Nexo Revista Científica
Vol. 34, No. 01, pp. 61-73/Marzo 2021

ISSN-E 1995-9516
Universidad Nacional de Ingeniería

http://revistas.uni.edu.ni/index.php/Nexo
https://doi.org/10.5377/nexo.v34i01.11285

Behaviour examples for synthesizing automaton models by temporal formulas

Ejemplos de comportamiento para sintetizar modelos de autómatas mediante fórmulas temporales

Dmitry V. Pashchenko[1*], Alexey I. Martyshkin[2], Dmitry A. Trokoz[1], Tatyana Yu. Pashchenko[3], Mikhail Yu. Babich[4], Mikhail M. Butaev[4]

[1] Penza State Technological University, 440039, Russia, Penza, 1/11 Baydukova lane/Gagarina St., 1/11
[2] Department of Computational Automatons and Systems, Penza State Technological University, 440039, Russia, Penza, 1/11 Baydukova lane/Gagarina St., 1/11
[3] Sub-department of Management and Economic Security, Penza State University, 440026, Russia, Penza, Krasnaya Street, 40
[4] JSC Research and Production Enterprise "Rubin", 440000, Russia, Penza, Baydukova St., 2

[*]dmitry.pashcenko@gmail.com

(recibido/received: 15-December-2020; aceptado/accepted: 01-February-2021)

ABSTRACT

The paper deals with researching and developing the methods that make it possible to account behaviour examples when synthesizing automaton models by temporal formulas. Definitions of the terms and concepts used in work are given; the problem of synthesizing automaton systems according to the specification in the form of temporal formulas and behaviour examples is formulated; a promising algorithm for reducing the problem of synthesizing automaton systems to the Boolean formula satisfiability problem is described; an analysis of the domain and other approaches is carried out. New methods of taking into account behaviour examples in the synthesis of automaton systems according to a specification given in the form of temporal formulas are proposed. Algorithms for constructing graphs of scripts and methods for dividing graphs into clusters are described; they are designed to increase the efficiency of representing behaviour examples used for coding the behaviour examples in the form of Boolean formulas. An experimental study of the proposed methods of accounting for behaviour examples and basic approaches to the presentation of behaviour examples is carried out. The experimental results showed the superiority of the newly developed methods regarding the presentation of scripts in the form of temporal formulas. In summary, the main conclusions of the work carried out are presented.

Key words: automaton model, discrete-event simulation, finite state automaton, conflicting situation, methods of accounting for behaviour examples.

RESUMEN

El artículo aborda el tema de la investigación y desarrollo de métodos que permitan dar cuenta de ejemplos de comportamiento al sintetizar modelos de autómatas mediante fórmulas temporales. Se dan definiciones de los términos y conceptos utilizados en el trabajo; se formula el problema de sintetizar sistemas de autómatas según la especificación en forma de fórmulas temporales y ejemplos de comportamiento; se describe un algoritmo prometedor para reducir el problema de sintetizar sistemas de autómatas al problema

de satisfacibilidad de la fórmula booleana; Se realiza un análisis del dominio y otros enfoques. Se proponen nuevos métodos para tener en cuenta ejemplos de comportamiento en la síntesis de sistemas de autómatas según una especificación dada en forma de fórmulas temporales. Se describen algoritmos para construir gráficos de scripts y métodos para dividir gráficos en grupos; están diseñados para aumentar la eficiencia de la representación de ejemplos de comportamiento utilizados para codificar los ejemplos de comportamiento en forma de fórmulas booleanas. Se lleva a cabo un estudio experimental de los métodos propuestos para dar cuenta de ejemplos de comportamiento y enfoques básicos para la presentación de ejemplos de comportamiento. Los resultados experimentales mostraron la superioridad de los métodos recientemente desarrollados con respecto a la presentación de guiones en forma de fórmulas temporales. En resumen, se presentan las principales conclusiones del trabajo realizado.

Palabras clave: modelo de autómata, simulación de eventos discretos, autómata de estado finito, situación de conflicto, métodos de contabilización de ejemplos de comportamiento.

1. INTRODUCTION

The synthesis of automata models is a common problem. Its field of application ranges from software verification and control system synthesis (Vashkevich & Biktashev, 2016; Peter Faymonville et al., 2017; Biktashev & Vashkevich, 2013; Dubinin et al., 2016) and (Dubinin & Drozdov, 2016; Pashchenko et al., 2020; Pashchenko et al., 2020; Volchikhin et al., 2013) to bioinformatics problems and formal description of parallel algorithms and processes (Vashkevich & Biktashev, 2011; Vashkevich & Biktashev, 2011; Vashkevich et al., 2015). A common way to solve this problem is to reduce it to a Boolean formula's satisfiability problem (Vashkevich & Vashkevich, 1996). Typically, linear temporal logic formulas are used to define the specification of a synthesized system. In this area, a new promising approach to synthesizing an automaton model with a constraint on the size of the system has recently appeared (Peter Faymonville et al., 2017). However, only the temporal logic formulas are often not enough to specify all the synthesized system features. Sometimes the task is to synthesize automaton systems based only on behaviour examples. The purpose of the paper is to study effective methods of accounting for behaviour examples and compare the methods with other approaches to presenting behaviour examples.

