graph-walking automata. Restricted to strings, these are 2NFA or 2DFA with a fixed number of pebbles, which, once dropped somewhere on the tape, can be detected at the later visits to the same square, but cannot be lifted anymore.

Automata with drop-once pebbles are a special case of pebble automata of the general form, which received much attention in the literature. Pebble automata were introduced by Blum and Hewitt [2]: when such an automaton sees any pebble it has earlier dropped, it may lift it and carry it to another square. In particular, Blum and Hewitt [2] proved that one-pebble 2DFA recognize only regular languages—a result later extended to one-pebble alternating automata by Goralčík et al. [8]. Later, Globerman and Harel [7] introduced a special case of pebble automata with *nested pebbles*: in these automata, pebbles are numbered, and at every moment, pebbles $1, \ldots, i$ are placed somewhere, while pebbles $i+1, \ldots, m$ are in the automaton's hands, so that the automaton is allowed either to lift pebble i or to drop pebble $i+1$; these automata recognize only regular languages [7, 4]. The research on pebble automata progressed from automata on strings to *tree-walking automata with nested pebbles*, introduced by Engelfriet and Hoogeboom [4]: their computational complexity was studied by Samuelides and Segoufin [18], Bojańczyk et al. [3] investigated their expressive power and established hierarchies on the number of heads, and some further results on the one-pebble case were recently obtained by Martynova and Okhotin [14]. On the subject of determinization complexity, Geffert and Ištoňová [5] proved that any lower bound on the complexity of determinizing one-pebble 2NFA would imply a close lower bound on the complexity of determinizing standard 2NFA, making this another possible approach to the open problem of Sakoda and Sipser [17].

Turning to two-way automata with k drop-once pebbles and their size complexity, these automata can be regarded as a special case of two-way automata with k nested pebbles studied by Globerman and Harel [7], who proved that their transformation to two-way automata without pebbles requires k-fold exponentially many states. For automata with drop-once pebbles, Martynova and Okhotin [15] proved that every n-state deterministic two-way finite automaton with k drop-once pebbles ($2\mathrm{DFA}_{k\downarrow}$) can be transformed to a standard 2DFA with only $O(n^{k+1})$ states—a substantial improvement from the case of nested pebbles. A matching lower bound of $\Omega(n^{k+1})$ states on the complexity of this transformation will be proved in this paper, and the methods developed will ultimately lead to a lower bound on the determinization complexity of nondeterministic two-way automata with one drop-once pebble ($2\mathrm{NFA}_{1\downarrow}$).

Once a definition of automata with drop-once pebbles is given in Section 2, the next Section 3 presents the simple case of the lower bound: it is proved that transforming a 2DFA with one drop-once pebble ($2\mathrm{DFA}_{1\downarrow}$) to a standard 2DFA requires $\Theta(n^2)$ states in the worst case. In general, all known direct lower bound methods for two-way automata apply only to their special cases: consider Sipser's [19] arguments for sweeping automata, the results on 2DFA and 2NFA over a unary alphabet by Mereghetti and Pighizzini [16], Geffert et al. [6] and Kunc and Okhotin [12], and the recent work of Clerici Lorenzini et al. [13] on bounded languages. However, there is a known indirect approach: to prove lower bounds on the size of one-way DFA, and then combine them with the well-known complexity of the 2DFA to DFA transformation to obtain a lower bound on the size of a 2DFA; Globerman and Harel [7], among others, used this method, and it has turned out to be particularly effective in the present paper. The argument in Section 3 proceeds by presenting a language L_m recognized by a $(2m+1)$-state $2\mathrm{DFA}_{1\downarrow}$, and showing that every DFA for this language requires at least $(m!)^m$ states, which implies a lower bound of $\frac{1}{9}n^2$ states on the complexity of transforming an n-state $2\mathrm{DFA}_{1\downarrow}$ to a 2DFA.

This lower bound is generalized to the case of k pebbles in Section 4. There, a language $L_{m,k}$ recognized by a $(2m+1)$-state $2\mathrm{DFA}_{k\downarrow}$ is presented, for which every DFA must have at least $(m!)^{m^k}$ states. This implies that $\Omega(n^{k+1})$ states are necessary in a 2DFA to simulate an n-state $2\mathrm{DFA}_{k\downarrow}$.

The final results of the paper, presented in Section 5, The proof uses a language K_m recognized by a $2\text{NFA}_{1\downarrow}$ with approximately $3m \log_2 m$ states, such that every DFA for the same language must have at least $(m!)^{m!}$ states. This leads to lower bounds on the complexity of several transformations, the most interesting of them being that determinization of n-state $2\text{NFA}_{1\downarrow}$ requires at least $2^{\frac{n}{6} - \frac{n \log_2 \log_2 n}{2 \log_2 n}}$ states in every $2\text{DFA}_{1\downarrow}$.

2 Drop-once pebble automata

This paper is concerned with an extension of the classical two-way finite automata equipped with a fixed number drop-once pebbles, that is, with pebbles that can be put at some square of the tape only once, and, once placed, cannot be moved anymore.

The automaton operates on an input tape containing the input string delimited by a left end-marker ($\vdash$) and a right end-marker ($\dashv$). It has finitely many states and is equipped with a reading head, which scans one of the symbols of the tape at every moment. It also has $k \geqslant 0$ distinguishable pebbles; at each moment of the computation, every pebble is either in the automaton's hands or placed at one of the squares of the tape. The automaton begins its computation in its initial state, with the head at the left end-marker, carrying all k pebbles in its hands. At every step the automaton sees the symbol in the current tape square and all pebbles placed at that square. Using the current state and this information, the automaton decides what to do according to its transition function, which instructs it to place a subset of pebbles at the current square, move the head to the left or to the right, and enter a new state. In the deterministic variant, the action is completely determined by the state and the contents of the square, and in the nondeterministic case, the transition function defines the set of possible actions.

▶ **Definition 1** (Martynova and Okhotin [15]). *A two-way nondeterministic finite automaton with k drop-once pebbles ($2NFA_{k\downarrow}$) is a sextuple $\mathcal{A} = (\Sigma, Q, Q_0, P, \delta, F)$, where:*

- *Σ is a finite alphabet, with $\vdash, \dashv \notin \Sigma$;*
- *Q is a finite set of states;*
- *$Q_0 \subseteq Q$ is the set of initial states;*
- *P is a finite set of pebbles, with $|P| = k$;*
- *$\delta \colon Q \times 2^P \times \Sigma \times 2^P \to 2^{Q \times \{-1,+1\} \times 2^P}$ is a transition function: if $\delta(q, S, a, T) \ni (q', d, T')$, this means that if the automaton is in the state $q \in Q$, has pebbles $S \subseteq P$ in its hands, and observes the symbol $a \in \Sigma \cup \{\vdash, \dashv\}$ and pebbles $T \subseteq P$ in the current square, then it may leave the pebbles from T' in the current square and move in the direction $d \in \{-1, +1\}$ in the state q';*
- *$F \subseteq Q$ is the set of accepting states, effective at the right end-marker ($\dashv$).*

The transition function and the set of accepting states must satisfy the following conditions: $\delta(q, S, a, T) \ni (q', d, T')$ implies $T \subseteq T' \subseteq S \cup T$, that is, pebbles that were already lying at this square will remain there, and any subset of pebbles in hand can be dropped now; if $a = \vdash$, then $d = +1$, and if $a = \dashv$, then $q \notin F$, $d = -1$, meaning that the automaton cannot move beyond either end-marker, and stops in an accepting configuration.

