On the problem of vertex optimization of bracketed automata

T. V. Generalova, B. F. Melnikov, A. A. Vylitok

***Abstract*—A new formalism for the specification of context-free languages is considered.**

In this formalism, the application of an auxiliary alphabet and the imposition of additional conditions make it possible to obtain an extension of the class of nondeterministic finite automata. This approach allows receiving a mechanism that recognizes context-free languages.

Despite the fact that we define the class of context-free languages, this formalism is similar to the nondeterministic finite automata. This circumstance allows using classic algorithms of the equivalent transformation of nondeterministic finite automata for objects of formalism that specifies the context-free languages.

As such a formalism, automata of a special kind, so-called bracketed automata, are considered. We examine the algorithm for constructing a bracketed automaton according to the given context-free grammar. Then we give an example of a context-free grammar with iterations for the model of arithmetic expressions.

Also, we consider some equivalent transformations of bracketed automata. We represent a new special alphabet and prove that on the basis of the alphabet, for each bracketed automaton the usual nondeterministic finite automaton can be constructed. Vise versa, for each nondeterministic finite automaton over a new alphabet, it is possible to construct an equivalent bracketed automaton.

Everything done in the paper makes it possible to apply various algorithms of equivalent transformations of nondeterministic finite automata, such as constructing of a minimal automaton, universal automaton, etc., and obtain objects of the considered formalism which is more acceptable in terms of some characteristics, for example, with fewer numbers of the vertices or the edges.

We give an example that shows that with a given transformation it is possible to reduce the number of vertices of the automaton, but it is not always possible to obtain an automaton with a minimum number of vertices.

***Keywords*—nondeterministic finite automata, context-free languages, the extension of nondeterministic automata, algorithms for equivalent transformation.**

I. Introduction and motivation

There exist a lot of formal systems for describing context-free languages. Along with systems of the generative type, for example, a grammar, there are recognition systems that are algorithms, possibly in the form of an automaton.

Context-free languages are specified by pushdown automata [1]. There also exist some other approaches for the description of the languages. For instance, a graphical method of the language representation is considered in [2]. We also mention the well-known book [3] and a series of the papers [4], [5], [6].

In this paper, we consider a new formalism for describing context-free languages [7]. On the one hand, it is an extension of the class of nondeterministic finite automata (NFA).

On the other hand, it can be considered as nondeterministic finite automaton over a special alphabet. Each element of this alphabet symbolizes *a pair of brackets*: the opening bracket and the closing one. We suppose that the sets of opening and closing brackets are not intersected.

In [8], a formalism for representing a special extension of finite automata was introduced. They were called generalized nondeterministic finite pseudo-automata. Unlike usual automata constructions, these automata do not indicate the concrete paths for defining the considered word of the given regular languages. This formalism gives only an algorithm for answering the question, whether or not the given word belongs to the considered language.

The extensions of the class of finite automata usually define the same class of regular languages. We can say that non-determinism was the first such extension and was considered firstly in terms of our approach, apparently, in [9]. We mention the works of one of the authors of this paper [10], [11] for other extensions of the class of finite automata.

However, we note that these extensions of the class of the automata *do not* expand the class of the languages, they define. This class remains the class of regular languages. In our case, *our* extension of the class of finite automata gives an extension of the class of corresponding languages, and the class of context-free languages is obtained.

We can get the context-free language due to only some of the words accepting by the automaton construct the total language.

Thus, everything done in the paper makes it possible to apply various algorithms of equivalent transformations of nondeterministic finite automata, such as:

- *constructing the equivalent automaton with the minimal possible number of states (simply so-called minimal automaton);*
- *constructing the equivalent automaton with the minimal possible number of edges;*
- *constructing the equivalent universal automaton;*
- *constructing an automaton according to the basis one;*
- *etc.,*

and obtain in this way objects of the formalism which are more acceptable in terms of some characteristics, for example, with fewer numbers of vertices, edges, etc.

In [13], it is shown how by means of successive transformations it is possible to get an equivalent for any given automaton for a given regular language. This is possible by virtue of the existence of a canonical finite automaton, which

Received 5.08.2018.

Tatiana V. Generalova, Lomonosov Moscow State University (email: tanya.generalova@gmail.com).

Boris F. Melnikov, Russian State Social University (email: bf-melnikov@yandex.ru).

Alexey A. Vylitok, Lomonosov Moscow State University (email: vylitok@cs.msu.su).

in turn is equivalent to the automata considered (participating in the transformation).

As algorithms for the equivalent transformation of the nondeterministic finite automaton, algorithms for combining several states into one are considered, as well as an algorithm for adding loops. The latter algorithm makes it possible to add loops that were not present in the original finite automaton, but are present in the equivalent basis one.

The algorithms of equivalent transformations of nondeterministic finite automata mentioned by us are described by various authors, including the author of this paper. Among many publications on this topic, we mention those that, from our point of view, are most interesting and are related to the questions considered in this paper: ours [10] – [15], and the publication of other authors [16], [17], [18].

The structure of this paper is as follows. In Section II, we briefly give the notion of the nondeterministic finite automaton: ordinary and extended. Besides, we recall the notion of *D-graphs* and *perfect pushdown automata*. (The concept of an extended pushdown automaton is used to define the concept of a perfect pushdown automaton.) The idea of designing D-graph was to represent the context-free languages graphically.

In Section III, we define bracketed automata over a special alphabet and their languages. Then we prove the theorem that the language recognized by a bracketed automaton is always a context-free language.

In Section IV, we present one of the possible methods of constructing bracketed automaton according to a context-free grammar. In particular, we present an algorithm for constructing bracketed automaton according to a given context-free grammar. Then we show the corresponding example of the application of such an algorithm.

The example of the equivalent transformation of bracketed automata is considered in Section V. (Language descriptions are called equivalent if they specify the same language.)

