ISSN 0146-4116, Automatic Control and Computer Sciences, 2021, Vol. 55, No. 7, pp. 670–701. © Allerton Press, Inc., 2021.
Russian Text © The Author(s), 2020, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2020, No. 3, pp. 260–303.

Efficient Equivalence Checking Technique for Some Classes of Finite-State Machines

V. A. Zakharov* (ORCID: 0000-0002-3794-9565)

HSE University, Moscow, 101000 Russia

**e-mail: zakh@cs.msu.ru*

Received August 25, 2020; revised September 7, 2020; accepted September 9, 2020

Abstract—Finite transducers, two-tape automata, and biautomata are related computational models descended from the concept of finite-state automaton. In these models an automaton controls two heads that read or write symbols on the tapes in the one-way mode. The computations of these three types of automata show many common features, and it is surprising that the methods for analyzing the behavior of automata developed for one of these models do not find suitable utilization in other models. The goal of this paper is to develop a uniform technique for building polynomial-time equivalence checking algorithms for some classes of automata (finite transducers, two-tape automata, biautomata, and single-state pushdown automata) which exhibit certain features of the deterministic or unambiguous behavior. This new technique reduces the equivalence checking of automata to solvability checking of certain systems of equations over the semirings of languages or transductions. It turns out that such a checking can be performed by the variable elimination technique which relies on some combinatorial and algebraic properties of prefix-free regular languages. The main results obtained in this paper are as follows: 1. Using the algebraic approach a new algorithm for checking the equivalence of states of deterministic finite automata is constructed; time complexity of this algorithm is $O(n \log n)$. 2. A new class of prefix-free finite transducers is distinguished and it is shown that the developed algebraic approach provides the equivalence checking of transducers from this class in quadratic time (for real-time prefix-free transducers) and cubic time (for prefix-free transducers with ε-transitions) relative to the size of analyzed machines. 3. It is shown that the equivalence problem for deterministic two-tape finite automata can be reduced to the same problem for prefix-free finite transducers and solved in cubic time relative to the size of the analyzed machines. 4. In the same way it is proved that the equivalence problem for deterministic finite biautomata can be solved in cubic time relative to the size of analyzed machines. 5. By means of the developed approach an efficient equivalence checking algorithm for the class of simple grammars corresponding to deterministic single-state pushdown automata is constructed.

Keywords: transducer, two-tape automaton, biatomaton, simple grammar, equivalence checking, prefix-free language, language equation, decision procedure

DOI: 10.3103/S014641162107018X

INTRODUCTION

Finite transducers, two-tape finite state automata (2-FSAs), and finite biautomata are related computational models descended from the concept of finite state automaton. In these models a finite state program controls two heads that read or write symbols on the tapes in one-way mode. Finite transducers and two-tape automata compute the same class of rational binary relations on words but the computations are performed differently: transducers read words on the input tape and write on the output tape whereas 2-FSAs read words alternately on two input tapes. Both models are easily convertible into each other, and, therefore, when rational relations are concerned many authors do not distinguish them. Biautomata run on a single input tape and recognize linear context-free languages by reading input words from both ends. The relationship between finite transducers and biautomata is based on the simple fact noted in [1]: for every rational binary relation R the set of words $L = \{uv^{-1} : (u,v) \in R\}$ is a linear context-free language.

However, these models, despite the similarity of their computing capabilities, have completely different applications. When studying and applying transducers, it is customary to restrict the consideration to real-time transducers which write on the output tape only in response to reading the next symbol on the input tape. Real-time transducers are widely used in many fields as diverse as computational linguistics

and text processing [2], bioinformatics [3], verification and optimization of reactive and distributed systems [4–7]. Optimization and verification problems for information processing systems that arise in these applications can be reduced to the equivalence checking and minimization problems for finite transducers (see [7–9]).

The applications of multitape automata (m-FSAs) are not so much extensive. In [10] it was shown that optimization of string predicates in alignment declaration language used for string manipulations in database systems can be reduced to minimization problem for m-FSAs. In [11] it was noted that the equivalence problem for sequential programs which include only unary predicates of the form $p(x)$ and assignment statements of the form $x := f(x)$ is mutually reducible to the equivalence problem for deterministic multitape automata (m-DFSAs). The difficulty of solving the latter significantly hindered the development of formal methods for program optimization.

Biautomata were introduced in [12] and (independently) in [13] (in these papers they were called linear automata) as a more primitive alternative to push-down automata (PDAs) in processing context-free languages (CFLs) generated by linear grammars. Subsequently, in several publications a detailed study of the computational capabilities of biautomata [14–16] and the algorithmic aspects of their behavior analysis problems [14, 17] was carried out. Biautomata are one of the simplest models of computation capable of recognizing irregular CFLs, and it may be assumed that they will find suitable applications in syntactic analysis and text processing. And in this case, equivalence checking and minimization algorithms for this model could be very useful.

These three computational models have many common features, and it is surprising that, until now, the problems of analyzing their behavior have not been considered from a general point of view. The ultimate goal of this paper is to develop a uniform technique for designing efficient (in particular, polynomial-time) equivalence checking algorithms for some related classes of automata (finite transducers, two-tape automata, biautomata, stateless push-down automata) which exhibit deterministic or unambiguous behavior. To move the earth one needs a fulcrum. In order to find such a pivot, it is advisable to outlook the known results on the decidability of the equivalence problem for finite transducers, multitape and push-down automata and choose an approach which will suit all these automata models. We select finite transducers as a starting point due to the fact that in many papers the equivalence problem was studied in much detail earlier for various classes of transducers.

The study of the equivalence problem for transducers (they are also called finite state machines) began in the early 60s. It can be divided into three stages. Initially, Fisher and Rozenberg [18] proved that this problem is undecidable for finite transducers by reducing to it the post correspondence problem. Soon afterward Griffiths [19] managed to extend this result to the case of real-time transducers, and later Ibarra [20] showed that undecidability of the equivalence problem is retained even when transducers operate on a single-letter output alphabet.

From the proofs presented in [18–20] it is clear that undecidability takes place only when some input words may have arbitrary many images. Therefore, at the next stage the efforts were aimed at studying the equivalence problem for bounded-valued transducers. At first, it was shown that this problem is decidable for functional [21, 22], deterministic [23], and finitely ambiguous [24] transducers. Moreover, in [24] it was proved that equivalence checking can be performed in polynomial time when transducers are unambiguous and, in particular, deterministic. Polynomial time algorithms were also proposed for checking whether a real-time transducer is bounded-valued [25], or k-valued for a fixed k [24]. A transducer is called k-valued if for every input word the image has at most k output words. As for the equivalence problem for k-valued transducers, its decidability was established in [26] but the decision technique used in this paper does not yield any complexity estimates. A more advanced approach based on the decomposition of any k-valued transducer into a sum of finitely many functional transducers was developed by Weber in [27] (see also [28]). He estimated an upper bound for the length of a witness of nonequivalence of k-valued transducers and thus proved that the equivalence problem for this class of transducers is decidable in double exponential time. The decision technique proposed in [28] relies on pure combinatorial considerations; this explains such a high upper bound of its complexity.

In an effort to improve the complexity of equivalence checking procedures for bounded-valued transducers, Sakharovich and de Souza undertook a thorough revision of the approach proposed in [27, 28]. In a series of papers [29–33] they refined substantially the decomposition of a k-valued transducers and proposed a more advanced equivalence checking technique which is more oriented towards the structure of the transducers. These improvements made it possible to build a polynomial time procedure for checking k-valuedness of real-time transducers and an exponential time decision procedure for checking equivalence of k-valued transducers. A simpler straightforward technique was developed also in [34]. It does not

require preliminary decomposition of analyzed transducers, works directly with their transition relations, and allows to analyze the behavior of real-time transducers operating over semigroups. In [34] it was shown that the equivalence of k-valued transducers can be checked in exponential time when a semigroup of output elements is embeddable in such a group for which the word problem is solvable in polynomial time.

As it can be seen from the above short review on the equivalence problem for finite transducers, bounded-valued transducers were in the focus of study for a long time. The aim of the research was to delimit as precise as possible the boundary between decidable and undecidable cases of the equivalence problem. The largest among the known classes of transducers for which the equivalence problem is decidable is the class of transducers of bounded length-degree [35]. In [36] it was proved that the equivalence of transducers from this class is decidable in triple exponential time. Such a high complexity is unlikely to allow the using of decision procedures developed so far in any applications. Perhaps, it might be reasonable to change the purpose of research and look for the classes of transducers for which the equivalence problem can be solved efficiently. In [24] it was proved that the equivalence of unambiguous transducers is decidable in polynomial time. But the authors of [24] considered the unambiguity of finite transducers only relative to their reading actions. It is possible that weakening this property will allow us to distinguish a new, more extensive class of automata for which the equivalence problem will also be decidable in polynomial time.

Undecidability of the equivalence problem for nondeterministic 2-FSAs was proved in [18]. But soon afterward Bird [37] showed that this problem is decidable for 2-DFSAs. The same conclusion also follows from the results established by Valiant in [38]. An interesting feature of 2-DFSAs is that the inclusion problem for this class of automata is undecidable (see [37]). In [39] Valiant's algorithm was modified yielding an upper bound on time complexity $2^{O(n^6)}$. Later, Friedman and Greibach discovered how to check equivalence of 2-DFSAs in polynomial time. In [40] they presented two algorithms. The first one is easy for understanding but it has exponential time complexity. The second algorithm is far more sophisticated; it is built upon the ideas of the first one and checks the equivalence of 2-DFSA in time $O(n^{12})$. In the concluding remarks the authors of [40] explained how to improve this algorithm to yield an upper bound on time complexity $O(n^4)$. Finally, in 1991 the decidability of equivalence problem for deterministic automata with arbitrary number of tapes was established by Harju and Karhumaki in [41]. By using several remarkable results about ordered groups they showed how to check whether two nondeterministic multitape automata have the same number of accepting runs on each input. From the decision techniques proposed in [41] it follows that equivalence problem for n-DFSA is in **co-NP**. Strangely enough, but over past three decades no significant progress has been made in the study of this problem. Nevertheless, paper [42] should be highlighted. Its author replaced the using of group theory in the decision strategy developed in [41] with results on matrix algebras and built a simple randomized algorithm for deciding equivalence of n-DFSAs in polynomial time.

Perhaps, further progress in the study of n-DFSAs can be achieved by establishing and using the relationships between multitape automata and other computational models. Given close relationships between finite transducers and 2-FSAs, it can be assumed that that equivalence procedures developed for certain classes of finite transducers could be customized for 2-DFSA analysis.

Computing capabilities of biautomata were studied in [12, 14, 43]. Loukanova [12] proved that nondeterministic biautomata can recognize all CFLs generated by linear grammars. As for the computing power of deterministic biautomata, in [43] it was shown that it is incomparable with that of deterministic push-down automata: a deterministic biautomaton can recognize a language of palindromes which is not a deterministic CFL, but deterministic biautomata cannot recognize nonlinear CFLs. The authors of [14] conducted a thorough study of deterministic biautomata and established a number of algorithmic and closure properties for this class of automata. One of the most significant problem which remained open (see [14]) is the equivalence problem for deterministic biautomata. Although it has not yet been seriously investigated, an intensive research on this problem has been carried out for some other related computational models, such as push-down automata and CFLs.

In [44] it was proved that the equivalence problem is undecidable for context-free grammars and push-down automata. From the proof presented in this paper it follows that undecidability takes place for linear context-free grammars (and, hence, for nondeterministic biautomata) as well. But a few years later Korenjak and Hopcroft [45] discovered that the equivalence problem is decidable for simple grammars ($LL(1)$ grammars in Greibach normal form) that correspond to deterministic stateless push-down automata. The resolving algorithm presented in [45] is, in essence, a solvability checking procedure for a system of language equations built according to production rules of a grammar. It was shown that $LL(1)$-languages

possess the prefix property, and the solvability of such systems can be detected through equivalent transformations. The authors of [45] showed that these equivalent transformations can be performed in such a way that the length of each equation in a system is no more than exponential of the size of a grammar; this brings double-exponential time complexity of the checking algorithm. It is also worth noting that, as in the case of multitape automata, the inclusion problem for simple grammars is algorithmically unsolvable (see [46]). Caucal in [47] overcame the effect of double-exponential explosion. He followed the same approach as that proposed in [45] but used a bit more involved properties of prefix-free languages in equivalent transformations of systems of language equations. This allowed him to check the solvability of such systems by means of variable elimination technique and reduce the time complexity of decision procedure to single exponential. The appearance of the exponential factor is due to the fact that the length of the shortest word in a CFL can exponentially depend on the size of the grammar. Bastien, Czyzowicz, et al. improved the complexity of Caucal's algorithm in [48] by using polynomial time algorithm for comparing pairs of words represented by straight-line programs (context-free grammars generating single words) and obtained thus a polynomial time equivalence checking algorithm for simple grammars (earlier, a polynomial algorithm of higher complexity was proposed in [49]).

The above review leads to several important conclusions.

With a few exceptions, all works devoted to the study of the equivalence problem for the above-mentioned automata models of computations are of a fragmentary nature: the authors distinguished a certain class of automata and tried to construct an equivalence checking algorithm that essentially uses some characteristic features of computation of automata of this class. Despite the well-known and quite obvious relationships of computations of automata from different classes, attempts have been made very rarely to transfer the results and adapt the methods for solving the equivalence problem obtained for some classes of automata to other classes of computation models. In this regard, a question arises about the possibility of constructing such a method for checking the equivalence of automata models of computations, which, on the one hand, would have a certain universality and would be equally applicable to automata of different types, and, on the other hand, would allow one to obtain efficient algorithms for solving the equivalence problem by taking into account the specific features of the computations of some classes of automata. In most papers, the authors pursued the quite natural goal of distinguishing the largest possible classes of automata with a decidable equivalence problem. However, as the "solvability range" of this problem expanded, the equivalence checking algorithms for automata became more complicated and less practical. Push-down automata are a good example. The decidability of the equivalence problem for relatively narrow classes of DPDAs was established in [38, 45, 50]; the proposed algorithms had no less than exponential time complexity. Later, equivalence checking algorithms that are polynomial in time or have a complexity close to this estimate were constructed for these classes of automata (see [48, 51, 52]). At the same time, many researchers [53–59] have step by step expanded the "solvability domain" of the equivalence problem for DPDAs. The techniques for constructing decision procedures became more and more sophisticated, and, finally, Senizergues [60] was able to prove the decidability of the equivalence problem for arbitrary DPDAs. However, the proposed decision procedure turned out to be extremely hard for understanding and completely impractical: the only thing that we managed to find out about its complexity is that it can be estimated by a primitive recursive function [61]. This example shows that often, as the field of application of resolving algorithms expands, their applied value decreases significantly. Thus, it is desirable to develop such an approach to designing equivalence checking algorithms for various types of automata, which would ensure the high efficiency of the resolving procedures constructed with its help, while meeting certain requirements that are imposed on the structure of the analyzed automata. As can be seen from [62], this approach can lead to rather interesting practical results.

To achieve this goal, one needs to choose the optimal approach to the construction of resolving algorithms. Among the variety of methods for checking the equivalence of programs, three approaches can be distinguished as the most popular: the estimating of the length of a counterexample, the analysis of joint computations, and the checking of the solvability of systems of equations. In the first of them, the length of the shortest counterexample is estimated - the input data on which nonequivalent programs of a given size or structure produce different results. If such an estimate is available, then to check the equivalence of two programs (automata), it is sufficient to compare the results of their computations on all input data, the size of which does not exceed this estimate. This method was successfully used in [21, 27, 28, 41]. The disadvantage of this technique is that it requires an exhaustive search in a large amount of input data, and it cannot be used to build fast verification algorithms. When joint computations are analyzed, a transition system is constructed for a pair of programs (automata) under consideration; the computations in such a transition system simulate all possible pairs of computations of these programs on the same input data. In the transition system constructed thus those states of joint computations are regarded accepting which

clearly indicate, whenever they are reached, that these programs are capable of producing different results on the same inputs. Thus, the equivalence checking problem for programs is reduced to the emptiness checking problem in the class of transition systems that represent the joint computations of these programs.

The joint computation technique has found application in many works [33, 34, 36–38, 40, 51, 55, 57, 59]. The fundamental difficulty in using it is the choice of a suitable transition system for modeling joint computations of two programs: such systems must be sufficiently detailed to match the semantics of the analyzed programs and to distinguish the results of their computations, and, at the same time, they must be simple enough for effective solutions to the emptiness problem. For example, the joint computation of two DPDAs can be very easily simulated using an automaton with two stacks, but the emptiness problem for this computation model is undecidable. One of Valiant's achievements is that in [38] he was able to distinguish a class of DPDAs whose joint computations can be simulated using automata with a single push-down store, and, thus, to obtain a solution to the equivalence problem for DPDAs with a finite number of turns. Paper [63] is devoted to the study of the necessary and sufficient conditions that must be satisfied by transition systems for their successful application in the method of joint computation.

In the algebraic approach to the equivalence checking the computations of any program (automaton) are described by means of a system of equations: a variable is assigned to each point (control state) of the program, and the equations of the system specify how the results computed by the program in some states depend on the data that were computed by the program in the neighboring states. If we add to such a system an equation that declares equal two such variables that correspond to the outputs of these programs, then we obtain a system of equations, which has a solution iff the analyzed programs are equivalent. Therefore, to check the equivalence of programs, it is enough to be able to check the solvability of the corresponding systems of equations. The advantages of this approach are obvious: when a problem is tackled algebraically one can rely on the vast arsenal of expressive means and methods of modern algebra. The algebraic method for checking equivalence was used in [45, 47, 48, 52, 58, 60, 64]. The relationship between the equivalence problem for automata and that of checking the solvability of equations over sets of words was investigated in [65].

