ISSN: 0975-766X
CODEN: IJPTFI
Available Online through *Research Article*
www.ijptonline.com

APPLICATION OF FINITE-STATE-MACHINE MODELS FOR THE TRANSFORMATION OF SYNTAX DIAGRAMS

Vladimir Mikhailovich Polyakov, Yuri Dmitrievich Ryazanov
Belgorod State Technological University named afterV.G. Shukhov,
Russia, 308012, Belgorod, Kostyukov str., 46.
Belgorod State Technological University named afterV.G. Shukhov,
Russia, 308012, Belgorod, Kostyukov str., 46.

Received on 14-08-2016 *Accepted on 20-09-2016*

Abstract.

The article deals with the questions of syntax diagrams equivalence transformations. To carry out equivalence transformations there was suggested a finite-state-machine model of a syntax diagram, which is a finite set of non-deterministic automata of Rabin-Scott with ε-transitions. There are presented an algorithm of transforming a syntax diagram into a finite-state-machine model and algorithm of transforming a finite-state-machine model into a syntax diagram. It is demonstrated that if an automaton in a model is replaced with an equivalent automaton, the obtained automata set would also be a syntax diagram model. On this basis there was suggested a method of syntax diagrams equivalence transformations, which consists in the following. A finite-state-machine model is built for a certain syntax diagram. The automaton of the finite-state-machine model is transformed into an equivalent automaton by using algorithms, known in the automata theory. Then, according to the transformed finite-state-machine model a syntax diagram is built. The obtained syntax diagram would be equivalent to the initial syntax diagram. The article demonstrates the examples of using this method to exclude the excess nodes in a syntax diagram, to obtain a pseudo-deterministic syntax diagram and to reduce the number of nodes in a syntax diagram.

Key words: formal language, finite-state automaton, syntax diagram, equivalence transformations.

Introduction.

Syntax diagrams (SD) is a visual method of setting a formal language, which is used for recording programming languages [1, 2], and for designing the language processing software [3 – 9]. In the process of developing programsan initial SD has to be transformed to an equivalent SD (two syntax diagrams are equivalent, if the languages, set by them, are equivalent), meeting the certain requirements. For example, to design linear complexity translators [8, 9] the deterministic syntax diagrams are used ;at this, the size of translator depends on the size of syntax diagram. So, to

make a translator smaller, it's reasonable to use the «compact» syntax diagrams [10]. The equivalence SD transformation algorithms are presented in a number of works [2 – 6, 10 – 12]. Many transformation algorithms are based on evident or intuitive reasoning and their correctness is not always confirmed. This article suggests using the known algorithms of finite-state automata transformation [13 – 19], the correctness of which is rigorously proven, to carry out the SD equivalence transformation. The article defines asyntax diagram and Rabin-Scott's non-deterministic automata withε-transitions and suggests a finite-state-machine model of a syntax diagram, which is afinite set of automata; demonstrates, that if in a syntax diagram model an automaton is replaced with another automaton, equivalent to the first one, then the new set of automat a would be a model of the initial SD;determines therules of building a SD model and rules of building a syntax diagram from its finite-state-machine model, describes the algorithms of the finite-state automata equivalent transformations, gives the examples of using them for SD transformations. The examples of finite-state automata transformations are not analyzed in detail, only their results are demonstrated.

The main part.

1. Syntaxdiagram

Asyntax diagram may be denoted with the setD = (T, N, S, G, F), where T – is a finite set of terminals; N – is a finite set of non-terminals; $S \in N$ – initial non-terminal; G = (V, E) – a directed graph, where

$V = V_T \cup V_N \cup V_u \cup V_{entry} \cup V_{exit}$, where

V_{entry} – a set of entry points, $|V_{entry}| = |N|$;

V_T – asetof terminal nodes;

V_N – asetofnon-terminal nodes;

V_u – a finite set of nodes;

V_{exit} – a set of exit points, $|V_{exit}| = |N|$;

$E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$, where

$E_1 \subseteq \{(a, b) \mid a \in V_{entry}, b \in V_u\}$– a set ofentry curves;

$E_2 \subseteq \{(a, b) \mid a \in V_u, b \in V_{exit}\}$ – a set ofexit curves;

$E_3 \subseteq \{(a, b) \mid a \in V_u, b \in V_T \cup V_N\}$ – a set ofcurves, exiting thenodes;

$E_4 \subseteq \{(a, b) \mid a \in V_T \cup V_N, b \in V_u\}$ – a set ofcurves, entering the nodes;

$E_5 \subseteq \{(a, b) \mid a \in V_u, b \in V_u\}$ – a set of[epsilon]-curves, connecting the nodes;

$F:V_T \cup V_N \rightarrow T \cup N$ – reflection of the set of nodes to the set of terminals and non-terminals.

