Some More on the Modeling Context-Free Languages by Nondeterministic Finite Automata

Tatiana Generalova

***Abstract*—A new formalism for the specification of context-free languages is presented. In this formalism, a generalization of the class of nondeterministic finite automata can be obtained by using an auxiliary alphabet and imposing additional conditions. Received mechanism, so-called bracketed automata, can be used for recognizing context-free languages.**

At the same time this formalism is similar to the nondeterministic finite automata and this fact allows using classic algorithms of the equivalent transformation of nondeterministic finite automata for objects of formalism that specifies the context-free languages.

An algorithm for constructing a bracketed automaton according to the given context-free grammar is considered.

An important problem that arises in the design and practical implementation of automation systems for constructing translators are optimization issues; we mean here both of the compilers themselves and of the generated executable code.

To obtain optimized variants of translators, different methods are used in practice; one of them is to obtain various equivalent descriptions of the compiled language. In most situations, when developing compilers, we need to use context-free languages or some related constructions.

In some cases, the formalism we have described makes it possible to construct a minimal object from the point of view of the number of states that describes the special extension of the class of finite automata.

***Keywords*—compiler building automation systems, nondeterministic finite automata, context-free languages, 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].

In this paper, we consider a new formalism for describing context-free languages [3]. We offer some examples that show transforming context-free grammars into objects of this formalism.

An important problem that arises in the design and practical implementation of automation systems for constructing translators are optimization issues; we mean here both of the compilers themselves and of the generated executable code. To obtain optimized variants of translators, different methods are used in practice; one of them (and, apparently, the first in order of application) is to obtain various equivalent descriptions of the compiled language.

Received 31.05.2021.

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

These equivalent descriptions can be optimized for different parameters; in a large number of cases, more compact descriptions lead to more successful versions of the resulting executable programs [4], [5].

The "compact description" concept itself can be formalized in a very large number of ways. For example, for regular construction, the minimum possible (or close to it) number of states of a deterministic (or, in some situations, nondeterministic) finite automaton which defines the given language is usually used.

However, in most situations, when developing compilers, we need to use context-free languages (or some related constructions.) For these languages, several variants of designating them in the form of a graph are described; but, apparently, none of these options has yet become a standard.

In [6], [7], [8], [9], [10], [11] various variants of minimization of ordinary finite nondeterministic automata are proposed. We hope, described methods can be useful for various problems in the theory of compilers.

In some cases, the formalism we have described makes it possible to construct a minimal object from the point of view of the number of states that describes the special extension of the class of finite automata.

Moreover, in [12], we defined an equivalent push-down automaton (which is an instance of the standard formalism of the theory of languages), which can also be called a special version of a minimal automaton.

The further direction of work is to obtain minimal automata on the basis of various variants of minimization of ordinary finite nondeterministic automata proposed in previous papers.

II. Preliminaries

We shall use the notation from [13] for nondeterministic finite automata (NFA for short).

Definition 1: [13] 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.

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

$$4\ 0\ 4\ {-}4\ 3\ {-}3\ {-}4\ 0\ 3\ {-}3\ 0 \tag{2}$$

is a word over $\mathbb{Z}_{(10)}$.

Definition 2: 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 $u \in [{\mathbb{Z}_{(n)}}^*]$.

Thus, 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 (2) is word matched by brackets.

Definition 3: We define a bracketed automaton B as follows:

$$B = (\, Q, \Sigma, \text{ϒ}, S, F, n \,), \tag{3}$$

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

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

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

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

are fulfilled.

Functions ϒ_γ and ϒ_ζ do not contain other values.

We shall denote that tuple $(\, Q, \Sigma, \text{ϒ}_\gamma, S, F \,)$ can be considered as an ordinary nondeterministic finite automaton for the bracketed automaton (3).

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 $(\, Q, \mathbb{Z}_{(n)}, \text{ϒ}_\zeta, S, F \,)$.

Let us define the language of the automaton (3) and denote it as $\mathcal{L}(B)$.