Arbitrary systems of language equations are hard for solution – they can be used to specify any recursively enumerated languages. But, as shown in [66], even systems of linear language equations in the general case are undecidable. Here, the properties of the languages used as coefficients in the equations are of decisive importance. One of these useful properties is the prefix property of languages – a language L is called prefix-free if none of the words w in L is a prefix of other words of this language. Relying precisely on this property of languages, the authors of [45] managed to prove the decidability of the equivalence problem for simple context-free languages. In subsequent papers [47, 48] devoted to the study of this problem, the prefix property also played a decisive role. The importance of this property for the efficient solution of language equations was noted in [65]: the solvability of the equivalence problem for finite prefix substitutions in given CFLs was proved, as well as the equivalence problem for one class of nondeterministic transducers. Using the prefix property, Senizergues [52] was able to show that the equivalence problem for DPDAs with a finite number of turns belongs to the complexity class **co-NP**. These results certify that in order to construct efficient equivalence checking algorithms it is advisable to focus on the classes of automata, the structure of which is closely related to prefix-free languages.

The purpose of this paper is to distinguish such classes of automata (transducers, multitape automata, biautomata, and push-down automata), for which an effective solution of the equivalence problem can be obtained using the algebraic method, based on the prefix properties of regular languages.

The main contributions of this paper are as follows.

1. A new equivalence checking technique for machines of various types is developed. In this method, the problem of checking the equivalence of automata is reduced to the problem of checking the solvability of systems of linear language equations, and this problem is solved using traditional algebraic techniques with an essential use of the particular properties of prefix-free regular languages.

2. Using the developed algebraic technique, a new equivalence checking algorithm for deterministic finite state automata is constructed, which has time complexity $O(n \log n)$.

3. A new class of prefix-free finite transducers is distinguished and it is shown that the equivalence checking of transducers from this class can be performed by a new method in quadratic time for real-time prefix transducers) and in cubic time for general prefix transducers.

4. It is shown that the equivalence problem for deterministic two-tape finite automata can be reduced to the equivalence problem for prefix-free finite transducers and can be solved in cubic time.

5. The decidability of the equivalence problem for deterministic finite biautomata in cubic time is established.

In this paper it is also shown that the proposed technique for checking the solvability of systems of language equations can also be used to analyze systems of nonlinear equations and, with its help, we obtain simple equivalence checking algorithms for some classes of PDAs and context-free grammars, in particular, for $LL(1)$ grammars and their corresponding stateless DPDAs.

The paper is organized as follows. Section 1 contains a minimal set of basic concepts from the theory of formal languages that are common to all subsequent sections. In Section 2, using the simplest computing model of deterministic finite state automata as an example, we describe the main principles of the algebraic method for checking the equivalence of automata models of computations this paper deals with. Application of the described method for more complex types of automata, which are considered in this paper, requires the use of some properties of regular prefix-free languages. Therefore, Section 3 is devoted to prefix-free languages: it is shown that the class of prefix-free languages is closed with respect to certain language-theoretic operations (concatenation, iteration, left and right quotients), automata-based methods for describing prefix-free languages are considered, and the complexity of performing these operations on prefix-free languages is established. Section 4 introduces the basic concepts of the theory of finite transducers (they are also called generalized finite state machines). Section 5 describes a new class of real-time transducers – prefix-free transducers – and shows how the equivalence problem for transducers from this class can be reduced to the solvability checking problem for systems of linear language equations in which the coefficients and free terms are finite words that form finite prefix-free languages. In the same section a quadratic time algorithm for checking the solvability of these systems of language equations is presented; it is based on the Gaussian variable elimination method, which is used in linear algebra to solve systems of linear equations. This algorithm relies on the properties of the prefix-free languages that were established in Section 3. In Section 6 a more general class of nondeterministic prefix-free homogeneous transducers is studied; these transducers allow ε-transitions without reading the input letters. In linear equations specifying the behavior of transducers from this class, coefficients and free terms are regular prefix-free languages, which are represented by deterministic finite state automata. The solvability checking of such systems of equations, based on the variable elimination method, is somewhat more complicated. It also uses the specific properties of prefix-free languages, established in Section 3, and makes it possible to check the equivalence of prefix-free homogeneous transducers with ε-transitions in cubic time. Next, in Section 7 we study the equivalence problem for deterministic two-tape automata. It is shown that the class of binary relations (transductions) recognized by automata from this class coincides with the class of binary relations computed by prefix-free homogeneous transducers with ε-transitions. The discovered correspondence provides a possibility to translate deterministic two-tape automata into prefix-free homogeneous transducers and, using the results of Section 6, construct a cubic time algorithm for checking the equivalence of deterministic two-tape automata. In Section 8 the method for checking the solvability of systems of linear language equations, developed in Section 3, is suitably adapted to solve the equivalence problem for deterministic finite biautomata; it is shown that this problem can be solved in a cubic time as well. The variable elimination method is applicable not only for checking the solvability of systems of linear language equations; Section 9 shows how this method can be used to effectively check the solvability of systems of language equations generated by $LL(1)$-grammars. Here, the decisive role is also played by the prefix-free property of $LL(1)$-languages. Prospects for the further development and application of the algebraic approach proposed in this paper to the designing of equivalence checking algorithms for various types of automata are discussed in the final Section 10.

1. BASIC NOTIONS

An *alphabet* is any nonempty finite set of letters $\Sigma = \{a_1, \ldots, a_k\}$. A *word* over an alphabet Σ is any finite sequence $w = a_{i_1} a_{i_2} \ldots a_{i_n}$ of letters in Σ. The empty word is denoted by ε, and the set of all words over an alphabet Σ is denoted by Σ^*. The length $|w|$ of a word w is the number of letters in w. Given a pair of words $u = a_{i_1} \ldots a_{i_n}$ and $v = a_{j_1} \ldots a_{j_m}$ we write uv for their *concatenation* which is the word $a_{i_1} \ldots a_{i_n} a_{j_1} \ldots a_{j_m}$. Given a word $w = a_{i_1} a_{i_2} \ldots a_{i_n}$ we write w^{-1} for its *reverse* $a_{i_n} \ldots a_{i_2} a_{i_1}$.

Suppose that a word w is a concatenation of words u and v, i.e. $w = uv$. Then the word u is a *prefix* of w, the word v is a *suffix* of w, and the word w is an extension of u. In the case of $w = uv$ we say that u is the *left quotient* of w with v ($u = w/v$ in symbols), and v is the *right quotient* of w with u ($v = w\backslash u$ in sym-

bols). Two words w_1 and w_2 are *compatible*, if one of them is a prefix of the other; otherwise, the words are *incompatible*.

A *language* L over an alphabet Σ is any subset of Σ^*. The *concatenation* of languages L_1 and L_2 is the language $L_1L_2 = \{uv : u \in L_1, v \in L_2\}$. If $L_1 = \varnothing$ or $L_2 = \varnothing$, then $L_1L_2 = \varnothing$. If $L_1 = \{u\}$ then we write uL_2 for the concatenation of L_1 and L_2. The *iteration of a language* L is the language $L^* = \{\varepsilon\} \cup L \cup LL \cup \ldots$. The *reverse of a language* L is the language $L^{-1} = \{w^{-1} : w \in L\}$. A *left quotient* of a language L with a word w is the language $L/w = \{u : uw \in L\}$ of all left quotients of words in L with w, and a *right quotient* of a language L with a word w is the language $L\backslash w = \{u : wu \in L\}$ of all right quotients of words in L with w. It should be noted that if no word in L has w as a suffix (prefix) then $L/w = \varnothing$ (respectively, $L\backslash w = \varnothing$).

A *transduction* over alphabets Σ and Δ is any subset T of $\Sigma^* \times \Delta^*$. The *domain of a transduction* T is the language $Dom(T) = \{u : \exists w : (u, w) \in T\}$, and the *image* of T is the language $Im(T) = \{w : \exists u : (u, w) \in T\}$. A language $Im(T, u) = \{w : (u, w) \in T\}$ is called the *image of a word* u *in a transduction* T. The *concatenation of transductions* R_1 and R_2 is the transduction $R_1R_2 = \{(u_1u_2, v_1v_2) : (u_1, v_1) \in R_1, (u_2, v_2) \in R_2\}$.

We denote by $Reg(\Sigma)$ the family of all regular languages over an alphabet Σ. Regular languages are usually specified by means of finite automata. A *deterministic finite automaton* (DFA) over an alphabet Σ is quadruple $A = \langle Q, q_0, F, \varphi \rangle$, where Q is a finite set of states, q_0 is an *initial state*, $F \subseteq Q$ is a subset of *accepting states*, and $\varphi : Q \times \Sigma \to Q$ is a partial transition function. Triples (q, x, q') such that $\varphi(q, x) = q'$ are called *transitions* of DFA A and depicted as $q \xrightarrow{x} q'$. A DFA A *accepts* a word $w = a_1a_2 \ldots a_n$, if there exists a sequence of transitions $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$, such that q_0 is the initial state and q_n is an accepting state of A. A DFA A *specifies* a regular language $L(A)$, which is the set of all those words w accepted by A. The *size* of a DFA A is the number $|Q|$ of its state. In what follows a DFA A with an initial state q will be denoted as $A(q)$.

To specify finite families of regular languages we will use deterministic finite automata with multiple sets of accepting states (multi-DFAs). A multi-DFA over an alphabet Σ is a tuple $D = \langle Q, q_0, F_1, \ldots, F_m, \varphi \rangle$, where Q, q_0, and φ have the same meanings as in DFAs, and $F_1, \ldots, F_m$ are subsets of accepting states. Such multi-DFA D represents the family of regular languages $\{L(D_1), \ldots, L(D_m)\}$, where $D_i = \langle Q, q_0, F_i, \varphi \rangle$ is an ordinary DFA for every $i, 1 \le i \le m$.

It can be naturally expected that DFAs and multi-DFAs have no useless states. A DFA (multi-DFA) A is called *trimmed* if its accepting states are reachable from every state of A. In what follows, it will always be assumed that all DFAs and multi-DFAs we deal with in this paper are trimmed.

2. ALGEBRAIC APPROACH TO THE EQUIVALENCE CHECKING OF AUTOMATA

An algebraic technique checks the equivalence of two machines (automata, programs) in two stages. At the first stage, a system of equations is constructed, which serves as an algebraic specification of the equivalence checking problem for two given machines. The choice of an algebra in which a suitable system of equations can be constructed depends, first of all, on the operational semantics of the class of machines for which the equivalence problem is studied. At the second stage, the solvability of the constructed system of equations is checked. Verification is carried out by applying equivalent transformations; these transformations either bring the system under consideration to a certain reduced form which always has a solution, or reveal the occurrence of inconsistent equations in the system, which indicates that such a system has no solutions. The applied equivalent transformations can take into account the specific features of the analyzed automata, such as determinism, unambiguity, prefix-free property, etc.

We begin with explaining the basic principles of this approach by the example of the equivalence problem for deterministic finite state automata: given a DFA A and a pair of its states p and q, check the equivalence of these states of the automaton. Two states p and q of a DFA $A = \langle Q, q_0, F, \varphi \rangle$ are called equivalent, if for DFAs $A(p) = \langle Q, p, F, \varphi \rangle$ and $A(q) = \langle Q, q, F, \varphi \rangle$ which have the initial states p and q respectively, the equality $L(A(p)) = L(A(q))$ holds.

Consider a DFA $A = \langle Q, q_0, F, \varphi \rangle$ over an alphabet Σ. To check the equivalence of states p and q we build a system of equations $\mathscr{E}(A, p, q) = \mathscr{E}(A) \cup \{X_p = X_q\}$, which is divided into two parts. The subsystem $\mathscr{E}(A)$ is a specification of DFA A in algebra of regular expressions, whereas the equation $X_p = X_q$ just

expresses a conjecture of equivalence of states p and q. One may construct this system and check its solvability for any DFA, but, for the sake of simplicity, in what follows it will be assumed that DFA A is trimmed, i.e., from every its state some accepting state is reachable.

To build the system of equations $\mathscr{E}(A)$ we associate with every state $r \in Q$ some variable X_r and a constant δ_r, which equals 1 if r is an accepting state, or, otherwise, it equals 0. The equations of the system $\mathscr{E}(A)$ are composed of regular expressions (regexes). The values of regexes are languages over the alphabet Σ. Regexes are built of variables X_r associated with states $r \in Q$, and constants by means of concatenation $\cdot$ and alternation $+$; the available constants are 0, 1, and the letters of the alphabet Σ. The values of constants 0 and 1 are, respectively, the empty language $L_0 = \varnothing$ and the language $L_1 = \{\varepsilon\}$, which includes only the empty word. The value of every constant $a \in \Sigma$ is the language $L_a = \{a\}$, which includes only the one-letter word a. We will implicitly use the identities of the algebra of regular expressions: $0 \cdot H = H \cdot 0 = 0$, $0 + H = H + 0 = H$

The system of equations (1) is well known in automata theory – it is used in some algorithms which build regexes for languages recognized by DFAs (see., e. g., [67]). It is as follows:

$$\mathscr{E}(A) = \left\{ X_r = \sum_{a \in \Sigma} a \cdot H_{r,a} + \delta_r : r \in Q \right\}, \tag{1}$$

where $H_{r,a} = \begin{cases} X_{\varphi(r,a)}, & \text{if a value of the transition function } \varphi(r,a) \text{ is defined,} \\ 0, & \text{otherwise.} \end{cases}$

It was proved in the textbook [67], for every trimmed DFA A the system of equations (1) has the unique solution $\{X_r = L(A(r)) : r \in Q\}$, and all languages $L(A(r))$ are nonempty. This implies that the states p are q equivalent iff the system of equations $\mathscr{E}(A, p, q)$ has a solution. So, the problem of checking the equivalence of DFA states is reduced to the problem of checking the solvability of a system of linear equations in the algebra of regexes.

The solution of a system of linear equations $\mathscr{E}(A, p, q)$ is carried out by the Gaussian method of complete elimination of variables, traditional for linear algebra. When this method is used to check the equivalence of DFA states, the systems of equations $\mathscr{E}_t$ that arise at each iteration consist of two parts: the first part is a subsystem of basic equations for some trimmed DFA A' (this automaton can be different from the initially given DFA A), and the second part is a set of verification equations of the form $X_i = X_j$, in both parts of which are variables. A verification equation $X_i = X_j$ from the system of equations $\mathscr{E}_t$ and the variable X_i from the left side of this equation are called reduced if X_i has no other occurrences in the system. A system of equations is called reduced if all verification equations of this system are reduced. It is easy to see that any reduced system of equations has a solution.

If a system of equations $\mathscr{E}_t$ is not reduced, a variable X_i from the left-hand side of at least one verification equation $X_i = X_j$ has other occurrences in the equations of the system $\mathscr{E}_t$. Such a variable X_i will be called an active variable of this system of equations $\mathscr{E}_t$. In particular, the initial system of equations $\mathscr{E}_1 = \mathscr{E}(A, p, q)$, corresponding to the equivalence problem for the states p, q of a given DFA A is not reduced, and the variable X_p from the left-hand side of the verification equation $X_p = X_q$ is an active variable.

Checking the solvability of the system of equations $\mathscr{E}(A, p, q)$ is carried out iteratively by applying equivalent transformations. These transformations deactivate one by one the variables of the analyzed system until either a reduced system of equations is obtained, which, as noted, always has a solution, or equations of the form $X_i = 0$ or $0 = 1$ are found, which are incompatible with the rest of the equations of the system, and thus certify to the unsolvability of this system.

At the beginning of the first iteration we have $\mathscr{E}_1 = \mathscr{E}(A, p, q)$. At each iteration t, an active variable is selected in an unreduced system of equations $\mathscr{E}_t$, this variable is deactivated ("excluded"), and then (if necessary) the canonical form of the system $\mathscr{E}_{t+1}$ of equations is restored. The rules for equivalent transformations at the t-th iteration are as follows.

(1) Removal of identities. Equations of the form $X = X$ are removed from $\mathscr{E}_t$.

(2) Reducedness checking. If there are no active variables in the system $\mathscr{E}_t$, then $\mathscr{E}_t$ is a reduced system of equations. Then the procedure terminates and announces the solvability of the initial system $\mathscr{E}(A, p, q)$.

(3) Elimination of variables. If the system $\mathscr{E}_t$ includes a verification equation $X_i = X_j$, where X_i is an active variable, then in all other equations of the system $\mathscr{E}_t$ all occurrences of the variable X_i are replaced with the variable X_j by applying the substitution $\{X_i/X_j\}$.

After this step the equation $X_i = X_j$ becomes reduced, and the total number of reduced variables in the system $\mathscr{E}_t$ increases by 1. However, the application of the substitution $\{X_i/X_j\}$ to the equation of the system $\mathscr{E}_t$ may have a side effect: two basic equations of the form

$$X_j = \sum_{a\in\Sigma} a \cdot H'_{j,a} + \delta'_j,$$
$$X_j = \sum_{a\in\Sigma} a \cdot H''_{j,a} + \delta''_j,$$

with the same variable X_j in their left-hand sides appear in the system $\mathscr{E}_t$. This deviation of the system of equations from the canonical form must be repaired.

(4) Elimination of duplicate basic equations. Remove from the system $\mathscr{E}_t$ one of the two equations with the same variable X_j in their left-hand sides and add to the system $|\Sigma|$ equations $H'_{j,a} = H''_{j,a}$ for every letter $a \in \Sigma$, and also an equation $\delta'_j = \delta''_j$. It is easy to see that after applying this transformation, an equivalent system of equations will be obtained, which may contain nonstandard equations of the form $X_r = 0$ or $c_1 = c_2$, where c_1, c_2 are constants.

(5) Removal of nonstandard equations. If among the added equations there are equations of the form $X_r = 0$ or $0 = 1$, then the verification procedure terminates and announces the unsolvability of the original system of equations $\mathscr{E}(A, p, q)$. If there are no such equations, i.e. among the added equations there are nonstandard equations (identities) of the form $0 = 0$ or $1 = 1$ only, then these identities is removed from the system.

If, as a result of applying these rules, the verification procedure has not yet completed its work, then a system of equations $\mathscr{E}_{t+1}$ will be obtained, which is equivalent to the system $\mathscr{E}_t$, but has a larger number of reduced variables. After that, the verification procedure proceeds to the next iteration $t + 1$.

Proposition 1. For every trimmed DFA A and a pair of its states p, q the proposed procedure after being applied to the system of equations $\mathscr{E}_1 = \mathscr{E}(A, p, q)$ eventually terminates and gives a correct answer to the question about the solvability of the system $\mathscr{E}_1$.

