On the Size Complexity of Deterministic Frequency Automata[⋆]

Rūsiņš Freivalds,[1,2] Thomas Zeugmann[2] and Grant R. Pogosyan[3]

[1] Institute of Mathematics and Computer Science, University of Latvia,
Raiņa bulvāris 29, Riga, LV-1459, Latvia
Rusins.Freivalds@mii.lu.lv
[2] Division of Computer Science, Hokkaido University,
N-14, W-9, Sapporo 060-0814, Japan
thomas@ist.hokudai.ac.jp
[3] Division of Natural Sciences, International Christian University,
Mitaka, Tokyo 181-0015, Japan
grant@icu.ac.jp

Abstract. Austinat, Diekert, Hertrampf, and Petersen [2] proved that every language L that is (m, n)-recognizable by a deterministic frequency automaton such that $m > n/2$ can be recognized by a deterministic finite automaton as well. First, the size of deterministic frequency automata and of deterministic finite automata recognizing the same language is compared. Then approximations of a language are considered, where a language L' is called an *approximation* of a language L if L' differs from L in only a finite number of strings. We prove that if a deterministic frequency automaton has k states and (m, n)-recognizes a language L, where $m > n/2$, then there is a language L' approximating L such that L' can be recognized by a deterministic finite automaton with no more than k states.

Austinat *et al.* [2] also proved that every language L over a single-letter alphabet that is $(1, n)$-recognizable by a deterministic frequency automaton can be recognized by a deterministic finite automaton. For languages over a single-letter alphabet we show that if a deterministic frequency automaton has k states and $(1, n)$-recognizes a language L then there is a language L' approximating L such that L' can be recognized by a deterministic finite automaton with no more that k states. However, there are approximations such that our bound is much higher, i.e., $k!$.

1 Introduction

The notion of frequency computation was introduced by Rose [18] as an attempt to have an absolutely deterministic mechanism with properties similar to probabilistic algorithms. The definition was as follows. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote

[⋆] The research was supported by Grant No. 09.1570 from the Latvian Council of Science and the Invitation Fellowship for Research in Japan S12052 by the Japan Society for the Promotion of Science.

the set of all natural numbers. A function $f\colon \mathbb{N} \to \mathbb{N}$ is (m, n)-computable, where $1 \le m \le n$, $m, n \in \mathbb{N}$, iff there exists a recursive function $R\colon \mathbb{N}^n \to \mathbb{N}^n$ such that, for all n-tuples $(x_1, \cdots, x_n) \in \mathbb{N}^n$ of mutually distinct natural numbers,

$$\mathrm{card}\{i \mid (R(x_1, \cdots, x_n))_i = f(x_i)\ ,\ 1 \le i \le n\} \ge m\ ,$$

where $(R(x_1, \cdots, x_n))_i$ denotes the ith component of $R(x_1, \cdots, x_n)$.

McNaughton [16] cites in his survey a problem (posed by Myhill) whether f has to be recursive if m is close to n. This problem was answered by Trakhtenbrot [20] by showing that f is recursive whenever $2m > n$. On the other hand, Trakhtenbrot [20] proved that, if $2m = n$ then nonrecursive functions can be (m, n)-computed. Kinber [14, 13] extended the research by considering frequency enumeration of sets. The class of (m, n)-computable sets equals the class of recursive sets if and only if $2m > n$. The notion of frequency computation can be extended to other models of computation. Frequency computation in polynomial time was discussed in full detail by Hinrichs and Wechsung [11].

For resource bounded computations, the behavior of frequency computability is completely different: for example, whenever $n' - m' > n - m$, it is known that under any reasonable resource bound there are sets which are (m', n')-computable, but not (m, n)-computable. However, scaling down to finite automata, the analogue of Trakhtenbrot's [20] result holds again: the class of languages (m, n)-recognizable by deterministic frequency automata equals the class of regular languages if and only if $2m > n$ (cf. Austinat *et al.* [2]). Conversely, as shown by Austinat *et al.* [2], for $2m \le n$, the class of languages (m, n)-recognizable by deterministic frequency automata is uncountable for a two-letter alphabet. A stronger result concerning sets separable by finite automata was claimed by Kinber [13], and this result would imply the results mentioned above as a corollary. However, as shown by Tantau [19], who gave a counter-example, Kinber's [13] Theorem 3 does not hold.

When restricted to a one-letter alphabet, then every (m, n)-recognizable language is regular. This was shown by Kinber [14] and also by Austinat *et al.* [2].

Frequency computations became increasingly popular when relations between frequency computation and computation with a small number of queries was discovered [1, 2, 3, 4, 5, 8, 10, 15].