Graph Gcan be represented as a set of connected graphsG_1, $G_2,\ldots,G_{|N|}$. Graph G_i corresponds to the non-terminal $N_i \in N$ and is called the syntax diagram component. The description of graph G_i corresponds to the above mentioned description of graph G, excepting that graph G_i has only one entry point and only one exit point. In Fig. 1 there is presented a SD, consisting of two components. The capital letters S and A denote non-terminals, the small letters (a,b,c) denote terminals.

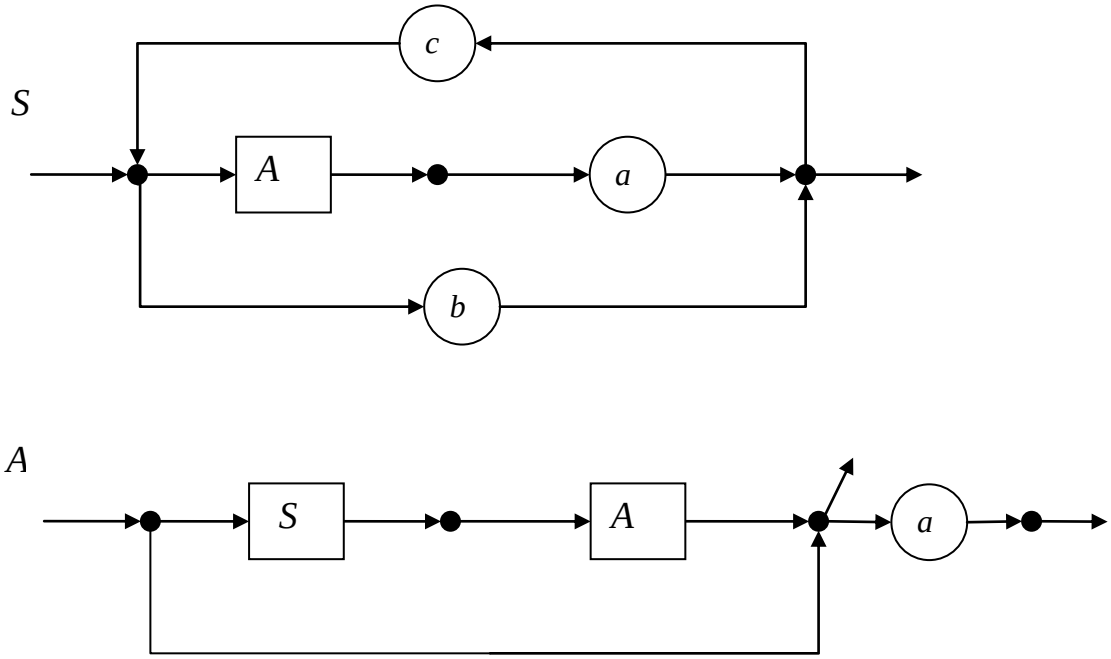

Fig. (1). –Syntaxdiagram.

A component is denoted with a certain non-terminal, has only one entry point and only one exit point and a finite set of nodes, terminal and non-terminal ones. The entry and exit points are not depicted in the component's diagram. A terminal node is depicted with a circle, in which a terminal symbol is written, a non-terminal node is depicted with a square, in which a non-terminal symbol is written. A node isdepicted in the diagram with aheavy point. No curves enter the entry point and a finite set of curves exit from it (the entry curves of the component). The nodes, which are entered by entry curves, are called initial. No curves exit from the exit point and a finite set of curvesenter it (the exit curves of the component). The nodes, from which the exit curves exit, are called final. The curves, connecting the nodes, are calledε-curves. Let's consider a certain path in the component G_i from node to some final node in this component. Let's goal on git and add symbols, recorded in terminal and non-terminal nodes, to an initially empty stringα in the course of it. The stringαwill be called a string, which connects node u with the final node of component G_i. A set of strings, connecting node with a certain final node, makes up alanguage L(u). In the similar way we will

determine a string, connecting the entry point of component $G_i$with its exit point. A set of such stringsmakes up a language $L(G_i)$. It's obviousthat the language $L(G_i)$ is equal tothe sum of all$L(u_j)$, whereu_j – is an initial node of componentG_i.