2. MATERIALS AND METHODS

The Boolean formula satisfiability problem is the problem of finding such an assignment of propositional variables included in the Boolean formula so that the formula becomes true. For the Boolean formula satisfiability problem, it can contain only variables, parentheses, and operations AND, OR, NOT. An existential quantifier is implied in the Boolean formula satisfiability problem for the connection between the propositional variables included in it. The satisfiability problem for a Boolean formula with quantifiers is an extension of the Boolean formula satisfiability problem, in which, along with the existential quantifier, the universal quantifier can be used to connect variables. Synthesis with a system size constraint is an approach to the automaton model synthesis with a constraint on the size of the final system and on the number of visits to rejecting states. The problem of synthesis with the system size constraint can be represented as a problem of the solvability of a system of constraints, even in conditions where other approaches to synthesis are unsolvable, for example, in the synthesis of asynchronous or distributed systems (Vashkevich, 2004). Linear temporal logic is logic with operators that allows us to work with time. With its help, we can set the order of phenomena and their interactions in time. In addition to the usual logical operators, linear temporal logic formulas support the following operators:

• $X\varphi$ (next) - the formula φ must be satisfied in the next state.
• $F\varphi$ (finally) - the formula φ must be satisfied in one of the following states.
• $G\varphi$ (globally) - the formula φ must be satisfied in every state.
• $\varphi U\psi$ (until) - the formula φ must be satisfied at least until the moment when the formula ψ is satisfied (which must be satisfied for sure now or in the future).

• $\varphi R \psi$ (release) - the formula ψ must be satisfied up to the state (including the current state) in which φ is first satisfied, if φ is never satisfied, then ψ must always be satisfied.
• $\varphi W \psi$ (weak until) - the formula φ must be satisfied at least until the moment when the formula ψ is satisfied, if ψ will never be satisfied, then φ must always be satisfied.
• $\varphi M \psi$ (strong release) - the formula ψ must be satisfied up to the state (including this state), when φ is first satisfied (which must be satisfied now or in the future).

Let Σ be a finite set of propositional variables and a linear temporal formula φ defined over this set, then the language of the formula φ, denoted by $\mathscr{L}(\varphi)$, consists of infinite sequences of states $\sigma \in (2^K)$.
An example of a linear temporal logic formula: $G(\neg g_0 \vee \neg g_1) \wedge G(r_0 \rightarrow F g_0) \wedge G(r_1 \rightarrow F g_1)$.

This expression specifies the behaviour of an arbiter system, which, after a request r_0 or r_1, must sooner or later issue the appropriate permission, while it is prohibited to issue both permissions simultaneously.
The universal co-Buchi automaton $\mathcal{A}$ over a finite alphabet Σ is given by the quadruple $\langle Q, q0, \delta, F \rangle$, where Q is a finite set of the automaton states; $q_0 \in Q$ is an initial state of the automaton, δ: $Q \times 2\ \Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is a set of rejecting states. Let an infinite word $\sigma \in (2^{\Sigma})^*$ is given; the initiation of the given word on the automaton $\mathcal{A}$ generates an infinite sequence of states $q_0 q_1 q_2 \ldots \in Q^*$. The start is considered admitting if it contains only a finite number of rejecting states. The automaton $\mathcal{A}$ admits the word σ if all starts of the given the word by the automaton $\mathcal{A}$ were admitting. The automaton $\mathcal{A}$ language is denoted as $\mathscr{L}(\mathcal{A})$ and is a set $\{\sigma \in (2^{\Sigma})^* \mid \mathcal{A} \text{ takes } \sigma\}$. Further, universal co-Buchi automata are depicted as directed graphs with vertices Q and a symbolic representation of the relation δ in the form of propositional Boolean formulas $\mathbb{B}$ (Σ). Rejection states F will be marked with double lines.

Let Σ is a set of propositional variables; we divide it into two parts: I being variables controlled by their environment and O being variables controlled by the system. In this definition, we mean by the environment the external environment with which the automaton system interacts, and the automation system itself will be understood as the system. Thus, it is assumed that the variables from set I will be changed only by the external environment, and the variables from set O will be changed only by the automaton system. The transition system $\mathcal{T}$ is given by the triple ⟨T, $t0$, τ⟩, where T is a finite set of states; t_0 is an initial state, and τ: $T \times 2^I \rightarrow 2^0 \times T$ is the transition function. If for a given state $t \in T$ and variables $\boldsymbol{i} \neq \boldsymbol{i}' \in 2I$ it follows from $\tau(t, \boldsymbol{i}) = (\boldsymbol{o}, \_)$ and $\tau(t, \boldsymbol{i}') = (\boldsymbol{o}', \_)$ that $\boldsymbol{o} = \boldsymbol{o}'$, then the system of transitions is called the Moore transition system, otherwise. the Mealy transition system.

Starting an infinite sequence $\boldsymbol{i_0 i_1} \ldots \in (2^I)^*$ on $\mathcal{T}$ generates an infinite trace $(\{t_0\} \cup \boldsymbol{i_0} \cup\ )(\{t_1\} \cup \boldsymbol{i_1} \cup \boldsymbol{o_1}) \ldots \in (2^{T \cup I \cup O})^*$, where $\tau(t_j, i_j) = (o_j, t_{j+1})$ for every $j \geq 0$. The path $\omega \in (2^{I \cup O})^*$ is the projection of the trace onto propositional variables. The set of all paths generated by the transition system $\mathcal{T}$ will be denoted by $Paths(\mathcal{T})$. The transition system realizes the linear temporal logic formula if $Paths\ \mathcal{T} \subseteq \mathscr{L}(\varphi)$.

The product of the transition system $\mathcal{T} = \langle T, t_0, \tau \rangle$ and the co-Buchi automaton $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$ is a traversal graph $\mathcal{G} = \langle V, E \rangle$, where $V = T \times Q$ is the set of vertices and $E \subseteq V \times V$ is a set of edges such that ((t, q), (t ', q')) $\in E \Leftrightarrow \exists \boldsymbol{i} \in 2^I.\ \exists \boldsymbol{o} \in 2^0.\ \tau(t, \boldsymbol{i}) = (\boldsymbol{o}, t\ ')$ and $(q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta$.