Definition 4: Let us suppose that for the sequence of states $q_0, q_1, \ldots, q_m \in Q$ the following conditions

- $q_0 \in S$, $q_m \in F$,
- $\text{ϒ}(q_k, q_{k+1}) \ni (a_k, i_k)$ 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 5: 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 (3) is the context-free language.

III. 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.

A. An algorithm of transforming context-free grammars into 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 [14].

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 ϒ_γ) and label 0 (for transition function ϒ_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.

The method for transforming a grammar into a graph presented in this algorithm was proposed by A.A. Vylitok.

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:

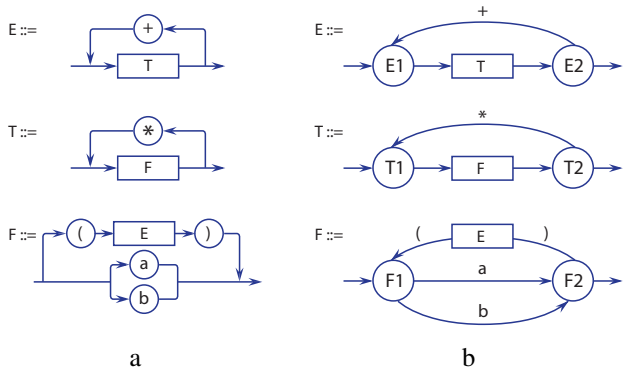
$$E \to T\,\{+T\}, \tag{4}$$

$$T \to F\,\{*F\}, \tag{5}$$

$$F \to a\,|\,b\,|\,(E). \tag{6}$$

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.a) are constructed in an obvious manner.



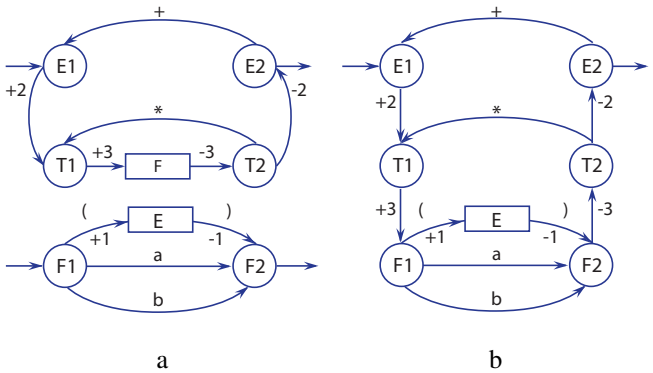
a b

Figure 1. a) Syntax diagrams for the expressions (4)–(6), Step 1; b) Graphs based on the syntax diagrams Fig. 1a, Step 2

Transform each diagram into a graph according to step 2 of the algorithm (Fig. 1b).

Substitute the transition to the subgraph T and returning from one into the graph E (Fig. 2a).

Similarly, substitute the transition to subgraph and returning from one for F (Fig. 2b).



a b

Figure 2. a) Graphs based on the syntax diagrams Fig. 1a, Step 2; b) Graph based on the syntax diagrams Fig. 1a, Step 3

Implement recursion for the obtained graph; for this, make a transition to E_1 and returning from E_2 (Fig. 3).

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

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

$$E_1 \underset{+2}{\longrightarrow} T_1 \underset{+3}{\longrightarrow} F_1 \xrightarrow{a} F_2 \underset{-3}{\longrightarrow} T_2 \xrightarrow{*} T_1 \underset{+3}{\longrightarrow}$$

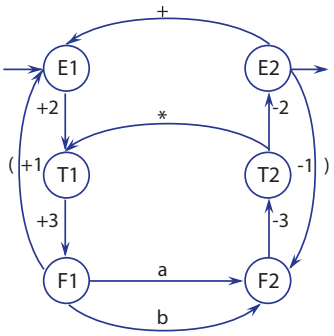


Figure 3. Graph based on the syntax diagram Fig. 1a

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

IV. Some examples of transforming context-free grammar to the bracketed automata

A. For-statement

Let us consider some examples of transforming context-free grammars according to the Algorithm 1. Syntax diagrams are taken from [15].