For an input string $w = a_1 \dots a_\ell \in \Sigma^*$, let $a_0 = \vdash$ and $a_{\ell+1} = \dashv$. A configuration of $\mathcal{A}$ on w is any pair (q, i, f), where $q \in Q$ is the current state, $i \in \{0, \dots, \ell + 1\}$ is the position of the head, and $f \colon P \to \{0, \dots, \ell + 1\}$ is a partial function that defines the position of each pebble, and is undefined for pebbles in hand. A computation of the automaton on w is any sequence of configurations (r_0, i_0, f_0), …, (r_j, i_j, f_j), …, possibly infinite, where

- *the initial configuration (r_0, i_0, f_0) satisfies $r_0 \in Q_0$, $i_0 = 0$ and f_0 is undefined on all arguments;*

- *every next configuration* (r_j, i_j, f_j)*, if it exists, satisfies* $(r_j, d_j, T') \in \delta(r_{j-1}, \{\, p \mid f_{j-1}(p)$ *is undefined*$\}, a_{i_{j-1}}, \{\, p \mid f_{j-1}(p) = i_{j-1} \})$*,* $i_j = i_{j-1} + d_j$*,* $f_j(p) = i_{j-1}$ *for all* $p \in T'$ *and* $f_j(p) = f_{j-1}(p)$ *for all* $p \notin T'$*.*

The string w *is accepted if at least one computation on it eventually arrives at a configuration* $(r_{acc}, \ell + 1, f)$*, with* $r_{acc} \in F$*.*

An automaton is called deterministic (2DFA$_{k\downarrow}$*) if* $|Q_0| = 1$ *and* $|\delta(r, S, a, T)| \leqslant 1$ *for all* $r \in Q$*,* $a \in \Sigma \cup \{\vdash, \dashv\}$ *and* $S, T \subseteq P$*.*

An automaton is called a two-way finite automaton (2NFA, 2DFA) if $P = \varnothing$*.*

Since 2NFA with any number of nested pebbles are known to recognize only regular languages [7], the same applies to 2NFA with drop-once pebbles: indeed, the freedom of choosing the order of dropping the pebbles cannot increase the expressive power of the model, because any desired order can be stored in the current state. However, the transformation of n-state automata with k nested pebbles to 2DFA requires k-fold exponentially many states in n [7]. For 2NFA$_{k\downarrow}$, no better transformation is known. But in the deterministic case, automata with drop-once pebbles can be transformed to ordinary 2DFA very efficiently.

▶ **Theorem A** (Martynova and Okhotin [15])**.** *For every* n*-state 2DFA with one drop-once pebble (2DFA*$_{1\downarrow}$*), there exists a 2DFA with* $5n^2 + 2n$ *states recognizing the same language.*

For every n*-state 2DFA with* k *drop-once pebbles (2DFA*$_{k\downarrow}$*), the same language is recognized by a 2DFA with* $5^k \cdot k! \cdot 3^{\frac{k(k+1)}{2}} \cdot n^{k+1} = \mathrm{const} \cdot n^{k+1}$ *states, where the constant depends on* k*.*

3 Lower bound on the transformation of 2DFA with one drop-once pebble to 2DFA

The first result of this paper is that the transformation of n-state 2DFA$_{1\downarrow}$ to 2DFA with $5n^2 + 2n$ states by Martynova and Okhotin [15] is actually optimal up to a constant factor. In this section it will be proved that for some 2DFA$_{1\downarrow}$ the transformation requires $\Omega(n^2)$ states in a 2DFA.

This proof, as well as all the subsequent proofs in this paper, uses the permutation group S_m, which is assumed to contain elements $\{0, \ldots, m-1\}$. Let f_1 and f_2 be the generators of the S_m, namely, f_1 is a grand cycle $(0 \to 1 \to \ldots \to m-1 \to 0)$, whereas f_2 exchanges 0 with 1. Note that throughout the paper f_1 and f_2 are used both as permutations and as symbols of various alphabets: wherever they are concatenated, e.g., as $f_1 f_2 f_1$, this always means a string of symbols, and composition or evaluation of permutations is always written explicitly, as in $f_1 \circ f_2$ or $f_1(i)$.

The automata in this section are defined over a fixed alphabet $\Sigma = \{f_1, f_2, c, a, b\}$, where the symbols f_1 and f_2 are the generators of S_m. For every string $v \in \Sigma^*$, the following notation for the composition of all permutations listed in v is introduced. Consider all occurrences of f_1 and f_2 in v, that is, $v = v_0 f_{i_1} v_1 \ldots f_{i_\ell} v_\ell$, where $\ell \geqslant 0$, $v_0, \ldots, v_\ell \in \{c, a, b\}^*$ and $i_1, \ldots, i_\ell \in \{1, 2\}$. Then, the composition $f_{i_\ell} \circ \ldots \circ f_{i_1}$, of all permutations in v, with the first symbol applied first, is denoted by $f_v \in S_m$. For example, $f_{cf_2ccf_2aaabb}$ is the identity permutation, because $f_2 \circ f_2 = \mathrm{id}$.