The problem of synthesizing reactive automata systems is to find a minimal automaton system that will implement the given specification. There are many approaches to solving this problem, but one of the most successful is the reduction to the Boolean formula satisfiability problem (Biktashev & Vashkevich, 2013). The main advantage of this approach and the speed at which the target system is located is the fact that when improving programs for solving the Boolean formula satisfiability problem, the work of the search algorithm for the reactive target system will also improve. Usually, the specification for the reactive target system is given in linear temporal logic formulas; however, it is often necessary to indicate behaviour examples of the sought system. Within this paper's frameworker, we will refer to the sequences of pairs of vectors $\boldsymbol{i} \in 2 \mid I \mid$

and $\boldsymbol{o} \in 2 \mid O \mid$ defining the state of variables controlled by the ensystem's environment and variables as scripts. A set of several scripts will be called behaviour examples. A separate pair that makes up the script will be called a script element. A reactive system is considered to implement behavior examples to reproduce every script in a given set. Behaviour examples allow us to specify additional system properties that were not presented in the LTL specification. Also, the task of synthesizing a reactive system is often posed only by behaviour examples (Finkbeiner & Schewe, 2007).

Annotation function λ: $T \times Q \to \{\bot\} \cup \mathbb{N}$ is a function that assigns to the vertices of the starting graph either a natural number k, or $\bot$ if the vertex is unreachable. An annotation function is considered correct if the following conditions are met: a natural number is assigned to a pair of initial states (t_0, q_0); a natural number k is assigned to the pair of states (t, q), then for all $\boldsymbol{i} \in 2^I$ and $\boldsymbol{o} \in 2^O$ such that $\tau\ (t, \boldsymbol{i}) = (\boldsymbol{o}, t')$ and $(q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta$, a greater or equal, or strictly greater natural number must be assigned to the pair (t', q'), if $q' \in F$. That is, $\lambda\ (t', q') \triangleright q'k$, where $\triangleright$ q' :=>, if $q' \in F$, $\geq$ otherwise /.

Let us describe the construction of a system of constraints for given $\mathcal{T}$, $\mathcal{A}$, and λ, which is solvable only if the annotation function is correct. By definition, the correctness of the annotation function can be proved by checking all transitions in the graph, for this, we code $\mathcal{T}$, $\mathcal{A}$ and λ with the following propositional variables:
• The transition function τ of the transition system $\mathcal{T}$ is represented by two propositional variables: $o_t, \boldsymbol{i}$ for each outgoing variable controlled by the system $o \in O$ and $\tau_{t, i, t'}$, denoting the transition from state t to state t'. Let $(t, t') \in T \times T$ and $\boldsymbol{i} \in 2^I$, then $\tau_{t, i, t'} = True \Leftrightarrow \tau\ (t, \boldsymbol{i}) = (\_, t')$ and $o_{t, i} = True \Leftrightarrow \tau\ (t, \boldsymbol{i}) = (\boldsymbol{o}, \_)$, where $o \in \boldsymbol{o}$.

• The transition relations δ: $Q \times 2^{I \cup O} \times Q$ of the co-Buchi automaton $\mathcal{A}$ can be represented as a formula δ t, q, i, q' with the variables $o_{t, i}$ in such a way that the assignment $\boldsymbol{o}$ for the variables $o_{t,i}$ satisfies $\delta_{t, q, i, q'} \Leftrightarrow (q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta$.

• For convenience, we divide the annotation function into two parts: $\lambda^{\mathbb{B}}$: $T \times Q \to \mathbb{B}$, which represents the reachability of the vertex, and $\lambda_{\#}$: $T \times Q \to \mathbb{N}$. For each $t \in T$ and $q \in Q$, we introduce the variable $\lambda_{t, q}^{\mathbb{B}}$ = True $\Leftrightarrow$ the vertex (q, t) in the starting graph is reachable from the initial vertex, and the variable $\lambda_{t, q}^{\#}$ represented by a bit vector and equal to the value $\lambda\ (t, q)$.
Using the propositional variables described above, the Boolean formula is compiled that checks the correctness of the annotation function.

$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda_{t,q}^{\mathrm{B}} \to \bigwedge_{q' \in Q} \bigwedge_{i \in 2^I} \left(\delta_{t,q,i,q'} \to \bigwedge_{t' \in T} (\tau_{t,i,t'} \to \lambda_{t',q'}^{B} \wedge \lambda_{t',q'}^{\#} \triangleright \lambda_{t,q}^{\#}) \right) \right), \qquad (1)$$

If for the given $\mathcal{T}$, $\mathcal{A}$ and λ the propositional variables satisfy the system of constraints, then the annotation function is correct (Heule & Verwer, 2013). There are several different approaches to the synthesis of automaton systems, which allow simultaneously taking into account the specification in the form of linear temporal logic formulas and behaviour examples. Among them there are the following methods:

- An iterative approach based on reducing to the Boolean formula satisfiability problem, the work of which consists of several stages: representing scripts in the form of a script tree, which is then represented as a Boolean formula; generating an automaton system according to the obtained formula; checking the synthesized automaton system using the model checking approach (Finkbeiner & Schewe, 2013) for compliance with linear temporal formulas; if counterexamples are identified at the current stage, they are added to the script tree with a special mark and the whole process is repeated anew. Thus, an automaton that implements a given specification is gradually deduced, or a fact that denies the existence of such a system is revealed. An important point in the described approach is the use of software tools for solving the Boolean formula satisfiability problem that support an incremental search for a solution (Clarke et al.,1999), which

allow not to completely rerun the search for a solution after adding another counterexample.

- Another approach is the method (Vashkevich & Biktashev, 2016), which is based on reducing to the satisfiability problem for a Boolean formula with quantifiers and checking models with a constraint on the length of the tested paths. This approach is somewhat similar to the previous one, but now the boundary used in model validation to check the length of the paths being tested, is sequentially increased instead of adding counterexamples. After generating the automaton, as in the previous method, a check is performed again whether the obtained automaton satisfies the LTL properties; if not, then the boundary used for checking models is increased and the operation is repeated anew. There is also a modification of this approach, in which a formula with quantifiers is translated into a satisfiability formula by an almost exponential expansion of universality quantifiers.