2 Deterministic Frequency Automata

For finite automata the definition of frequency computation is not so obvious. First, let us fix the necessary notations. We assume familiarity with finite automata theory, cf., e.g., Hopcroft and Ullman [12]. Let Σ be any finite alphabet, and let Σ^* be the free monoid over Σ. Every subset $L \subseteq \Sigma^*$ is said to be a *language*. The elements of Σ^* are called *strings*, and we use $|x|$ to denote the *length of a string* $x \in \Sigma^*$. By $\chi_L\colon \Sigma^* \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$, we denote the *characteristic function* of L, i.e., for all $x \in \Sigma^*$ we set

$$\chi_L(x) = \begin{cases} 1, & \text{if } x \in L\ ; \\ 0, & \text{if } x \notin L\ . \end{cases}$$

To define deterministic frequency automata we extend the notion of a deterministic finite automaton as follows (cf. Austinat *et al.* [2]).

Let $\mathcal{A} = [Q, \Sigma, \#, \delta, q_0, \tau, n]$, where $n \in \mathbb{N}$, $n \geq 1$, is a number, Q is a finite set of states, q_0 is the initial state, Σ is a finite alphabet and $\#$ is a new symbol such that $\# \notin \Sigma$. The mapping $\delta\colon Q \times (\Sigma \cup \{\#\})^n \to Q$ is the transition function, and we call $\tau\colon Q \to \mathbb{B}^n$ the type of state. The type of state is used for the output. We refer to $\mathcal{A}$ as *deterministic frequency automaton.*

Next, we formally describe the behavior of a deterministic frequency automaton $\mathcal{A}$. Let $n \in \mathbb{N}$, $n \geq 1$, and let $x = (x_1, \dots, x_n) \in (\Sigma^*)^n$ be an input vector. We define $|x| = \max\{|x_i| \mid 1 \leq i \leq n\}$, and $q \circ x = \delta^*(q, (x_1\#^{\ell_1}, \dots, x_n\#^{\ell_n}))$, where $\delta^*\colon Q \times ((\Sigma \cup \{\#\})^n)^*$ is the usual extension of δ on n-tuples of strings, and $\ell_i = |x| - |x_i|$ for all $1 \leq i \leq n$. Then the output of $\mathcal{A}$ is defined to be the type $\tau(q_0 \circ x)$. We refer to such an automaton as n-DFA for short.

A language $L \subseteq \Sigma^*$ is said to be (m, n)*-recognized* by an n-DFA $\mathcal{A}$ iff for each n-tuple $(x_1, \dots, x_n) \in (\Sigma^*)^n$ of pairwise distinct strings the tuples $\tau(q_0 \circ x)$ and $(\chi_L(x_1), \dots, \chi_L(x_n))$ coincide on at least m components. A language $L \subseteq \Sigma^*$ is called (m, n)-recognizable iff there is an n-DFA $\mathcal{A}$ that (m, n)-recognizes L.

Frequency computation is not much similar to probabilistic computation. The advantages of probabilistic algorithms over deterministic ones are based on the effect that at some moments the algorithm has a choice of several possible continuations of the computation process but there is no information which one suits better. For example, if the language is $L_{3,5} = \{1^n \mid n \text{ is divisible by 3 or 5}\}$ then a probabilistic algorithm has a choice: whether to test divisibility by 3 or divisibility by 5.

Frequency algorithms have no such option. Nonetheless, frequency automata can have size complexity advantages over deterministic automata as well. To see this, consider the language $L_{2015} \subseteq \{1\}^*$ defined as

$$L_{2015} = \{1^n \mid n = 2015\} \; . \tag{1}$$

A deterministic finite automaton recognizing this language needs to have 2016 states. On the other hand, there is a 1-state 100-DFA $\mathcal{A}$ that $(99, 100)$-recognizes the language L_{2015}. The 100-DFA $\mathcal{A}$ rejects all 100-tuples, i.e., it always outputs $\tau(q_0 \circ x) = (b_1, \dots, b_{100}) \in \mathbb{B}^{100}$, where $b_i = 0$ for all $i = 1, \dots, 100$. But nonetheless $\mathcal{A}$ does $(99, 100)$-recognize the language L_{2015}, since it can only be wrong on at most one of the 100 strings in any 100-tuple given as input.

This idea can be easily extended. Consider $L_{2015,2158} \subseteq \{1\}^*$ defined as

$$L_{2015,2158} = \{1^n \mid n = 2015 \text{ or } n = 2158\} \; . \tag{2}$$

Then there exists a 1-state 100-DFA $\mathcal{A}$ such that $\mathcal{A}$ does $(98, 100)$-recognize the language $L_{2015,2158}$. Again, $\mathcal{A}$ rejects all 100-tuples, but nonetheless recognizes the language $L_{2015,2158}$.

Maybe, the only advantage of deterministic frequency automata over deterministic finite automata is to save size by producing errors on a constant number of fixed input words? Not at all, some nonregular and even nonrecursive languages can be recognized by deterministic frequency automata.