Proof. After every iteration t the number of reduced variables in the system of equations $\mathscr{E}_{t+1}$ increases by at least one, and, therefore, the procedure will eventually terminate. Transformations of systems of equations $\mathscr{E}_t$ according to rules 1) and 3) –5) are equivalent transformations, and, therefore, at each iteration the system of equations is equivalent to the original system $\mathscr{E}(A, p, q)$. Since a reduced system of equations always has a solution, the conclusion about the solvability of the system of equations $\mathscr{E}(A, p, q)$, which the procedure takes at step 2), is correct. Since for a reduced DFA the system of equation $\mathscr{E}(A)$ always has the unique solution $X_r = L(A(r)) \neq \varnothing$ for every state $r \in Q$, the conclusion about the unsolvability of the system of equations $\mathscr{E}(A, p, q)$, which the procedure takes at step 5), is correct.

Thus, we arrive at

Theorem 1. *The equivalence problem for reduced DFAs is decidable in time $O(n \log n)$, where n is the size of a DFA.*

Proof. By Proposition 1, the decision procedure presented above correctly checks the equivalence of a pair of states of any DFA A. The number of iterations of this procedure does not exceed the number of states of this automaton. To reduce computational overhead, data processed by the procedure for checking the solvability of systems of equations $\mathscr{E}(A, p, q)$ are represented by reference structures. For example, the set of occurrences of each variable in the regexes of a system of equations $\mathscr{E}(A, p, q)$ can be represented by a weakly balanced tree: the root of the tree represents the name of the variable X_r, and leaf nodes are used to refer to all occurrences of the variable X_r in the regexes of the system of equations. Two basic operations performed by the solvability checking procedure are the comparing of variable names and the substitution of one variable for all occurrences of another; these operations can be performed in time linear of the height of the trees corresponding to these variables.

A substitution X/Y can be applied in two ways:

1. The root of the tree corresponding to one of these variables is appended to one of the leaf nodes closest to the root of the tree representing the other variable, and the resulting tree is named X.

2. The roots of both trees corresponding to these variables X and Y are appended to a new root, and the resulting tree is named X.

Since the total number of occurrences of all variables in the equations of the system $\mathscr{E}(A, p, q)$ does not exceed $|\Sigma| n$, such an implementation of substitution operation makes it possible to maintain all trees corresponding to the variables so that their heights do not exceed the value $O(\log n)$. Thus, each iteration of procedure for checking the solvability of the system of equations $\mathscr{E}(A, p, q)$ can be performed in a time which is proportional to the logarithm of the size of DFA A.

Although Proposition 1 refers to the application of the solvability checking procedure only to systems of equations generated by trimmed DFAs, this limitation is not very much essential. To check the solvability of systems of equations $\mathscr{E}(A, p, q)$ for arbitrary DFAs, it is sufficient to equip the procedure described above with two additional rules of equivalent transformations:

(6) Substitution of constant 0. If a system $\mathscr{E}_t$ includes an equation $X_i = 0$, then in all other equations of the system $\mathscr{E}_t$ all occurrences of the variable X_i are replaced with constant 0 by applying the substitution $\{X_i/0\}$.

(7) Propagation of 0. If a system $\mathscr{E}_t$ includes an equation $0 = a_{i_1} \cdot X_{j_1} + \cdots + a_{i_k} \cdot X_{j_k}$, then this equation is removed from the system, and instead of it, the equations $X_{j_1} = 0, \ldots, X_{j_k} = 0$ are added to the system.

The most important rule for the successful application of the described technique for checking the solvability of systems of equations is the rule for eliminating duplicate equations (rule 4), which restores the canonical form of the system after applying the substitution. Rules of this type largely depend on the specific features of the automata under consideration. In the case of finite state automata, such a feature is the determinism of the automaton. Thanks to this property an equation

$$a_{i_1} \cdot X_{j_1} + \cdots + a_{i_k} \cdot X_{j_k} = a_{i_1} \cdot Y_{j_1} + \cdots + a_{i_k} \cdot Y_{j_k}$$

can be decomposed into a series of more simple equations $X_{j_1} = Y_{j_1}, \ldots, X_{j_k} = Y_{j_k}$, the left-hand sides of which are variables, and thus the canonical form of a system of equation is restored. For example, for equations describing a nondeterministic finite state automaton, such a decomposition is impossible. Therefore, the elimination of variables is applicable to check the solvability of systems of linear language equations that are generated only by such automata (machines, programs) that have a certain "unambiguity" of their behavior. In particular, this feature of the behavior of automata displays itself as prefix property of those languages that appear in the systems of equations specifying the computations of automata. We study this property in more detail in the next section of the paper.

3. PREFIX-FREE LANGUAGES AND THEIR BASIC PROPERTIES

Our method for checking the equivalence of finite state machines essentially relies on certain basic properties of fully incompatible prefix-free regular languages. A language L is called prefix-free if all its words are pairwise incompatible. By this definition, the empty language $\varnothing$ is prefix-free, and $L = \{\varepsilon\}$ is the only prefix-free language which includes the empty word ε. Two languages L_1 and L_2 are called compatible if every word in any of them is compatible with some word in the other. We say that a language L_1 is extension-free from a language L_2 if no word in L_2 is an extension of any word in L_1. Languages L_1 and L_2 are called fully incompatible if they are extension-free from each other, i.e. if every two words $w_1 \in L_1$ and $w_2 \in L_2$ are incompatible.

Propositions below follow immediately from the definitions of incompatibility, prefix-free and extension-free properties of languages. These propositions, although very simple, are the keystones of decision procedures presented in Sections 5, 6, and 8.

Proposition 2. If a language L_1 is extension-free from a language L_2 then for every word w and a language L

- L_1 is extension-free from L_2/w,
- $L_1\backslash w$ is extension-free from $L_2\backslash w$, and
- $L_1 L$ is extension-free from L_2.

Proposition 3. If languages L_1 and L_2 are extension-free from both languages L_3 and L_4, then $L_1 \cup L_2$ is extension-free from $L_3 \cup L_4$.

Proposition 4. If languages L_1 and L_2 are fully incompatible then for any language L the languages L_1L and L_2 are fully incompatible as well.

Proposition 5. For any prefix-free languages L_1 and L_2, and a word w the languages L_1L_2 and $L_1\backslash w$ are prefix-free.

Proposition 6. For any pair of fully incompatible prefix-free languages L_1 and L_2 the language $L_1 \cup L_2$ is prefix-free.

Proposition 7. If a language L_1 is prefix-free and a language L_2 is extension-free from a language L_3, then the language L_1L_2 is extension-free from the language L_1L_3.

Proposition 8. For any pair of fully incompatible prefix-free languages L_1 and L_2 the language $L_1^*L_2$ is prefix-free.

Proof. Suppose that L_1 and L_2 are fully incompatible prefix-free languages.

1. If $\varepsilon \in L_1$, then $L_2 = \varnothing$, since the empty word ε is compatible with any word. Hence, the language $L_1^*L_2 = \varnothing$ is prefix-free.

2. Consider the case of $\varepsilon \notin L_1$. Assume that $L_1^*L_2$ is not prefix-free, and let $w' = u_1' \ldots u_k'v'$, where $\{u_1', \ldots, u_k'\} \subseteq L_1, v' \in L_2$, be the shortest word in $L_1^*L_2$, which is compatible with some other word $w'' = u_1'' \ldots u_m''v''$ from the set $L_1^*L_2$, where $\{u_1'', \ldots, u_m''\} \subseteq L_1, v'' \in L_2$.

If $k > 0$, then a nonempty word u_1' is compatible either with v'' (in the case of $m = 0$), or with u_1'' (in the case of $m > 0$). The former is impossible, since L_1 and L_2 are fully incompatible languages. As far as the language L_1 is prefix-free, the latter is possible only when $u_1' = u_1''$. But this implies that shorter words $u_2' \ldots u_k'v'$ and $u_2'' \ldots u_m''v''$ are also compatible contrary to the choice of w'.

If $k = 0$, then v' is compatible either with u_1'' (in the case of $m > 0$), or with v'' (in the case of $m = 0$). The former is impossible, since L_1 and L_2 are fully incompatible languages. As far as the language L_2 is prefix-free, the latter is possible only when $v' = v''$. This means that $w' = w''$ which contradicts to the choice of w'.

The contradictions obtained bring us to the conclusion that the language $L_1^*L_2$ is prefix-free.

Proposition 9. If a prefix-free language L_1 is fully incompatible with a language L_2, and both languages L_1 and L_2 are extension-free from a language L_3, then the language $L_1^*L_2$ is extension-free from $L_1^*L_3$.

The proof of Proposition 9 follows the same line of reductio ad absurdum reasoning as that of Proposition 8.

Thus, some language-theoretic operations such as union, concatenation, left and right quotients preserve such properties as incompatibility, prefix-free and extension-free properties.

Proposition 10. Let L' and L'' be finite nonempty prefix-free compatible languages. Then there exist unique partitions $L' = \bigcup_{i=1}^{n} L_i'$ and $L'' = \bigcup_{i=1}^{n} L_i''$ of these languages such that for every $i, 1 \le i \le n$, one of the subsets L_i' or L_i'' is a singleton $\{w\}$, and all words from the other are extensions of w.

Proof. By induction on the cardinality of $L' \cup L''$.The base case $|L'| = 1$ or $|L''| = 1$ is obvious. As for the induction step, consider any shortest word u in the set $L' \cup L''$. Suppose that $w \in L'$. Then we take the singleton $\{w\}$ as L_1', and all words in L'' which are compatible with w as L_1''. Since w is one of the shortest words in $L' \cup L''$, both L' and L'' are compatible languages, and the set of words L_1'' is nonempty and includes only the extensions of w. Since L' is a prefix-free language, no word from L_1'' is an extension or a prefix of any word from $L' - \{w\}$. Thus, a pair of subsets L_1' and L_1'' is as required in Proposition 10 and such that $w \in L_1'$ is uniquely determined. The languages $L' - \{w\}$ and $L'' - L_1''$ are nonempty, prefix-free and compatible; by the induction hypothesis, they have the unique partitions that meet the requirements of Proposition 10.

Such partitioning of a compatible pair of prefix-free languages $L' = \bigcup_{i=1}^{n} L'_i$ and $L'' = \bigcup_{i=1}^{n} L''_i$ will be called the splitting of L' and L''. The pairs of corresponding subsets L'_i and L''_i, $1 \le i \le n$, of a splitting will be called its fractions. Proposition 10 will be in demand in the construction of an equivalence checking algorithm for real-time transducers in Section 5.

We denote by $PFReg(\Sigma)$ a subfamily of all prefix-free regular languages. DFAs that specify prefix-free languages have an important characteristic property which facilitates manipulations with such automata.

Proposition 11. ([68, 69]) Let $A = \langle Q, q_0, F, \varphi \rangle$ be a trimmed DFA over an alphabet Σ. Then the regular language $L(A)$ is prefix-free language iff $\varphi(q, x)$ is undefined for every accepting state q in F and every letter x in Σ, i.e. the accepting states of A do not have outgoing transitions..

In view of Proposition 11, if $L(A)$ is a prefix-free language then all accepting states of A can be safely merged into a single accepting state f which has no outgoing transitions. An automaton $A = \langle Q, q_0, \{f\}, \varphi \rangle$ of this kind will be called prefix-free DFA. Analogously, each family of regular languages $\mathcal{F} = \{L_1, \ldots, L_m, L_{m+1}\}$, such that $L_1, \ldots, L_m$ are pairwise fully incompatible prefix-free languages that are extension-free from L_{m+1}, can be represented by a multi-DFA $D = \langle Q, q_0, \{f_1\}, \ldots, \{f_m\}, F_{m+1}, \varphi \rangle$ in which $f_1, \ldots, f_m$ have no outgoing transitions. Such a multi-DFA which represents $\mathcal{F}$ will be called prefix-free multi-DFA.

Decision procedures presented in Sections 5, 6, and 8 mainly apply five operations to regular languages: right quotient $L\backslash w$, left quotient L/a, union $L_1 \cup L_2$, concatenation $L_1 L_2$, and combined iteration $L_1^* L_2$. Prefix-free DFA and multi-DFA representations of languages give certain advantages when performing these operations.

Given a word $w = x_1 \ldots x_n$ and a multi-DFA $D = \langle Q, q_0, F_1, \ldots, F_m, \varphi \rangle$ which represents a family of regular languages $\{L_1, \ldots, L_m\}$, it suffices only to find a w-successor q' of the initial state q_0 (i.e. such a state q' for which there exists a sequence of transitions $q_0 \xrightarrow{x_1} \cdots \xrightarrow{x_n} q'$) to build a multi-DFA representation $D' = \langle Q, q', F_1, \ldots, F_m, \varphi \rangle$ of the family $\{L_1\backslash w, \ldots, L_m\backslash w\}$.

Given a letter $a \in \Sigma$ and DFA $A = \langle Q, q_0, F, \varphi \rangle$ it is easy to find the subset of all a-predecessors $F' = \{q' : q' \xrightarrow{a} q, q \in F\}$ of the accepting states and build a DFA representation $A' = \langle Q, q_0, F', \varphi \rangle$ for the left quotient $L(A)/a$.

If a multi-DFA $D = \langle Q, q_0, F_1, \ldots, F_m, \varphi \rangle$ represents a pair of regular languages L_1 and L_2, then their union $L_1 \cup L_2$ is specified by a DFA $B' = \langle Q, q_0, F_1 \cup F_2, \varphi \rangle$.

By merging the accepting state of a prefix-free DFA which represents a language L_0 and the start state a multi-DFA which represents a family of languages $L_1, \ldots, L_m$ we construct the representation for the family of concatenations $\{L_0 L_1, \ldots, L_0 L_m\}$.

Finally, if there is a prefix-free multi-DFA $D = \langle Q, q_0, \{f_1\}, \ldots, \{f_m\}, F, \varphi \rangle$, then by 1) adding a copy q'_0 of the initial state q_0 (along with the outgoing transitions), 2) redirecting to q'_0 all transitions leading to the state q_0, and 3) merging the initial state q_0 and the accepting state f_1, we build a multi-DFA $D' = \left\langle Q \cup \{q'_0\}, q_0, \{f_2\}, \ldots, \{f_m\}, F, \varphi' \right\rangle$, which represents the family of languages $L_1^* L_2, \ldots, L_1^* L_m, L_1^* L_{m+1}$.

All these operations on languages can be performed in time linear of the size of given DFA specifications and, moreover, the size of the resulting automata increases by no more than 1.

4. FINITE TRANSDUCERS

A finite transducer (briefly, transducer) over a finite input alphabet Σ and a finite output alphabet Δ is a triple $\pi = \langle Q, F, \to \rangle$, where Q is a finite set of states, $F \subseteq Q$ is a subset of final states, and $\to$ is a finite transition system of the type $Q \times (\Sigma \cup \{\varepsilon\}) \times \Delta^* \times Q$. Sometimes we will write $\pi(q_0)$ to emphasize that a state q_0 is distinguished in Q as the initial state of π. Quadruples (q, x, u, q') in $\to$ are called transitions and depicted as $q \xrightarrow{x|u} q'$. If x is a letter in Σ then a transition $q \xrightarrow{x|u} q'$ will be called Σ-transition. Transitions of the form $q \xrightarrow{\varepsilon|u} q'$ are called ε-transitions. It is easy to see that Σ-transition $q \xrightarrow{a|u} q'$

means that a transducer when being at the state q, and reading the letter a on the input tape writes the word u on the output tape, and passes to the state q'. When a transducer fires a ε-transition $q \xrightarrow{\varepsilon|u} q'$ it does not access the input tape, writes the word u on the output tape, and passes to the state q'. By the size of a transition $q \xrightarrow{x|u} q'$ we mean the number $|u|+1$. The size $|\pi|$ of a transducer π is the sum of the sizes of the transitions of π plus the number of states $|Q|$.

A run of a transducer π from a state q to a state q' on an input word h is any finite sequence of transitions

$$q \xrightarrow{x_1|u_1} q_1 \xrightarrow{x_2|u_2} \cdots \xrightarrow{x_{n-1}|u_{n-1}} q_{n-1} \xrightarrow{x_n|u_n} q', \tag{2}$$

such that $h = x_1 x_2 \ldots x_n$. The pair of words (h,u), where $u = u_1 u_2 \ldots u_n$, is called a label of a run (2). We will write $q \xrightarrow{h|u}_* q'$ to indicate that a transducer π has a run (2) labelled with (h,u) from a state q to a state q'. If $q' \in F$, then a run (2) is called final. A run (2) of a transducer π labelled with (ε,u) is called *on*-run. For each state q there exists an empty run $q \xrightarrow{\varepsilon|\varepsilon}_* q$ from a state q to itself which is the empty sequence of transitions.

The set of pairs of words $TR(\pi,q) = \{(h,u) : q \xrightarrow{h|u}_* q', q' \in F\}$, which are the labels of all final runs of a transducer π from a state q, is called a transduction relation computed by the transducer π in the state q. It should be noted that when q is a final state of a transducer π, the transduction relation $TR(\pi,q)$ includes the label $(\varepsilon,\varepsilon)$. States q_1 and q_2 of a transducer π are called equivalent ($\pi(q_1) \sim \pi(q_2)$ in symbols), if an equality $TR(\pi,q_1) = TR(\pi,q_2)$ holds. The equivalence problem for transducers is that of checking, given a transducer π and a pair of its states q_1 and q_2, whether $\pi(q_1) \sim \pi(q_2)$ holds.

A transducer π is called real-time if it does not have ε-transitions, i.e. $\rightarrow \subseteq Q \times \Sigma \times \Delta^* \times Q$. Unlike the case of finite state automata, in some transducers one can not get rid of ε-transition, and, therefore, real-time transducers have significantly less computational power than transducers with ε-transitions.

A real-time transducer π is called

• deterministic, if for every input letter a and a state q it has at most one transition of the form $q \xrightarrow{a|u} q'$;

• functional (or, single-valued), if for every state q the transduction relation $TR(\pi,q)$ is a function;

• k- ambiguous, if for every state q and an input word h there is at most k final runs of π from the state q on the word h;

• k-valued, if for every state q and an input word h the transduction relation $TR(\pi,q)$ contains at most k images of h, i.e. $|Im(TR(\pi,q),h)| \le k$;

• of length-degree k if for every state q and an input word h, the set of distinct lengths of the images w of the word h in the transduction relation $TR(\pi,q)$ contains at most k numbers, i.e. $|\{|w| : w \in \mathrm{Im}(TR(\pi,q),h)\}| \le k$.