2.1. Theoretical research

Today, perhaps the only way of presenting behaviour examples, that suggests the possibility of adding them to the specification when using the synthesis approach with a constraint on the size of the system is to represent behaviour examples in the form of linear temporal logic formulas. Since this method does not require modifications of the algorithm for reducing to the Boolean formula satisfiability problem used in the bounded synthesis approach, we will consider it as the starting point for further comparisons of the results with other proposed methods. To represent behaviour examples in the form of linear temporal formulas, it is necessary to sequentially replace each transition in the list of elements of a specific script with a construction of the form $\boldsymbol{i}_k \rightarrow \boldsymbol{o}_k \wedge X(\boldsymbol{i}_{k+1} \rightarrow \boldsymbol{o}_{k+1} \wedge \ldots)$, where the vector $\boldsymbol{i}$ specifies the state of environment variables in this element of the script, the vector $\boldsymbol{o}$ specifies the state of the system variables, and X denotes the temporal logic operator “next”.

Suppose that $I = \{e_{11}, e_{12}, e_2, e_3, e_4\}$, $O = \{z_1, z_2, z_3\}$ and there are the following scripts

$$\begin{array}{l} (e_{11} \mid z_1) \rightarrow (e_2 \mid) \rightarrow (e_{12} \mid z_2) \rightarrow (e_2 \mid) \\ (e_{11} \mid z_1) \rightarrow (e_4 \mid z_3) \end{array}, \quad (2)$$

Then the temporal formulas defining these scripts will look like this

$$\begin{array}{l} (e_{11} \wedge \neg e_4 \wedge \neg e_{12} \wedge \neg e_3 \wedge \neg e_2) \\ \rightarrow (z_1 \wedge \neg z_2 \wedge \neg z_3 \wedge X((e_2 \wedge \neg e_{11} \wedge \neg e_4 \wedge \neg e_{12} \wedge \neg e_3) \\ \rightarrow (\neg z_1 \wedge \neg z_2 \wedge \neg z_3 \wedge X((e_{12} \wedge \neg e_{11} \wedge \neg e_4 \wedge \neg e_3 \wedge \neg e_2) \\ \rightarrow (z_2 \wedge \neg z_1 \wedge \neg z_3 \wedge X((e_2 \wedge \neg e_{11} \wedge \neg e_4 \wedge \neg e_{12} \wedge \neg e_3) \\ \rightarrow (z_1 \wedge \neg z_2 \wedge \neg z_3)))))) \\ (e_{11} \wedge \neg e_4 \wedge \neg e_{12} \wedge \neg e_3 \wedge \neg e_2) \\ \rightarrow (z_1 \wedge \neg z_2 \wedge \neg z_3 \wedge X((e_4 \wedge \neg e_{11} \wedge \neg e_{12} \wedge \neg e_3 \wedge \neg e_2) \\ \rightarrow (z_3 \wedge \neg z_1 \wedge \neg z_2)))))) \end{array}, \quad (3)$$

As can be seen from the example, the disadvantage of this approach is a strong increase in size of the formula defining the specification of the system, namely, $(|I| + |O|) \cdot |SC|$, where $|I|$ is the size of the set of environment variables, $|O|$ is the size of the set of variables of the system, and $|SC|$ is the number of script elements in the given behaviour examples. Due to an increase in the formula, the amount of time required for construction of co-Buchi automaton increases, and the co-Buchi automaton itself also increases directly, which ultimately leads to an inefficient growth of the corresponding Boolean formula and an increase in the time spent on synthesizing the target automaton system. In addition, this method does not take into account

the peculiarities of the behaviour examples.

Another way to represent system behavior examples is to represent them as a tree or graph of scripts. The original algorithm for constructing a script tree was described in (Vashkevich, N.P. (2004; Eén & Sörensson, 2003); we will use its modified version in the given work. A tree or script graph is a tree or, respectively, a graph, where the state of the environment variables and the expected state of the system variables are written on each edge. In traversing the tree or graph using the projection of conditions on its edges, we can get all the scripts from the list of scripts on which it is constructed. Initially, the tree contains only one vertex, i.e., its root, then script elements are sequentially added according to the following principle: if there is an edge from the current vertex according to the condition presented in the script element, then a transition is performed along this edge to the next vertex; if there is no such edge, then a new vertex is added to which a new edge is drawn from the current vertex with the necessary condition; after that, a transition to the created vertex is performed. For example, let's say we have the following scripts:

$$\begin{array}{l}
(e_2|) \\
(e_2|) \to (e_{11}|z_1) \to (e_2|) \to (e_{12}|z_2) \to (e_2|) \\
(e_3|z_1) \to (e_2|) \to (e_{12}|z_2) \to (e_2|) \\
(e_2|) \to (e_{11}|z_1) \to (e_2|) \to (e_{12}|z_2) \to (e_3|z_1) \to (e_2|) \to (e_{12}|z_2) \to (e_2|) \\
(e_3|z_1) \to (e_2|) \to (e_{12}|z_2) \to (e_3|z_1) \to (e_2|) \to (e_{12}|z_2) \to (e_2|) \\
(e_4|z_3) \\
(e_2|) \to (e_{11}|z_1) \to (e_4|z_3) \\
(e_3|z_1) \to (e_4|z_3)
\end{array}, \qquad (4)$$

The script tree constructed according to the algorithm described above for the given behaviour examples, is shown in Figure 1.

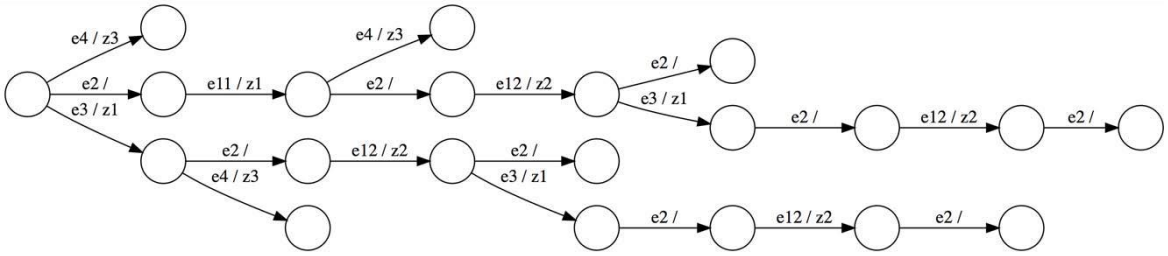

Figure 1. An option of the script tree for the given behaviour examples

As we can see from the example, the script tree allows us to more efficiently define behaviour examples by reducing the amount of information required to represent scripts.