Such transformations show that for each bracketed automaton, on the base on the new alphabet the equivalent ordinary nondeterministic finite automaton can be built. Vise versa, for each nondeterministic finite automaton over a new alphabet, it is possible to construct an equivalent bracketed automaton.

In some cases with the help of such transformations, we can obtain a finite automaton with the minimum possible number of vertices.

We study this transformation more thoroughly and offer an example in which we show that this transformation does not always allow to obtain a minimal bracketed automaton. Thus, in the most general case, this transformation cannot be considered as an algorithm for minimization of the bracketed automata.

In Conclusion (Section VI), we briefly summarize the results of the paper and formulate the main directions for further research on this topic.

II. Preliminaries

A. Nondeterministic finite automaton: classical definitions and additional information

We shall use the notation from [14] for nondeterministic finite automata (NFA for short).

Definition 1: [14] Let

$$K = (Q, \Sigma, \delta, S, F) \tag{1}$$

be a nondeterministic finite automaton that defines language, denoted $\mathcal{L}(K)$, where

- Q is a finite set of states,
- S is a set of the initial states of the finite state control, $S \subseteq Q$,
- F is a set of the final states, $F \subseteq Q$, and
- δ is a transition function
$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q),$$
where $\mathcal{P}(Q)$ is a finite subset of Q. Remark, that we shall admit ε-transitions. □

We get all possible next states of automaton due to state transition function according to the given "current" state and "current" input symbol [1]. The NFA in the current state goes into each of its possible states in accordance with the read symbol. It is assumed that an automaton accepts an input string if any of its parallel instances reaches an accepting state.

We shall also use another method for specifying the transition function of the finite automata in addition to the described one. For this method, we shall define transition function

$$\gamma : Q \times Q \to \mathcal{P}(\Sigma \cup \{\varepsilon\}),$$

and condition $\gamma(q, r) \ni a$ is fulfilled if and only if

$$\delta(q, a) \ni r \text{ [11].}$$

We assume that $q, r \in Q$ and $a \in \Sigma \cup \{\varepsilon\}$.

B. A nondeterministic pushdown automaton and an extended one

Definition 2: [1] A nondeterministic pushdown automata is defined by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$

where

- Q is a finite set of states,
- Σ is the input alphabet,
- Γ is a finite set of symbols,
- δ is a transition function
$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \in \Gamma \to Q \times \Gamma^*$$
- $q_0 \in Q$ is the initial state,
- $z \in \Gamma$ is the stack start symbol, and
- $F \subseteq Q$ is the set of final states. □

Definition 3: [1] We shall represent a pushdown list as a string of symbols Γ^* with the topmost symbol written on the left. □

Definition 4: [1] Let an extended pushdown automaton (PDA for short) be a 7-tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

where δ is a mapping from a finite subset of

$$Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*$$

to the finite subsets of $Q \times \Gamma^*$ and all other symbols have the same meaning as before. □

We give some more information from [1].

Definition 5: [1] A *configuration* of the automaton is a triple

$$(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*,$$

where

- q represents a current state of the finite control,
- w represents the unused portion of the input; the first symbol of w is under the input head; if $w = \varepsilon$, then it is assumed that all of the input tapes have been read,
- α represents the contents of the pushdown list; the leftmost symbol of α is the topmost pushdown symbol; if $\alpha = \varepsilon$, then the pushdown list is assumed to be empty. □

A *move* by pushdown automaton will be represented by the binary relation $\vdash$ on configurations. For ordinary pushdown automaton, we write

$$(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha) \tag{2}$$

if $\delta(q, a, Z)$ contains (q', γ) for any $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Sigma^*$, $Z \in \Gamma$, and $\alpha \in \Gamma^*$.

A configuration of the extended pushdown automaton is as before and a move by PDA will be represented as

$$(q, aw, \alpha\gamma) \vdash (q', w, \beta\gamma) \tag{3}$$

if $\delta(q, a, \alpha)$ contains (q', β) for $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $\alpha \in \Gamma^*$. In this move, the string α is replaced by the string β on top of the pushdown list. In our further considerations, we shall consider extended pushdown automata. □

Let us suppose that current state of finite control P is q, the current input symbol is a, and the finite-length string on top of the pushdown list is α. If $a \neq \varepsilon$, then (3) states that in such configuration P may go into a configuration in which the finite control is now in state q', the input head has been shifted one square to the right, and the topmost string α on the pushdown list has been replaced by the string β of pushdown list symbols.

An initial configuration of P is one of the form (q_0, w, Z_0) for some $w \in \Sigma^*$. That is, the finite state control is in the initial state, the input contains the string to be recognized, and the pushdown list contains only the symbol Z_0. A final configuration is one of the form (q, ε, α), where

$$q \in F,\ \alpha \in \Gamma^*.$$

Definition 6: [1] We say that a string $w \in \Sigma^*$ is accepted by P if

$$(q, w, Z_0) \overset{*}{\vdash} (q, \varepsilon, \alpha)$$

for some $q \in F$ and $\alpha \in \Gamma^*$.

The language defined by PDA P, is

$$L(P) = \{w \,|\, (q_0, w, Z) \overset{*}{\vdash} (q, \varepsilon, \alpha),\ q \in F,\ \alpha \in \Gamma^*\}.\ \square$$

Extended pushdown automata can replace a finite-length string of symbols on top of the pushdown list by some other finite-length string in a single move. The original version of PDA could replace only the topmost symbol on the top of the pushdown list on a given move.

Unlike an ordinary PDA, an extended pushdown automaton is capable of making moves when its pushdown list is empty.

C. D-graphs

The notion of *D-graphs* was designed with the purpose to represent the context-free languages graphically [5]. We introduce auxiliary notions and notations. Let $\Sigma_($ and $\Sigma_)$ be non-intersecting alphabets and there is a bijection

$$\phi : \Sigma_(\to \Sigma_).$$

We shall denote a nonempty set $\mathcal{P} \subseteq \Sigma_(\times \Sigma_)$ as *D-set*. We shall call the language generated by the grammar

$$S \to \Lambda \,|\, a\, S\, b\, S,\ (a,\ b)\ \in \mathcal{P},$$

as *D-language*.