The string of language, set by a syntax diagram, can be deducted by the following rule:

1. Record a string, belonging to language$L(G_1)$, where componentG_1 corresponds to the initial non-terminal.

2. If the string contains a certain non-terminalN_i, replace it with the string of language$L(G_i)$ and go to step 2, otherwise – finish the deduction.

From the rule of deducting a language string the following assertions result.

Assertion 1.If in a syntax diagram component $G_i$is replaced with component G_i', so that$L(G_i') = L(G_i)$, there will be obtained a SD, equivalent to the initial SD.

Assertion 2.Let in a syntax diagram component G_i correspond to non-terminalN_i, componentG_j – correspond to non-terminalN_j and $L(G_i) = L(G_j)$. Then exclusion of component $G_j$from the syntax diagramand replacement of non-terminal$N_j$with non-terminal$N_i$in a syntax diagram, willgive a new SD, equivalent to the initial SD.

2. Finite-state automaton

A non-deterministic finite-state automaton of Rabin-Scott with [epsilon]-transitions [13] maybe denoted with the set $A = (X, S, S_0, [delta], Q)$, where

X – a finite non-empty set of input symbols (input alphabet);

S – a finite non-empty set of states (state alphabet);

S_0 – a finite non-empty set of initial states, $S_0 \subseteq S$;

[delta] : $S \times \{X \cup \{[epsilon]\}\} \rightarrow 2^S$ – transition function, which assigns a subset of states to a pair (state, input symbol) or to a pair (state, [epsilon]). Here 2^S – isa Boolean of the set$S$, [epsilon] – the indicator of symbol's absence.

Q – a finite set of final states, $Q \subseteq S$.

Automaton A can be presented with a graph, in which nodes correspond to states. Node s_i is connected with a curve, noted with symbol$x$ ($x \in X$), with nodes_j ,if [delta](s_i,x) = $S'$and$s_j \in S'$. If [delta](s_i,[epsilon]) = $S'$and$s_j \in S'$, then node$s_i$is connected with nodes_j with an unmarked curve. The stringα, obtained by writing symbols from the graph curves of automaton$A$, forming a path from state$s_i$to the final state, is allowed by state s_i. These to fall strings, allowed by state s_i, forms the language $L(s_i)$. The stringα, allowed by some initial state of automaton$A$, is allowed by automaton$A$. These to fall strings, allowed by automaton$A$, forms the language $L(A)$.

3. The finite-state-machine model ofa syntax diagram

The finite-state-machine (FSM) model of a SD, the graph of which is$G = \{G_1, G_2,\ldots,G_n\}$, is a set ofnon-deterministic automata of Rabin-Scott with [epsilon]-transitions$A = \{A_1, A_2,\ldots,A_n\}$, so that$L(A_i) = L(G_i)$.

Component G_i corresponds to automaton $A_i = (X, S^i, S_0^i, [delta]^i, Q^i)$, in which

– the input alphabet Xis a set of terminals and non-terminals of a syntax diagram;

– the state alphabet $S^i$is a set of nodes of component G_i;

– the set of initial states $S_0^i$is the set of initial nodes of component G_i;

– these to final states$Q^i$is the set of final nodes of component G_i;

– the transition function is determined in the following way:

1. $s_j \in [delta]^i(s_i,x)$ if and only if there is a path in a syntax diagram from nodes $_i$to node s_j through the only node, containing symbolx;

2. $s_j \in [delta]^i(s_i,[epsilon])$ if and only if there is a curve in a syntax diagram from nodes$_i$ to nodes$_j$.

Inorder to build afinite-state machine model of a SD, for each component $G_i \in G$ the corresponding automaton A_i must be built according to the above mentioned rules.