Let's develop the idea of combining behaviour examples into a script tree, and let’s move on to the script graph. A script graph differs from a script tree in that now the final elements of the scripts are also merged in addition to combining the initial elements of the scripts. The process of building a script graph starts in the same way as building a tree, but now back edges are also remembered for each vertex. After the end of the first stage, all vertices from which no edges outgo merge into one final vertex. Further, starting from the final vertex, a recursive traversal of the graph occurs along the back edges in such a way that if several back edges originate from the vertex with the same condition, then the vertices to which these edges lead merge into one, after which these vertices are passed. The process is repeated again until it reaches the root. It is

worth noting one key point of this algorithm: when traversing and merging vertices along back edges, it is very important not to allow a situation when we add the ability to get new scripts that were absent in the original list. For a better understanding, we will give an example of such a situation: for example, after the first step, we have a tree, which is shown in Figure 2.

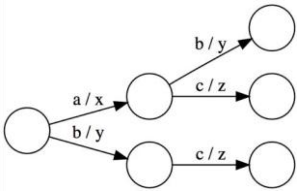

Figure 2. A script tree in which a path conflict will occur when merging vertices

Note that if we do not take into account the case described above, then, when merging vertices along back edges, we get the following script graph (Figure 3).

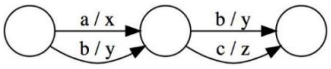

Figure 3. A graph of scripts in which unintended transitions are possible

The graph shown in Figure 3 displays the script $(b \mid y) \rightarrow (b \mid y)$ that was absent in the original list of scripts. This can lead to the synthesis of an incorrect automaton system and cause problems with the search or even the absence of a solution for the Boolean formula satisfiability problem if the formulas of linear temporal logic in the specification prohibit or contradict such transitions. To avoid the described problems, we can use the approach applied in the algorithm for minimizing finite state automata, with the help of which the vertices are divided into equivalence classes and then a new script graph is constructed from them. As an example, Figure 4 shows a script graph for the same behaviour examples for which the script tree was constructed above:

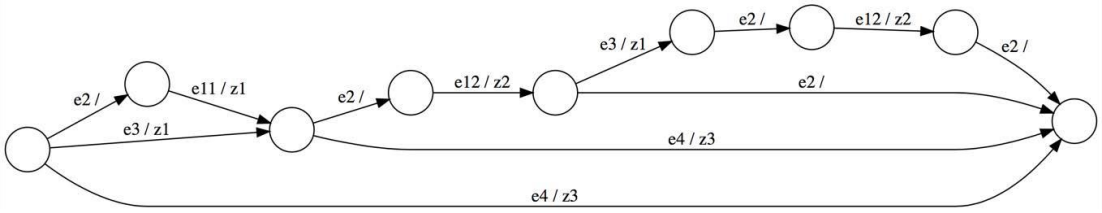

Figure 4. An example of a script graph

Using a script graph can further reduce the amount of information needed to present behaviour examples. The methods described later in this chapter will be based, in one way or another, on the search for matching the vertices of a script tree or graph to the vertices of the target transition system.

As mentioned above, the methods of accounting for behaviour examples in the synthesis of automaton models are based on the search for a mapping between the script graph's vertices and the states of the transition system. To do this, we introduce a new type of Boolean variables, $S_{t,j,}$ defining the correspondence between vertices and states. Thus, $S_{t,\,j}$ is true if and only if the state t of the transition system corresponds to the vertex j of the script graph. We also denote by $out\ (j)$ the set of edges outgoing from the vertex j and represented in the form of triplets from the vertex to which the edge enters and conditions for the state of variables controlled by the environment and variables controlled by the system. To confirm the correctness of the established correspondence between the script graph's vertices and the states of the transition system, it is necessary for each vertex to check the correspondence between the edges outgoing from it and the transitions from the transition system. In other words, if the state t corresponds to the vertex j, then for each

edge outgoing from the current vertex with conditions ***i*** and ***o*** to the vertex *j*' there must be a transition in the system of transitions from the state *t* to the state *t*Ö with the state of the environment variables equal to the vector ***i*** and the system variables equal to vector ***o*** and the state t' thus must correspond to the vertex *j*'. The predicate that sets this condition is shown below.

$$\bigwedge_{t\in T}\bigwedge_{j\in ST} \rightarrow \bigwedge_{(j',i,o)\in out(j)} \bigvee_{t'\in T} (\tau_{t,i,t'} \wedge O_{t,i} \wedge S_{t',j'}), \qquad (5)$$

Where the variable $\tau_{t,i,t'}$ from the original encoding sets the transition from the state *t* to the state *t*' when the environment variables are equal to the vector ***i***, and the vector of variables $\boldsymbol{o}_{t,i}$ sets the state of the environment variables equal to the vector ***o*** when this transition is performed.

This expression can be used to check the correctness of a given match and search for it when passing it to the program for finding the assignment of variables at which the formula will be executed. The only additional condition that must be added to the formula is $S_{t0,j0}$, which asserts the correspondence between the initial state of the transition system and the root vertex of the script graph.