For each $m \geqslant 2$, the witness language $L_m \subseteq \Sigma^*$ is defined as follows (where $|v|_c$ denotes the number of occurrences of c in v).

$$L_m = \{\, xcva^i b^j \mid x, v \in \{f_1, f_2, c\}^*,\ i, j \in \{0, \ldots, m-1\},\ |v|_c = j,\ f_v(i) = 0 \,\}$$

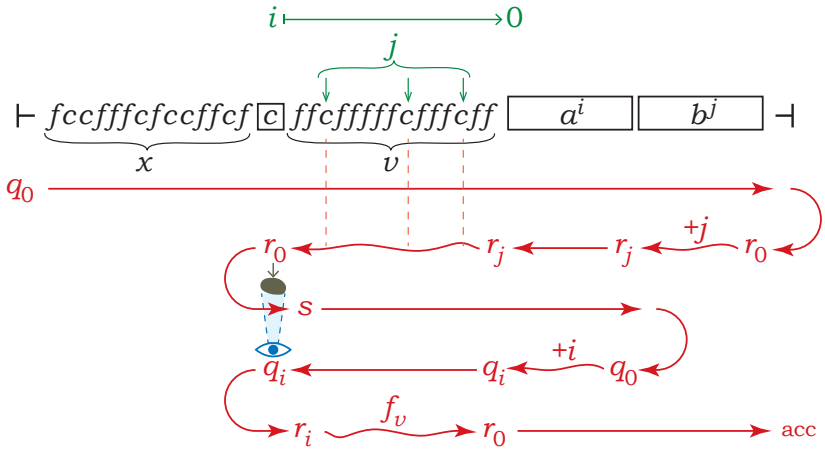

Figure 1 A $(2m+1)$-state $2\text{DFA}_{1\downarrow}$ recognizing the language L_m.

Here the number j defines the position of a particular instance of c in the string. Then, starting from this c, it is sufficient to test the value of the permutation on the argument i. A small deterministic automaton with one drop-once pebble recognizes this language by dropping its pebble at that particular c.

▶ **Lemma 2.** *For every $m \geqslant 3$, there exists a $2DFA_{1\downarrow}$ with $2m+1$ states that recognizes the language L_m.*

Proof. The automaton is equipped with a single pebble $P = \{p\}$. Define the set of states as $Q = \{q_0, \ldots, q_{m-1}, r_0, \ldots, r_{m-1}, s\}$, with $|Q| = 2m+1$. The initial state is q_0.

The computation of the automaton is illustrated in Figure 1. First, it makes a left-to-right sweep to verify that $w \in \{f_1, f_2, c\}^* a^* b^*$. This uses the states q_0, q_1 and q_2, with all transitions defined for configurations with a pebble in hand. Next, the automaton moves from right to left and counts the number of bs. Once it finishes reading the bs in a state r_j, it continues to the left, decrementing the index of the state until it locates the j-th symbol c from the right. Having found this symbol, the automaton drops the pebble upon it and starts moving to the right in a new state s.

The automaton moves to the right in the state s until it reaches the leftmost b (or the right end-marker, if the number of bs is zero), where it turns around to move left in the state q_0. Note that the earlier segment of the computation involving the states q_0, q_1 and q_2 used transitions with the pebble in hand; at the present stage, the pebble has already been dropped, and completely new transitions in the same states can be defined.

Now the automaton moves from right to left and counts the number of as. It finishes reading the block of as in a state q_i, and then continues leftwards until it encounters the pebble, where it turns around in the state r_i, again reusing these states, now for the case of the pebble already dropped. Having entered the state q_i at this point, the automaton sequentially evaluates $f_v(i)$ while moving from left to right, ignoring all symbols except f_1 and f_2. Upon reaching the right end-marker $\dashv$, it accepts if $f_v(i) = 0$ and rejects otherwise. ◀

It is claimed that every 2DFA without pebbles recognizing L_m must have at least $\Omega(m^2)$ states. a quadratic number of states in m. The idea of the proof is to obtain a very large lower bound on the size of every one-way DFA recognizing this language, and then use it to infer the desired lower bound on the size of a 2DFA.

▶ **Lemma 3.** *Every DFA recognizing L_m has at least $(m!)^m$ states.*

Proof. Since f_1 and f_2 are interpreted as generators of the symmetric group S_m, for every permutation $\pi \in S_m$ there exists a string v with $\pi = f_v$. For each $\pi \in S_m$, let u_π be any such string.

For example, if $m = 3$, then there are six permutations, and one can fix the strings implementing them as $u_{012} = \varepsilon$, $u_{021} = f_1 f_2$, $u_{102} = f_2$, $u_{120} = f_1$, $u_{201} = f_1 f_1$, $u_{210} = f_2 f_1$.

Next, consider the following set of $(m!)^m$ strings.

$$\left\{ cu_{\pi_{m-1}} cu_{\pi_{m-2}} \dots cu_{\pi_0} \;\middle|\; \pi_{m-1}, \pi_{m-2}, \dots, \pi_0 \in S_m \right\}$$

The proof is by showing that every DFA recognizing L_m must finish reading the strings from this set in pairwise distinct states. This is done by constructing, for every two strings $w = cu_{\pi_{m-1}} cu_{\pi_{m-2}} \dots cu_{\pi_0}$ and $w' = cu_{\pi'_{m-1}} cu_{\pi'_{m-2}} \dots cu_{\pi'_0}$ from the above set, a continuation w_0, such that one of the strings ww_0 and $w'w_0$ is in L_m, and the other is not.

Since $w \neq w'$, there exists an index j with $\pi_j \neq \pi'_j$; let j be the least among all such indices. Then, $\pi_0 \circ \dots \circ \pi_{j-1} = \pi'_0 \circ \dots \circ \pi'_{j-1}$ and

$$\Pi = \pi_0 \circ \dots \circ \pi_j \neq \pi'_0 \circ \dots \circ \pi'_j = \Pi'.$$

Because $\Pi \neq \Pi'$, there exists an element i with $\Pi(i) \neq \Pi'(i)$. Let $\rho \in S_m$ be any permutation satisfying $\rho(\Pi(i)) = 0$. Then, define $w_0 = u_\rho a^i b^j$. It is claimed that w_0 is the desired separating string satisfying $ww_0 \in L_m$ and $w'w_0 \notin L_m$.

To see how this works, consider the example for $m = 3$, and the following two distinct strings from the above set: $w = cu_{102} cu_{012} cu_{120} = cf_2 ccf_1$ and $w' = cu_{120} cu_{021} cu_{120} = cf_1 cf_1 f_2 cf_1$. Then, $j = 1$, because $120 = 120$ and $012 \neq 021$. The compositions of the last $j + 1$ permutations are $\Pi = 120 \circ 012 = 120$ for w, and $\Pi' = 120 \circ 021 = 102$ for w'. These compositions differ on the argument $i = 1$, as $\Pi(1) = 2 \neq 0 = \Pi'(1)$. Hence, ρ can be any permutation satisfying $\rho(2) = 0$, and one can take $\rho = f_1$. This yields a separating string $w_0 = u_\rho a^i b^j = f_1 ab$. To see that the concatenation $ww_0 = cf_2 ccf_1 f_1 ab$ is in L_3, consider that a specifies an argument 1, while b points to the suffix $ccf_1 f_1 ab$, and this suffix represents a permutation $f_1 \circ f_1$ satisfying $(f_1 \circ f_1)(1) = 0$, hence the entire string is in L_3. The other concatenation $w'w_0 = cf_1 cf_1 f_2 cf_1 f_1 ab$ has b point to the suffix $cf_1 f_2 cf_1 f_1 ab$, in which the permutation satisfies $(f_1 \circ f_1 \circ f_2 \circ f_1)(1) = 1$, and therefore $w'w_0$ is not in L_3.