Theorem 1 (Austinat *et al.* [2]). *There exists a nonrecursive language that is $(1,2)$-recognizable by a 2-DFA.*

3 Size Complexity

In this section we study the size complexity of deterministic frequency automata and compare it to the size of ordinary deterministic automata recognizing the same language or an approximation of it.

First, we extend the example shown in (1). This directly yields the following theorem showing that the advantage of deterministic frequency automata with respect to their size complexity over deterministic finite automata can be arbitrarily large.

Theorem 2. *For every $s \in \mathbb{N}$, $s \geq 1$, there is a language L_s such that, for every $n \geq 1$ there is an n-DFA $\mathcal{A}$ having one state that $(n-1,n)$-recognizes L_s, but every deterministic finite automaton recognizing L_s needs at least s states.*

Proof. Let $s \geq 1$ be arbitrarily fixed and let L_s be any language defined as

$$L_s = \{1^m \mid m = m_0\} ,$$

where m_0 is a number such that every deterministic finite automaton needs at least s states to recognize the language L_s. The desired n-DFA $\mathcal{A}$ can then be easily defined such that $Q = \{q_0\}$ and such that for all n-tuples of strings $x \in (\{1\}^*)^n$ the mapping $\tau(q_0 \circ x)$ returns the n-tuple containing only zeros. Hence, L_s is $(n-1,n)$-recognized by $\mathcal{A}$, since $\mathcal{A}$ rejects all input strings. But an error can happen only once, i.e., if 1^{m_0} is part of the input. $\square$

Clearly, Theorem 2 can be easily generalized along the lines of the example shown in (2). So the more interesting question is whether or not there is always a (huge) gap in the size of deterministic frequency automata and deterministic finite automata provided they accept roughly the same language. Here by "roughly" we mean that we allow the deterministic finite automaton to accept an approximation of the language accepted by the corresponding deterministic frequency automaton.

Austinat *et al.* [2] proved that in the case $m > n/2$ every language (m,n)-recognized by an n-DFA is also recognizable by a deterministic finite automaton. There were no size estimates of the deterministic frequency automata and the deterministic finite automata, respectively, in [2] but a careful optimization of the construction given in [2] proves the following theorem.

Theorem 3. *Let any pair (m,n), where $m > n/2$, be arbitrarily fixed, and let $\mathcal{A}$ be any n-DFA having k states. If there is a language L which is (m,n)-recognized by the n-DFA $\mathcal{A}$ then there exists a language L' differing from L only in a finite number of strings such that L' can be recognized by a deterministic finite automaton with 2^{k+3} states.*

It is well-known that for nondeterministic finite automata and probabilistic 1-way automata the size gap to deterministic finite automata recognizing the same language is exponential [17, 6, 7, 9]. But now we consider approximations and compare n-DFA and deterministic finite automata. Our next theorem shows that the gap expressed in Theorem 3 between the sizes of deterministic frequency automata and deterministic finite automata is not necessary provided $|\Sigma| \geq 2$.

Theorem 4. *Let any pair (m, n), where $m > n/2$, be arbitrarily fixed, and let $\mathcal{A}$ be any n-DFA having k states. If there is a language L which is (m, n)-recognized by the n-DFA $\mathcal{A}$ then there exists a language L' differing from L only in a finite number of strings such that L' can be recognized by a deterministic finite automaton with k states.*

Proof. This proof is nonconstructive. This means that we do not present an effective construction how to transform a program for (m, n)-recognition of a language into a program for deterministic recognition of the same language. Instead we show that such a transformation can be done using a finite amount of additional information and we show that such an additional information cannot fail to exist but we do not show how to obtain such additional information effectively. Since $|\Sigma| \geq 2$, we assume without loss of generality that $\{0, 1\} \subseteq \Sigma$.

By $[\alpha_1, \alpha_2, \cdots, \alpha_k/\beta_1, \beta_2, \cdots, \beta_k]$, where $\alpha_i, \beta_i \in \{0, 1\}$, $1 \leq i \leq k$, we denote the set of all strings $x \in \Sigma^*$ such that

$$\begin{aligned} \chi_L(x\alpha_1) &= \beta_1 \ , \\ \chi_L(x\alpha_1\alpha_2) &= \beta_2 \ , \\ &\cdots \\ \chi_L(x\alpha_1\alpha_2\cdots\alpha_k) &= \beta_k \ . \end{aligned}$$

We start to describe a noneffective "construction" of a tree denoted by S. This tree is defined inductively. In principle, the tree might be infinite but we prove below that the "construction" results in a finite tree. We will use the resulting tree as the finite additional information about the given frequency automaton.