The equivalence problem was shown to be decidable for these families of real-time transducers, see [21–24, 28, 35]. In the next Section we introduce a new class of real-time transducers for which this problem is also decidable.

5. EQUIVALENCE CHECKING OF PREFIX-FREE REAL-TIME TRANSDUCERS

Let $\pi = \langle Q, q_0, F, \rightarrow \rangle$ be a real-time transducer. For every pair of states q, q' and an input letter x we denote by $Out_\pi(q,q',x)$ the set of all output words u such that $q \xrightarrow{x|u} q'$ is a transition of π. A real-time transducer π is called prefix-free if for every its state q, an input letter x, and a pair of different transitions $q \xrightarrow{x|u'} q'$ and $q \xrightarrow{x|u''} q''$ the output words u' and u'' are incompatible. The choice of the name ``prefix-free'' is due to the following obvious characteristic property of transducers of this kind: for every state $q \in Q$ and an input letter $x \in \Sigma$ all languages from the family $\{Out_\pi(q,x,p) : p \in Q\}$ are prefix-free and pairwise fully incompatible.

Clearly, every deterministic transducer is prefix-free but not vice verse. Prefix-free transducers have a certain ``unambiguity'' property: for every state q of a prefix-free transducer π and a pair $(h,u) \in TR(\pi,q)$ there is the only run of π from the state q labelled with (h,u). However, one have to keep in mind that this property differs significantly from the unambiguity property of transducers in its traditional sense (see [24,

25, 27, 28, 31]): the latter requires that for every input word h there exists at most one run of a transducer π labelled with (h, w) for some output word w. It should be noticed also that some prefix-free real-time transducers do not have such properties as k-ambiguity, k-valuedness, or even bounded length-degree, i.e. the class of prefix-free transducers is not subsumed by any of previously known decidable cases of equivalence problem for real-time finite transducers.

Example 1. Consider a prefix-free real-time transducer $\pi = \langle\{q_0, q_1\}, \{q_1\}, \to\rangle$, which operates over $\Sigma = \{a, b\}$ и $\Delta = \{c, d\}$ and has three transitions $q_0 \xrightarrow{a/c} q_0$, $q_0 \xrightarrow{a/dd} q_0$, and $q_0 \xrightarrow{b/\varepsilon} q_1$. For every input word $h_n = a^n b$ the lengths of outputs in the runs of π from the state q_0 vary from n to $2n$. Hence, the transducer π is not a transducer of bounded length-degree.

Our equivalence checking technique for prefix-free real-time transducers is based on manipulations with regular expressions of some special kind. In the same way as it was done for DFAs in Section 1, we associate with each transducer π a system of linear equations E_π, which specifies the transductions realized at all states of π. To check the equivalence $\pi(q') \sim \pi(q'')$ we add to the set of equations $\mathscr{E}_\pi$ the equivalence requirement which is an equation of the form $X' = X''$, and then verify whether the resulting system of equations has a solution. To this end we use Gaussian elimination of variables. In the case of an arbitrary transducer the advantages of this approach are doubtful, but for prefix-free transducers the splitting effect of compatible prefix-free languages (see Proposition 10) provides a suitable means for solving efficiently the systems of equations that correspond to the equivalence problem.

For the sake of clarity and to make the notation simpler we will use some assumptions concerning a transducer to be analyzed:

1. An input alphabet $\Sigma = \{a_1, \ldots, a_k\}$ and an output alphabet Δ are disjoint; we will use symbols x, y, z to denote arbitrary letters from Σ, and symbols u, v, and w to denote words from Δ^*.

2. A transducer is trimmed, i.e., a final state is reachable from any state of π.

Regular expressions (regexes, for short) are built of variables $X_1, X_2, \ldots$, constants 0,1, letters from Σ, and words from Δ^* by means of multiplication $\cdot$ and addition $+$. Regexes are interpreted on the semiring of transductions. The constants 0, 1, every letter $x \in \Sigma$, and every word $u \in \Delta^*$ are understood as transductions $\varnothing$, $\{(\varepsilon, \varepsilon)\}$, $\{(x, \varepsilon)\}$ and $\{(\varepsilon, u)\}$. Multiplication $\cdot$ and addition $+$ are interpreted, respectively, as concatenation and union of transductions. Clearly, addition $+$ is commutative, and the equality $x \cdot u = u \cdot x$ holds for every letter $x \in \Sigma$ and a word $u \in \Delta^*$. We will also implicitly use identities: $0 \cdot T = T \cdot 0 = 0, 0 + T = T, 1 \cdot T = T \cdot 1 = T$.

We will focus on linear regexes of two types. A Δ-regex is either a constant 0, or any expression of the form $u_1 \cdot X_{i_1} + u_2 \cdot X_{i_2} + \ldots + u_m \cdot X_{i_m}$, where variables may have multiple occurrences. If words $u_1, u_2, \ldots, u_m$ are pairwise incompatible then such a Δ-regex is called prefix-free. A Σ-regex is any expression of the form $a_1 \cdot E_1 + a_2 \cdot E_2 + \ldots + a_k \cdot E_k + w_1 + \cdots + w_m$, where $E_i, 1 \le i \le k$, are Δ-regexes, $\Sigma = \{a_1, a_2, \ldots, a_k\}$ and $\{w_1, \ldots, w_m\} \subseteq \Sigma^*$. When referring to such regexes as $E(X_1, \ldots, X_n)$ (for Δ-regexes) or $G(X_1, \ldots, X_n)$ (for Σ-regexes), we emphasize that $X_1, \ldots, X_n$ are the only variables involved in them..

Let $\pi = \langle Q, F, \to\rangle$ be a real-time transducer over Σ and Δ. With each state p we associate a variable X_p, and for every pair $p \in Q$ and $x \in \Sigma$ we build a Δ-regex $E_{p,x} = \sum_{q \in Q} \sum_{u \in Out_\pi(p,x,q)} u \cdot X_q$. Clearly, if the transducer π is prefix-free then all Δ-regexes $E_{p,x}$ thus defined are prefix-free as well. For every state p denote by δ_p either the constant 1 if $p \in F$, or the constant 0 otherwise. Then the transducer π can be algebraically specified by the system of equations

$$\mathscr{E}_\pi = \left\{ X_p = \sum_{x \in \Sigma} x \cdot E_{p,x} + \delta_p : p \in Q \right\}.$$

Proposition 12. For every real-time transducer π the system of equations $\mathscr{E}_\pi$ has the unique solution $\{X_p = TR(\pi, p) : p \in Q\}$.

Proof. The fact that the set of transductions $TR(\pi, p)$, $p \in Q$, taken as values of corresponding variables X_p, provides a solution to the system of equations $\mathscr{E}_\pi$, follows immediately from the definitions of $TR(\pi, p)$ and equations of this system.

To prove that this solution is unique we may take an arbitrary solution $X_p = T_p$, $p \in Q$, of the system of equations $\mathscr{E}_\pi$ and show by induction on the length of an input word h, that for every pair (h, w) and a state $p \in Q$ the pair (h, w) is in T_p iff $(h, w) \in TR(\pi, p)$.

As it can be seen from the definition of the equations in $\mathscr{E}_\pi$, a pair (ε, w) appears in T_p iff $w = \varepsilon$ and $p \in F$; the latter is equivalent to $(\varepsilon, w) \in TR(\pi, p)$.

Suppose that the relationship $(h, w) \in T_p \Leftrightarrow (h, w) \in TR(\pi, p)$ holds for every state p, an output word w, and an input word h of length less than n. Consider an arbitrary input word $h' = yh$ of length $n+1$. Since T_p is a component of a solution to $\mathscr{E}_\pi$, the equality $T_p = \sum_{x\in\Sigma}\sum_{q\in Q}\sum_{u\in Out_\pi(p,x,q)} x \cdot u \cdot T_q + c_p$ holds. Therefore, for every output word w the pair (h', w) is in T_p iff $(h, w\backslash u) \in T_q$ for some state q and an output word $u \in Out_\pi(p, y, q)$. By the induction hypothesis, the latter is equivalent to the fact that $(h, w\backslash u) \in TR(\pi, q)$, and π has a transition $p \xrightarrow{y|u} q$; which means that $(h, w) \in TR(\pi, p)$.

Corollary 1. For every pair of states $p, q \in Q$ the equivalence $\pi(p) \sim \pi(q)$ holds iff the system of equations $\mathscr{E}_\pi \cup \{X_p = X_q\}$ has a solution.

This corollary shows, that to verify the equivalence of real-time transducers it suffices to learn how to check the solvability of systems of equations which are certain extensions of the system $\mathscr{E}_\pi$, corresponding to transducers. The solvability of some extensions is quite obvious. We say that a system of linear equations

$$\mathscr{E} = \mathscr{E}_\pi(X_1, \ldots, X_n) \cup \{Y_j = E_j(X_1, \ldots, X_n) : 1 \le j \le m\},$$

is reduced if the sets of variables $\{X_1, \ldots, X_n\}$ and $\{Y_1, \ldots, Y_m\}$ are disjoint, and the right-hand sides $E_j, 1 \le j \le m$, are some regexes.

Proposition 13. Every reduced system of equations $\mathscr{E}$ has the unique solution.

Proof. Follows immediately from Proposition 12.

Some other extensions of the systems $\mathscr{E}_\pi$ have no solutions.

Proposition 14. If a system of equations $\mathscr{E}_\pi(X_1, \ldots, X_n) \cup \left\{\sum_{i=1}^{\ell} u_i \cdot X_i = \sum_{j=1}^{m} v_j \cdot X_j\right\}$ has a solution then the languages $L_1 = \{u_1, \ldots, u_\ell\}$ and $L_2 = \{v_1, \ldots, v_m\}$ are incompatible.

Proof. By Proposition 12, the system of equation $\mathscr{E}_\pi$ has nonzero solution $X_1 = T_1, \ldots, X_n = T_n$. If the word u_i is incompatible with the words $v_1, \ldots, v_m$, then $(\varepsilon, u_i)T_i \cap \bigcup_{j=1}^{m} (\varepsilon, v_j)T_j = \varnothing$, and, hence, $\sum_{i=1}^{\ell} u_i \cdot T_i \ne \sum_{j=1}^{m} v_j \cdot T_j$

Proposition 15. If a system of equations $\mathscr{E}_\pi(X_1, \ldots, X_n) \cup \left\{X_1 = \sum_{i=1}^{\ell} u_i \cdot X_i\right\}$ has a solution, and the set of words $\{u_1, \ldots, u_\ell\}$ if prefix-free, then $\ell = 1$ and $u_1 = \varepsilon$, i.e. $\sum_{i=1}^{\ell} u_i \cdot X_i \equiv X_1$.

Proof. By Proposition 12, the system of equations $\mathscr{E}_\pi$ has the unique solution $X_1 = T_1, \ldots, X_n = T_n$ and $T_i \ne \varnothing$ for every $i, 1 \le i \le n$. Suppose that the equality $T_1 = \bigcup_{i=1}^{\ell} (\varepsilon, u_i)T_i$ holds. Consider any input word h in $Dom(T_1)$ and finite languages from $Im(T_i, h), 1 \le i \le \ell$. On the one hand,

$$T_1 = \bigcup_{i=1}^{\ell} (\varepsilon, u_i)T_i \Rightarrow Im(T_1, h) = \bigcup_{i=1}^{\ell} u_i Im(T_i, h).$$

On the other hand, since the set of words $\{u_1, \ldots, u_\ell\}$ is prefix-free, the languages $u_1 Im(T_1, h)$ and $\bigcup_{i=2}^{\ell} u_i Im(T_i, h)$ have no common words. Therefore, the cardinality of the set $\bigcup_{i=2}^{\ell} u_i Im(T_i, h)$ equals 0. However, the equality $Im(T_1, h) = u_1 Im(T_1, h)$ implies $u_1 = \varepsilon$. Since ε is compatible with any word, we conclude that $\ell = 1$.

It should be emphasized that this proof relies on the fact that π is a real-time transducer, and, hence, for every input word h transductions T_i which are the solutions of the system $\mathscr{E}_\pi$ have finitely many images of h. When transducers with ε-transitions are concerned (see Section 6), Proposition 15 is not valid.

Below we present an iterative procedure which checks the solvability of the system of equations $\mathscr{E}_1 = \mathscr{E}_\pi \cup \{X_p = X_q\}$ for a prefix-free transducer π by bringing this system to an equivalent reduced form. At the beginning of each iteration t the procedure gets at the input a system of equations of the form

$$\mathscr{E}_t = \mathscr{E}_{\pi_t} \cup \{X_{j_i} = E_i : 1 \le i \le N_t\},$$

where

- π_t is some prefix-free transducer (not necessarily a given transducer π),
- $X_{j_i}, 1 \le i \le N_t$, are variables (not necessarily pairwise different),
- $E_i, 1 \le i \le N_t$, are prefix-free nonzero Δ-regexes which depend only on those variables that occur in the subsystem $\mathscr{E}_{\pi_t}$.

Equations from the subsystem $\mathscr{E}_{\pi_t}$ will be called basic equations, and all other equations will be called verification equations. If a variable X_i on the left-hand side of a verification equation $X_i = E_i$ does not occur elsewhere in the system $\mathscr{E}_t$, then X_i will be called reduced variable, and the corresponding equation will be called reduced equation. All other verification equations and variables on the left-hand sides of these equations will be called active. Clearly, if the system of equations $\mathscr{E}_t$ does not contain active variables, then it is reduced. Our approach follows the idea of Gaussian variable elimination techniques and deactivates all variables of $\mathscr{E}_t$ one after another by applying certain equivalent transformations to $\mathscr{E}_t$. This process terminates in two cases:

- no active variables occur in $\mathscr{E}_t$; then, by Proposition 13, the system $\mathscr{E}_t$ has a solution, and this implies $\pi(p) \sim \pi(q)$;
- some active equation of the system $\mathscr{E}_t$ contradicts the conclusions of Proposition 14 or Proposition 15; then $\mathscr{E}_t$ has no solutions, and, therefore, the states p and q of the transducer π are recognized as nonequivalent.

At the tth iteration the following equivalent transformations are applied to a system of equations $\mathscr{E}_t$.

(1) Removal of identities. Equations of the form $X = X$ are removed from $\mathscr{E}_t$.

(2) Reducedness checking. If $\mathscr{E}_t$ has no active equations then the procedure terminates and announces the solvability of the system due to Proposition 13.

(3) Elimination of variables. Select an active equation $X_i = E_i$ in $\mathscr{E}_t$. If X_i is involved in the Δ-regex E_i with a nonzero coefficient then the procedure terminates and announces the unsolvability of the system due to Proposition 15. Otherwise, in all other equations of the system $\mathscr{E}_t$ all occurrences of the variable X_i are replaced with the regex E_i by means of the substitution $\{X_i/E_i\}$.

After this step the variable X_i and the equation $X_i = E_i$ become reduced, and the number of active variables in $\mathscr{E}_t$ decreases. But the application of the substitution $\{X_i/E_i\}$ has a side effect: nonstandard equations of the form

(A) $E = G$, where E is a non-variable Δ-regex, and G is Σ-regex,

(B) $E' = E''$, where E', E'' are non-variable Δ-regexes, may appear in the system $\mathscr{E}_t$. It may also happen that

(C) several basic equations of the form $X = G$ with the same variable X on their left-hand sides arise in $\mathscr{E}_t$. These deviations of the system of equations from the established canonical form should be repaired.

(4) Removal of nonstandard equations $E = G$. To this end

- for every equation of the form $E(X_1, \ldots, X_\ell) = G$ replace all the occurrences of variables $X_i, 1 \le i \le \ell$, in the nonzero Δ-regex E with Σ-regexes G_i from the right-hand sides of basic equations $X_i = G_i$ in the subsystem E_{π_t}, then bring the resulting expression $E(G_1, \ldots, G_\ell)$ to the standard form of Σ- regex using commutativity law $u \cdot x = x \cdot u$ for letters in Σ and Δ, and distributivity law $T \cdot R + T \cdot S = T \cdot (R + S)$;
- for every pair of basic equations $X = G'$ and $X = G''$ with the same left-hand side replace one of these equations with the equation $G' = G''$.

After this step all equations of the form $E = G$ disappear and all basic equations $X = G$ will have pairwise different left-hand side variables. But this is achieved by inserting to the system nonstandard equations of the form

(D) $G' = G''$, where G', G'' are Σ-regexes.

(5) Removal of nonstandard equations $G' = G''$. Consider an equation

$$\sum_{i=1}^{k} a_i \cdot E_i' + w_1' + \cdots + w_m' = \sum_{i=1}^{k} a_i \cdot E_i'' + w_1'' + \cdots + w_n''.$$

If $\{w_1', \ldots, w_m'\} \neq \{w_1'', \ldots, w_n''\}$, then the procedure terminates and announces the unsolvability of the system. The same decision is made when for some $i, 1 \leq i \leq k$, one of the Δ-regexes E_i' or E_i'' is 0, whereas the other is a nonzero expression. It is easy to see that in both cases the left and the right sides of this equation evaluate differently for any nonempty transductions taken as values of variables.

Otherwise, remove this equation from the system and insert instead of it the equations $E_i' = E_i'', 1 \leq i \leq k$, where E_i' and E_i'' are nonzero Δ-regexes. Thus, all equations of the form $G' = G''$ disappear from the system due to the introduction of new equations of the form $E' = E''$. After this step equations of the form (B) are the only nonstandard equations that remain in the system.