Resuming the proof of the lemma, each of the two strings ww_0 and $w'w_0$ has a unique suffix of the form cx, where $x \in \Sigma^*$ and $|x|_c = j$. For ww_0, this suffix is $cu_{\pi_j} cu_{\pi_{j-1}} \dots cu_{\pi_0} u_\rho a^i b^j$, and since $(\rho \circ \Pi)(i) = 0$, this string is in L_m. For the other string $w'w_0$, the permutation in the similarly defined suffix satisfies $(\rho \circ \Pi')(i) \neq 0$, and therefore $w'w_0 \notin L_m$. ◀

This lower bound on the size of a DFA implies a lower bound on the size of a 2DFA by the means of the following known precise tradeoff between these two models.

▶ **Theorem B** (Kapoutsis [9])**.** *For every 2DFA with n states, there exists a DFA with $n(n^n - (n-1)^n) + 1$ states recognizing the same language.*

▶ **Lemma 4.** *Let α be a constant factor with $0 < \alpha < \frac{1}{2}$. Then, for each $m > e^{\frac{2}{1-2\alpha}}$, every 2DFA recognizing L_m must have at least αm^2 states.*

Sketch of a proof. Assume that for some $m > e^{\frac{2}{1-2\alpha}}$ there exists a 2DFA with $n < \alpha m^2$ states that recognizes L_m. Then, by Theorem B, there exists a DFA recognizing L_m with at most $n(n^n - (n-1)^n) + 1$ states. Consequently, by Lemma 3,

$$(m!)^m \leqslant n(n^n - (n-1)^n) + 1 \leqslant n^{n+1} < \left(\alpha m^2\right)^{\alpha m^2 + 1},$$

and, after some calculations, this inequality leads to a contradiction. ◀

Now a desired lower bound on the $2DFA_{1\downarrow}$ to 2DFA state complexity tradeoff is inferred from Lemmata 2 and 4 as follows.

▶ **Theorem 5.** *For every constant factor $\beta < \frac{1}{8}$, the number of states in a 2DFA necessary and sufficient to recognize every language defined by an n-state deterministic two-way automaton with one drop-once pebble ($2DFA_{1\downarrow}$) is at least βn^2, for all $n > 3e^{\frac{4}{1-8\beta}}$.*

Combined with the known upper bound from Theorem A, this leads to the following bounds on the number of states, which are precise up to a constant factor.

▶ **Corollary 6.** *The number of states in a 2DFA necessary and sufficient to recognize every language defined by an n-state deterministic two-way automaton with one drop-once pebble ($2DFA_{1\downarrow}$) is $\varphi_1(n) = \Theta(n^2)$. More precisely, it is contained within the bounds $\frac{1}{9}n^2 \leqslant \varphi_1(n) \leqslant 5n^2 + 2n$, with the lower bound established for all $n \geqslant 3e^{36} \approx 1.3 \cdot 10^{16}$.*

4 Lower bound on the transformation of 2DFA with k drop-once pebbles to 2DFA

This section establishes a lower bound on the complexity of tranforming n-state $2DFA_{k\downarrow}$ to a 2DFA, which coincides with the upper bound by Martynova and Okhotin [15] up to a constant factor. The proof applies to all $k \geqslant 1$, and hence this result is more general than the result in Theorem 5.

By Theorem A, every n-state 2DFA with k drop-once pebbles can be transformed to a 2DFA with $5^k k! 3^{\frac{k(k+1)}{2}} n^{k+1} = O(n^{k+1})$ states recognizing the same language. The goal is to prove that $\Omega(n^{k+1})$ states are needed in the worst case.

The witness language $L_{m,k}$ generalizes the language L_m from Section 3. It is defined over a larger alphabet: instead of the symbols b and c, now there are k different symbols b_i corresponding to different pebbles, and accordingly k different symbols c_i.

$$\Sigma_k = \{f_1, f_2, a\} \cup C_k \cup B_k, \qquad \text{where } C_k = \{c_0, \ldots, c_{k-1}\} \text{ and } B_k = \{b_0, \ldots, b_{k-1}\}$$

Each string in the language $L_m = L_{m,0}$ ended with a suffix b^j, which was used to locate a particular instance of c. In the new language, the suffix is $b_0^{j_0} \ldots b_{k-1}^{j_{k-1}}$, and it represents a way to locate particular instances of symbols $c_0, \ldots, c_{k-1}$ in the string. In the end, c_0 becomes the starting point of testing one value of a permutation, as defined below.

$$\begin{aligned} L_{m,k} = \{ xc_0v_0 \ldots c_{k-1}v_{k-1}a^i b_0^{j_0} \ldots b_{k-1}^{j_{k-1}} \mid\ & 0 \leqslant j_t \leqslant m-1 \text{ for all } t,\ 0 \leqslant i \leqslant m-1, \\ & x \in \{f_1, f_2, c_0, \ldots, c_{k-1}\}^*, \\ & v_t \in \{f_1, f_2, c_0, \ldots, c_t\}^* \text{ and } |v_t|_{c_t} = j_t \text{ for all } t, \\ & f_{v_0 \ldots v_{k-1}}(i) = 0 \} \end{aligned}$$

Here a permutation $f_v \in S_m$ is defined for $v \in \Sigma_k^*$ in the same way as for L_m, and $|v_t|_{c_t}$ denotes the number of occurrences of c_t in v_t.

This new language can be efficiently recognized by placing the pebbles $p_{k-1}, \ldots, p_0$ at the corresponding symbols $c_{k-1}, \ldots, c_0$, one by one, and then checking the value of $f_{v_0 \ldots v_{k-1}}$ on i starting from pebble p_0 at c_0.

▶ **Lemma 7.** *For every $m \geqslant k+1$ there exists a $2DFA_{k\downarrow}$ with $2m+1$ states that recognizes the language $L_{m,k}$.*

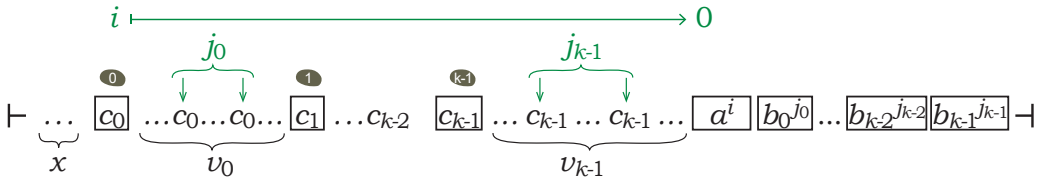

Figure 2 How a 2DFA$_{k\downarrow}$ in Lemma 7 recognizes the language $L_{m,k}$.