– There is a single vertex of the zero level in the tree (called the root). All the strings $x \in \Sigma^*$ are assigned to it.
– If the vertex of the p-th level is already in the tree with the set

$$[\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p]$$

used as label to it, we "consider" whether the sets

$$\begin{aligned} &[\alpha_1, \alpha_2, \cdots, \alpha_p, 0/\beta_1, \beta_2, \cdots, \beta_p, 0] \quad \text{and} \\ &[\alpha_1, \alpha_2, \cdots, \alpha_p, 0/\beta_1, \beta_2, \cdots, \beta_p, 1] \end{aligned}$$

are infinite. If the two sets are infinite then we add two $(p + 1)$-th level vertices to the tree and label them by the sets

$$\begin{aligned} &[\alpha_1, \alpha_2, \cdots, \alpha_p, 0/\beta_1, \beta_2, \cdots, \beta_p, 0] \quad \text{and} \\ &[\alpha_1, \alpha_2, \cdots, \alpha_p, 0/\beta_1, \beta_2, \cdots, \beta_p, 1] \ , \end{aligned}$$

respectively. We say that these two vertices

$$\begin{array}{ll}[\alpha_1, \alpha_2, \cdots, \alpha_p, 0/\beta_1, \beta_2, \cdots, \beta_p, 0] & \text{and} \\ [\alpha_1, \alpha_2, \cdots, \alpha_p, 0/\beta_1, \beta_2, \cdots, \beta_p, 1] & \end{array}$$

are *strictly higher* than the vertex v labeled by

$$[\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p] \tag{3}$$

If at least one of these two sets is finite or empty then no vertex is added in the result of this "consideration." In this case, we continue as follows. If the two sets

$$\begin{array}{ll}[\alpha_1, \alpha_2, \cdots, \alpha_p, 1/\beta_1, \beta_2, \cdots, \beta_p, 0] & \text{and} \\ [\alpha_1, \alpha_2, \cdots, \alpha_p, 1/\beta_1, \beta_2, \cdots, \beta_p, 1] & \end{array}$$

are infinite, then we add two $(p+1)$-th level vertices to the tree S and label them by the sets

$$\begin{array}{ll}[\alpha_1, \alpha_2, \cdots, \alpha_p, 1/\beta_1, \beta_2, \cdots, \beta_p, 0] & \text{and} \\ [\alpha_1, \alpha_2, \cdots, \alpha_p, 1/\beta_1, \beta_2, \cdots, \beta_p, 1]\ , & \end{array}$$

respectively. Again, we say that these two vertices are strictly higher than v. If at least one of these two sets is finite or empty then no vertex is added.

Notice two properties of the sets assigned to the vertices of the tree S.

(1) If a vertex
$$[\alpha_1, \alpha_2, \cdots, \alpha_{p+1}/\beta_1, \beta_2, \cdots, \beta_{p+1}]$$
is strictly higher than
$$[\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p]$$
then the following inclusion holds:
$$[\alpha_1, \alpha_2, \cdots, \alpha_{p+1}/\beta_1, \beta_2, \cdots, \beta_{p+1}] \subseteq [\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p]\ .$$

(2) If two vertices
$$\begin{array}{ll}[\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p] & \text{and} \\ [\gamma_1, \gamma_2, \cdots, \gamma_r/\delta_1, \delta_2, \cdots, \delta_r] & \end{array}$$
are distinct vertices in the tree S then there exist strings
$$\begin{array}{ll}x \in [\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p] & \text{and} \\ y \in [\gamma_1, \gamma_2, \cdots, \gamma_r/\delta_1, \delta_2, \cdots, \delta_r] & \end{array}$$
and a string $z \in \Sigma^*$ such that $(xz \in L) \not\Leftrightarrow (yz \in L)$.

Property (1) is an immediate consequence of the construction.

The second property is proved separately for the following two cases.

Case (a). One of the vertices is strictly higher than the other one, e.g., $p > r$.

By the definition of strictly higher (cf. (3)) we then have $\alpha_i = \gamma_i$ and $\beta_i = \delta_i$ for $i = 1, \ldots, r$ and so, by Property (1) and the transitivity of set inclusion, $[\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p] \subseteq [\alpha_1, \alpha_2, \cdots, \alpha_r/\beta_1, \beta_2, \cdots, \beta_r]$. Then we take any arbitrarily chosen string $x \in [\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p]$, and any string $y \in [\alpha_1, \alpha_2, \cdots, \alpha_r/\beta_1, \beta_2, \cdots, \beta_r] \setminus [\alpha_1, \alpha_2, \cdots, \alpha_r, \alpha_{r+1}/\beta_1, \beta_2, \cdots, \beta_r, \beta_{r+1}]$, and define $z = \alpha_1 \cdots \alpha_r \alpha_{r+1}$, i.e., the concatenation of $\alpha_1, \ldots, \alpha_{r+1}$. By construction x, y, and z clearly satisfy Property (2).

The subcase $p < r$ is handled *mutatis mutandis*.

Case (b). None of the vertices $[\alpha_1, \cdots, \alpha_p/\beta_1, \cdots, \beta_p]$ and $[\gamma_1, \cdots, \gamma_r/\delta_1, \cdots, \delta_r]$ is strictly higher than the other one.