From the definition of asyntax diagram's FSM model the following assertion results:

Assertion3.Ifinasyntaxdiagram'sFSMmodel$A = \{A_1, A_2,\ldots,A_i,\ldots,A_m\}$ automaton$A_i$is replaced with an equivalent automatonA_i' ($L(A_i') = L(A_i)$), then$A' = \{A_1, A_2,\ldots,A_i',\ldots,A_m\}$ also would be a FSM modelof the considered syntax diagram.

Byasyntax diagram's FSM model$A = \{A_1, A_2,\ldots,A_m\}$ we canunambiguously build a SD. For this purpose each automaton$A_i \in A$should be transformed to a component $G_i$of the syntax diagram. Transformation is doneaccording to the following rules.

1. The set of nodes of component $G_i$is made up with the set of states of automatonA_i.

2. If in the automaton$s_j \in [delta]^i(s_i,[epsilon])$, then in component G_i acurve is drawn from node s_i to node s_j. If in the automaton $s_j \in [delta]^i(s_i,x)$, where x is a terminal, then a terminal node is added to component G_i, the x symbol is written into it, a curve is drawn from nodes$_i$to this node, and from it – a curve to nodes$_j$. Ifxisanon-terminal, then the similar actions are done, except that a non-terminal node is added to componentG_i.

3. The nodes, corresponding to initial states, are made initial(entry curves are drawn to them).

4. The nodes, corresponding to final states, are made final (exit curvesare drawn from them).

The application of a finite-state-machine model for transforming syntax diagrams consists in the following. According to the given SDa FSM model is built. The automaton of a FSM model is transformed to an equivalent automaton by using the known algorithms of the automata theory. Then a syntax diagram is built by the transformed FSM model. The obtained SD will be equivalent to the original SD.

Let's consider the application of this approach to carry out some syntax diagrams transformations.

4. Excluding the excess nodes from a syntax diagram

In the set of nodes of a syntax diagram component two types of excess nodes can be singled out:

1. Anode belongs to excess nodes set of the 1-sttype, if there is no path from it to some other initial node of the component;

2. Anode belongs to excess nodes set of the 2-ndtype, if there is no path from it to some other final node of the component.

No path from the initial node to the final node goes through an excess node, so the excess node doesn't influence these to strings, drawn in the syntax diagram. Asyntax diagram with excess nodes can be transformed to an equivalent syntax diagram without texcess nodes. Forth is purpose let us build a FSM model of a syntax diagram$A = \{A_1, A_2,\ldots,A_n\}$. In each automaton$A_i \in A$weexclude the excess states, using, for example, the method, described in [19]. For this purpose let us do the following actions.

1. Find the setEof reachable states according to the algorithm:

1. Assign$E = S_0^i$, whereS_0^i – is the set of initial states of automatonA_i.

2. If the automaton's graph hasthe curve (s_j,s_k), where$s_j \in E$and$s_k \notin E$, then add the state$s_k$to the setEand performstep 2, otherwise – finish the algorithm.

2. Find the set of unreachable states of automatonA_i, excluding the states of setEfrom the set of all states of the automaton.

3. Obtain automaton$A_i^{-1}$by altering the orientation ofcurves of automatonA_i. Consider the initial states of automatonA_i^{-1}as the final states of automatonA_i.

4. Find the set of unreachable states of automatonA_i^{-1}. This will be a set of excess states of automatonA_i.

5. Remove all the unreachable and excess states in the automaton$A_i$to gether with exit and entry curves.

Let's trans form the FSM model of a syntax diagram, in which each automaton contains no excess states, into a syntax diagram without excess nodes.

Infig. 2 an example of excluding the excess nodes from a SD is given. Fig. 2 apresents theoriginal syntax diagram, fig. 2б – the FSM modelof a syntax diagram, fig. 2в – automaton without excess states, fig. 2г – syntax diagram without excess nodes.