Suppose that for any D-set $\mathcal{P} \subseteq \Sigma_(\times \Sigma_)$ records *Left*($\mathcal{P}$) and *Right*($\mathcal{P}$) denote the sets

$$\{a \in \Sigma_(\,|\, \exists\, b \in \Sigma_),\ (a, b) \in \mathcal{P}\}$$

and

$$\{b \in \Sigma_) \,|\, \exists\, a \in \Sigma_(,\ (a, b) \in \mathcal{P}\},$$

respectively.

Definition 7: [5] We define D-graph as follows:

$$D = (V, \Sigma, \mathcal{P}, \lambda, P_0, F),$$

where:

- V is a finite set of vertices,
- Σ is an alphabet of marks of the edges,
- $\mathcal{P}$ is a D-set, union $E(D)$ = *Left*($\mathcal{P}$) ∪ *Right*($\mathcal{P}$) is a set of oriented loaded edges,
- $P_0 \subseteq V$ is the initial vertex,
- $F \subseteq V$ is the set of finite vertices, and
- $\lambda : E(D) \to V \times (\Sigma \cup \{\varepsilon\}) \times V$ is the edge position function in D-graph. □

We denote some edge in D-graph as π. The elements of tuple

$$(P, a, Q) \in V \times (\Sigma \cup \{\varepsilon\}) \times V,$$

$$(P, a, Q) = \lambda(\pi),$$

where

- P is the initial vertex,
- a is the mark, and
- Q is the finite vertex.

We denote $\omega(\pi)$ as *edge mark*, $\omega(\pi) = a$.

Definition 8: We define the edge mark notion recursively: edge mark of empty (trivial) route is empty. For the route $T\pi$, where

- $\pi \in E(D)$
- $\lambda(\pi) = (P, a, Q)$ for some
  - $P, Q \in V$
  - $a \in \Sigma \cup \{\varepsilon\}$,

edge is $\omega(T\pi) = \omega(T)\pi$.

For edge π:

- $beg(\pi)$ is the *initial vertex of the edge*, $beg(\pi) = P$,
- $end(\pi)$ is the *finite vertex of the edge*, $end(\pi) = Q$. □

We consider the example of D-graph (Fig. 1):

$$D_1 = (\{a, b\}, \{P, Q\}, \{(1, 2), (1, 3)\}, \lambda, P, \{Q\}),$$

where

$$\lambda(1) = P\, a\, P,\quad \lambda(2) = P\, b\, Q,\quad \lambda(3) = Q\, b\, Q.$$

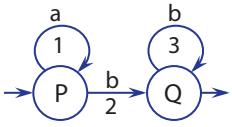

Fig. 1. An example of D-graph.

D. Perfect pushdown automaton

Definition 9: [6] Pushdown automaton is a tuple

$$M = (\Sigma, K, \Gamma, Z_0, \delta, p_0, F),$$

where

- Σ is a finite input alphabet,
- K is a finite set of states,
- Γ is a finite alphabet of pushdown list symbols,
- Z_0 is the symbol that appears initially on the pushdown list (the start symbol),
- p_0 is the initial state of the finite control,
- $F \subseteq K$ is the finite set of final states,
- $\delta \subseteq K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \ K \times \Gamma^*)$ is the *finite set of commands.* □

The notion of the command is convenient for comparing pushdown automata with D-graphs.

The language $L(M)$ accepted by the automaton is determined using the concept of the configuration and the binary relationship of attainability, given on the set of configurations.

We shall assume that

$$p, q \in K, \quad a \in \Sigma \cup \{\varepsilon\}, \quad X, Z \in \Gamma, \quad \gamma \in \Gamma^*.$$

The command is denoted by a record $(p, a, Z) \to (q, \gamma)$.

Let

$$p \in K, \quad x, y \in \Sigma^*, \quad \gamma \in \Gamma^*,$$
$$((p_0,\, x,\, Z_0),\ (p,\, y,\, \gamma)) \in \vdash_M^*.$$

Then we call the triple (p, y, γ) the configuration of the pushdown automaton. The configuration (p_0, x, Z_0) is called the initial configuration, the configuration (p, ε, γ), where $p \in F$, is the final one.

The language can be defined by the formula

$$L(M) = \{x \in \Sigma^* \mid \exists\, (p\ \in F, \gamma \in \Gamma^*)$$
$$(p_0, x, Z_0) \vdash_M^*\ (p, \varepsilon, \gamma)\}.$$

Now we introduce the notion of a perfect pushdown automaton.

Definition 10: [6] Let the pushdown automaton satisfies the following conditions:

- the trace of any command is $+X$ or $-X$ for some $X \in \Gamma$,
- a configuration has the form $(p, x, Z_0\gamma)$, where $\Gamma\ \in (\Gamma \backslash \{Z_0\})$,
- the final configuration has the form $(f, \epsilon, Z_0), f \in F$.

Then we call the automaton perfect. □

The following lemma is used in further consideration.

Lemma 1: [6] For a pushdown automaton M, there exists a perfect pushdown automaton M' such that

$$L(M) = L(M'). \square$$

In [6] for perfect pushdown automaton some following notions are introduced: a *memory* of configuration of automaton, a concept of *computation* over memory, a *mark* of the calculation, a *trace* of the calculation, a *final memory*, and a *length* of calculation.

Also, the concept of a *route* is introduced. Let $(p, +Z)$ be the *initial memory* of some empty route. Then the pair (p, Z) is the *vertex of the automaton* (and the route). If p is the final state, then the vertex (p, Z) is said to be *final*. The pair (p_0, Z_0) is called the *input vertex*.

III. Bracketed automata

In this section, we define bracketed automata and their languages. After that, we prove that the language recognized by the bracketed automaton is always a context-free language.