The Fig. 4 shows a syntax diagram for *For-statement*. We construct graph for the diagram shown on the figure Fig. 4. The vertices O_x are auxiliary vertices.

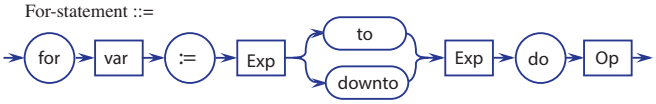


Figure 4. Syntax diagram for “For-statement”

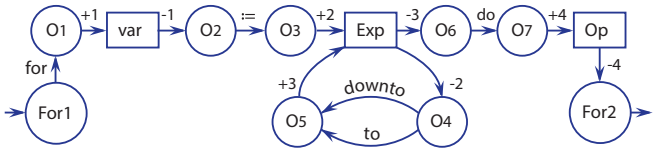


Figure 5. Graph based on the syntax diagram Fig. 4.

We do not show the transitions to Exp and Op. We can implement them with the help of pairwise replacement of edges:
$O_3 \to Exp$ to $O_3 \to Exp_1$, $O_5 \to Exp$ to $O_5 \to Exp_1$, $Exp \to O_6$ to $Exp_2 \to O_6$, $Exp \to O_4$ to $Exp_2 \to O_4$.

B. While-statement

Let us consider the construction of a bracketed automaton with the example of a *While-statement*.

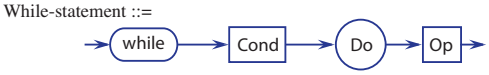


Figure 6. Syntax diagram for “While-statement”.

The resulting graph for the *While-statement* does not contain transitions to Op and Exp. They can easily be

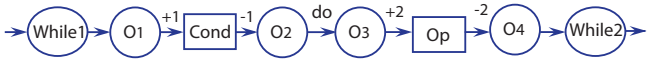


Figure 7. Graph based on the syntax diagram Fig. 6.

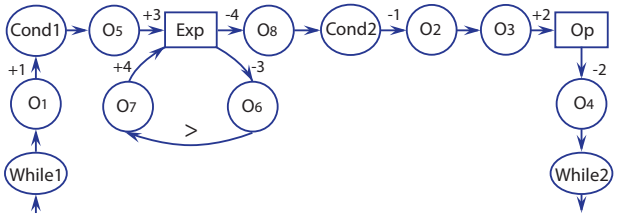


Figure 8. Resulting graph for “While-statement”.

implemented with the help of pairwise replacement of edges: $O_3 \to Op$ to $O_3 \to Op_1$, $Op \to O_4$ to $Op_2 \to O_4$, $O_5 \to Exp$ to $O_5 \to Exp_1$, $O_7 \to Exp$ to $O_7 \to Exp_1$, $Exp \to O_6$ to $Exp_2 \to O_6$, $Exp \to O_8$ to $Exp_2 \to O_8$.

C. *Repeat-statement*

Let us consider the construction of a bracketed automaton with the example of a *Repeat-statement*.

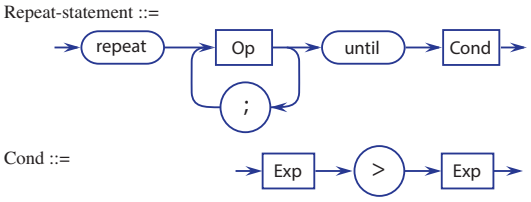


Figure 9. Syntax diagrams for “Repeat-statement” and condition “Cond”.

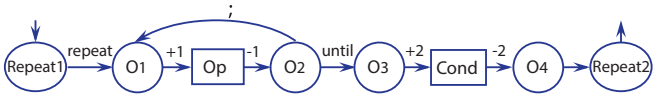


Figure 10. Graph based on the syntax diagram for “Repeat-statement”, Fig. 9.