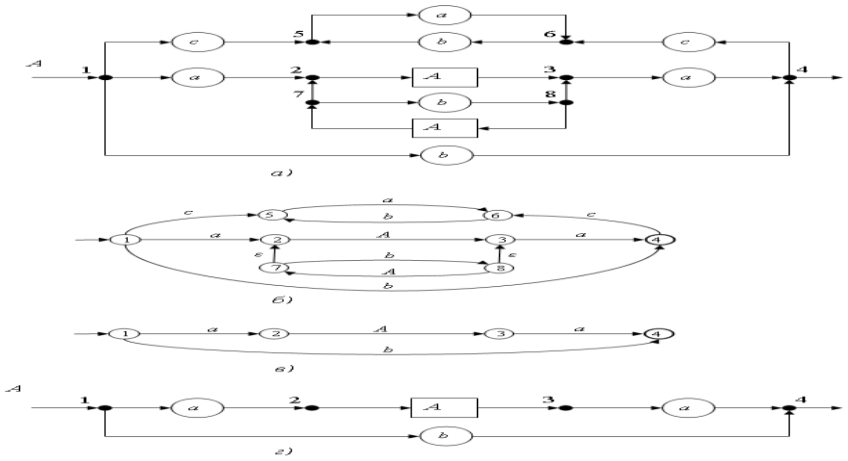

Fig. (2). Excluding the excess nodes from a syntax diagram.

5. Transforminga syntax diagram toapseudo-deterministic syntax diagram

Asyntax diagramiscalledpseudo-deterministic (PSD) [9], if:

1. it has no [epsilon]-curves;

2. each component has only one initial node;

3.ForeachnodeofaSDit'strue, that any two curves, exiting this node, go to nodes, containing various symbols.

The transformation of a SD to a PSD is the first stage in the process of transforming a SD to a deterministic SD [10]. Deterministic syntax diagrams are used for designing the efficient language processing software[8, 9].

To transform arandom SD to a PSD let's build a FSM model of a SD and transform each automaton to adeterministic one. The algorithms of transforming a non-deterministic automaton of Rabin-Scott with [epsilon]-transitions to adeterministic one are described in the works [13, 18, 19] etc.

A non-deterministic automaton of Rabin-Scott with[epsilon]-transitions can be transformed to a deterministic one in two stages:

1. eliminating the [epsilon]-transitions (transforming the automaton$A_{[epsilon]}$with [epsilon]-transitions to anon-deterministic automaton$A_н$without [epsilon]-transitions);

2. transforming the automaton$A_н$without [epsilon]-transitions to adeterministic automaton$A_д$.

To eliminate the [epsilon]-transitions a concept of [epsilon]-closure of the state s_i ([epsilon](s_i)) is used. The [epsilon]-closure of the state s_i is a set of states, containing the set s_i and all sets, achievable from itonly through [epsilon]-transitions.

The automaton$A_н$is built in the following way.

The states of automaton$A_н$are [epsilon]-closures ofstates of automaton$A_{[epsilon]}$.

The initial states of automaton$A_н$are [epsilon]-closures ofthe initial states ofautomaton$A_{[epsilon]}$and [epsilon]-closures of allstates, which areachievable frominitial states only through [epsilon]-transitions.

The final states of automaton$A_н$are [epsilon]-closures of states, which contain at least one final state of automaton$A_{[epsilon]}$.

The transition function of automaton$A_н$is determined in the following way. Theautomaton$A_н$transitsfrom [epsilon](s_i) under the action of input symbolxto [epsilon]-closuresof states, to which automaton$A_{[epsilon]}$transits from states, belonging to [epsilon](s_i) under the action of symbol x.

The automaton$A_д$is built in the following way.

The states of automaton $A_д$ aresubsets of states of automaton$A_н$.

The initial state of automaton $A_д$ is the subset of all initial states of automaton$A_н$.

The final states of automaton $A_д$ are subsets, which contain at least one final state of automaton$A_н$.

The transition function of automaton$A_д$is determined in the following way. The automaton $A_д$ transits from the subset S_i under the action of input symbolx to the subset S_j. The states belongs to the subsetS_j, if and only if the automaton$A_н$transits to the states under the action of symbol xfrom at least one state, belonging to the subsetS_i.

Infig. 3 an example of transforming a syntax diagram to apseudo-deterministic syntax diagram is given. Fig. 3a presents the original syntax diagram, fig. 3б – the FSM model of a syntax diagram, fig. 3в – automaton without [epsilon]-transitions, fig. 3г – deterministic automaton and fig. 3д – a PSD.