The size of the resulting formula is of $\mathcal{O}(n^2 \cdot |SG|^2 \cdot |O|)$ clauses, and it contains $\mathcal{O}(n \cdot (2^{|I|} \cdot |O| + |SG|))$ variables, where $n = |T|$ is the size of the transition system; |SG| is the size of the script graph, |I| and |O| are the sizes of the sets of variables controlled by the environment and the system, respectively. It is assumed that this predicate will be added to the original expression presented in (Peter Faymonville et al., 2017), which will increase the number of variables included in the original formula by $n \cdot |SG|$. However, it is worth noting that at the moment, the proposed formula does not use the predicate from the original formula anymore, which states that for each state, there is a transition from it for all variants of states of environment variables.

$$\bigwedge_{t\in T}\bigwedge_{i\in 2^I}\bigvee_{t'\in T} \tau_{t,i,t'}, \qquad (6)$$

To do this, we modify our formula by changing the condition that specifies the correspondence between transitions in the script graph and in the transition system. This modification changes the meaning of the original formula. Now, if the state *t* corresponds to the vertex *j*, then for each edge outgoing from the given vertex with conditions ***i*** and ***o*** to the vertex *j* ', and if the transition in the transition system leads from the state *t* to the state *t*' with the state of the environment variables equal to the vector ***i***, then the state of the system variables must be equal to the vector ***o***, and the state *t* 'must correspond to the vertex *j*'. The modified formula is as follows:

$$\bigwedge_{t\in T}\bigwedge_{j\in ST} S_{t,j} \rightarrow \bigwedge_{(j.i,o)\in out(j)} \bigvee_{t'\in T} (\tau_{t,i,t'} \rightarrow (o_{t,i} \wedge S_{t',j'})), \qquad (7)$$

It is believed that the updated formula will allow us to quickly find the right solution or the fact that there is none.

Before moving on to methods based on reducing the Boolean formula's satisfiability problem with quantifiers, it is worth dwelling on modifying the script graph, namely, dividing it into clusters. This approach is aimed at an even stronger compression of information about behaviour examples, which leads to a simplification of the resulting Boolean formula. Consider two approaches to clustering, i.e., vertex clustering and global clustering.

First, we will consider the vertex clustering approach, as it involves less change in the script graph. It consists of the fact that the conditions on the script graph's edges are divided into conditions for variables controlled by the environment and conditions controlled by the system. Further, the edges outgoing from the vertex are

divided into clusters according to the conditions for the environment variables; for this, a new imaginary vertex is added, to which an edge leads from the original vertex with the condition for the environment variables and from which the edges come with conditions for the variables controlled by the system and associated with this cluster. No new vertex is added for clusters with only one edge.

Now let's look at the second approach. In contrast to vertex clustering, it is assumed that all edges in the graph are split into clusters at once when using the global clustering method. For this purpose, the conditions on the edges are divided by the type of variables, as in the previous version. Further, global clusters associated with conditions on variables controlled by the environment are constructed using all the edges of the graph. After that, the edges located inside of such clusters are divided again into subclusters, this time connected by the conditions for the variables controlled by the system.

2.2. Methods of accounting for behaviour examples when reducing to the Boolean formula satisfiability problem with quantifiers

The methods of accounting for behaviour examples when reducing to the Boolean formula satisfiability problem are based on the same approach to finding a correspondence between a script graph and a transition system as methods when reducing to a formula without quantifiers. The main difference from the quantifier-free notation in expression (Peter Faymonville et al., 2017) was the presence of a universality quantifier for environment variables, which made it possible to represent the variables $\tau_{t,\,t'}$ and $\boldsymbol{o}_t$ as functions of the variables $\boldsymbol{i}$. This solution made it possible to reduce the size of the formula itself and the number of variables included in it; due to this, it becomes possible to more efficiently code the predicate, which was not available in the early version of the methods the Boolean formula satisfiability.

The first version of the formula is obtained by modifying the quantifier-free version with an amendment to the new conditions. The meaning of the new predicate is as follows: if the state t corresponds to the vertex j, then for each edge outgoing from the current vertex with the conditions $\boldsymbol{i}$ and $\boldsymbol{o}$ to the vertex j ', if at the moment the environment variables are in the state $\boldsymbol{i}$, then there must be a transition in the transition system, which leads from the state t to the state t', and the state of the system variables must be equal to the vector $\boldsymbol{o}$ and the state t' must correspond to the vertex j '. The new expression looks like this

$$\bigwedge_{t\in T}\bigwedge_{j\in SG} S_{t,j} \to \bigwedge_{(j.i,o)\in out(j)} \bigvee_{t'\in T} (i \to (\tau_{t,t'} \wedge O_t \wedge S_{t',j'})), \tag{8}$$

We would like to point out immediately that the variables $\boldsymbol{i}$ and $\boldsymbol{o}_t$ standing under the operator $\bigvee_{t'\in T}$ do not depend on t ', so they can be taken out from under its action without any losses

$$\bigwedge_{t\in T}\bigwedge_{j\in SG} S_{t,j} \to \bigwedge_{(j.i,o)\in out(j)} \to O_t \wedge \bigvee_{t'\in T} (\tau_{t,t'} \wedge S_{t',j'}), \tag{9}$$

The size of the first version of the formula is of $\mathcal{O}$ ($n\cdot$ | SG |) $^2\cdot$ (| O |) + | I | + n)) clauses and the number of new variables is still $\mathcal{O}$ ($n \cdot$ | SG |).