The resulting graph for the *Repeat-statement* does not contain transitions to Op and Exp. They can easily be implemented with the help of pairwise replacement of edges: $O_1 \to Op$ to $O_1 \to Op_1$, $Op \to O_2$ to $Op_2 \to O_2$, $O_5 \to Exp$ to $O_5 \to Exp_1$, $O_7 \to Exp$ to $O_7 \to Exp_1$, $Exp \to O_6$ to $Exp_2 \to O_6$, $Exp \to O_8$ to $Exp_2 \to O_8$.

V. Conclusion

In the article, we showed some examples of converting context-free grammars into bracketed automata, a new formalism for defining context-free languages. The main feature of the bracketed automata is the ability to consider them in two ways, including using nondeterministic finite automata, over a newly extended alphabet.

In some cases, the formalism we have described makes it possible to construct a minimal object from the point of view of the number of states that describes the special extension of the class of finite automata.

The further direction of work is to obtain minimal automata on the basis of various variants of minimization of ordinary finite nondeterministic automata proposed in previous papers.

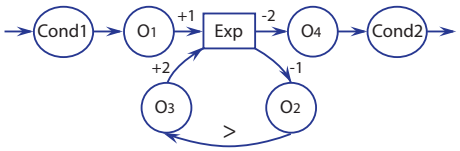


Figure 11. Graph based on the syntax diagram for condition “Cond”, Fig. 9.

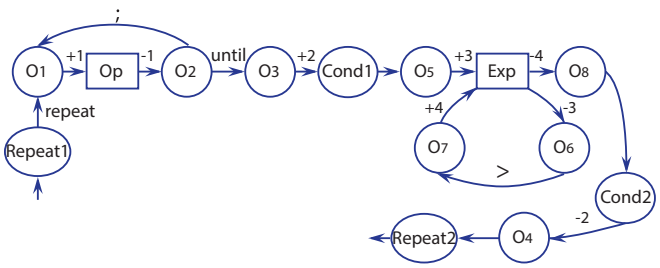


Figure 12. Resulting graph for “Repeat-statement”.

References

[1] Aho A., Ullman J. “The theory of parsing, translation and compiling”, V.1, Prentice-Hall, INC, Englewood Cliffs, N. J., 1972, pp. 1002.

[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] 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).

[4] Wirth N. “Compiler Construct”, M.: DMK-Press, 2013, pp. 193.

[5] Wirth N., Gutknecht J., “Porject Oberon. The design of an operating system and compiler”. 2005, pp. 441.

[6] “Some more on the finite automata”, Melnikov B.F., Vakhitova A.A. Journal of Applied Mathematics and Computing, 1998, V. 5, no. 3, pp. 495-505.

[7] Melnikov B.F., Melnikova A.A., “Some properties of the basis finite automaton”, Korean Journal of Computational and Applied Mathematics, 2002, V.9, no.1, pp. 135-150.

[8] Melnikov B.F., Sciarini-Guryanova N.V., “Possible edges of a finite automaton defining a given regular language”, Korean Journal of Computational and Applied Mathematics, 2002, V. 9, no. 2, pp. 475-485.

[9] Melnikov B., Sayfullina M., “On some algorithms of equivalent transformations of nondeterministic finite automata”, Izvestiya of universites, Mathematics. 2009, no.4, pp.67-72 (in Russian), (English translation: Mel’nikov B., Sayfullina M., Some algorithms for equivalent transformations of nondeterministic finite automata. Russian Mathematics, (Izv.VUZ). 2009, no. 4, pp. 54-56.)

[10] Melnikov B., “Extended nondeterministic finite automata”, Fundamenta Informaticae, 2010, V. 104, no. 3, pp. 255-265.

[11] Dolgov V., Melnikov B., “The construction of a universal finite automaton. Part I: From theory to practical algorithms”. Vestnik of Voronezh State University, S: Physics. Mathematics, 2013, no. 2. pp. 173-181 (in Russian).

[12] Generalova T., Melnikov B., Vylitok A., “On the extension of the finite automata class for context-free languages specification”, International Journal of Open Information Technologies, 2018, V.6, no.8, pp.1-8. (http://injoit.ru/index.php/j1/article/view/602)

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

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

[15] Grogono P., “Programming in Pascal”, 1982, M.: Mir, pp. 378.