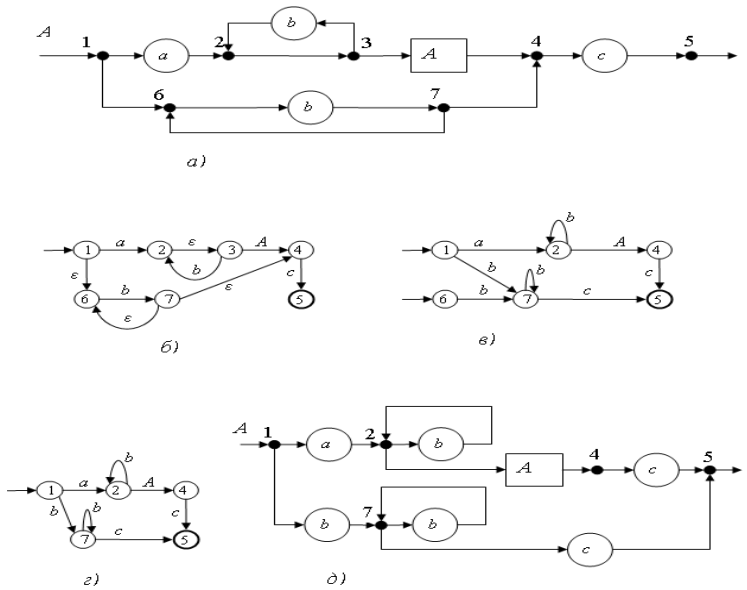

Fig. (3). Transforming a syntax diagram to a pseudo-deterministic syntax diagram.

6. Reducing the number of nodes in components of pseudo-deterministic syntax diagrams

The FSM model of a PSD is a set of deterministic automata. Each automaton of a model can be transformed to a minimal automaton. The transformation algorithms are described in works [14 – 19] etc.

A minimal automaton contains no excess states and no pairs of equivalent states. Exclusion of the excess states is described above. To provide these con d condition we can use an algorithm, based on dividing a set of automaton's states to equivalent states classes and replacement of each class with a single state (uniting a class of equivalent states in to one state).

To divide a set of states to equivalent states classes a concept of k-equivalent states is used. Let's define $L^k(s_i)$ as a set of words with the length of k, allowed by $states_i$. $Statess_i and s_j$ are called k-equivalent, if$L^k(s_i) = L^k(s_j)$. These to fall k-equivalent to each other states forms a class of k-equivalent states. To find the partition of a set of states into equivalent states classes, the partitions into classes of 0-, 1-, …,k-equivalent states are determined successively. The process is finished, when at acertain step the classes ofk-equivalent states coincide with classes of (k – 1)-equivalent states. At this moment he partition into classes of k-equivalent states will be presented by partition into classes of equivalent states.

Two states of the automaton are 0-equivalentin case if both of them are final or if both of them are non-final. $Thestatess_i and s_j$ would bek-equivalent, if they are (k – 1)-equivalent and the states s_i' = [delta](s_i, x) ands_j' = [delta](s_j, x) and also (k – 1)-equivalent for allx of the automaton's alphabet.

Infig. 4 an example of reducing the number of nodes in a syntax diagram is given. Fig. 4apresents the original syntax diagram, fig. 4б – the FSM modelof a syntax diagram, fig. 4в – the minimal automaton, fig. 3г – SD with the lesser number of nodes, than in SD in figure 4a.

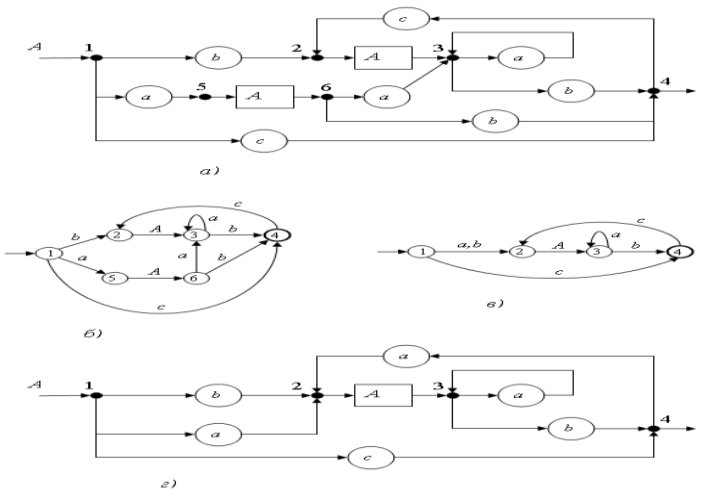

Fig. (4). Reducing the number of nodes in a syntax diagram.