Now we turn to the second version of the predicate to solve a Boolean formula's satisfiability problem with quantifiers. The second version of the formula proposes to use the previously proposed global clustering principle. To do this, we divide this graph into clusters by environment variables and then denote the set of vectors characterizing clusters as IC. Also, now the set out (j, $\boldsymbol{i}$) contains only those edges outgoing from the vertex j that lie in the cluster for the vector of environment variables $\boldsymbol{i}$. The set of vertices in the corresponding cluster will be denoted as ST ($\boldsymbol{i}$). Suppose the environment variables are in the state $\boldsymbol{i}$ and the vertex j corresponds to the state t. In that case,hen there must be a corresponding transition in the transition system for each edge outgoing from this vertex and lying in the cluster associated with the vector $\boldsymbol{i}$, with the

condition ***o*** to the vertex *j* ’.

$$\bigwedge_{i\in IC} i \rightarrow \bigwedge_{t\in T}\bigwedge_{j\in SG(i)} S_{t,j} \rightarrow \bigwedge_{(j',o)\in out(j,i)} O_t \wedge \bigvee_{t'\in T}(\tau_{t,t'} \wedge S_{t',j'}), \quad (10)$$

The external operator $\bigwedge_{i\in IC}$ fixes the state of the environment variables, which should make it easier to check and find dependencies for the variables $t_{t,t'}$ and o_t. The size of the second version of the formula is of $\mathcal{O}$ (| IC | · (| I | + n· | SG | ^{2}· (| O | + n))) clauses. This version of the formula is beneficial for relatively small sizes of | IC | , otherwise, it can lead to an increase in the time required to find the right solution.
As in the case of the quantifier-free formula, we use the predicate of the transition function completeness for the third version of the formula from the original formula.

$$\bigwedge_{t\in T}\bigwedge_{t'\in T}\tau_{t,t'}, \quad (11)$$

Also, the vertex clustering approach will be applied in the current version of the formula for greater efficiency; this makes it possible to define the script graph more efficiently, without suffering from cases with a large number of clusters. We denote the set of vectors specifying the state of environment variables associated with clusters for a vertex *j* as (*j*).

$$\bigwedge_{t\in T}\bigwedge_{j\in SG} S_{t,j} \rightarrow \bigwedge_{i\in IC(j)} i \rightarrow \bigwedge_{(j',o)\in out(j,i)} O_t \wedge \bigwedge_{t'\in T}(\tau_{t,t'} \wedge S_{t',j'}), \quad (12)$$

The asymptotics of the new formula is similar to the first version and amounts to $\mathcal{O}$ (n· | SG | ^{2}· (| O | + | I | + n)), but despite this, it is expected that this formula will be more efficient due to more efficient work with constraints on variables defining the transition function of the target transition system, and using vertex clustering.

3. ANALYSIS OF THE RESULTS

In the experimental part, the proposed methods of accounting for behaviour examples when synthesizing automaton models using temporal formulas are compared, and an analysis of the results will be presented. Testing was performed on a computer running OS Ubuntu 16.04, having an Intel Core i5 5257U processor, and 4GB RAM. The transformation of linear temporal properties into a co-Buchi automaton was carried out using the “spot” utility, “cryptominisat” was used as a software tool for finding a solution to the satisfiability problem for a Boolean formula, and rareqs was used as a means for solving the Boolean formula satisfiability problem with quantifiers. To process the input data, we used a modified version of the BoSy program (BoSy, 2016) written with the use of the Swift programming language, into which the methods proposed in work are embedded. We also used Python scripts to generate behaviour examples and represent scripts in the form of temporal formulas. A part of the test set of entities presented at the SYNTCOMP competition for the synthesis of reactive systems was used as tests. Random behaviour examples were generated for each entity to test the effectiveness of the methods of accounting for behaviour examples using the program written in the Python language. The generation was carried out as follows: an automaton system was delivered to the program input, for which it was necessary to construct behaviour examples presented in the form of a directed graph with conditions for environment variables and system variables indicated on the transitions, and the number of scripts that need to be generated. Further, the length n was randomly determined for each script within the range from 2 to | the automaton system size | ·5. After that, starting in the initial state randomly each time, an outgoing transition to the next state was selected, and a condition for the variables of this transition was added to the script. If there are multiple environment variable assignments that satisfy the transition condition, then one of them is randomly selected. After a new specification was generated in the form of linear temporal formulas and behaviour examples, it was passed to the input of another utility,

which converted the behaviour examples into linear temporal logic formulas for comparison with the base implementation. The resulting data were delivered to the input of the tested program. During testing, the running time of each individual step of the bounded synthesis algorithm was measured and the total running time of all steps. About twenty starts were made for each input data and each method, among which the average time at each stage was selected, and before that, five starts were previously performed without time measurements.

The experiments' results for approaches based on reducing the Boolean formula satisfiability problem and approaches based on reducing the satisfiability problem of a Boolean formula with quantifiers are presented in the graphs shown in Figure 5 and Figure 6, respectively. The vertical scale in the graphs represents the time in seconds, and the horizontal scale gives a number of entities for which a solution was found. Each entity's running time was determined as the total running time of all stages of the “bounded synthesis” algorithm.

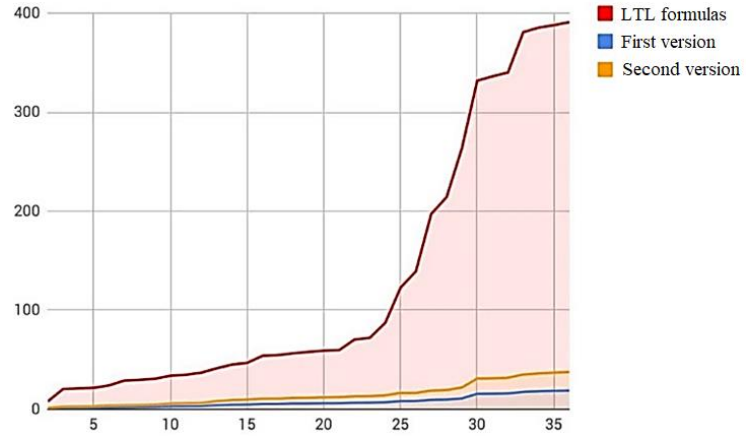

Figure 5. Comparison of the work speed for various methods when reducing to the Boolean formula satisfiability problem