(6) Removal of nonstandard equations $E' = E''$. The decisive importance here is that

Δ-regexes E' and E'' are prefix-free. This is because the initial system of equations $\mathscr{E}_1$ stems from a prefix-free transducer π, and all transformations to which the system of equations is subjected retain, as it follows from Propositions 4-7, the prefix-free property of Δ-regexes occurred in the equations. To remove a nonstandard equation of the form (B)

$$\sum_{i=1}^{\ell} u_i \cdot X_i' = \sum_{j=1}^{m} v_j \cdot X_j'' \tag{3}$$

check the compatiblity of prefix-free languages $L' = \{u_1, \ldots, u_\ell\}$ and $L'' = \{v_1, \ldots, v_m\}$. If some word in one of these languages is incompatible with every word in the other language then the procedure terminates and announces the unsolvability of the system due to Proposition 14. Otherwise the procedure makes a splitting $L' = \bigcup_{i=1}^{n} L_i'$ and $L'' = \bigcup_{i=1}^{n} L_i''$ of these languages (see Proposition 10), removes the equation (3) from the system $\mathscr{E}_t$, and inserts for every fraction $L_i' = \{u_{i_0}\}$ and $L_i'' = \{v_{i_1}, \ldots, v_{i_r}\}$ (or $L_i' = \{u_{i_1}, \ldots, u_{i_r}\}$ and $L_i'' = \{v_{i_0}\}$) a new equation $X_{i_0}' = (v_{i_1} \backslash u_{i_0}) \cdot X_{i_1}'' + \ldots + (v_{i_r} \backslash u_{i_0}) \cdot X_{i_r}''$ (or, respectively, $X_{i_0}'' = (u_{i_1} \backslash v_{i_0}) \cdot X_{i_1}' + \ldots + (u_{i_r} \backslash v_{i_0}) \cdot X_{i_r}'$). Proposition 5 guarantees that Δ-regexes in the right-hand sides of the inserted equations are prefix-free.

After this step we obtain the system of equations $\mathscr{E}_{t+1}$, which is equivalent to $\mathscr{E}_t$, but has a smaller number of active variables than $\mathscr{E}_t$.

It is worth paying attention to the fact that this solvability checking procedure is very similar to (and was inspired by) Martelli–Montanari algorithm [70] for computing the most general unifiers of two terms. And just as in this unification algorithm, the rules (1)–(6) for converting the system of equations to the reduced form can be applied in any order.

Proposition 16. For every prefix-free real-time transducer π and a pair of its states p, q the procedure above when being applied to the system of equations $\mathscr{E}_1 = \mathscr{E}_\pi \cup \{X_p = X_q\}$ terminates and correctly detects the solvability of $\mathscr{E}_1$.

Proof. Since at every stage t the procedure decreases the number of active variables at least by one, it eventually terminates. The transformations 1) and 3)–6) that the system $\mathscr{E}_t$ undergoes are equivalent transformations; therefore, at every stage t the system of equations $\mathscr{E}_t$ is equivalent to the initial system $\mathscr{E}_1$. Propositions 13–15 certify that the decision made by the procedure is correct for the system $\mathscr{E}_t$.

Thus, we arrive at

Theorem 2. *The equivalence problem for prefix-free real-time transducers is decidable in quadratic time.*

Proof. By Proposition 16, the decision procedure described above checks the equivalence $\pi(p) \sim \pi(q)$ for every finite prefix-free real-time transducer π. The number of stages to complete this procedure does not exceed the number of states of π. All Δ- regexes occurred in the systems of equations $\mathscr{E}_t$, can be represented as dictionary trees by means of reference data structures to avoid duplications. These data structures enable to perform all operations on the equations such as application of substitutions, compatibility

checking, splitting of equations, etc. in time linear of the size of $\mathscr{E}_t$. It is also important to keep in mind that the usage of reference structures makes it possible to implement these operations so that the size of the systems $\mathscr{E}_t$ does not increase. Since the initial system of equations $\mathscr{E}_t$ can be built in time linear of the size of π, the equivalence checking $\pi(p) \sim \pi(q)$ can be performed in time quadratic of the size of π.

6. EQUIVALENCE CHECKING OF PREFIX-FREE TRANSDUCERS WITH ε-TRANSITIONS

In this section we distinguish a new class of prefix-free transducers with ε-transitions and show how to check their equivalence by means of the same technique as was developed in Section 5 for the analysis of prefix-free real-time transducers, i.e. by checking the solvability of systems of transduction equations. However, now not separate words, but regular prefix-free languages will be the coefficients of these equations.

Given a transducer $\pi = \langle Q, F, \to \rangle$ over an input alphabet Σ and an output alphabet Δ, we say that a state q is Σ-state (ε-state), if no ε-transitions at all (respectively, only ε-transitions) outgo from q. States of both types are called homogeneous. A transducer π is called homogeneous if all its states are homogeneous. Every transducer can be easily converted into a homogeneous one: if transitions of both types outgo from a state q then it can be splitted into two states q' and q'' so that q' inherits from q only its Σ-transitions, and q'' inherits from q only its ε-transitions. Transitions that previously reached q are cloned and directed to both states q' and q''. In what follows in this section we will consider only homogeneous transducers.

The set of states Q of a homogeneous transducer can be partitioned into a subset Q_Σ of Σ-states, and a subset Q_ε of ε-states. In the same way, the set of final states F is divided into a subset F_Σ of final Σ-states, and a subset F_ε of final ε-states. Since a homogeneous transducer may have cycles on n-transitions, it can, in contrast to a real-time transducer, write arbitrary many words on its output tape in response to reading a single input letter. To characterize all possible responses of a homogeneous transducer π to the events at the beginning of a computation or reading a letter on the input tape we introduce four classes of languages:

1. For every pair of Σ-states p, q and an input letter $a \in \Sigma$ define a set of words

$$Out_\pi^\Sigma(p,a,q) = \{w : p \xrightarrow{a|w}_* q\},$$

which π outputs when running from p to q after reading a single letter a.

2. For every Σ-state p and an input letter $a \in \Sigma$ define a set of words

$$Fin_\pi^\Sigma(p,a) = \{w : p \xrightarrow{a|w}_* q, q \in F_\varepsilon\},$$

which π outputs when running from p to some ε-state after reading a single letter a.

3. For every ε-state p and a Σ-state q define a set of words

$$Out_\pi^\varepsilon(p,q) = \{w : p \xrightarrow{\varepsilon|w}_* q\},$$

which π outputs when it makes some ε-run from p to q.

4. For every ε-state p define a set of words

$$Fin_\pi^\varepsilon(p) = \{w : p \xrightarrow[*]{\varepsilon|w} q, q \in F_\varepsilon\},$$

which π outputs when making some ε-run from p to some final ε-state.

Similar to real-time transducers, we say that a homogeneous transducer π is prefix-free if for every $x \in \Sigma \cup \{\varepsilon\}$, a state q, and a pair of different transitions $q \xrightarrow{x|u'} q'$ and $q \xrightarrow{x|u''} q''$ the output words u' and u'' are incompatible. The proposition below follows immediately from this definition.

Proposition 17. Let $\pi = \langle Q_\Sigma \cup Q_\varepsilon, F_\Sigma \cup F_\varepsilon, \to \rangle$ be a prefix-free homogeneous transducer. Then,

1. For every Σ-state p and an input letter $a \in \Sigma$ all languages from the family $\{Out_\pi^\Sigma(p,a,q) : q \in Q_\Sigma\}$ are pairwise fully incompatible prefix-free regular languages which are extension-free from the regular language $Fin_\pi^\varepsilon(p,a)$.

2. For every ε-state p all languages from the family $\{Out_\pi^\varepsilon(p,q) : q \in Q_\Sigma\}$ are pairwise fully incompatible prefix-free regular languages which are extension-free from the regular language $Fin_\pi^\varepsilon(p)$.

Proof. Regarding each ε-transition $p \xrightarrow{\varepsilon|w} q$ of π, where $w = x_1 \ldots x_m$, as the sequential composition $p \xrightarrow{x_1} \cdots \xrightarrow{x_m} q$ of transitions, one could build for every ε-state p such a finite state multi-automaton which (1) operates on the output alphabet Δ, (2) has p as the initial state, and (3) has a family of sets of accepting states which includes all singletons $\{q\}, q \in Q_\Sigma$, and F_ε. The prefix-free property of π guarantees that such an automaton is a prefix-free multi-DFA which specifies the family of languages $\{Out^\varepsilon_\pi(p,q) : q \in Q_\Sigma\}$ and $Fin^\varepsilon_\pi(p)$.

Following the same reasoning, one can just as easily build for every Σ-state p of π and an input letter $a \in \Sigma$ a prefix-free multi-DFA which specifies the family of languages $\{Out^\Sigma_\pi(p,a,q) : q \in Q_\Sigma\}$ and $Fin^\varepsilon_\pi(p,a)$.

Note that the multi-DFAs described in the proof of this proposition can be built in time linear of the size of π, and the size of every such a multi-DFA does not increase the size of π. An equivalence checking algorithm for prefix-free homogeneous transducers presented below carries out the analysis and transformation of families of regular languages $Out^\Sigma_\pi(p,a,q)$, $Fin^\Sigma_\pi(p,a)$, $Out^\varepsilon_\pi(p,q)$, and $Fin^\varepsilon_\pi(p)$, using the indicated multi-DFAs as their formal specifications.

First, we show how to build a system of equations $\mathscr{E}_\pi$, which specifies transduction relations realized at Σ-states of a homogeneous transducer $\pi = \langle Q_\varepsilon \cup Q_\Sigma, F_\varepsilon \cup F_\Sigma, \to \rangle$. Equations are composed of generalized regexes which, as in the case of real-time transducers, are constructed from variables $X_1, X_2, \ldots$, constants 0,1, and letters in Σ, but now instead of letters from Δ we will use regular languages from $Reg(\Delta)$ as constants. Every such constant $L \in Reg(\Delta)$ is interpreted as a transduction $\{(\varepsilon, w) : w \in L\}$. Of course, for such new constants the equalities $L_1 + L_2 = L_3$ or $L_1 \cdot L_2 = L_3$ hold whenever $L_1 \cup L_2 = L_3$ or $L_1 L_2 = L_3$. We may refer to every constant $L \in Reg(\Delta)$ by the name (description) of a DFA which specifies L.

A generalized Δ-regex is an expression of the form $L_0 + L_1 \cdot X_1 + \ldots + L_n \cdot X_n$, where $L_i \in Reg(\Delta)$ for every $i, 0 \le i \le n$. A generalized Δ-regex is called prefix-free if the languages $L_1, \ldots, L_n$ are prefix-free, pairwise fully incompatible and extension-free from L_0. Clearly, every generalized Δ-regex E can be represented by a multi-DFA D_E, which specifies the family of regular languages $L_0, L_1, \ldots, L_n$. A generalized Σ-regex is any expression of the form $a_1 \cdot E_1 + \cdots + a_k \cdot E_k + L$, where $\Sigma = \{a_1, \ldots, a_k\}$, $L \in Reg(\Delta)$, and $E_1, \ldots, E_k$ are generalized Δ-regexes.

Given a homogeneous transducer $\pi = \langle Q_\varepsilon \cup Q_\Sigma, F_\varepsilon \cup F_\Sigma, \to \rangle$, we define a system of equations $\mathscr{E}_\pi$ in the same way as for real-time transducers. With each Σ-state p we associate a variable X_p, and for every input letter $a \in \Sigma$ we build a generalized Δ-regex $E_{p,a} = \sum_{q \in Q_\Sigma} Out^\Sigma_\pi(p,a,q) \cdot X_q + Fin^\Sigma_\pi(p,a)$. Clearly, if π is a prefix-free transducer then every generalized Δ-regex $E_{p,a}$ is prefix-free as well. The behavior of π is specified by a system of equations

$$\mathscr{E}_\pi = \{X_p = \sum_{a \in \Sigma} a \cdot E_{p,x} + L_p : p \in Q_\Sigma\},$$

where $L_p = 1$, if $p \in F_\Sigma$, and $L_p = 0$ otherwise.

Proposition 18. For every homogeneous transducer π the system of equations $\mathscr{E}_\pi$ has a unique solution $\{X_p = TR(\pi, p) : p \in Q_\Sigma\}$.

Proof. Follows the same line of inductive reasoning as the proof of Proposition 12.

For every state $p \in Q_\Sigma \cup Q_\varepsilon$ of a homogeneous transducer π define a generalized Δ-regex $E_{\pi,p}$:

$$E_{\pi,p} = \begin{cases} X_p, \text{ if } p \in Q_\Sigma, \\ \sum_{q \in Q_\Sigma} Out^\varepsilon_\pi(p,q) X_q + Fin^\varepsilon_\pi(p), \text{ if } p \in Q_\varepsilon. \end{cases}$$

Proposition 19. For every homogeneous transducer π and a pair of its states $p', p'' \in Q_\Sigma \cup Q_\varepsilon$ the system of equations $\mathscr{E}_\pi \cup \{E_{\pi,p'} = E_{\pi,p''}\}$ has a solution iff $\pi(p') \sim \pi(p'')$.

Proof. By Proposition 18, the system of equations $\mathscr{E}_\pi$ has the unique solution $X_p = TR(\pi, p)$ for every Σ-state p. Then, as it can be seen from definitions of sets $Out^{\varepsilon}_{\Pi}(p,q)$ and $Fin^{\varepsilon}_{\Pi}(p)$, for such an evaluation of variables Δ-regexes $E_{\pi,p'}$ and $E_{\pi,p''}$ take the values $TR(\pi, p')$ и $TR(\pi, p'')$.

Thus, the equivalence problem $\pi(p') \overset{?}{\sim} \pi(p'')$ for homogeneous transducers is reduced to the solvability checking problem for systems of transduction equations $\mathscr{E}_\pi \cup \{E_{\pi,p'} = E_{\pi,p''}\}$. Although in general case the latter is undecidable, it has an efficient decision procedure when π is a prefix-free transducer.

Below we present a solvability checking procedure for systems of equations of the form $\mathscr{E}_\pi \cup \{E'_1 = E''_1, \ldots, E'_m = E''_m\}$, where π is a prefix-free homogeneous transducer, and E'_i, E''_i are prefix-free Δ-regexes for all $1 \le i \le m,$. It is based on the same Gaussian variable elimination principle which makes it possible to bring the initial system $\mathscr{E}_1$ to an equivalent reduced form

$$\mathscr{E}_{\pi'}(X_1, \ldots, X_n) \cup \{Y_j = E_j(X_1, \ldots, X_n) : 1 \le j \le r\},$$

by deactivating one by one variables in $\mathscr{E}_1$.

The equations of every such system $\mathscr{E}_t = \mathscr{E}_\pi \cup \{E'_1 = E''_1, \ldots, E'_m = E''_m\}$ are divided into basic, reduced and active as in the case of real-time transducers. At every iteration the following equivalent transformations are applied to a system of equations $\mathscr{E}_t$.

(1) Removal of identities. All active equations $E' = E''$ that are identities are removed from $\mathscr{E}_t$. To this end, it is enough to check the equivalence of the multi-DFAs $D_{E'}$ and $D_{E''}$, that represent Δ-regexes E' and E''.

(2) Reducedness checking. If $\mathscr{E}_t$ has no active equations then the procedure terminates and announces the solvability of the system due to Proposition 18.

(3) Processing of active equations. Select in E_t an active equation

$$\sum_{i=1}^{n} L'_i X_i + L'_0 = \sum_{i=1}^{n} L''_i X_i + L''_0. \tag{4}$$

If $L'_i = L''_i$ for all $i, 1 \le i \le n$, then the procedure terminates and announces a verdict on the insolvability of the system. This decision can be explained as follows. Since (4) is not an identity, the languages L'_0 and L''_0 are different. As far as the languages $L'_i, 1 \le i \le n$, are extension-free from both L'_0 and L''_0, for any transductions $T_1, \ldots, T_n$ it is true that

$$L'_0 \cap \bigcup_{i=1}^{n} L'_i T_i = L''_0 \cap \bigcup_{i=1}^{n} L''_i T_i = \varnothing.$$

Therefore, the left and the right sides of (4) always evaluate differently.

If $L'_j \ne L''_j$ for some $j, 1 \le j \le n$, then select the shortest witness of their nonequivalence, i.e. the shortest output word u in the nonempty language $\bigcup_{i=1}^{m} (L'_i \oplus L''_i)$. Let us assume for the sake of definiteness that $u \in L'_1$ and $u \notin L'_1$. Since Δ-regex $\sum_{i=1}^{n} L'_i X_i + L'_0$ is prefix-free, the languages $L'_2, \ldots, L'_n$ are extension-free from u. Therefore, taking into account the fact that u is the shortest word in $\bigcup_{i=1}^{n} \left(L'_i \oplus L''_i \right)$, we conclude that the languages $L'_1, \ldots, L'_n$ are extension-free from u as well. So, since u is extension-free from L'_0, for any transductions $T_1, \ldots, T_n$ it is true that

$$\sum_{i=1}^{n} L'_i T_i + L'_0 = \sum_{i=1}^{n} L''_i T_i + L''_0 \Rightarrow T_1 = \sum_{i=1}^{n} (L''_i \backslash u) T_i + (L''_0 \backslash u).$$

Thus, $\mathscr{E}_t$ is equivalent to the system $\mathscr{E}_t \cup \left\{ X_1 = \sum_{i=1}^{n} (L''_i \backslash u) X_i + (L''_0 \backslash u) \right\}$.

Consider this additional equation

$$X_1 = \sum_{i=1}^{n}(L_i''\backslash u)X_i + (L_0''\backslash u). \quad (5)$$

If $L_0''\backslash u = L_1''\backslash u = \ldots = L_n''\backslash u = \varnothing$, then the procedure terminates and announces the unsolvability of $\mathscr{E}_t$, since, by Proposition 18, the subsystem $\mathscr{E}_\pi$ has nonempty transductions as the solutions.

Otherwise, it should be noticed that $\varepsilon \notin L_1''\backslash u$ because of the choice of u. Therefore, by Arden's Theorem [67, 71], Eq. (5) is equivalent to the equation

$$X_1 = \sum_{i=2}^{n}(L_1''\backslash u)^*(L_i''\backslash u)X_i + (L_1''\backslash u)^*(L_0''\backslash u). \quad (6)$$

Thus, we can add Eq. (6) to the system $\mathscr{E}_t$ instead of (5) and replace all occurrences of variable X_1 in other equations of the system $\mathscr{E}_t$ with Δ-regex from the right-hand side of (6). As a result, we obtain an equivalent system of equations in which a new reduced Eq. (6) appeared, and the number of active variables decreases by 1.