Proof. Let the set of pebbles be $P = \{p_0, \ldots, p_{k-1}\}$. The conditions to be checked by the automaton are illustrated in Figure 2, along with the desired final positions of its pebbles.

Note that the location of c_{k-1} in the definition of $L_{m,k}$ is specified exactly as the location of c in L_m. The new automaton begins by reading the number j_{k-1} from the suffix $b^{j_{k-1}}$ and then moving through v_{k-1} while counting the occurrences of c_{k-1}, until it locates the desired occurrence and drops p_{k-1} there, in the same way as in Lemma 2.

Then, for every next ℓ-th pebble, with $0 \leqslant \ell \leqslant k-2$, the automaton repeats a similar procedure: first, it reads j_ℓ from b^{j_ℓ}, and then moves to pebble $p_{\ell+1}$ straight away. Starting from the $(\ell+1)$-th pebble, it moves through v_ℓ while counting c_ℓ until it finds the right c_ℓ to drop the ℓ-th pebble at.

Once pebble p_0 is in place, the automaton retrieves the value of i from a^i, gets back to pebble p_0 and, starting from it, verifies that the value of the permutation of i is 0.

It remains to see that it is sufficient to use $2m+1$ distinct states. As in Lemma 2, the automaton uses states q_i and r_i, with $i \in \{0, \ldots, m-1\}$, and an extra state s. The states q_i are used for checking the general form of the string, for locating the position of c_ℓ after reaching $c_{\ell+1}$, and also for reading i from a^i once all pebbles are in place; these cases differ by the number of pebbles in hand, and hence the automaton may have different behaviour defined. The state s is used for the moving to the right after having placed each pebble. The states r_i are used for reading each j_ℓ from b_ℓ, and also for computing $f_{v_0 \ldots v_{k-1}}(i)$. ◀

▶ **Lemma 8.** *Every DFA recognizing $L_{m,k}$ must have at least $(m!)^{m^k}$ states.*

Proof. As in Lemma 3, for every permutation $\pi \in S_m$, let $u_\pi \in \{f_1, f_2\}^+$ be a string with $f_{u_\pi} = \pi$.

The $(m!)^{m^k}$ strings needed to prove the lemma correspond to different functions $\theta\colon \{0, \ldots, m-1\}^k \to S_m$. For every such function, the corresponding string w_θ should encode the given m^k permutations separated by symbols from $\{c_0, \ldots, c_{k-1}\}$, so that each permutation could be found by counting these symbols from the right, as in the definition of $L_{m,k}$. Such a function can be regarded as an m-ary tree of depth k, with permutations in its leaves, and with the children of each node numbered from right to left, as in Figure 3. The sequence of child numbers from root to leaf is the address of that leaf. Then the yield of this tree, with appropriately inserted markers $c_0, \ldots, c_{k-1}$, becomes the string w_θ.

Formally, w_θ is defined inductively on the subtree depth $\ell \in \{1, \ldots, k\}$, for all functions $\theta\colon \{0, \ldots, m-1\}^\ell \to S_m$.

- For the base case $\ell = 1$, the m permutations are listed one after another, each preceded with its own c_0.

$$w_\theta = c_0 w_{\theta(m-1)} \ldots c_0 w_{\theta(0)}.$$

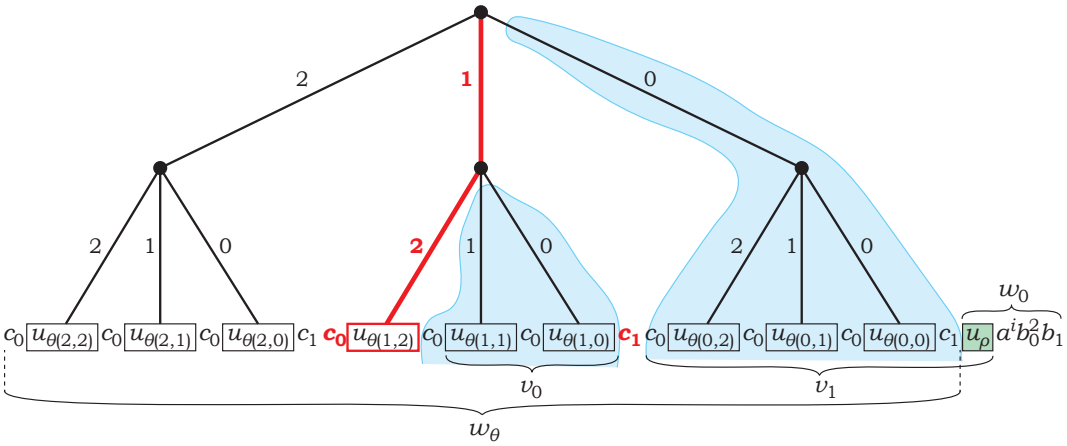

Figure 3 A tree representing a function $\theta\colon \{0,\dots,m-1\}^k \to S_m$, for $k=2$ and $m=3$.

- For the induction step, assume that w_θ has been defined for $\ell \geqslant 1$ and for all functions $\theta\colon \{0,\dots,m-1\}^\ell \to S_m$. Now, w_θ should be defined for every function $\theta\colon \{0,\dots,m-1\}^{\ell+1} \to S_m$. For each value j of the first argument of θ, denote by θ_j the function $\theta_j\colon \{0,\dots,m-1\}^\ell$ defined by $\theta_j(j_\ell,\dots,j_0)=\theta(j,j_\ell,\dots,j_0)$. Then the desired string lists the substrings for θ_j, each followed by c_ℓ.

$$w_\theta = w_{\theta_{m-1}} c_\ell \dots w_{\theta_0} c_\ell$$

Note that, by this definition, different values of $\theta(j_{k-1},\dots,j_0)$ are listed in the string w_θ in the reverse lexicographical order ($\succ$) of their arguments.

Let $\mathcal{A}$ be any DFA recognizing $L_{m,k}$, and consider the states reached by $\mathcal{A}$ after reading the strings w_θ, for different functions $\theta\colon \{0,\dots,m-1\}^k \to S_m$. It is claimed that all these $(m!)^{m^k}$ states must be pairwise distinct, which will imply the desired lower bound on the number of states.