Let j be the least number less than $\min\{p, r\}$ such that $\alpha_i = \gamma_i$ and $\beta_i = \delta_i$ for all $i = 1, \ldots, j$. Note that we allow $j = 0$ to denote the case that the highest such vertex is the root, i.e., the vertex of level zero.

Consequently, we then have that $\alpha_{j+1} \neq \gamma_{j+1}$ or $\beta_{j+1} \neq \delta_{j+1}$. But by construction we know that $\alpha_{j+1} \neq \gamma_{j+1}$ cannot occur. Therefore, we know that $\beta_{j+1} \neq \delta_{j+1}$ must hold, i.e., $\gamma_{j+1} = \overline{\beta}_{j+1}$, where $\overline{b}$ denotes the logical negation of $b \in \{0, 1\}$. This in turn implies that

$$[\alpha_1, \ldots, \alpha_{j+1}/\beta_1, \ldots, \beta_{j+1}] \cap [\alpha_1, \ldots, \alpha_{j+1}/\beta_1, \ldots, \overline{\beta}_{j+1}] = \emptyset \ , \qquad (4)$$

i.e., these two sets partition $[\alpha_1, \ldots, \alpha_j/\beta_1, \ldots, \beta_j]$. So by Property (1) and Equality (4) we also have

$$[\gamma_1, \cdots, \gamma_r/\delta_1, \cdots, \delta_r] \cap [\alpha_1, \ldots, \alpha_{j+1}/\beta_1, \ldots, \beta_{j+1}] = \emptyset \ . \qquad (5)$$

Therefore, we can take any string $x \in [\alpha_1, \alpha_2, \cdots, \alpha_p/\beta_1, \beta_2, \cdots, \beta_p]$, and any string $y \in [\gamma_1, \cdots, \gamma_r/\delta_1, \cdots, \delta_r]$. Furthermore, we set $z = \alpha_1 \cdots \alpha_{j+1}$.

So, if $j = 0$ then x and y are as above, and $z = \alpha_1$. Thus, $\chi_L(x\alpha_1) = \beta_1$ and $\chi_L(y\alpha_1) \neq \beta_1$. In the general case that $j > 0$ we directly obtain that $\chi_L(xz) = \beta_{j+1}$ and $\chi_L(yz) \neq \beta_{j+1}$ (cf. (5)), and Property (2) is shown.

Claim: The tree S has at most k vertices.

Suppose that the tree S has at least $(k+1)$ vertices. Take an n-tuple of pairwise distinct strings from each of the sets, and, again nonconstructively, appropriate $z^1, \ldots, z^{k+1}$ (such that Property (2) will be applicable). Denote the resulting tuples by $(x_1^1 z^1, \cdots, x_n^1 z^1)$, $(x_1^2 z^2, \cdots, x_n^2 z^2)$, $\ldots$, $(x_1^{k+1} z^{k+1}, \cdots, x_n^{k+1} z^{k+1})$.

The n-DFA $\mathcal{A}$ has only k states but we have taken $(k+1)$ many n-tuples of strings. Hence there are two distinct values i and j such that, after reading $(x_1^i z^i \cdots, x_n^i z^i)$ and $(x_1^j z^j \cdots, x_n^j z^j)$, the n-DFA $\mathcal{A}$ is in the same state. So, by the choice of the z^ℓ, $\ell = 1, \ldots, k+1$, we know that $z^i = z^j =: z$ and that Property (2) is applicable. Since we also have $m > n/2$, if the n-tuple corresponding to the output on $(x_1^i z \cdots , x_n^i z)$ contains no less than m correct values then the n-DFA cannot produce an n-tuple as output which contains m many correct values on the input $(x_1^j z, \cdots , x_n^j z)$. This is a contradiction, and the claim is shown.

Now we add one more level to the tree S in order to construct a new tree S'. Denote the highest level of S by s, We have already shown that $s \leq k-1$. Every vertex of the highest level and only these vertices have the following property. If the vertex is

$$[\alpha_1, \alpha_2, \cdots , \alpha_p / \beta_1, \beta_2, \cdots , \beta_p]$$

then there may exist a $\gamma \in \Sigma$ such that the sets

$$\begin{array}{ll}[\alpha_1, \alpha_2, \cdots , \alpha_p, \gamma / \beta_1, \beta_2, \cdots , \beta_p, 0] & \text{and} \\ [\alpha_1, \alpha_2, \cdots , \alpha_p, \gamma / \beta_1, \beta_2, \cdots , \beta_p, 1] & \end{array}$$

are infinite. In this case we add

$$\begin{array}{ll}[\alpha_1, \alpha_2, \cdots , \alpha_p, \gamma / \beta_1, \beta_2, \cdots , \beta_p, 0] & \text{and} \\ [\alpha_1, \alpha_2, \cdots , \alpha_p, \gamma / \beta_1, \beta_2, \cdots , \beta_p, 1] & \end{array}$$

as new vertices of the k-th level over the vertex

$$[\alpha_1, \alpha_2, \cdots , \alpha_p / \beta_1, \beta_2, \cdots , \beta_p] \ .$$