But the application of the substitution to the equations of the system $\mathscr{E}_t$ may give the same side effect as in the case of real-time transducers: we may obtain nonstandard equations of the form

(A) $E = G$, where E is a non-variable Δ-regex, and G is a Σ-regex, or

(B) $E' = E''$, where E', E'' are non-variable Δ-regexes, or, moreover,

(C) Several basic equations of the form $X = G$ with the same variable X in their left-hand sides may appear in $\mathscr{E}_t$. The canonical form of the system of equations can be restored using the same transformations 4) and 5) that were described in Section 5. Propositions 2-9 guarantee that after these transformations (including the application of the substitution) all Δ- regexes in the obtained system of equations $\mathscr{E}_{t+1}$ remain prefix-free.

Proposition 20. For every prefix-free homogeneous transducer π and a pair of its states p', p'' the procedure above when being applied to the system of equations $\mathscr{E}_1 = \mathscr{E}_\pi \cup \{E_{\pi,p'} = E_{\pi,p''}\}$ terminates and correctly detects the solvability of $\mathscr{E}_1$.

Proof. At every stage t the procedure either terminates and correctly detects the (un)solvability of the system $\mathscr{E}_t$, or yields an equivalent system of equations $\mathscr{E}_{t+1}$, which has less active variables than $\mathscr{E}_t$. As a result, the procedure either finds that $\mathscr{E}_1$ has no solutions, or builds an equivalent reduced system of equations which, by Proposition 18, certifies the solvability of the initial system $\mathscr{E}_1$.

Theorem 3. *The equivalence problem for prefix-free homogeneous transducers is decidable in cubic time.*

Proof. Decidability follows from Propositions 19 and 20. The number of iterations the solvability checking procedure described above needs to complete does not exceed the number of states of the analyzed transducer π. As it can be seen from the definition of languages $Out_\pi^\Sigma(p,a,q)$, $Out_\pi^\varepsilon(p), Fin_\pi^\Sigma(p,a), Fin_\Sigma^\varepsilon(p)$, multi-DFAs which specify Δ-regexes in the system of equations $\mathscr{E}_\pi$ and a verification equation $E_{\pi,p'} = E_{\pi,p''}$ can be extracted without any computational overhead from the transition system of the analyzed transducer π. In Section 4 it was shown that all manipulations with Δ- regexes in the solvability checking procedure can be performed in linear time and without increasing the total size of the used multi-DFAs. The only two operations that require more time are checking the equality of regular languages $L(A') = L(A'')$, and computing the shortest witness, i.e. a word $u \in L(A') \oplus L(A'')$ in symmetric difference of regular languages specified by DFAs A' and A''. These operations can be performed in time quadratic of the size of DFAs (see [67, 72]).

In the next sections we show how to use prefix-free homogeneous transducers to check the equivalence of deterministic two-tape automata and deterministic biautomata.

7. EQUIVALENCE CHECKING OF DETERMINISTIC TWO-TAPE AUTOMATA

A two-tape finite state automaton (briefly, 2-FSA) over disjoint alphabets Σ and Δ is a quadruple $M = \langle S_1, S_2, F, \dashrightarrow \rangle$ such that S_1, S_2 is a partitioning of a finite set $S = S_1 \cup S_2$, a set $F \subseteq S$ is a subset of

final states, and, $---\rightarrow$ is a transition relation of the type $(S_1 \times \Sigma \times S) \cup (S_2 \times \Delta \times S)$. Transitions are depicted as $s \overset{z}{---\rightarrow} s'$, where $z \in \Sigma$ when $s \in S_1$, and $z \in \Delta$ when $s \in S_2$. We denote by $---\rightarrow_\Delta$ the projection of a transition relation $---\rightarrow$ on the alphabet a, i.e. $---\rightarrow_\Delta = ---\rightarrow \cap S_2 \times \Delta \times S$.

A run of a 2-FSA M is any sequence of transitions

$$s \overset{z_1}{---\rightarrow} s_1 \overset{z_2}{---\rightarrow} \cdots \overset{z_{n-1}}{---\rightarrow} s_{n-1} \overset{z_n}{---\rightarrow} s'. \tag{7}$$

A run (7) is called final, if $s' \in F$. We say that a 2-FSA M accepts a pair of words $(h, w) \in \Sigma^* \times \Delta^*$ in a state s, if the automaton M has such a final run (7) that h is a projection of the word $z_1 z_2 \ldots z_{n-1} z_n$ on the alphabet Σ, and w is a projection of the same word $z_1 z_2 \ldots z_{n-1} z_n$ on the alphabet Δ. A transduction relation recognized by a 2-FSA M in a state s is the set $TR(M, s)$ of all pairs of words (h, w) accepted by M in this state. A 2-FSA M is called deterministic (2-DFSA), if for every letter z and a state s it has at most one transition of the form $s \overset{z}{---\rightarrow} s'$.

States s' and s'' of a 2-FSAs M are equivalent ($M(s') \sim M(s'')$ in symbols), if $TR(M, s') = TR(M, s'')$. The equivalence problem for 2-FSAs is that of checking, given a 2-FSA M and a pair of its states s' and s'' whether $M(s') \sim M(s'')$ holds.

The relationship between 2-FSAs and transducers is quite obvious: it is enough to require that a two-tape automaton, instead of reading the letters on the second tape, write the same letters on this tape, and we turn a 2-FSA into a homogeneous transducer which realizes the same transduction relations at its states. If a 2-FSA $M = \langle S_1, S_2, F, ---\rightarrow \rangle$ has no cycles on states from the set S_2 (i.e., it can not read arbitrary long on the second tape), then M can be converted into an equivalent real-time transducer. However, till now it remained unclear to which class of transducers 2-DFSAs correspond. But as soon as the class of prefix-free homogeneous transducers was distinguished, the answer to this question becomes obvious.

Consider an arbitrary 2-FSA $M = \langle S_1, S_2, F, ---\rightarrow \rangle$, and define a transducer $\pi_M = \langle S_1 \cup S_2, F, \rightarrow \rangle$, so that for every pair of states s, s' and every letter $x \in \Sigma \cup \Delta$:

- π_M has a transition $s \xrightarrow{x|\varepsilon} s'$ iff $s \in S_1$, $x \in \Sigma$, and M has a transition $s \overset{x}{---\rightarrow} s'$.
- π_M has a transition $s \xrightarrow{\varepsilon|y} s'$ iff $s \in S_2$, $y \in \Delta$, and M has a transition $s \overset{y}{---\rightarrow} s'$.

Proposition 21. Let M be a 2-DFSA and π_M is a transducer as defined above. Then it is a prefix-free homogeneous transducer, and $TR(M, s) = TR(\pi_M, s)$ holds for every state $s \in S_1 \cup S_2$.

Proof. As it can be seen from the definition of transition relation of π_M, for every its state s there exists a final run labelled with a pair of words (h, w) iff 2-DFSA M, accepts this pair of words in the state s. Therefore, $TR(M, s) = TR(\pi_M, s)$.

As for prefix-free property, it is worth noting that if $S_1 = \{s_1, \ldots, s_m\}$, then for every state $s \in S_2$ the multi-DFA $D = \langle S, s, \{s_1\}, \ldots, \{s_m\}, F \cap S_2, ---\rightarrow_\Delta \rangle$ is prefix-free.

By combining Proposition 21 and Theorem 3, we arrive at

Theorem 4. *The equivalence problem for deterministic two-tape finite state automata is decidable in cubic time.*

8. EQUIVALENCE CHECKING OF DETERMINISTIC BIAUTOMATA

A finite biautomata (BA) over an alphabet Σ is quadruple $B = \langle Q, F, Left, Right \rangle$, where Q is a finite set of states, $F \subseteq Q$ is a subset of final states, and $Left$ and $Right$ are transition relations of the type $Q \times \Sigma \times Q$.

Any biautomaton B has two heads that read letters on both ends of an input word. The left head reads letters on the left side of a word, and the right head do the same on the other side of the same word. In one step of a computation BA B chooses one of the heads, reads the next symbol by it, and moves to a new state according to a transition relation $Left$ (if it chooses the left head) or $Right$ (if it chooses the right head). A computation ends when the heads meet somewhere on the tape. If at the end of a computation B is at a final state then it accepts the input word.

Formally, computations of a biautomaton are defined using the concept of configuration. A configuration of a biautomaton B is a pair $\alpha = (q, w)$, where q is a state of B, and w is a word over Σ (a part of the input word that the biautomaton has yet to read). A relation $\vdash$ on the set of configurations of a biautomaton B is defined for every pair of its states q, q', a word w, and a letter a as follows: $(q, aw) \vdash (q', w)$ holds if $(q, a, q') \in Left$, and $(q, wa) \vdash (q', w)$ holds if $(q, a, q') \in Right$. A run of a biautomaton B on an input word w from a state q is any sequence of configurations

$$\alpha_0 \vdash \alpha_1 \vdash \ldots \alpha_{n-1} \vdash \alpha_n, \tag{8}$$

such that $\alpha_0 = (q, w)$. A run (8) is called final iff $\alpha_n = (q', \varepsilon)$ for some $q' \in F$. A biautomaton B accepts an input word w in a state q if it has a final run on w from q. A language recognized by a biautomaton B in a state q is a set $L(B, q)$ of all words accepted by B in this state. We write $\mathcal{L}(BA)$ for the class of languages recognized by BAs.

States q' and q'' of a biautomaton B are equivalent ($B(q') \sim B(q'')$ in symbols), if $L(B, q') = L(B, q'')$. The equivalence problem for biautomata is that of checking, given an automaton B and a pair of its states q' and q'', whether $B(q') \sim B(q'')$ holds.

In [12, 14, 43] certain specific subclasses of BAs were distinguished. A BA $B = \langle Q, F, Left, Right \rangle$ is called deterministic (DBA) if the set of states Q is partitioned into subsets Q_L and Q_R such that transition relations $Left$ and $Right$ are (partial) functions $Left : Q_L \times \Sigma \to Q$ and $Right : Q_R \times \Sigma \to Q$. DBA B is weak from the right (DWBA) if for each $F \subseteq Q_L$ and every state $q \in Q_R$ there exists a unique $a \in \Sigma$, such that $Right(q, a)$ is defined. Loukanova [12] proved that the class of languages $\mathcal{L}(BA)$ coincides with the class **Lin** of linear CFLs, and, therefore, the equivalence problem for BAs is undecidable. Jiraskova and Klima found in [14] that the class of languages $\mathcal{L}(DWBA)$ coincides with the class **DLin** of deterministic linear CFLs, and this implies that the equivalence problem for DWBAs is decidable. But in [43] it was also established that $\mathcal{L}(DBA)$ is incomparable with the family of deterministic CFLs; this means that the technique developed in [60] for the equivalence checking of DPDAs is not applicable for solving the same problem for DBAs; thus, the question of the (un)decidability of this problem for DBAs remained open.

We show how to check the equivalence of DBAs by exploiting an obvious relationship between DBAs and prefix-free homogeneous transducers and by adapting appropriately the solvability checking techniques for systems of linear equations developed in Section 6.

Rosenberg [1] found a simple correspondence between 2-FSAs and linear context-free grammars: L is a linear CFL iff there exists such a 2-FSA M and its state s that $L = \{uv^{-1} : (u, v) \in TR(M, s)\}$ holds. Since linear CFLs are recognized by BAs, and 2-FSAs realize the same transduction relations as homogeneous transducers, it is possible to convert BAs into transducers.

Given a BA $B = \langle Q, F, Left, Right \rangle$ over an alphabet Σ consider a transducer $\pi_B = \langle Q, F, \to \rangle$ which has the same set of letters Σ for its input and output alphabets and whose transition relation is defined as follows: for every pair of states p, q and a letter a a quadruple $p \xrightarrow{a|\varepsilon} q$ (or $p \xrightarrow{\varepsilon|a} q$) is a transition of π_B iff $(p, a, q) \in Left$ (or $(p, a, q) \in Right$, respectively).

Proposition 22. For every BA $B = \langle Q, F, Left, Right \rangle$, the corresponding transducer π_B as defined above and a state $q \in Q$

1. $L(B, q) = \{uv^{-1} : (u, v) \in TR(\pi_B, q)\}$;

2. if BA B is deterministic then, then π_B is a prefix-free homogeneous transducer.

Proof. Follows immediately from the definitions of DBA, a language $L(B, q)$, recognized by BA B, transduction relation $TR(\pi, q)$, realized by a transducer π, and the correspondence between B and π_B.

We say that a transducer π generates a catenation language $LC(\pi, q) = \{uv^{-1} : (u, v) \in TR(\pi, q)\}$ in a state q. States p and q of a transducer π are called ctn-equivalent ($\pi(p) \sim_{ctn} \pi(q)$ in symbols), if $LC(\pi, p) = LC(\pi, q)$. Thus, by Proposition 22, the equivalence problem for DBAs is reduced to the ctn-equivalence problem for prefix-free homogeneous transducers.

To check the ctn-equivalence $\pi(p) \sim_{ctn} \pi(q)$ for prefix-free homogeneous transducers we adapt the technique developed in Section 6 for checking the transduction equivalence $\pi(p) \sim \pi(q)$: build a system of language equations $\hat{\mathcal{E}}_\pi$, which specifies the catenation languages generated by π, add to this system a ver-

ification equation $X_p = X_q$ and check the solvability of the extended system of equations using Gaussian variable elimination. But this time regular expressions, from which the equations are built, are interpreted over the semiring of languages over an alphabet Σ.

Equations of a system $\hat{\mathscr{E}}_\pi$ are constructed from regexes which are formed from variables $X_1, X_2, \ldots$, constants 0,1, and regular languages from the family $Reg(\Sigma)$. These languages are regarded as constants, and they are referred by names (descriptions) of DFAs that specify these languages. A $\hat{\Delta}$-regex $\hat{E}$ is any expression of the form $L_0 + L_1 \cdot X_1 + \ldots + L_n \cdot X_n$, where $L_i \in Reg(\Sigma)$ for all $i, 0 \le i \le n$. Although this time transducers we deal with have the same alphabet Σ for input and output tapes, we retain the terminology for the sake of uniformity. A $\hat{\Delta}$-expression $\hat{E}$ is called prefix-free if languages $L_1, \ldots, L_n$ are prefix-free, pairwise fully incompatible, and extension free from L_0. Such a $\hat{\Delta}$- regex can be represented by a multi-DFA $D_{\hat{E}}$, which specifies the family of regular languages $L_1, \ldots, L_n, L_0$. A $\hat{\Sigma}$-regex is any expression $\hat{G}$ of the form $\hat{E}_1 \cdot a_1 + \cdots + \hat{E}_k \cdot a_k + L$, где $\Sigma = \{a_1, \ldots, a_k\}$, $L \in Reg(\Sigma)$, and $\hat{E}_1, \ldots, \hat{E}_k$ are $\hat{\Delta}$-expressions.

Given a homogeneous transducer $\pi = \langle Q_\varepsilon \cup Q_\Sigma, F_\varepsilon \cup F_\Sigma, \to \rangle$, the system of equations $\hat{\mathscr{E}}_\pi$ is defined as expected. For every state $p \in Q$ and letter $a \in \Sigma$ we build a $\hat{\Delta}$-regex $\hat{E}_{p,a} = \sum_{q \in Q_\Sigma} Out^\Sigma_\pi(p,a,q) \cdot X_q + Fin^\Sigma_\pi(p,a)$. It is easy to see that if π is a prefix-free transducer then every $\hat{\Delta}$-regex $\hat{E}_{p,a}$ is prefix-free as well. The family of catenation languages generated by π is specified by a system of equations

$$\hat{\mathscr{E}}_\pi = \left\{ X_p = \sum_{a \in \Sigma} \hat{E}_{p,a} \cdot a + L_p : p \in Q_\Sigma \right\},$$

where $L_p = 1$, if $p \in F_\Sigma$, and $L_p = 0$ otherwise. Following the same line of reasoning as in the case of Proposition 12 and taking into account the changes made to the definitions of regexes, it is easy to prove

Proposition 23. For every homogeneous transducer π the system of equations $\hat{\mathscr{E}}_\pi$ has a unique solution $\{X_p = LC^{-1}(\pi, p) : p \in Q_\Sigma\}$.

To build verification equations we define for every state p of a homogeneous transducer π a $\hat{\Delta}$-regex $\hat{E}_{\pi,p}$:

$$\hat{E}_{\pi,p} = \begin{cases} X_p, \text{ if } \ p \in Q_\Sigma, \\ \sum_{q \in Q_\Sigma} Out^\varepsilon_\pi(p,q) \cdot X_q + Fin^\varepsilon_\pi(p), \text{ if } \ p \in Q_\varepsilon. \end{cases}$$

Proposition 24. For every homogeneous transducer π and every pair of states $p', p'' \in Q_\Sigma \cup Q_\varepsilon$ the system of equations $\hat{\mathscr{E}}_\pi \cup \{\hat{E}_{\pi,p'} = \hat{E}_{\pi,p''}\}$ has a solution iff $\pi(p') \sim_{ctn} \pi(p'')$.

Proof. As it follows from Proposition 23, the system of equations $\hat{\mathscr{E}}_\pi$ has the unique solution $X_p = LC^{-1}(\pi, p)$ for every state $p \in Q_\Sigma$. Looking at the definitions of sets $Out^\varepsilon_\pi(p,q)$ and $Fin^\varepsilon_\pi(p)$, one can easily notice that for such a solution $\hat{\Delta}$-regexes $\hat{E}_{\pi,p'}$ and $\hat{E}_{\pi,p''}$ take the values $LC^{-1}(\pi, p')$ и $LC^{-1}(\pi, p'')$.

The solvability of systems of equations $\hat{\mathscr{E}}_\pi \cup \{\hat{E}'_1 = \hat{E}''_1, \ldots, \hat{E}'_m = \hat{E}''_m\}$, where π is a prefix-free homogeneous transducer, and $\hat{E}'_i, \hat{E}''_i, 1 \le i \le m$, are prefix-free $\hat{\Delta}$-regexes, can be checked by the same variable deactivating scenario as described in Section 6, i.e. by bringing the system to an equivalent reduced form. At every stage the checking procedure applies the following equivalent transformations to a current system of equations $\hat{\mathscr{E}}_t$.

1. Removal of identities $\hat{E} = \hat{E}$.
2. Reducedness checking.
3. Processing of active equations $\hat{E}' = \hat{E}''$.
4. Removal of nonstandard equations $\hat{E} = \hat{G}$.