Conclusion. The article suggests a finite-state-machine model of a syntax diagram and a method, which allows using finite-state-machine models for equivalence transformations of syntax diagrams. There are given examples of using the suggested method for carrying out some of the transformations. The application of this method is not restricted to the analyzed transformations. This work canbe further developed by searching other transformations, which can be carried out by means of the suggested method.

Inferences. Based on the above-mentioned we can conclude that application of the suggested method allows doing the equivalent transformations of syntax diagrams correctly, using the known algorithms of the finite automata theory.

Acknowledgements. The paper is prepared with support of Russian Foundation of Fundamental Research (grant № 16-07-00487).

References.

1. Jensen, K.andN.Wirth,1975. Pascal User Manual and Report. Springer-Verlag, New York,pp: 167.
2. Jensen, K.andN.Wirth,1996. Pascal Standard Iso. JacksonLibri, pp: 290.
3. Legalov, A.I. Translators: development methods. Date Views 22.09.2016 http://www.softcraft.ru/translat/.
4. Legalov, A.I., D.A.Shvets and I.A. Legalov, 2007.Formal Languages and Translators. Siberian Federal University, Krasnoyarsk, pp: 213.
5. Karpov,Yu.G.,2005.The Theory and Technology of Programming. Fundamentals of Translators. BHV-Petersburg, Saint-Petersburg, pp: 272.
6. Sverdlov, S.Z., 2007.Programming Languages and Methods of Translation. Peter, Saint-Petersburg, pp: 638.
7. Martynenko, B.K., 2014.Syntactic Charts and Graph-Schemes in the SYNTAX-Technology. Computer Tools in Education, 2: 3–19.
8. Ryazanov, Y.D. and M.N. Seval'neva, 2013.The Analysis of Syntax Diagrams and Automatic Generation of Linear-Time Programs-Recognizer. Belgorod State University Scientific Bulletin. History. Political science. Economics. Information technologies, 8(151): 128–136.
9. Polyakov, V.M. and Y.D. Ryazanov, 2013.Algorithm for not Recursive Linear-Time Programs-Recognizer Design from Deterministic Syntax Diagrams. Bulletin of BSTU named after V.G. Shukhov, 6: 194–199.
10. Ryazanov, Y.D., 2015. Transformation of Nondeterministic Syntax Diagrams into Deterministic Syntax Ones. Proceedings of Voronezh State University. Series: Systems analysis and information technologies, 1: 139–147.

11. Ryazanov,Yu.D., 2015. Minimizing the Number of Components Deterministic Syntax Diagram Based Strong Equivalence. Information Systems and Technologies, 1(87): 94–100.

12. Polyakov, V.M. andYu.D. Ryazanov, 2015.VIRT Charts To Multiport Component Syntactic Charts Transformation. Global Journal of Pure and Applied Mathematics, 11(5): 3939–3952.

13. Rabin, M.O. and D. Scott, 1959. Finite automata and their decision problems. IBM J. Research and Development, 3(2): 115–125.

14. Huffman, D.A., 1954.The synthesis of sequential machines. J. Franklin Inst. 257: 3–4, 161–190 and 275–303.

15. Moore,E. F.,1956.Gedanken experiments on sequential machines. In C. E. Shannon and J. McCarthy, Automata Studies, Princeton Univ. Press, pp: 129–153.

16. Hopcroft,J.E., 1971.An n log n algorithm for minimizing states in a finite automaton. Theory of machines and computations, New York: Academic Press, pp: 189–196.

17. David,J., 2012.Average complexity of Moore's and Hopcroft's algorithms. Theoretical Computer Science, 417: 50–65.

18. Hopcroft, J.E.,R. Motwani andJ.D. Ullman, 2013. Introduction to Automata Theory, Languages, and Computation (3rd ed.). Pearson.,pp: 496.

19. Brauer, W.,2012.AutomatenTheorie: EineEinfuhrung in die. Theorieendlicher. Automaten, B.G. Teubner, Stuttgart, pp: 272.