A. Special automata for accepting context-free languages

For every n from set $\mathbb{N}_0$, we shall consider the sets

$$\mathbb{N}_{(n)} = \{\, 1, 2, \ldots, n{-}1, n \,\}$$

and

$$\mathbb{Z}_{(n)} = \{\, -n, -(n{-}1), \ldots, -2, -1, 0, 1, 2, \ldots, n{-}1, n \,\}.$$

Each element i of $\mathbb{N}_{(n)}$ symbolizes i-numbered pair of brackets. Also i symbolizes i-numbered opening bracket, and $-i$ denotes corresponding closing bracket.

Sometimes, we shall consider the set $\mathbb{Z}_{(n)}$ as the *alphabet* containing $2n{+}1$ symbols, i.e. ${\Sigma_{(n)}}^*$, and, therefore, we shall consider words and languages over $\mathbb{Z}_{(n)}$.

For example, we can say that

$$4\ 0\ 4\ {-4}\ 3\ {-3}\ {-4}\ 0\ 3\ {-3}\ 0 \tag{4}$$

is a word over $\mathbb{Z}_{(17)}$.

Definition 11: For given $n \geq 0$, we define language $[{\mathbb{Z}_{(n)}}^*]$. We call it the language matched by brackets and every word of this language is a word matched by brackets.

We define the word matched by brackets recursively:

- ε and 0 are words matched by brackets;
- if w and v are words matched by brackets, then $u = wv$ is also word matched by brackets;
- if w is a word matched by brackets, $i \in \mathbb{N}_{(n)}$, then we denoted u a word matched by brackets, and
$$u = iw - i;$$
- other word is not word matched by brackets.

The word $v \in [{\mathbb{Z}_{(n)}}^*]$ is called matched prefix if it is a prefix of the word $u \in [{\mathbb{Z}_{(n)}}^*]$. □

Let us comment on this definition. We get a new word matched by brackets from the existing one by putting the latter in brackets and (or) assigning to it another word matched by brackets. For example, the word (4) is word matched by brackets.

Definition 12: We define a bracketed automaton B as follows:

$$B = (\, Q, \Sigma, \text{ϒ}, S, F, n \,), \tag{5}$$

where

- Q is a finite set of states,
- Σ is a given alphabet,
- S is a set of initial states of finite state control,
- F is a set of final states of Q,
- $n \in \mathbb{N}_0$ defines the bracket set $\mathbb{Z}_{(n)}$, and
- ϒ is a transition function of the type
$$\text{ϒ} : Q \times Q \to \mathcal{P}\left((\Sigma \cup \{\varepsilon\}) \times \mathbb{Z}_{(n)}\right).$$

We consider that we define simultaneously the functions ϒ_γ and ϒ_ζ

$$\text{ϒ}_\gamma : Q \times Q \to \mathcal{P}\,(\Sigma \cup \{\varepsilon\}),$$
$$\text{ϒ}_\zeta : Q \times Q \to \mathcal{P}\left(\mathbb{Z}_{(n)}\right).$$

In this case, if condition

$$\text{ϒ}(q', q'') \ni (a, i)$$

is fulfilled for the states $q', q'' \in Q$, then we assume that conditions

$$\gamma(q', q'') \ni a \quad \text{and} \quad \zeta(q', q'') \ni i$$

are fulfilled.

Functions γ and ζ do not contain other values. □

We shall denote that tuple $(Q, \Sigma, \gamma, S, F)$ can be considered as an ordinary nondeterministic finite automaton for the bracketed automaton (5). We shall denote $\mathcal{L}_\gamma(B)$ a language defined by this automaton. We shall also designate a notation $\mathcal{L}_\zeta(B)$ for the language of the automaton $(Q, \mathbb{Z}_{(n)}, \zeta, S, F)$.

Let us define the language of the automaton (5) and denote it as $\mathcal{L}(B)$.

Definition 13: Let us suppose that for the sequence of states $q_0, q_1, \ldots, q_m \in Q$ the following conditions

- $q_0 \in S$, $q_m \in F$,
- $\gamma(q_k, q_{k+1}) \ni (a_k, i_k)$ for each $k \in \{0, \ldots, m-1\}$,
- $i_1\, i_2 \ldots i_m \in [\mathbb{Z}_{(n)}{}^*]$

are fulfilled. Then we believe that the word $a_1 a_2 \ldots a_m$ belongs to $\mathcal{L}(B)$. The language $\mathcal{L}(B)$ does not contain other words. □

Definition 14: The bracketed automata B_1 and B_2 are called equivalent if $\mathcal{L}(B_1) = \mathcal{L}(B_2)$. □

Theorem 1: Language recognized by the automaton (5) is the context-free language.

Proof. We shall prove this fact by constructing corresponding pushdown automaton in the following way. We shall use the *extended* pushdown automaton accepting words by empty of *pushdown list* (see the strict definition in [1, Ch. 2.5.2]).

For a given bracketed automaton B (5), we shall consider pushdown automaton

$$\mathcal{P} = (\, Q \cup \{p_0\},\, \Sigma,\, \Gamma,\, \delta,\, p_0,\, z_0,\, F\,), \tag{6}$$

where

- Q, Σ and F coincide with corresponding objects of automaton B,
- $\Gamma = \mathbb{N}_{(n)} \cup \{z_0\}$,

and transition function δ is defined in the following way:

- for each s of the set S, the condition

  $$\delta(p_0, \varepsilon, z_0) \ni (s, z_0)$$

  is fulfilled;
- an edge of an automaton will be called the transition of the automaton from one state to another; for each edge $\gamma(q', q'') \ni (a, i)$ of the automaton B (where $a \in \Sigma \cup \{\varepsilon\}$ and $i \in \mathbb{Z}_{(n)}$) and each $z \in \Gamma$ the following condition holds:
  - if $i = 0$, then $\delta(q', a, z) \ni (q'', z)$,
  - if $i > 0$, then $\delta(q', a, \varepsilon) \ni (q'', i)$,
  - if $i < 0$, then $\delta(q', a, -i) \ni (q'', \varepsilon)$; let us note once again, that $-i$ is a letter of the alphabet $\mathbb{Z}_{(n)}$.