These transformations are exactly the same as described in Section 6. The only difference concerns the removing of nonstandard equations of the form $\hat{G}' = \hat{G}''$, where $\hat{G}', \hat{G}''$ are Σ-regexes.

5. Removal of nonstandard equations $\hat{G}' = \hat{G}''$.

If a system includes an equation

$$\sum_{i=1}^{k} \hat{E}_i' \cdot a_i + L' = \sum_{i=1}^{k} \hat{E}_i'' \cdot a_i + L'',$$

then remove it from the system and insert the equations $\hat{E}_i' + L'/a_i = \hat{E}_i'' + L''/a_i, 1 \le i \le k$, instead of it. It is easy to see that as a result of applying this transformation we obtain an equivalent system of equations. By Proposition 3, $\hat{\Delta}$-regexes on both sides of new equations remain prefix-free. Thus, all nonstandard equations of the form $\hat{G}' = \hat{G}''$ disappear from the system $\hat{\mathscr{E}}_t$, and we obtain an equivalent system of equations $\hat{\mathscr{E}}_{t+1}$ with fewer active variables. As in Section 6, multi-DFAs are used for the representation and processing of $\hat{\Delta}$-regexes. These considerations, supported by Propositions 23 and 24, lead to

Theorem 5. *The equivalence problem for deterministic finite biautomata is decidable in cubic time.*

The equivalence checking procedure described here is applicable only to deterministic DBAs. The decisive role in it is assigned to the rules of (3) deactivation of variables and (5) removal of nonstandard equations of the form $\hat{G}' = \hat{G}''$. These rules require that the transducer π_B is prefix-free, which is a consequence of the determinism of the biautomaton B.

The decision procedure can be greatly simplified for some classes of DBAs. For example, if a biautomaton does not have the ability to read letters for an arbitrarily long time, while remaining all along on the right side of a word, then it can be translated into a prefix-free real-time transducer, for which the verification of ctn-equivalence of states can be carried out more efficiently than for prefix-free homogeneous transducers.

Given a DBA $B = \langle Q_L, Q_R, F, Left, Right \rangle$, its transition function $Right : Q_R \times \Sigma \to Q_L \cup Q_R$ can be extended in the usual way to the words in Σ^*: for every state $q \in Q_R$, a word $w \in \Sigma^*$ and a letter $x \in \Sigma$ we will assume $Right^*(q, \varepsilon) = q$ and $Right^*(q, wx) = Right(p, x)$, if $Right^*(q, w) = p \in Q_R$. We say that a DBA B is deterministic acyclic on the right biautomaton (DABA), if $F \subseteq Q_L$ and $Right^*(q, w) \ne q$ holds for every state $q \in Q_R$ and a nonempty word $w \ne \varepsilon$. As can be seen from this definition, every deterministic weak from the right biautomaton (DWBA) is acyclic on the right. Moreover, a strict inclusion $\mathscr{L}(DWBA) \subset \mathscr{L}(DABA) \subset \mathscr{L}(DBA)$ holds, since

• a language $\{a^n b^n : n \ge 0\} \cup \{a^n c^n : n \ge 0\}$ is, obviously, in the class $\mathscr{L}(DABA)$, but it does not belong to the family of languages $\mathscr{L}(DWBA)$;

• a language $\{a^n (bc^*)^n : n \ge 0\} : n \ge 0\}$ is, obviously, in the class $\mathscr{L}(DBA)$, but it does not belong to the family of languages $\mathscr{L}(DABA)$.

For every DBA $B = \langle Q_L, Q_R, F, Left, Right \rangle$ we build a transducer $\pi_B = \langle Q_L, F, \to \rangle$ such that its transition relation $\to$ is defined as follows: for every pair of states $p, q \in Q_L$, a letter $x \in \Sigma$ and a word $w \in \Sigma^*$ the transducer π_B has a transition $p \xrightarrow{x|w} q$ iff one of the conditions below is satisfied

1. $Left(p, x) = q$ and $w = \varepsilon$, or

2. $Left(p, x) = r \in Q_R$ and $Right^*(r, w) = q$.

For the proposed alternative translation of biautomata into real-time transducers the following Proposition is true.

Proposition 25. For every DABA B the corresponding machine π_B is a prefix-free real-time finite transducer such that for every its state $q \in Q_L$ the equality $L(B, q) = \{uv^{-1} : (u, v) \in TR(\pi_B, q)\}$ holds.

Proof. Since the biautomaton is acyclic on the right, for every its state $r \in Q_R$ the extended transition function $Right^*(r, w)$ is defined on a finite set of words w. Therefore, as it follows from the definition of transition relation for π_B, for every letter $x \in \Sigma$ each state $p \in Q_L$ has only finitely many outgoing transitions of the form $p \xrightarrow{x|w} q$. As far as BA B is deterministic, for every state $q \in Q_R$ the set of words

$\{w : Right^*(q,w) \in Q_L\}$ is prefix-free. Therefore, as it follows from the definition of transition relation for the transducer π_B, for every state p and a letter x the sets of words $Out_{\pi_B}(p,q,x)$, where $q \in Q_L$, are pairwise fully incompatible prefix-free languages. Hence, π_B is a prefix-free real-time transducer. And, finally, as it was noticed in Proposition 22, the equality $L(B,q) = \{uv^{-1} : (u,v) \in TR(\pi_B,q)\}$ follows immediately from the definition of the language $L(B,q)$ and the transition relation for π_B.

Thus, the equivalence problem for DABAs can be reduced to the problem of checking the ctn-equivalence of the states of real-time prefix-free finite transducers, and this problem can be solved by checking the solvability of the corresponding systems of linear equations in the same way as it was shown earlier for prefix-free homogeneous transducers. The only difference is that now the coefficients in Δ-regexes are not prefix-free regular languages but individual words. Therefore, to check the solvability of such systems of equations, one can use a modified procedure described in Section 5 and based on Proposition 10. This modification, in essence, concerns only the form of equations and leads to the following results.

Theorem 6. *The equivalence problem for deterministic acyclic on the right finite state biautomata (DABAs) is decidable in quadratic time.*

Corollary 2. The equivalence problem for deterministic weak from the right finite state biautomata (DABAs) is decidable in quadratic time.

Corollary 3. The equivalence problem for deterministic linear context-free grammars is decidable in quadratic time.

9. THE EQUIVALENCE PROBLEM FOR SIMPLE GRAMMARS

The Gaussian variable elimination technique can also be successfully applied to check the solvability of systems of nonlinear equations that describe languages generated by some context-free grammars, and, thus, to obtain an efficient solution to the equivalence problem for these grammars. For the first time, the authors of the [45] used an algebraic approach to solve the equivalence problem for the so-called. simple grammars. In this section, we show how the idea of the variable elimination method can be appropriately adapted to check the solvability of systems of equations describing the problem of checking the equivalence of nonterminals in simple grammars.

A context-free grammar over an alphabet Σ is a pair $\Gamma = \langle \mathcal{N}, \mathcal{P} \rangle$, which includes a finite set of nonterminals $\mathcal{N}$ and a finite set of production rules $\mathcal{P}$. The symbols in the set $\mathcal{N}$ (nonterminals) differ from the letters in the alphabet Σ (terminals). For the convenience of naming, we will agree to call sequences of letters in the alphabet Σ terminal words and denote them by symbols u, v, w, and call sequences of letters in the alphabet $\Sigma \cup \mathcal{N}$ sentences and denote them by symbols α, β, γ. Production rules are the pairs of the form $N \to \alpha$, where N is a nonterminal in the set $\mathcal{N}$, which is called a head of the rule, and α is a sentence, which is called a body of the rule. Given a grammar Γ, a 1-step derivation relation $\to_\Gamma$ on the set of sentences is defined: two sentences β_1, β_2 are in 1-step derivation relation $\beta_1 \to_\Gamma \beta_2$ iff $\beta_1 = \beta' N \beta'', \beta_2 = \beta' \alpha \beta''$ and the set $\mathcal{P}$ contains a rule $N \to \alpha$. A derivation relation $\xrightarrow{*}_\Gamma$, generated by a grammar Γ, is the reflexive transitive closure of 1-step derivation relation $\longrightarrow_\Gamma$. A language of a sentence α in a grammar Γ is the set of all terminal words $L(\Gamma, \alpha) = \{w : w \in \Sigma^*, \alpha \xrightarrow{*}_\Gamma w\}$ derived from α in Γ.

A nonterminal N is called irrelevant in a grammar Γ if $L(\Gamma, N) = \varnothing$. As it was shown in the textbook [67], all irrelevant nonterminals in a context-free grammar can be detected in quadratic time. All production rules that refer to irrelevant nonterminals can be removed from the grammar Γ; as the result one obtains such a grammar Γ' that for every nonterminal the equality $L(\Gamma, N) = L(\Gamma', N)$ holds. Therefore, when studying the equivalence problem, we restrict ourselves to considering context-free grammars that do not contain irrelevant nonterminals.

Two sentences α_1, α_2 are called equivalent in a grammar Γ ($\alpha_1 \sim_\Gamma \alpha_2$ in symbols) if $L(\Gamma, \alpha_1) = L(\Gamma, \alpha_2)$. It is easy to see that for context-free grammars, the equivalence relation of sentences is a congruence relation with respect to concatenation operation: the relationship $\beta \sim_\Gamma \gamma \Rightarrow \alpha' \beta \alpha'' \sim_\Gamma \alpha' \gamma \alpha''$ holds for any sentences $\alpha', \alpha'', \beta, \gamma$.

The equivalence problem for nonterminals is to check for any given grammar Γ and for any pair of its nonterminals N_1, N_2 whether the equivalence relation $N_1 \sim_\Gamma N_2$ holds.

We will solve the equivalence problem for a class of simple grammars. We say that a production rule $N \to x\alpha$ has a normal form if x is a terminal which is called a key of the rule, and $\alpha \in \mathcal{N}^*$ is a sequence

of nonterminals. A context-free grammar Γ is called simple [45] if all its production rules are in normal form and all rules with the same head have pairwise different keys.

As it can be seen from this definition, a simple grammar Γ is unambigous: for any nonterminal N and a word $w \in L(\Gamma, N)$ there exists a unique leftmost derivation of w from N. This property displays itself in the following property, which is important for solving the equivalence problem.

Proposition 26. Suppose that $\Gamma = \langle \mathcal{N}, \mathcal{P} \rangle$ is a simple grammar. Then for any sentences $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathcal{N}^*$ and a terminal word w it is true that

$$\alpha_1 \sim_\Gamma \alpha_2 \wedge \alpha_1 \xrightarrow{*}_\Gamma w\beta_1 \wedge \alpha_2 \xrightarrow{*}_\Gamma w\beta_2 \Rightarrow \beta_1 \sim_\Gamma \beta_2.$$

In [45] it was proved that for every simple grammar $\Gamma = \langle \mathcal{N}, \mathcal{P} \rangle$ and a nonterminal $N \in \mathcal{N}$ the language $L(\Gamma, N)$ is prefix-free. Therefore, as the authors of this paper noted, when simple grammars are analyzed, it is advisable to rely on some fundamental properties of prefix-free languages. These include the laws of left and right contraction of prefix-free languages with respect to concatenation operation.

Proposition 27. ([45]) For any prefix-free languages L, L', L'' the following relationships hold:

1. $LL' = LL'' \Leftrightarrow L' = L''$,

2. $L'L = L''L \Leftrightarrow L' = L''$.

An algorithm for checking the equivalence of two nonterminals M, N of an arbitrary simple grammar $\Gamma = \langle \mathcal{N}, \mathcal{P} \rangle$ operates in two stages: building of a system of equations $\mathcal{E}(\Gamma, M, N)$ which specifies the equivalence problem, and checking the solvability of the system of equations thus constructed.

The equation of the system $\mathcal{E}(\Gamma, M, N)$ are composed of expressions, and the expressions are constructed of variables $X_1, X_2, \ldots$ and constants by means of concatenation $\cdot$ and alternation $+$. Each nonterminal X is regarded as a variable. Constants are 0,1, and the terminals from Σ. All expressions are interpreted in the semiring of languages over Σ.

For every nonterminal X a basic equation $X = E_X$ is formed such that $E_X = \sum_{X \to \alpha \in \mathcal{P}} \alpha$. A system of equations which specifies the equivalence problem $M \overset{?}{\sim}_\Gamma N$ is that of the form $\mathcal{E}(\Gamma, M, N) = \mathcal{E}_\Gamma \cup \{M = N\}$, where $\mathcal{E}_\Gamma = \{X = E_X : X \in \mathcal{N}\}$.

Proposition 28. ([45]) For every simple grammar Γ which has no irrelevant nonterminals the system of basic equations $\mathcal{E}_\Gamma$ has the unique solution $X = L(\Gamma, X)$ for each nonterminal X.

Corollary 4. For every simple grammar Γ which has no irrelevant nonterminals the system of equations $\mathcal{E}(\Gamma, M, N) = \mathcal{E}_\Gamma \cup \{M = N\}$ has a solution iff $M \sim_\Gamma N$.

At the second stage, to check the equivalence $M \sim_\Gamma N$, the algorithm applies the procedure for checking the solvability of the systems of equations

$$\mathcal{E}_t = \mathcal{E}_\Gamma \cup \{\alpha_1 = \beta_1, \ldots, \alpha_m = \beta_m\}, \tag{9}$$

where $\alpha_i, \beta_i \in \mathcal{N}^*$. A verification equation $\alpha_i = \beta_i$ in system (9) is called reduced, if the sentence α_i is a single variable X_j, which has no other occurrences in the equations of the system; in this case the variable X_j is also called reduced. A system of equations (9) is called reduced if all its verification equations are reduced. As it follows from Proposition 28, any reduced system of equations has a solution. Based on this, to check the solvability of systems of equations of the form (9) the decision procedure applies equivalent transformations aimed at constructing a reduced system. At each iteration, this procedure either increases the number of reduced variables or finds inconsistent equations, which indicates that the system has no solution.

The procedure starts with the system of equation $\mathcal{E}_1 = \mathcal{E}(\Gamma, M, N)$, and at every iteration t it applies the following transformations to the current system of equations (9).

(1) Contraction of equations. Left and right contraction rules $X\alpha = X\beta \Rightarrow \alpha = \beta$ and $\alpha X = \beta X \Rightarrow \alpha = \beta$ are applied to both sides of verification equations (see Proposition 27). If the application of these transformations yields the identity $1 = 1$, then it is removed from the system. If the application of these transformations brings some verification equation to the form $X\alpha = 1$ or $1 = X\beta$, then the checking procedure terminates and announces that the system of equations $\mathcal{E}(\Gamma, M, N)$ has no solutions. If such

inconsistent equations do not arise, then at the end of this step all verification equations remaining in the system will have the form $X\alpha = Y\beta$, where X and Y are different variables.

(2) Reducedness checking. If a system $\mathscr{E}_t$ is such that all verification equations $\alpha_i = \beta_i$ are reduced, then $\mathscr{E}_t$ is a reduced system of equations. In this case the solvability checking procedure terminates and announces the solvability of the initial system of equations $\mathscr{E}(\Gamma, M, N)$.

(3) Constructing equations of the form $X = \alpha$. If the system $\mathscr{E}_t$ contains a verification equation $X\alpha = Y\beta$ such that $|\alpha| > 0$ then $|\beta| > 0$, then one needs to compare the lengths of the shortest terminal words u and v that may be derived from nonterminals X and Y: $X \xrightarrow{*}_{\Gamma} u$, $Y \xrightarrow{*}_{\Gamma} v$. We will assume that $|u| \le |v|$ (otherwise the left and right sides of the equation can be swapped). If it is impossible to build a leftmost derivation of the form $Y \xrightarrow{*}_{\Gamma} u\gamma$, then the procedure terminates and announces that the initial system of equations $\mathscr{E}(\Gamma, M, N)$ is unsolvable. But if $Y \xrightarrow{*}_{\Gamma} u\gamma$ holds for some sentence $\gamma \in N^*$, then the equation $X\alpha = Y\beta$ in the system $\mathscr{E}_t$ is replaced with a pair of equations $\alpha = \gamma\beta$ and $Y = X\gamma$. After such a replacement the system $\mathscr{E}_t$ will have at least one verification equation of the form $Y = X\gamma$ such that its left-hand side is a variable and it is different from the right-hand side.

(4) Reduction of variables. In the resulting system $\mathscr{E}_t$, an unreduced equation of the form $Y = \beta$ is selected in which the sentence β is different from the variable Y. If Y is a member of β then the procedure terminates and announces that the initial system of equations $\mathscr{E}(\Gamma, M, N)$ unsolvable. If the sentence β does not contain the variable Y, then the substitution $\{Y/\beta\}$ is applied to all other equations of the system. After the application of this substitution the verification equation $Y = \beta$ becomes reduced, and, thus, the number of reduced variables increases.

However, the application of the substitution $\{Y/\beta\}$ to the basic equations has a side effect: either two basic equations with the same left-hand side, or an equation of the form $X\beta = \sum_{a\in\Sigma} a \cdot \alpha_a$, where $|\ beta\ | > 0$, may appear in the system.

Therefore, further it is necessary to apply a transformation that restores the canonical form of the system.

(5) Removal of duplications. If two equations of the form $X = \sum_{a\in\Sigma} a \cdot \alpha_a$ and $X\beta = \sum_{a\in\Sigma} a \cdot \alpha'_a$, such that their left-hand sides start with the same variable appear in the system, then the second equation is removed from the system and instead of it verification equations $\alpha_a\beta = \alpha'_a, a \in \Sigma$ are added to the system. As a result of applying this transformation, the canonical form of the resulting system of equations will be restored, and the procedure for checking the solvability proceeds to the next iteration $t + 1$.

Proposition 29. For every simple grammar Γ and a pair of its nonterminals M, N the described procedure, after being applied to the system of equations $\mathscr{E}(\Gamma, M, N)$, always terminates and correctly detects the solvability of this system.

Proof. The described procedure always terminates since after each iteration t the number of reduced variables in the system $\mathscr{E}_{t+1}$ increases.