Let $\theta,\theta'\colon \{0,\dots,m-1\}^k \to S_m$ be any two distinct functions. The goal is to construct a string w_0 with the property that $w_\theta w_0 \in L_{m,k}$ whereas $w_{\theta'} w_0 \notin L_{m,k}$. Since the functions θ and θ' are different, they differ on some argument. Let $(j_{k-1},\dots,j_0)$ be the lexicographically least argument with $\theta(j_{k-1},\dots,j_0) \neq \theta'(j_{k-1},\dots,j_0)$; in other words, this is the address of the rightmost leaf that contains different permutations in the two trees. Then, all permutations to right of this leaf are the same in both trees, and hence their compositions are also the same.

$$\prod_{(i_{k-1},\dots,i_0)\prec(j_{k-1},\dots,j_0)} \theta(i_{k-1},\dots,i_0) = \prod_{(i_{k-1},\dots,i_0)\prec(j_{k-1},\dots,j_0)} \theta'(i_{k-1},\dots,i_0)$$

Therefore, with the permutations $\theta(j_{k-1},\dots,j_0)$ and $\theta'(j_{k-1},\dots,j_0)$ added to the compositions, the resulting compositions become different.

$$\Pi = \prod_{(i_{k-1},\dots,i_0)\preccurlyeq(j_{k-1},\dots,j_0)} \theta(i_{k-1},\dots,i_0) \neq \prod_{(i_{k-1},\dots,i_0)\preccurlyeq(j_{k-1},\dots,j_0)} \theta'(i_{k-1},\dots,i_0) = \Pi'$$

Because $\Pi \neq \Pi'$, there exists an element $i \in \{0,\dots,m-1\}$ with $\Pi(i) \neq \Pi'(i)$. Let $\rho \in S_m$ be a permutation with $\rho(\Pi(i))=0$. Then, $\rho(\Pi'(i)) \neq 0$. Then the desired separating string is defined as $w_0 = u_\rho a^i b_0^{j_0} \dots b_{k-1}^{j_{k-1}}$.

To see that the string $w_\theta w_0$ is in $L_{m,k}$, consider the path to the leaf with the address $(j_{k-1}, \ldots, j_0)$ in the tree for θ, and all subtrees to the right of this path. At the root node, the substring spawned off to the right, with u_ρ appended, is denoted by v_{k-1}. The substrings spawned off to off to the right at the remaining nodes of this path are denoted by v_{k-2}, $\ldots$, v_0; furthermore, v_0 contains the leaf $(j_{k-1}, \ldots, j_0)$ itself. Then, each substring v_t is in $\{f_1, f_2, c_0, \ldots, c_t\}^*$, and the number of symbols c_t in v_t is exactly j_t. At the same time, the permutation $f_{v_0 \ldots v_{k-1}}$ equals $\rho \circ \Pi$, and since $(\rho \circ \Pi)(i) = 0$, the string is in the language by definition.

In Figure 3, the string $w_\theta w_0$ is illustrated with the suffix $w_0 = u_\rho a^i b_0^{j_0} b_1^{j_1} = a^i b_0 b_0 b_1$, where $b_0^2 b_1$ points to the leaf $\theta(1,2)$, and the corresponding substrings v_1 and v_0 are marked.

The string $w_{\theta'} w_0$ is not in $L_{m,k}$, because for the path with the same address, the substrings v'_{k-1}, $\ldots$, v'_0 to the right of this path define the permutation $f_{v'_0 \ldots v'_{k-1}} = \rho \circ \Pi'$, with $(\rho \circ \Pi')(i) \neq 0$. ◀

A lower bound on the size of every 2DFA recognizing $L_{m,k}$ is proved using the same tools as in Section 3: that is, by putting together the lower bound on the size of a DFA for $L_{m,k}$ (Lemma 8) and the precise 2DFA to 1DFA tradeoff (Theorem B).

▶ **Lemma 9.** *Let $\alpha < \frac{1}{k+1}$. Then, for every $m > e^{\frac{2}{1-(k+1)\alpha}}$, any 2DFA recognizing $L_{m,k}$ must have at least αm^{k+1} states.*

▶ **Theorem 10.** *For every $k \geqslant 1$, and for every constant factor $\beta < \frac{1}{(k+1)2^{k+1}}$, the number of states in a 2DFA necessary and sufficient to recognize every language defined by an n-state 2DFA with k drop-once pebbles (2DFA$_{k\downarrow}$) is at least βn^{k+1}, for all $n > e^{\frac{10k}{1-(k+1)2^{k+1}\beta}}$.*

Using this together with the known transformation of 2DFA$_{k\downarrow}$ to 2DFA (Theorem A) yields the following bounds.

▶ **Corollary 11.** *For every $k \geqslant 1$, the number of states in a 2DFA necessary and sufficient to recognize every language defined by an n-state 2DFA with k drop-once pebble (2DFA$_{1\downarrow}$) is $\varphi_k(n) = \Theta(n^{k+1})$. More precisely, it is contained within the bounds $\frac{1}{(k+1)\cdot 2^{k+2}} \cdot n^{k+1} \leqslant \varphi_k(n) \leqslant 5^k \cdot k! \cdot 3^{\frac{k(k+1)}{2}} \cdot n^{k+1}$, with the lower bound established for all $n > e^{20k}$.*

5 Lower bounds on the transformation of 2NFA with one drop-once pebble to different kinds of automata

The final result of this paper is a lower bound on the determinization complexity for nondeterminitic automata with one drop-once pebble (2NFA$_{1\downarrow}$), as well as lower bounds on transforming this model to DFA, 2DFA, 2DFA$_{1\downarrow}$ and 2NFA.

It is known that even if equipped with k nested pebbles (which are more general than drop-once pebbles), two-way nondeterministic finite automata still recognize only regular languages. Furthermore, the complexity of transforming such an automaton to a 2NFA without pebbles was proved to be k-fold exponential by Globerman and Harel [7, Prop. 4.3, Thm. 4.7]. The upper bound from this result holds for 2NFA$_{k\downarrow}$ with k drop-once pebbles: in particular, 2NFA$_{1\downarrow}$ can be transformed to 2NFA with exponentially many states. One of the results of this section is an asymptotically matching lower bound for the latter transformation.

The first result to be established is a double exponential lower bound on the transformation of 2NFA$_{1\downarrow}$ to DFA. The argument uses witness languages K_m, for $m \geqslant 2$, defined over an alphabet $\Sigma = \{f_1, f_2, a, c, g_1, g_2\}$, where f_1 and f_2 denote the two generators of the permutation group S_m, while g_1 and g_2 are another set of generators for S_m. Every string

$v \in \Sigma^*$ defines two permutations: the permutation $f_v \in S_n$ is obtained as a composition of all occurrences of f_1 and f_2 in v while ignoring all other symbols, whereas the other permutation $g_v \in S_m$ is defined analogously, only using the generators g_1, g_2. Then the desired language is

$$K_m = \left\{ xcwa^i v \;\middle|\; x, w \in (\Sigma \setminus \{a\})^*,\ 0 \leqslant i \leqslant m-1,\ v \in \{g_1, g_2\}^*,\ g_{wv} = \mathrm{id},\ f_w(i) = 0 \right\}.$$

This language is generally similar to L_m from Section 3, except for the starting point of the permutation f_w being defined not by a unary address, but by another permutation g_{wv}. A small $2\mathrm{NFA}_{1\downarrow}$ recognizing this language will nondeterministically guess that starting point, drop a pebble there, and then check the conditions on f_w and on g_{wv} separately.