Let us consider a word $u \in \Sigma^*$ recognized by the automaton (5). Let u be

$$u = a_1\, a_2\, a_3\, \ldots\, a_n\,.$$

The operation of the automaton B while accepting of the word u can be represented as it shown on Fig. 2.

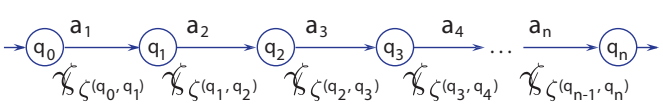

Fig. 2. The operation of the automaton B while accepting of the word u

This scheme can be interpreted as the operation of pushdown automaton over word u.

The command of a pushdown automaton is a production that translates the automaton from configuration to another one. We consider that the transitions from state to state are defined by the commands:

- $\delta(p_0, \varepsilon, z_0) \ni (q_0, z_0)$;

for each k from set $\{0, \ldots, n-1\}$:

- if $\zeta(q_k, q_{k+1}) > 0$,
  then $\delta(q_k, a_{k+1}, \varepsilon) \ni (q_{k+1}, \zeta(q_k, q_{k+1}))$,
- if $\zeta(q_k, q_{k+1}) = 0$ and $z \in \Gamma$,
  then $\delta(q_k, a_{k+1}, z) \ni (q_{k+1}, z)$,
- if $\zeta(q_k, q_{k+1}) < 0$,
  then $\delta(q_k, a_{k+1}, -\zeta(q_k, q_{k+1})) \ni (q_{k+1}, \varepsilon)$.

By the construction of the pushdown automaton, all transitions from state to state have one of the following forms:

1) $\delta(q, a, z) \ni (p, z)$,
2) $\delta(q, a, \varepsilon) \ni (p, z)$,
3) $\delta(q, a, z) \ni (p, \varepsilon)$, where $z \in \Gamma,\ p, q \in Q \cup \{p_0\}$.

Therefore, the proof that any word accepted by pushdown automaton (6) is also accepted by bracketed automaton (5) is carried out in a similar way according to the scheme in Fig. 2. □

In this paper, we do not prove that each context-free language can be represented by the bracketed automaton. This problem will be one of the topics of further work. We shall sketch only the possible ways of such the proof. The first way is connected with the construction of the bracketed automaton according to the pushdown automaton.

Let us denote that the form of the instructions 1–3 from the proof of Theorem 1 does not restrict the class of languages accepted by an extended pushdown automaton by empty pushdown list.

Therefore, the work of the pushdown automaton over the word u similar to shown in Fig. 1, and this scheme can also be interpreted as the work of the bracketed automaton.

Another way assumes that we can consider D-graph as the bracketed automaton.

D-graphs is known to specify exactly context-free languages [5], [6].

Theorem 2: The language is a context-free if and only if it is determined by some D-graph.

We shall remind briefly the main ways of such a proof [6]. At first, we prove that specific D-graph is equivalent to an arbitrary pushdown automaton constructively. Then we use the construction of D-graph from this proof to transform an arbitrary D-graph into an equivalent pushdown automaton.

B. Transformation of a pushdown automaton into a D-graph

The concept of the route is the key notion. In the case of a perfect magazine, the second element of the final vertex is always Z_0. The notion of the nest of the route is introduced. The sections of the route that form the nest are paired with each other. The aggregate of marks of successful routes of

the perfect pushdown automaton is the language accepted by the automaton.

Let $V(M)$ is a set of all vertices defined by empty sections of successful routes of the automaton M, $\mathcal{P}(M)$ is the set of all pairs of edges such that π_1 and π_2 form a pair in some successful route of the automaton M.

Then D-graph

$$Graph(M) = (\Sigma, V(M), \mathcal{P}(M), \lambda(M), (p_0, Z_0), \{(p, Z_0)|p \in F)\}),$$

where the position function $\lambda(M)$ matches the edge

$$\pi \in \textit{Left}(\mathcal{P}(M)) \cup \textit{Right}(\mathcal{P}(M))$$

to the triple $(begin(\pi), \omega(\pi), end(\pi))$, is equivalent to an automaton M. The D-graph $Graph(M)$ is called the *graph of a pushdown automaton*. This completes the construction of the D-graph according to the pushdown automaton.

C. Conversion of a D-graph into a pushdown automaton

The concept of a D-graph is used to prove the fact that for every D-graph there exists an equivalent pushdown automaton.

In [6], it is shown an algorithm that, from an arbitrary D-graph D, first obtains the equivalent D-graph G, which is the graph of some pushdown automaton, then converts the edges of the resulting D-graph into automaton commands. □

The proof of Theorem 1 can also be carried out using the D-graphs theory: we shall get D-graph if we substitute every edge in the bracketed automaton with ζ-mark 0 by a pair of edges with the unique opening and closing brackets.

IV. Context-free grammars to bracketed automata

In this section, we present a method of constructing a bracketed automaton in accordance with a given context-free grammar.

Firstly, we shall describe the general constructing algorithm, and then demonstrate its work by example.

The strict proof of the correctness of the considered algorithm, in other words the proof of the equivalence of the given context-free grammar and the obtained bracketed automaton, is the subject of the further publication.

A. An algorithm for transforming a context-free grammar to an equivalent bracketed automaton

Algorithm 1: *(Bracketed automaton construction from a given context-free grammar)*

Input: A context-free grammar.

Output: Corresponding bracketed automaton.

Method:

Step 1. Number all the nonterminals of context-free grammar and build a syntax diagram for each grammar production according to [19].

Step 2. Transform each diagram into a graph as follows:

a) transform the input and output edges into a pair of vertices marked with nonterminal that defines by syntax diagram; mark these vertices by indices 1 and 2, respectively; transform other edges into vertices with arbitrary pairwise different labels,

b) replace all the terminal vertices of diagrams by the edges with according terminal labels (for transition function γ) and label 0 (for transition function ζ).