The rules above define equivalent transformations of systems of equations. For rules (1), (4) and (5) this fact follows from Proposition 27 and congruence property of the relation $\sim_\Gamma$. Consider rule (3) and suppose that $X \xrightarrow{*}_{\Gamma} u$ and $Y \xrightarrow{*}_{\Gamma} u\gamma$. Then $X\alpha \xrightarrow{*}_{\Gamma} u\alpha$ and $Y\beta \xrightarrow{*}_{\Gamma} u\gamma\beta$, and, therefore, by Propositions 26 and 27, the equality $X\alpha = Y\beta$ implies $\alpha = \gamma\beta$ and $X\gamma = Y$. Thus, if $X \xrightarrow{*}_{\Gamma} u$ and $Y \xrightarrow{*}_{\Gamma} u\gamma$, then the equation $X\alpha = Y\beta$ is equivalent (in the framework of the whole system) to the pair of equations $\alpha = \gamma\beta$ and $Y = X\gamma$. Hence, rule (3) also defines an equivalent transformation. So, at each iteration t the procedure deals with the system of equation $\mathscr{E}_t$ which is equivalent to the initial system $\mathscr{E}(\Gamma, M, N)$.

The procedure correctly detects the solvability of the initial system of equations. We showed that every system of equations $\mathscr{E}_t$ is equivalent to the system $\mathscr{E}_\Gamma \cup \{M = N\}$. By Proposition 28, the unique solution of the system of equations $\mathscr{E}_\Gamma$ is such that $X = L(\Gamma, X)$ for all nonterminal variables. Note that production rules of simple grammars guarantee that $\varepsilon \notin L(\Gamma, X)$ for each nonterminal X. Therefore, the values $X = L(\Gamma, X)$ does not satisfy the equalities $X\alpha = 1$ or $X = \beta' X\beta''$. It is also clear that if an equality $X\alpha = Y\beta$ holds for these values of variables, then for every word $w \in L(\Gamma, X)$ the language $L(\Gamma, Y)$ contains either a prefix, or an extension of w. Therefore, rules (1), (3) and (4) correctly detect the unsolvabil-

ity of the system $\mathscr{E}_t$. Since every reduced system of equations has a solution, rule (2) correctly detects the solvability of the system $\mathscr{E}_t$ as well.

For a simple grammar Γ denote by $\nu(\Gamma)$ the value $\max\limits_{N\in\mathcal{N}} \min\limits_{w\in L(\Gamma,N)} |w|$. Via Proposition 29 we arrive at

Theorem 7. *The equivalence problem for nonterminals of any simple grammar* Γ *is decidable in time* $O(n^3\nu(\Gamma))$*, where n is the total size of the production rules of* Γ.

Proof. The decidability of the equivalence problem follows from Proposition 29. Reference data structures *are* used for representing the equations. When checking the solvability of the system of equations the procedure makes no more than n iterations, and at each of them, having a subsystem of verification equations of size m, it can apply rules (1) and (4) in time $O(m)$, rules (2) and (5) in time $O(n)$, and rule (3) in time $O(n\nu(\Gamma))$; in all cases the size of the subsystem of verification equation increases no more than by the value $O(n\nu(\Gamma))$. Hence, the expected complexity estimate follows.

CONCLUSIONS

In the opinion of the author of this paper, the potential capabilities of the proposed algebraic approach to the designing of efficient algorithms for checking the equivalence of various types of automata are not limited to the results presented here. Several directions can be outlined in which it is advisable to continue research aimed at improving the developed method and expanding the scope of its application.

First, it is quite natural to apply the algebraic approach to the equivalence checking ~~to~~ in other computational models whose behavior can be specified by means of algebraic equations. The main goal here is an attempt to construct a polynomial time algorithm for checking the equivalence of deterministic k-tape automata for. However, the properties of prefix-free languages that favor the construction of an efficient algorithm for checking the equivalence of deterministic two-tape automata and are expressed, for example, in Proposition 10, will apparently no longer be helpful in solving this more difficult problem. It is necessary, as far as possible, to find fundamentally new methods of restoring the canonical form of the system of equations after applying substitutions for reduced variables (see rules 4–6 in Section 5). It is likely that this could be achieved by studying the equivalence problem for some relatively narrow classes of multitape automata that can be translated into deterministic finite transducers operating over certain special semigroups (see [34]) such that their algebraic properties help to check effectively the solvability of systems of equations, just as the properties of prefix-free languages have proved to be useful.

Another direction in the development of the algebraic approach for checking the equivalence of automata is its application to the designing of minimization algorithms for various types of automata, and, first of all, for deterministic transducers operating over special semigroups. A successful example of constructing efficient algorithms for minimizing transducers based on equivalence checking procedures is presented in [8, 73]. One of the interesting problems for further research can be the problem of developing polynomial time algorithms for minimizing deterministic two-tape automata or biautomata. Here, you can also rely on the results of [10].

And, finally, the problem of designing efficient equivalence checking and minimization algorithms in the class of finite transducers operating over semigroups of substitutions worth an attention. As shown in [9], it is this automata-based model of computations that most closely matches the model of sequential imperative programs, the semantics of which is specified by the relation of logical-thermal equivalence [74]. Thus, on this line of research, the possibility of direct use of the algebraic method for checking the equivalence of automata for optimization of computer programs can be discovered.

FUNDING

This work was supported by the Russian Foundation for Basic Research, project no. 18-01-00854.

CONFLICT OF INTEREST

The author declares that he has no conflicts of interest.

REFERENCES

1. Rosenberg, A.L., A machine realization of the linear context-free languages, *Inf. Control,* 1967, vol. 10, no. 2, pp. 175–188.

2. Mohri, M., Finite-state transducers in language and speech processing, *Comput. Linguist.,* 1997, vol. 23, no. 2, pp. 269–311.
3. Roche-Lima, A. and Thulasiram, R.K., Bioinformatics algorithm based on a parallel implementation of a machine learning approach using transducers, *J. Phys.: Conf. Ser.,* 2012, vol. 341, no. 1, 012034.
4. Alur, R. and Černy, P., Streaming transducers for algorithmic verification of single-pass list-processing programs, *Proceedings of 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages,* 2011, pp. 599–610.
5. Thakkar, J., Kanade, R., and Alur, A., Transducer-based algorithmic verification of retransmission protocols over noisy channels, *Proceedings of IFIP Joint International Conference on Formal Techniques for Distributed Systems,* Springer, 2013, pp. 209–224.
6. Veanes, M., Hooimeijer, P., Livshits, B., et al., Symbolic finite state transducers: Algorithms and applications, *Proceedings of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages,* 2012, pp. 137–150.
7. Kozlova, D.G. and Zakharov, V.A., On the model checking of sequential reactive systems, *Proceedings of the 25th International Workshop on Concurrency, Specification and Programming,* Rostock, 2016, p. 233.
8. Mohri, M., Minimization algorithms for sequential transducers, *Theor. Comput. Sci.,* 2000, vol. 234, no. 2, pp. 177–201.
9. Zakharov, V.A. and Jaylauova, S.R., On the minimization problem for sequential programs, *Autom. Control Comput. Sci.,* 2017, vol. 51, no. 7, pp. 689–700.
10. Tamm, H., Nykänen, M., and Ukkonen, E., On size reduction techniques for multitape automata, *Theor. Comput. Sci.,* 2006, vol. 363, no. 2, pp. 234–246.
11. Luckham, D.C., Park, D.M., and Paterson, M.S., On formalized computer programs, *J. Comput. Syst. Sci.,* 1970, vol. 4, no. 3, pp. 220–249.
12. Loukanova, R., Linear context free languages, *International Colloquium on Theoretical Aspects of Computing,* Springer, 2007, pp. 351–365.
13. Bedregal, B., λ-ALN: Autômatos lineares não-determinísticos com λ-transições, *Tendencias Mat. Apl. Comput.,* 2011, vol. 12, no. 3, pp. 171–182.
14. Jiraskova, G. and Klima, O., Deterministic biautomata and subclasses of deterministic linear languages, *International Conference on Language and Automata Theory and Applications,* Springer, 2019, pp. 315–327.
15. Holzer, M. and Jakobi, S., Nondeterministic biautomata and their descriptional complexity, *International Workshop on Descriptional Complexity of Formal Systems,* Springer, 2013, pp. 112–123.
16. Klima, O. and Polak, L., On biautomata*, *RAIRO - Theor. Inf. Appl.,* 2012, vol. 46, no. 4, pp. 573–592.
17. Holzer, M. and Jakobi, S., Minimization and characterizations for biautomata, *Fundam. Inf.,* 2015, vol. 136, nos. 1–2, pp. 113–137.
18. Fischer, P.C. and Rosenberg, A.L., Multitape one-way nonwriting automata, *J. Comput. Syst. Sci.,* 1968, vol. 2, no. 1, pp. 88–101.
19. Griffiths, T., The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines, *J. ACM,* 1968, vol. 15, no. 3, pp. 409–413.
20. Ibarra, O., The unsolvability of the equivalence problem for ε-free NGSM's with unary input (output) alphabet and applications, *SIAM J. Comput.,* 1978, vol. 7, no. 4, pp. 524–532.
21. Blattner, M. and Head, T., Single-valued a-transducers, *J. Comput. Syst. Sci.,* 1977, vol. 15, no. 3, pp. 310–327.
22. Schutzenberger, M.P., Sur les relations rationnelles, *Proceedings of the Conference on Automata Theory and Formal Languages,* 1975, pp. 209–213.
23. Blattner, M. and Head, T., The decidability of equivalence for deterministic finite transducers, *J. Comput. Syst. Sci.,* 1979, vol. 19, no. 1, pp. 45–49.
24. Gurari, E.M. and Ibarra, O., A note on finite-valued and finitely ambiguous transducers, *Math. Syst. Theory,* 1983, vol. 16, no. 1, pp. 61–66.
25. Weber, A., On the valuedness of finite transducers, *Acta Inf.,* 1990, vol. 27, no. 8, pp. 749–780.
26. Culik, K. and Karhumäki, J., The equivalence of finite valued transducers (on HDT0L languages) is decidable, *Theor. Comput. Sci.,* 1986, vol. 47, pp. 71–84.
27. Weber, A., A decomposition theorem for finite-valued transducers and an application to the equivalence problem, *Proceedings of the 13th International Symposium on Mathematical Foundations of Computer Science,* Springer, 1988, pp. 552–562.
28. Weber, A., Decomposing finite-valued transducers and deciding their equivalence, *SIAM J. Comput.,* 1993, vol. 22, no. 2, pp. 175–202.
29. Béal, M.-P., Carton, O., Prieur, C., and Sakarovitch, J., Squaring transducers: An efficient procedure for deciding functionality and sequentiality, *Theor. Comput. Sci.,* 2003, vol. 292, no. 1, pp. 45–63.
30. Sakarovitch, J. and de Souza, R., On the decomposition of *k*-valued rational relations, *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science,* 2008, pp. 588–600.

31. Sakarovitch, J. and de Souza, R., On the decidability of bounded valuedness for transducers, *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science,* 2008, pp. 588–600.
32. Sakarovitch, J. and de Souza, R., Lexicographic decomposition of k-valued transducers, *Theory Comput. Syst.,* 2010, vol. 47, no. 3, pp. 758–785.
33. de Souza, R., On the decidability of the equivalence for k-valued transducers, *Proceedings of the 12th International Conference on Developments in Language Theory,* Springer, 2008, pp. 252–263.
34. Zakharov, V.A., Equivalence checking problem for finite state transducers over semigroups, *Proceedings of the 6th International Conference on Algebraic Informatics,* Springer, 2015, pp. 208–221.
35. Weber, A., On the lengths of values in a finite transducer, *Acta Inf.*, 1992, vol. 29, nos. 6–7, pp. 663–687.
36. de Souza, R., On the decidability of the equivalence for a certain class of transducers, *Proceedings of the 13th International Conference on Developments in Language Theory,* Springer, 2009, pp. 478–489.
37. Bird, M., The equivalence problem for deterministic two-tape automata, *J. Comput. Syst. Sci.,* 1973, vol. 7, no. 2, pp. 218–236.
38. Valiant, L.G., The equivalence problem for deterministic finite-turn pushdown automata, *Inf. Control,* 1974, vol. 25, no. 2, pp. 123–133.
39. Beeri, C., An improvement on Valiant's decision procedure for equivalence of deterministic finite turn pushdown machines, *Theor. Comput. Sci.,* 1976, vol. 3, no. 3, pp. 305–320.
40. Friedman, E.P. and Greibach, S.A., A polynomial time algorithm for deciding the equivalence problem for 2-tape deterministic finite state acceptors, *SIAM J. Comput.,* 1982, vol. 11, no. 1, pp. 166–183.
41. Harju, T. and Karhumäki, J., The equivalence problem of multitape finite automata, *Theor. Comput. Sci.,* 1991, vol. 78, no. 2, pp. 347–355.
42. Worrell, J., Revisiting the equivalence problem for finite multitape automata, *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming,* Springer, 2013, pp. 422–433.
43. Bedregal, B., Nondeterministic linear automata and a class of deterministic linear languages, *Preliminary Proceedings LSFA,* 2015, *pp.* 183–196.
44. Bar-Hillel, Y., Perles, M., and Shamir, E., On formal properties of simple phrase structure grammars, *Sprachtypol. Universalienforsch.,* 1961, vol. 14, pp. 143–172.
45. Korenjak, A.J. and Hopcro, J.E., Simple deterministic languages, *7th Annual Symposium on Switching and Automata Theory (SWAT 1966),* IEEE, 1966, pp. 36–46.
46. Friedman, E.P., The inclusion problem for simple languages, *Theor. Comput. Sci.,* 1976, vol. 1, no. 4, pp. 297–316.
47. Caucal, D., A fast algorithm to decide on simple grammars equivalence, *International Symposium on Optimal Algorithms,* Springer, 1989, pp. 66–85.
48. Bastien, C., Czyzowicz, J., Fraczak, W., and Rytter, W., Prime normal form and equivalence of simple grammars, *International Conference on Implementation and Application of Automata,* Springer, 2005, pp. 78–89.
49. Hirshfeld, Y., Jerrum, M., and Moller, F., A polynomial algorithm for deciding bisimilarity of normed context-free processes, *Theor. Comput. Sci.,* 1996, vol. 158, nos. 1–2, pp. 143–159.
50. Valiant, L.G. and Paterson, M.S., Deterministic one-counter automata, *J. Comput. Syst. Sci.,* 1975, vol. 10, no. 3, pp. 340–350.
51. Böhm, S. and Göller, S., Language equivalence of deterministic real-time one-counter automata is NL-complete, *International Symposium on Mathematical Foundations of Computer Science,* Springer, 2011, pp. 194–205.
52. Sénizergues, G., The equivalence problem for t-turn dpda is co-NP, *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming,* Springer, 2003, pp. 478–489.
53. Linna, M., Two decidability results for deterministic pushdown automata, *J. Comput. Syst. Sci.,* 1979, vol. 18, no. 1, pp. 92–107.
54. Meytus, V.Y., The equivalence problem for real-time strict deterministic pushdown automata, *Cybernetics,* 1989, vol. 25, no. 2, pp. 581–594.
55. Oyamaguchi, M., The equivalence problem for real-time DPDAs, *J. ACM,* 1987, vol. 34, no. 3, pp. 731–760.
56. Romanovskii, V.Y., Equivalence problem for real-time deterministic pushdown automata, *Cybernetics,* 1985, vol. 22, no. 2, pp. 162–175.
57. Rosenkrantz, D.J. and Stearns, R.E., Properties of deterministic top-down grammars, *Inf. Control,* 1970, vol. 17, no. 3, pp. 226–256.
58. Tomita, E., An extended direct branching algorithm for checking equivalence of deterministic pushdown automata, *Theor. Comput. Sci.,* 1984, vol. 32, nos. 1–2, pp. 87–120.
59. Ukkonen, E., The equivalence problem for some non-real-time deterministic pushdown automata, *J. ACM,* 1982, vol. 29, no. 4, pp. 1166–1181.
60. Sénizergues, G., The equivalence problem for deterministic pushdown automata is decidable, *Proceedings of the 24th International Colloquium on Automata, Languages, and Programming,* Springer, 1997, pp. 671–681.

61. Stirling, C., Deciding DPDA equivalence is primitive recursive, *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming,* Springer, 2002, pp. 821–832.
62. Madhavan, R., Mayer, M., Gulwani, S., and Kuncak, V., Automating grammar comparison, *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications,* 2015, pp. 183–200.
63. Zakharov, V.A., Program equivalence checking by two-tape automata, *Cybern. Syst. Anal.,* 2010, vol. 46, no. 4, pp. 554–562.
64. Olshansky, T. and Pnueli, A., A direct algorithm for checking equivalence of LL(k) grammars, *Theor. Comput. Sci.,* 1977, vol. 4, no. 3, pp. 321–349.
65. Karhumäki, J., Equations over finite sets of words and equivalence problems in automata theory, *Theor. Comput. Sci.,* 1993, vol. 108, no. 1, pp. 103–118.
66. Okhotin, A., Decision problems for language equations, *J. Comput. Syst. Sci.,* 2010, vol. 76, nos. 3–4, pp. 251–266.
67. Hopcro, J.E., Motwani, R., and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, 2006, 3rd ed.
68. Eilenberg, S., *Automata, Languages, and Machines,* Academic Press, 1974.
69. Han, Y., Salomaa, K., and Wood, D., State complexity of prefix-free regular languages, *DCFS,* 2006, pp. 165–176.
70. Martelli, A. and Montanari, U., An efficient unification algorithm, *ACM Trans. Program. Lang. Syst.,* 1982, vol. 4, no. 2, pp. 258–282.
71. Arden, D., Delayed-logic and finite-state machines, *Proceedings of 2-nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961),* IEEE, 1961, pp. 133–151.
72. Hopcroft, J.E., *A Linear Algorithm for Testing Equivalence of Finite Automata,* Def. Tech. Inf. Cent., 1971, vol. 114.
73. Zakharov, V.A. and Temerbekova, G.G., On the minimization of finite state transducers over semigroups, *Autom. Control Comput. Sci.,* 2017, vol. 51, no. 7, pp. 523–530.
74. Itkin, V.E., Logic-term equivalence of program schemes, *Kibern. Sist. Anal.,* 1972, no. 1, pp. 5–27.