The first task, the permutation identity testing, is done as follows.

▶ **Lemma 12.** *There exists a 2DFA with $3m\lceil \log_2 m \rceil$ states that tests an input string $v \in \{g_1, g_2\}^*$ for representing the identity permutation.*

Sketch of a proof. The automaton uses two types of states: $q_{i,j,b}$ and $r_{i,j}$, where $0 \leqslant i \leqslant m-1$, $0 \leqslant j \leqslant \lceil \log_2 m \rceil - 1$, and $b \in \{0, 1\}$.

To check that the permutation given by the input string is the identity, the automaton tests all arguments i one by one, from 0 to $m-1$. For each argument i, the automaton makes $\lceil \log_2 m \rceil$ sweeps over the input string. During the j-th sweep for i, the automaton checks whether the j-th bit of $g_v(i)$ coincides with the j-th bit of i. A left-to-right sweep begins in state $q_{i,j,b}$, where b is the j-th bit of i, traverses v while evaluating g_v in the first component, and arrives at state $q_{g_v(i),j,b}$. After checking that the j-th bit of $g_v(i)$ is indeed b, the automaton begins its way to the right in the state $r_{g_v(i),j}$, applying the inverses of all permutations on the way to the first component of the state. When it is back at the left end-marker, it will have *uncomputed* g_v and reached the state $r_{i,j}$, ready for to check the $(j+1)$-th bit.

The initial state is $q_{0,0,0}$. The accepting state is $q_{m-1,\lceil \log_2 m \rceil - 1, b_0}$, where b_0 is the $(\lceil \log_2 m \rceil - 1)$-th bit of $m-1$. ◀

▶ **Lemma 13.** *For every $m \geqslant 3$ there exists a $2\mathrm{NFA}_{1\downarrow}$ with $3m\lceil \log_2 m \rceil + m + 1$ states that recognizes K_m.*

Proof. The computation of the automaton is illustrated in Figure 4. First, the automaton makes a left-to-right sweep and verifies that w belongs to the language $\{f_1, f_2, g_1, g_2, c\}^* a^* \{g_1, g_2\}^*$.

Next, the automaton aims to drop the pebble in the correct place, and then begin evaluating the composition of f_1 and f_2 on the argument equal to the number of as. To this end, it first counts the number of as and moves leftwards, nondeterministically deciding at each occurrence of c whether to drop the pebble there, and keeping the count of as in its internal state.

Once the automaton drops the pebble, it turns around, still remembering the count of as, and begins evaluating f_w on that argument while moving to the right. Upon reaching the first a (or the right end-marker if $i = 0$), the automaton will have $f_w(i)$ computed in its state, and can check whether $f_w(i) = 0$.

Then the automaton moves back to the pebble. It remains to verify that the composition of g_1 and g_2 starting from the pebble is indeed the identity permutation. This is done using the automaton constructed in Lemma 12. The only difference is that the new automaton ignores symbols not in $\{g_1, g_2\}$ by simply skipping over them, and it treats the pebble on the tape as if it were the left end-marker. At the right end-marker, the automaton finally accepts if this composition of permutations was found to be the identity, and rejects otherwise.

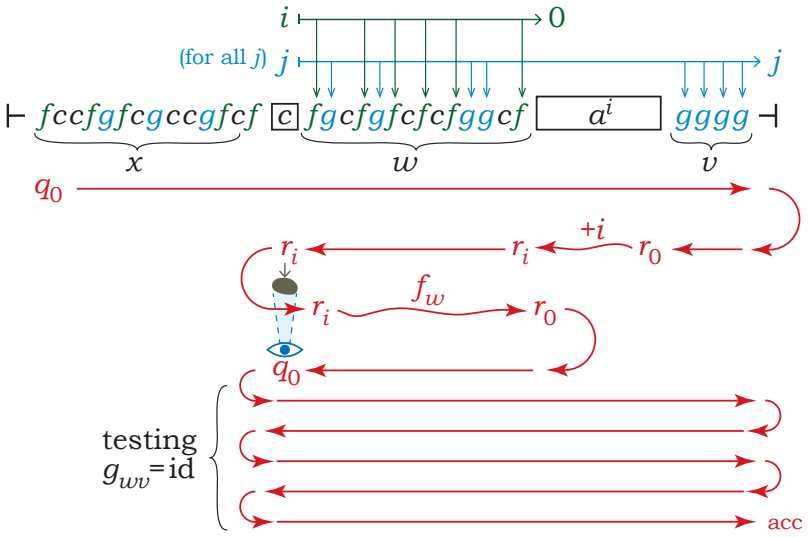

Figure 4 A $2\text{NFA}_{1\downarrow}$ with $3m\lceil\log_2 m\rceil + m + 1$ states recognizing the language K_m.

Turning to the number of states, after the pebble is placed, $m+1$ states are used for reading i and returning to the pebble, and $3m\lceil\log_2 m\rceil$ more states are used for permutation identity testing. The earlier part of the computation can reuse the same states, since transitions with the pebble in hand and with the pebble placed are disjoint. ◀

▶ **Lemma 14.** *Every DFA recognizing K_m must have at least $(m!)^{m!}$ states.*

Sketch of a proof. The proof proceeds by constructing a certain fixed vector of $m!$ permutations $(\gamma_1, \ldots, \gamma_{m!})$ and then arguing that every DFA for K_m must come to different states upon reading strings of the form $cv_{\gamma_{m!}}u_{\pi_{m!}}cv_{\gamma_{m!-1}}u_{\pi_{m!-1}}\ldots cv_{\gamma_1}u_{\pi_1}$, where $(\pi_1, \ldots, \pi_{m!})$ is any vector of permutations. ◀

A lower bound on the complexity of transforming $2\text{NFA}_{1\downarrow}$ to 2DFA is obtained by first inferring from Lemma 14 the necessary number of states in a 2DFA recognizing K_m.