Step 3. Combine received graphs.

For combining graphs for each i-numbered terminal A we shall implement the following:

a) substitute the entries into each corresponding vertex that also marked with A label by the transitions with label $+i$ into each vertex marked with A_1;

b) similarly, substitute the exits from the corresponding vertices (also labeled as A) by the transitions marked with $-i$ from each vertex marked with A_2.

c) delete the vertices marked with A. □

The bracketed automaton is determined by the constructed graph. The vertices of the graph are corresponding to automaton states. The transition function is determined by edges. The initial and final states are corresponding to the pair of vertices obtained from the initial symbol of the original grammar.

B. Example of constructing bracketed automaton according to a context-free grammar

Let us consider the example of context-free grammar with iterations for model arithmetic expressions for this algorithm:

$$E \to T\,\{+T\}, \tag{7}$$

$$T \to F\,\{*F\}, \tag{8}$$

$$F \to a \,|\, b \,|\, (E). \tag{9}$$

The nonterminal E symbolizes the expression, it is the initial symbol of the grammar or axiom. The nonterminal T denotes the term, the nonterminal F is the factor, and terminals a, b are the simple expressions (variables or constants).

The following syntax diagrams according to a given grammar (Fig. 3a) are constructed in an obvious manner.

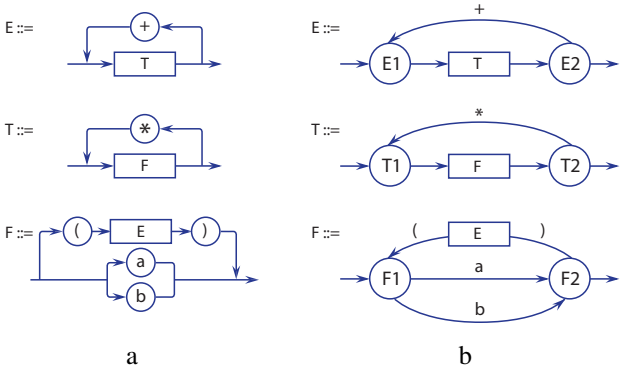

Fig. 3. a) Syntax diagrams for the expressions (7)–(9), Step 1; b) Graphs based on the syntax diagrams Fig. 3a, Step 2

Transform each diagram into a graph according to step 2 of the algorithm (Fig. 3b).

Substitute the transition to the subgraph T and returning from one into the graph E (Fig. 4a).

Similarly, substitute the transition to subgraph and returning from one for F (Fig. 4b).

Implement recursion for the obtained graph; for this, make a transition to E_1 and returning from E_2 (Fig. 5).

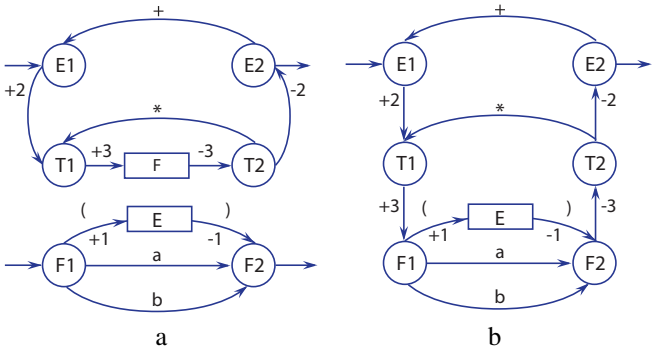

Fig. 4. a) Graphs based on the syntax diagrams Fig. 3a, Step 2; b) Graph based on the syntax diagrams Fig. 3a, Step 3

As a result, we obtained a graph for the arithmetic expressions.

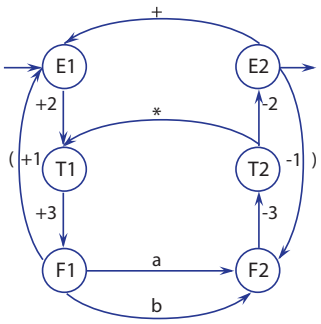

Fig. 5. Graph based on the syntax diagram Fig. 3a

We consider the example of parsing expression $a*(a+b)$:

$$E_1 \xrightarrow[+2]{} T_1 \xrightarrow[+3]{} F_1 \xrightarrow{a} F_2 \xrightarrow[-3]{} T_2 \xrightarrow{*} T_1 \xrightarrow[+3]{} \rightarrow F_1 \xrightarrow[+1]{(} E_1 \xrightarrow[+2]{} T_1 \xrightarrow[+3]{} F_1 \xrightarrow{a} F_2 \xrightarrow[-3]{} T_2 \xrightarrow[-2]{} \rightarrow E_2 \xrightarrow{+} E_1 \xrightarrow[+2]{} T_1 \xrightarrow[+3]{} F_1 \xrightarrow{b} F_2 \xrightarrow[-3]{} T_2 \xrightarrow[-2]{} \rightarrow E_2 \xrightarrow[-1]{)} F_2 \xrightarrow[-3]{} T_2 \xrightarrow[-2]{} E_2.$$

V. Equivalent Transformations of Bracketed Automata

Let us introduce a new alphabet $\Psi = \Sigma \times \mathbb{Z}_{(n)}$. It consists of the symbols of the alphabet Σ and the elements that symbolize the pair of brackets.

For this alphabet, we shall consider the ordinary nondeterministic finite automaton constructed on the base of the given bracketed automaton (3)

$$\mathcal{K}(B) = (\, Q, \Psi, \delta_{\chi}, S, F\,),$$

where the transition function δ_{χ} is constructed as follows:

- for every edge $\chi(q', q'') \ni (a, i)$ of the automaton B, the edge $\delta_{\chi}(q', (a, i)) \ni q''$ of the automaton $\mathcal{K}(B)$ does exist;
- anything else is not an edge of the automaton $\mathcal{K}(B)$.