Each newly added vertex either corresponds to some state of the n-DFA $\mathcal{A}$ that is already related to another vertex of S' or it corresponds to a state of $\mathcal{A}$ that yet has no vertex in S'. We transform S' into a graph which is no longer a tree by identifying vertices that correspond to the same state of the n-DFA $\mathcal{A}$. We continue adding new and new vertices in the same style. For every vertex that has been added in the process of transforming the tree S we try to add new vertices of a higher level but we identify them with vertices already constructed if no new state of the n-DFA $\mathcal{A}$ is employed. It is easy to see that the obtained graph (we call it S'') has no more than k vertices.

To construct an equivalent deterministic finite automaton we notice that in the construction above we distinguished between "infinitely many strings" and "a finite number of strings." Let d exceed the length of all considered "a finite number of strings." Then every string of length no less than d falls in one or several of the sets which are names of vertices in S''. Consider unions of such sets. We say that a union of the sets which are names of vertices in S'' is *consistent* if it is a union of type

$$[\alpha_1/\beta_1] \cup [\alpha_1, \alpha_2/\beta_1, \beta_2] \cup \cdots \cup [\alpha_1, \alpha_2, \cdots , \alpha_p/\beta_1, \beta_2, \cdots , \beta_p].$$

We say that a union is *complete* if it is not possible to add any other set which is a name of a vertex in S'' to this union.

It is easy to see that every consistent and complete union has incorporated a vertex of the upper-most level which is not present at any other consistent and complete union. Hence the number of consistent and complete unions does not exceed the number of vertices in the graph S'', i.e., it does not exceed k.

Consider a deterministic finite automaton that has states numbered by the consistent and complete unions of vertices of the tree S'. The initial state (for the empty input string) is the vertex of the level zero. The state numbered $[\alpha_1, \alpha_2, \cdots, \alpha_j / \beta_1, \beta_2, \cdots, \beta_j]$ is accepting if and only if $\beta_j = 1$. Since $m > n/2$, it is possible to construct the program for the deterministic finite automaton counting to which state the transition should take place. For strings that have a length exceeding d this automaton is equivalent to the given n-DFA $\mathcal{A}$ and its number of states does not exceed k. □

The formulation of Theorem 4 may seem to be over-complicated. Why another language is considered? It is well-known that a language differing from a regular languages only in a finite number of strings is itself regular. However, the number of states may be influenced very much if we neglect this finite number of strings (cf. Theorem 2, and Theorems 3 and 4, respectively).

Theorem 4 has a sensitive restriction: the considered deterministic frequency automata have parameters (m, n) with $m > n/2$. Looking at Theorem 1 we see that this restriction cannot be relaxed.

Therefore, we turn our attention to the unary case, i.e., only single-letter alphabets are allowed. Then the situation changes drastically, since the following theorem is known.

Theorem 5 (Austinat *et al.* [2], Kinber [13]). *Let any pair (m, n), where $0 < m \leq n$, be arbitrarily fixed, and let $\mathcal{A}$ be any n-DFA having k states. Then every language L over a single-letter alphabet that is (m, n)-recognized by the n-DFA $\mathcal{A}$ is also recognizable by a deterministic finite automaton.*

We complement Theorem 5 in terms of size complexity of the automata.

Theorem 6. *Let any pair (m, n), where $0 < m \leq n$, be arbitrarily fixed, and let $\mathcal{A}$ be any n-DFA having k states. If there is a language L over a single-letter alphabet Σ which is (m, n)-recognized by the n-DFA $\mathcal{A}$ then there exists a language $L' \subseteq \Sigma^*$ differing from L only in a finite number of strings such that L' can be recognized by a deterministic finite automaton with k states.*

Proof. Without loss of generality, let the single-letter alphabet $\Sigma = \{1\}$. If an n-DFA $\mathcal{A}$ does (m, n)-recognize a language $L \subseteq \Sigma^*$ then $\mathcal{A}$ $(1, n)$-recognizes L, too. Let Q be the set of states of $\mathcal{A}$. By assumption we know that $|Q| = k$. For all $q_b \in Q$ we consider the following sets of n-tuples of input strings:

$$T_b = \{(1^{m_1}, 1^{m_2}, \cdots, 1^{m_n}) \mid \mathcal{A} \text{ after reading } (1^{m_1}, 1^{m_2}, \cdots, 1^{m_n}) \text{ enters state } q_b\} ,$$

where all $m_i \geq 1$, i.e., all m_i are positive natural numbers. Note that the class T_b is labeled by the index b of the state q_b but not by the n-tuple of input strings.