▶ **Lemma 15.** *Any 2DFA recognizing K_m must have at least $m!$ states.*

Proof. Suppose there exists a 2DFA recognizing K_m with at most $m!-1$ states. Then, by Theorem B, there exists a DFA recognizing K_m with at most $(m!-1)^{m!}$ states. On the other hand, by Lemma 14, any DFA recognizing K_m requires at least $(m!)^{m!}$ states. Since $(m!-1)^{m!} < (m!)^{m!}$, a contradiction is obtained. ◀

Since there is a small $2\text{NFA}_{1\downarrow}$ for this language (Lemma 13), this leads to the following lower bound.

▶ **Theorem 16.** *For every $n \geqslant 3$, the number of states in a 2DFA necessary and sufficient to recognize every language defined by an n-state 2NFA with one drop-once pebble ($2\text{NFA}_{1\downarrow}$) is at least $2^{\frac{n}{3} - \frac{n\log_2\log_2 n}{\log_2 n}}$.*

In order to estimate the complexity of transforming $2\text{NFA}_{1\downarrow}$ to an ordinary 2NFA, one can use a known transformation of a 2NFA to a DFA with slightly less than 2^{n^2} states [9].

▶ **Lemma 17.** *Any 2NFA recognizing K_m must have at least $\sqrt{m!\cdot\log_2 m!}$ states.*

Proof. Suppose there exists a 2NFA recognizing K_m with fewer than $\sqrt{m! \cdot \log_2 m!}$ states. Then there exists a DFA recognizing K_m with fewer than $2^{\left(\sqrt{m! \cdot \log_2 m!}\right)^2} = 2^{m! \cdot \log_2 m!} = (m!)^{m!}$ states, which contradicts Lemma 14. ◀

▶ **Theorem 18.** *For every $n \geqslant 3$, the number of states in a 2NFA necessary and sufficient to recognize every language defined by an n-state 2NFA with one drop-once pebble ($2NFA_{1\downarrow}$) is at least $2^{\frac{n}{6} - \frac{n \log_2 \log_2 n}{2 \log_2 n}}$.*

The final result is a lower bound on the transformation of $2NFA_{1\downarrow}$ to $2DFA_{1\downarrow}$. It relies on the transformation of an n-state $2DFA_{1\downarrow}$ to a 2DFA with $5n^2 + 2n$ states given by Martynova and Okhotin [15].

▶ **Lemma 19.** *Let $m \geqslant 2$. Then any $2DFA_{1\downarrow}$ recognizing K_m must have at least $\sqrt{\frac{m!}{6}}$ states.*

Proof. Suppose there exists a $2DFA_{1\downarrow}$ recognizing K_m with fewer than $\sqrt{\frac{m!}{6}}$ states. Then, by Theorem A, there exists an ordinary 2DFA recognizing K_m with fewer than $6\left(\sqrt{\frac{m!}{6}}\right)^2 = m!$ states, which contradicts Lemma 15. ◀

▶ **Theorem 20.** *For every $n \geqslant 3$, the number of states in a 2DFA with one drop-once pebble ($2DFA_{1\downarrow}$) necessary and sufficient to recognize every language defined by an n-state 2NFA with one drop-once pebble ($2NFA_{1\downarrow}$) is at least $2^{\frac{n}{6} - \frac{n \log_2 \log_2 n}{2 \log_2 n}}$.*

6 Conclusion

This establishes an exponential separation between nondeterminism and determinism for two-way finite automata with one drop-once pebble—an intermediate case between one unrestricted pebble (which can be freely lifted and dropped again) and no pebbles at all. It is worth noting that the latter two cases, as proved by Geffert and Ištoňová [5], are equivalent with respect to the complexity of their determinization, and this complexity remains one of the most important, if not the most important open problem in automata theory.

References

1 Piotr Berman and Andrzej Lingas. On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1977.

2 Manuel Blum and Carl Hewitt. Automata on a 2-dimensional tape. In *8th Annual Symposium on Switching and Automata Theory, Austin, Texas, USA, October 18-20, 1967*, pages 155–160. IEEE Computer Society, 1967. doi:10.1109/FOCS.1967.6.

3 Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebble automata. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2006. doi:10.1007/11786986_15.

4 Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-walking pebble automata. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Păun, and Grzegorz Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 72–83. Springer, 1999. doi:10.1007/978-3-642-60207-8_7.

5 Viliam Geffert and Lubomíra Istonová. Translation from classical two-way automata to pebble two-way automata. *RAIRO Theor. Informatics Appl.*, 44(4):507–523, 2010. doi:10.1051/ita/2011001.

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

7 Noa Globerman and David Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theor. Comput. Sci.*, 169(2):161–184, 1996. doi:10.1016/S0304-3975(96)00119-3.

8 Pavel Goralcik, A. Goralciková, and Václav Koubek. Alternation with a pebble. *Inf. Process. Lett.*, 38(1):7–13, 1991. doi:10.1016/0020-0190(91)90208-Y.

9 Christos A. Kapoutsis. Removing bidirectionality from nondeterministic finite automata. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 544–555. Springer, 2005. doi:10.1007/11549345_47.

10 Christos A. Kapoutsis. Nondeterminism is essential in small two-way finite automata with few reversals. *Inf. Comput.*, 222:208–227, 2013. doi:10.1016/J.IC.2012.11.001.

11 Christos A. Kapoutsis. Two-way automata versus logarithmic space. *Theory Comput. Syst.*, 55(2):421–447, 2014. doi:10.1007/S00224-013-9465-0.

12 Michal Kunc and Alexander Okhotin. Describing periodicity in two-way deterministic finite automata using transformation semigroups. In Giancarlo Mauri and Alberto Leporati, editors, *Developments in Language Theory - 15th International Conference, DLT 2011, Milan, Italy, July 19-22, 2011. Proceedings*, volume 6795 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2011. doi:10.1007/978-3-642-22321-1_28.

13 Alessandro Clerici Lorenzini, Giovanni Pighizzini, and Luca Prigioniero. Two-way automata and bounded languages. In Giuseppa Castiglione and Sabrina Mantaci, editors, *Implementation and Application of Automata - 29th International Conference, CIAA 2025, Palermo, Italy, September 22-25, 2025, Proceedings*, volume 15981 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 2025. doi:10.1007/978-3-032-02602-6_6.

14 Olga Martynova and Alexander Okhotin. Nondeterministic tree-walking automata are not closed under complementation. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*, volume 334 of *LIPIcs*, pages 168:1–168:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.168.

15 Olga Martynova and Alexander Okhotin. A time to cast away stones: On a family of pebble automata. *Int. J. Found. Comput. Sci.*, 37(1):23–46, 2026. doi:10.1142/S0129054125410023.

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

17 William J. Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 275–286. ACM, 1978. doi:10.1145/800133.804357.

18 Mathias Samuelides and Luc Segoufin. Complexity of pebble tree-walking automata. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings*, volume 4639 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2007. doi:10.1007/978-3-540-74240-1_40.

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