Now we shall consider the "inverse" transformation. Let us consider a nondeterministic finite automaton

$$K = (\, \hat{Q}, \Psi, \hat{\delta}, \hat{S}, \hat{F}\,),$$

and define the corresponding bracketed automaton

$$\mathcal{B}(K) = (\, \hat{Q}, \Psi, \hat{\chi}_{\hat{\delta}}, \hat{S}, \hat{F}, n\,),$$

where the value n can be chosen based on the letter of the alphabet $\Psi = \Sigma \times \mathbb{Z}_{(n)}$, and the transition function $\hat{\chi}_{\hat{\delta}}$ is constructed in a similar way: the edge $\hat{\chi}_{\hat{\delta}}(q', q'') \ni (a, i)$ of the automaton $\mathcal{B}(K)$ exists for every edge $\hat{\delta}(q', (a, i)) \ni q''$ of the automaton K.

Theorem 3: Let K_1 and K_2 be some nondeterministic finite automata over the alphabet Ψ, and $\mathcal{L}(K_1) = \mathcal{L}(K_2)$. Then $\mathcal{L}(\mathcal{B}(K_1)) = \mathcal{L}(\mathcal{B}(K_2))$.

Proof. Let us consider a word v that belongs to the language $\mathcal{L}(K_1)$ and, therefore, to the language $\mathcal{L}(K_2)$.

We shall construct $\mathcal{B}(K_1)$ and $\mathcal{B}(K_2)$ according to the previous definition. We shall form two words from the word v:

- $v_{\zeta} \in \mathbb{Z}^*_{(n)}$, which contains the symbols of the word v belonging to the alphabet $\mathbb{Z}_{(n)}$ in the same order in which they occur in v,
- and $v_{\gamma} \in \Sigma^*$, which contains symbols from the alphabet Σ, in the same order in which they occur in v.

From $v \in \mathcal{L}(K_1)$ we get

$$v_{\gamma} \in \mathcal{L}(\mathcal{B}(K_1))$$

and

$$v_{\zeta} \in \mathcal{L}_{\zeta}(\mathcal{B}(K_1)).$$

Furthermore, $v_{\gamma} \in \mathcal{L}(\mathcal{B}(K_2))$ and $v_{\zeta} \in \mathcal{L}_{\zeta}(\mathcal{B}(K_2))$ as soon as $v \in \mathcal{L}(K_2)$. Therefore, $\mathcal{L}(\mathcal{B}(K_1)) = \mathcal{L}(\mathcal{B}(K_2))$. □

For example, consider the bracketed automata

$$\mathcal{B}_1 = (\{X, Y, Z\}, \{a, b\}, \hat{\chi}_{\hat{\delta}_1}, \{X\}, \{X, Y, Z\}, 1) \quad \text{and}$$

$$\mathcal{B}_2 = (\{A\}, \{a, b\}, \hat{\chi}_{\hat{\delta}_2}, \{A\}, \{A\}, 1),$$

shown on Fig. 6 (a, b).

They generate the language of bracketed systems. Bracketed system can consist of brackets ")", "(" , "[" and others. Language may have one or more types of brackets. In the bracketed systems a is an opening bracket, and b is a closing one.

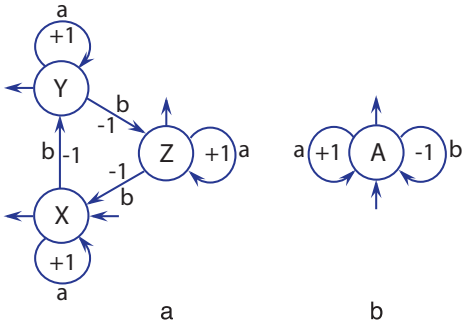

Fig. 6. a) Automaton $\mathcal{B}_1$; b) Automaton $\mathcal{B}_2$

The functions $\hat{\chi}_{\hat{\delta}_1}, \hat{\chi}_{\hat{\delta}_2}$ are determined from the images of the automaton on the figures.

It is not difficult to see that finite automata

$$\mathcal{K}(\mathcal{B}_1) = (\{X, Y, Z\}, \{(a, +1), (b, -1)\}, \hat{\delta}_1, \{X\}, \{X, Y, Z\})$$

and

$$\mathcal{K}(\mathcal{B}_2) = (\{A\}, \{(a, +1), (b, -1)\}, \hat{\delta}_2, \{A\}, \{A\})$$

are equivalent, and $\mathcal{K}(\mathcal{B}_2)$ is minimal in term of the quantity of vertices.

According to Theorem 3, the bracketed automata $\mathcal{B}_1$ and $\mathcal{B}_2$ are also equivalent. In addition, the bracketed automaton $\mathcal{B}_2$ has the smallest possible quantity of vertices.

Thus, Fig. 6 shows an example of an automaton, from which, using the transformation under consideration, we can obtain an automaton with a minimum number of vertices.

On Fig. 7 we give an example of an automaton ($\mathcal{C}_1$), which can be reduced to the automaton with a smaller number of vertices ($\mathcal{C}_2$) with the help of the considered transformation. But $\mathcal{C}_3$ is the equivalent automaton with the minimal number of the vertices.

As a result, with the help of the transformation under consideration, we obtained an automaton with a smaller number of vertices, but not a minimum.

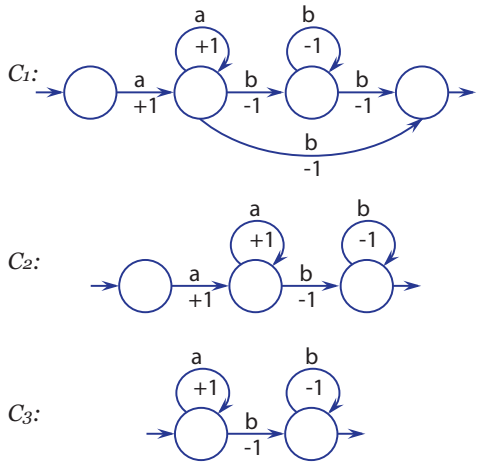

Fig. 7. The transformation of the automaton

We note that each of the automata $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ is a correct bracketed automaton [21].