Let $\mathcal{T}$ be the collection $\mathcal{T} = \{T_b \mid T_b \text{ is infinite }\}$. We define a relation between the sets T_b in $\mathcal{T}$. Let T_b and T_c be any sets in $\mathcal{T}$. Then we say that

$(1^{m_1}, 1^{m_2}, \cdots, 1^{m_n}) \in T_b$ *precedes* $(1^{s_1}, 1^{s_2}, \cdots, 1^{s_n}) \in T_c$ if there exists a string 1^v such that

$$m_1 + v = s_1, m_2 + v = s_2, \cdots, m_n + v = s_n \ .$$

Precedence of n-tuples induces precedence of sets in $\mathcal{T}$. Since all the sets $T_b \in \mathcal{T}$ are infinite, for every pair (T_b, T_c), either T_b precedes T_c or T_c precedes T_b or T_b does not precede T_c and T_c does not precede T_b. The precedence relation divides the sets into equivalence classes corresponding to states b and c in the same cycle.

Let k be the number of the states in $\mathcal{A}$. Then each set $T_b \in \mathcal{T}$ contains an n-tuple $(1^{m_1}, 1^{m_2}, \cdots, 1^{m_n})$ such that $m_1 + m_2 + \cdots + m_n \leq n \cdot k$.

It may be possible that the same automaton $\mathcal{A}$ $(1, n)$-recognizes several languages. Let L be one of these languages. Then it is possible to make assertions about which components of the n outputs of $\mathcal{A}$ on a certain n-tuple of input strings are correct. By the definition of frequency computation, for every n-tuple of input strings at least one of the outputs is correct.

Assume that each equivalence class of the sets T_b is represented by one set from the equivalence class:

$$(1^{m_1}, 1^{m_2}, \cdots, 1^{m_n}),\ (1^{p_1}, 1^{p_2}, \cdots, 1^{p_n}),\ \cdots,\ (1^{s_1}, 1^{s_2}, \cdots, 1^{s_n}) \ .$$

Let these classes contain $t_m, t_p, \cdots, t_s$ sets T_b, respectively. Let N be the least common multiple of $t_m, t_p, \cdots, t_s$.

Assume that one of the equivalence classes is represented by n-tuples of input strings

$$\begin{gathered} (1^{m_1}, 1^{m_2}, \cdots, 1^{m_n}) \ , \\ (1^{m_1+1}, 1^{m_2+1}, \cdots, 1^{m_n+1}) \ , \\ \cdots \\ (1^{m_1+t_m}, 1^{m_2+t_m}, \cdots, 1^{m_n+t_m}) \ . \end{gathered}$$

Then by analyzing the outputs on these n-tuples

$$\begin{gathered} (y_1^1, y_2^1, \cdots, y_n^1) \\ (y_1^2, y_2^2, \cdots, y_n^2) \\ \cdots \\ (y_1^{t_m}, y_2^{t_m}, \cdots, y_n^{t_m}) \end{gathered}$$

we can find one or several periodical sequences $f = \langle f(1), f(2), \cdots \rangle$ of elements $0, 1$ (any such sequence describes a language in a single-letter alphabet such that $f(n) = 1$ iff 1^n is in the language) with period t_m such that

$$\begin{gathered} (f(m_1) = y_1^1) \vee (f(m_2) = y_2^1) \vee \cdots \vee (f(m_n) = y_n^1) \\ (f(m_1 + 1) = y_1^2) \vee (f(m_2 + 1) = y_2^2) \vee \cdots \vee (f(m_n + 1) = y_n^2) \\ \cdots \\ (f(m_1 + t_m) = y_1^{t_m}) \vee (f(m_2 + t_m) = y_2^{t_m}) \vee \cdots \vee (f(m_n + t_m) = y_n^{t_m}) \ . \end{gathered}$$

The Chinese Remainder Theorem gives an algorithm of how to find such sequences. However, combining all these calculations give us only a sequence (or several sequences) with period $N \leq k!$. On the other hand, the states of the frequency automaton are periodically repeated with periods $t_m, t_p, \cdots, t_s$, respectively. The least common multiple of all these periods is N. Hence the language can be recognized by a deterministic finite automaton with N states.

There is no need to perform these calculations by a finite automaton. After performing these calculations we can reconstruct a full period of the length N and construct the program of the deterministic finite automaton.

However, a deeper analysis of the frequency automaton is needed, since our theorem promises a deterministic finite automaton with no more than k states, and not N states. Let L be a language $(1,n)$-recognized by the n-DFA $\mathcal{A}$. For each cycle

$$\begin{gathered}(1^{m_1}, 1^{m_2}, \cdots, 1^{m_n}) \ , \\ (1^{m_1+1}, 1^{m_2+1}, \cdots, 1^{m_n+1}) \ , \\ \cdots \\ (1^{m_1+t_m}, 1^{m_2+t_m}, \cdots, 1^{m_n+t_m}) \ .\end{gathered}$$

of the frequency automaton we can say which outputs

$$\begin{gathered}(y_1^1, y_2^1, \cdots, y_n^1) \\ (y_1^2, y_2^2, \cdots, y_n^2) \\ \cdots \\ (y_1^{t_m}, y_2^{t_m}, \cdots, y_n^{t_m})\end{gathered}$$

are correct and which are not. For every cycle we establish whether for some $i \in \{1, 2, \cdots, t_m\}$ all the outputs

$$y_i^1, y_i^2, \cdots, y_i^{t_m}$$

are correct. If there is such a cycle and such an i then the i-th output of this cycle provides correct results for all sufficiently long input strings.

If such an i does not exist for all cycles then the n-DFA $\mathcal{A}$ does not recognize L correctly, because, by Chinese Remainder Theorem, there is an n-tuple of input strings such that all the outputs of $\mathcal{A}$ are incorrect. □

However, Theorem 6 does not hold for all approximations.

Theorem 7. *Let any number $n \in \mathbb{N}$, $n \geq 1$, be arbitrarily fixed, and let $\mathcal{A}$ be any n-DFA having k states. If there is a language L over a single-letter alphabet Σ which is $(1,n)$-recognized by the n-DFA $\mathcal{A}$ then there exists a language $L' \subseteq \Sigma^*$ differing from L only in a finite number of strings such that L' can be only recognized by a deterministic finite automaton which has at least $k!$ states.*

References

[1] Ablaev, F.M., Freivalds, R.: Why sometimes probabilistic algorithms can be more effective. In: Gruska, J., Rovan, B., Wiedermann, J. (eds.) Mathematical Foundations of Computer Science 1986, Proceedings of the 12th Symposium, Bratislava, Czechoslovakia, August 25-29, 1986. Lecture Notes in Computer Science, vol. 233, pp. 1–14. Springer, Berlin (1986)

[2] Austinat, H., Diekert, V., Hertrampf, U., Petersen, H.: Regular frequency computations. Theoretical Computer Science 330(1), 15–21 (2005)

[3] Beigel, R., Gasarch, W., Kinber, E.: Frequency computation and bounded queries. Theoretical Computer Science 163(1-2), 177–192 (1996)

[4] Case, J., Kaufmann, S., Kinber, E.B., Kummer, M.: Learning recursive functions from approximations. J. Comput. Syst. Sci. 55(1), 183–196 (1997)

[5] Degtev, A.N.: On (m, n)-computable sets. In: Moldavanskij, D.I. (ed.) Algebraic Systems, pp. 88–99. Ivanovo Gos. Universitet (1981), (in Russian)

[6] Freivalds, R.: Recognition of languages with high probability of different classes of automata. Doklady Akademii Nauk SSSR 239(1), 60–62 (1978), (in Russian)

[7] Freivalds, R.: On the growth of the number of states in result of the determinization of probabilistic finite automata. Avtomatika i Vychislitel'naya Tekhnika 3, 39–42 (1982), (in Russian)

[8] Freivalds, R.: Complexity of probabilistic versus deterministic automata. In: Bārzdiņš, J., Bjørner, D. (eds.) Baltic Computer Science. Lecture Notes in Computer Science, vol. 502, pp. 565–613. Springer, Berlin (1991)

[9] Freivalds, R.: Non-constructive methods for finite probabilistic automata. International Journal of Foundations of Computer Science 19(3), 565–580 (2008)

[10] Harizanov, V., Kummer, M., Owings, J.: Frequency computations and the cardinality theorem. The Journal of Symbolic Logic 57(2) (1992)

[11] Hinrichs, M., Wechsung, G.: Time bounded frequency computations. Information and Computation 139(2), 234–257 (1997)

[12] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison–Wesley Publishing Company, Reading MA (1979)

[13] Kinber, E.B.: Frequency computations in finite automata. Cybernetics and Systems Analysis 12(2), 179–187 (1976)

[14] Kinber, E.B.: Frequency calculations of general recursive predicates and frequency enumerations of sets. Soviet Mathematics Doklady 13, 873–876 (1972)

[15] Kummer, M.: A proof of Beigel's cardinality conjecture. The Journal of Symbolic Logic 57(2), 677–681 (1992)

[16] McNaughton, R.: The theory of automata, a survey. Advances in Computers 2, 379–421 (1961)

[17] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3(2), 114–125 (1959)

[18] Rose, G.F.: An extended notion of computability. In: International Congress for Logic, Methodology and Philosophy of Science, Stanford University, Stanford, California, August 24 - September 2, 1960, Abstracts of contributed papers (1960)

[19] Tantau, T.: Towards a cardinality theorem for finite automata. In: Diks, K., Rytter, W. (eds.) Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002 Warsaw, Poland, August 26-30, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2420, pp. 625–636. Springer, Berlin (2002)

[20] Trakhtenbrot, B.A.: On the frequency computation of functions. Algebra i Logika 2(1), 25–32 (1964), (in Russian)