VI. Conclusion

Let us briefly summarize the results of the paper and formulate the main directions for further research on this topic. Thus, we considered the bracketed automata as a new formalism for defining context-free languages.

This formalism is similar to the nondeterministic finite automata. This fact makes it possible to use any algorithms for the equivalent transformation of nondeterministic finite automata, obtaining various algorithms for equivalent transformations for the formalism that defines the given context-free languages.

In this paper, it is shown that we can apply different algorithms of equivalent transformation to the automata considered in the paper. As such algorithms, we can consider algorithms of constructing the equivalent automaton having minimal possible number of states (so-called minimal automaton) [12] or number of edges [14], [20], the equivalent universal automaton, an automaton according to the basis one, etc.

Moreover, we can obtain objects of the proposed formalism which are more acceptable in terms of some characteristics, such as fewer numbers of vertices or edges.

We also denote that the well-known theorem of Chomsky-Schützenberger about the representation of a context-free language by the morphic image of the intersection of the Dyck language and the regular language [22] can be obtained as a consequence of the definition of the bracketed automaton language. A similar result for D-graphs was established by L. I. Stanevichene in [5].

Searching subclasses of bracketed automata in which the problem of vertex minimization is solved and finding the corresponding algorithms, by means of which, with certain transformations, a minimal bracketed automaton will be constructed may be the direction of further work.

References

[1] Aho A., Ullman J. *The theory of parsing, translation and compiling*, V.1, Prentice-Hall, INC, Englewood Cliffs, N.J., 1972.

[2] Vylitok A. *On a pushdown automata graph construction*, Vestnik of Moscow University, S. 15, *Computational Mathematics and Cybernetics*, 1996, no. 3, pp. 68–73 (in Russian).

[3] Ollongren A. *Definition of programming languages by interpreting automata*, (London: Academic Press, 1974).

[4] Stanevichene L. *On one way of studying contextless languages*, The Cybernetics, 1989, no. 4, pp. 135–136.

[5] Stanevičené L. *D-Graphs in Context-Free Language Theory*, Informatica (Journal of Lithuanian Academy of Sciences), 1997, V.8, no. 1. pp. 43–56.

[6] Stanevichene L. *On some definitions of context-free languages*, Programming and Computer Software, 1999, no. 5, pp.15–25.

[7] Vylitok A., Zubova M., Melnikov B. *On an extension of the class of finite automata for the specification of context-free languages*, Vestnik of Moscow University, S. 15, *Computational Mathematics and Cybernetics*, 2013, no. 1, pp. 39–45 (in Russian).

[8] Melnikov B., Melnikova A. *Pseudo-automata for Generalized Regular Expression*, International Journal of Open Information Technologies, 2018, V. 6, no. 1, pp. 1–8.

[9] Medvedev Y. *On the class of events accepting a representation in a finite automaton*, Automata, M, Foreign Literature, 1956, pp. 385–401.

[10] Melnikov B., Vakhitova A. *Some more on the finite automata*, The Korean Journal of Computational and Applied Mathematics, (Journal of Applied Mathematics and Computing), 1998, vol. 5, no. 3, pp. 495–506.

[11] Melnikov B. *Extended nondeterministic finite automata*, Fundamenta Informaticae, 2010, V. 104, no. 3, pp. 255–256.

[12] Melnikov B. *A new algorithm of the state-minimization for the nondeterministic finite automata*, The Korean Journal of Computational and Applied Mathematics, 1999, no. 2, pp. 277–290.

[13] Melnikov B., Sayfullina M. *On some algorithms of equivalent transformations of nondeterministic finite automata*, Izvestiya of universities. Mathematics, 2009, no. 4, pp. 67–72 (in Russian) (English translation: Mel'nikov B., Saifullina M. *Some algorithms for equivalent transformations of nondeterministic finite automata*, Russian Mathematics (Izv. VUZ), 2009, no. 4, pp. 54–56.)

[14] Melnikov B. *Once more on the edge-minimization of nondeterministic finite automata and the connected problems*, Fundamenta Informaticae, 2010, no. 3, pp. 267–283.

[15] Dolgov V., Melnikov B. *A Construction of Universal Final Automaton. I. From the Theory to the Practical Algorithms*, Bulletin of Voronezh State University, Series: Physics. Mathematics, 2013, no. 2, pp. 173–181.

[16] Jiang T., Ravikumar B. *Minimal NFA problems are hard*, SIAM J.Comput, 1993, V. 22, no. 6, pp. 1117–1141.

[17] Geldenhuys J., van der Merwe B., van Zijl L. *Reducing nondeterministic finite automata with SAT solvers* Springer. Finite-State Methods and Natural Language Processing, Lecture Notes in Computer Science, 2010, V. 6062, pp. 81–92.

[18] Polák L., *Minimizations of NFA using the universal automaton*, International Journal of Foundations of Computer Science, 2005, V. 16, no. 5, pp. 999–1010.

[19] Wirth N. *Algorithms + Data Structure = Programs*, Prentice-Hall PTR UPPER Saddle River, N.J., USA, 1978.

[20] Melnikov B., Melnikova A. *Edge-minimization of non-deterministic finite automata*, The Korean J. of Comp. and Appl. Math., September 2001, V. 8, pp. 469–479 (Journal of Applied Mathematics and Computing).

[21] Vylitok A., Zubova M, *The correct bracketed automata*, Problems of Theoretical Cybernetics. Materials of the XVII International Conference (Kazan, June 16-20, 2014), S. Problems of theoretical cybernetics, Fatherland Kazan, pp. 62–65.

[22] Chomsky N., Schützenberger M.-P. *The Algebraic Theory of Context-Free Languages*, Computer Programming and Formal Systems, P. Braffort and D. Hirschberg (eds.), North Holland, 1963, pp. 118–161.