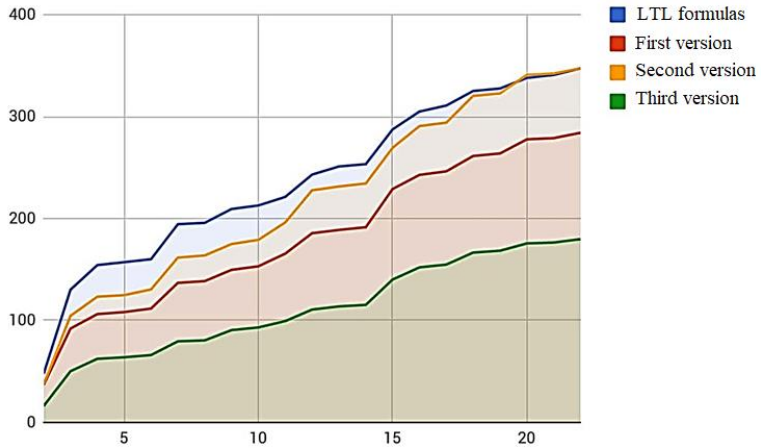

Figure 6. Comparison of the work speed for various methods when reducing to the Boolean formula satisfiability problem with quantifiers

As can be seen from the graph shown in Figure 5, for the variant with reducing the problem to the Boolean formula satisfiability, the first version of the formula showed itself in the best way, although it is worth noting that the second version is not far behind it. Such a large increase is justified by the fact that in the course of encoding for the Boolean formula satisfiability problem in the bounded synthesis version, an enumeration over $2^{|Å|}$ variables is performed, and the formula grows very much with an increase in the co-

Buchi automaton. There is no dependence on $2^{|Å|}$ in the proposed methods; therefore, the formula does not increase so much with the growth of the script graph.

The graph presented in Figure 6 shows that the leader is the third version of the encoding, which uses the predicate of the transition function completeness and vertex clustering; the performance gain is also significant, although it is not as large as for other versions.

After the experiments, the fact was confirmed that when the behaviour examples are represented in the form of linear temporal logic formulas, the time required for the construction of a co-Buchi automaton significantly increases due to the complexity of the LTL specification structure.

4. CONCLUSION

The paper shows new methods of accounting for behaviour examples in the synthesis of automaton models using temporal formulas based on the approach of synthesizing automaton models with a constraint on the target system's size. Also, new options for representing behaviour examples in the form of clustered script graphs were proposed. In addition, an experimental study was carried out, which showed the high efficiency of new methods and their multiple superiority over the basic methods.

ACKNOWLEDGMENTS

RFBR funded the reported study according to the research project № 19-07-00516**.**

REFERENCES

Biktashev, R.A., Vashkevich, N.P. (2013). Models of event-driven non-deterministic automata for the formal presentation of the main properties of control systems for parallel processes and resources. *Infocommunication technologies, 11*(3), 95-98.

BoSy. Reactive synthesis tool based on constraint solving. — 2016. — URL: https://github.com/reactive-systems/bosy

Clarke, E.M., Grumberg, O., Peled, D. (1999). Model checking. MIT press, USA.

Dubinin, V.N., & Drozdov, D.N. (2016). Design and implementation of control systems for discrete event systems based on hierarchical modular non-deterministic automata (Part 1. Formal model). Proceedings of higher educational institutions. *Volga region. Technical science, 1*(37),28-39.

Dubinin, V.N., Budagovsky, D.A., Drozdov, D.N., & Artamonov, D.V. (2016). Design and implementation of control systems for discrete event systems based on hierarchical modular non-deterministic automata (Part 2. Methods and tools) Proceedings of higher educational institutions. *Volga region. Technical science, 2*(38), 18-32.

Eén, N., Sörensson, N. (2003). Temporal induction by incremental SAT solving. Electr. *Notes Theor. Comp. Sci, 89*(4), 543–560.

Finkbeiner, B., Schewe, S. (2007). SMT-based synthesis of distributed systems. In: Proceedings of AFM.

Finkbeiner, B., Schewe, S. (2013). Bounded synthesis. *STTT, 15*(5-6), 519–539.

Heule, M.J., Verwer, S. (2013). Software model synthesis using satisfiability solvers. *Empir. Software Eng,*

18(4), 825–856.

Pashchenko, D.V., Martyshkin, A.I., Trokoz, D.A. (2020). Decomposition of Process Control Algorithms for Parallel Computing Systems Using Automata Models. Proceedings - 2020 International Russian Automation Conference, RusAutoCon 2020, 839-845, 9208165

Pashchenko, D.V., Trokoz, D.A., Martyshkin, A.I., Sinev, M.P., Svistunov, B.L. (2020). Search for a substring of characters using the theory of non-deterministic finite automata and vector-character architecture. *Bulletin of Electrical Engineering and Informatics*, *9*(3), 1238-1250

Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, Leander Tentrup. Encodings of Bounded Synthesis. International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2017. P. 354-370. doi:10.1007/978-3-662-54577-5_20

Vashkevich, N.P. (2004). *Nondeterministic automata in the design of parallel processing systems: Textbook.* - Penza: Publishing house of the Penza State University, - 280 p.

Vashkevich, N.P., & Biktashev, R.A. (2011). The Benefits of a Formal Language Based on the Concept of Non-determinism in the Structural Implementation of Parallel Systems of Logical Control of Processes and Resources / Proceedings of Higher Educational Institutions. Volga region. *Technical science, 1,* 3-11.

Vashkevich, N.P., & Biktashev, R.A. (2011). The benefits of a formal language based on the concept of non-determinism for functional description and transformation of algorithms for managing processes and resources in parallel systems. *Telecommunications, 1,* 18-25.

Vashkevich, N.P., & Vashkevich, S.N. (1996). *Non-deterministic automata and their use for the synthesis of control systems*. Part 1. Equivalent transformations of non-deterministic automata: Textbook. - Penza: Publishing house of the Penza State Technical University, - 88 p.

Vashkevich, N.P., Biktashev, R.A. (2016). Non-deterministic automata and their use for the implementation of parallel information processing systems: Monograph - Penza: PSU Publishing House, 394 P.

Vashkevich, N.P., Biktashev, R.A., Pashchenko, D.V., Kutuzov, V.V., & Sauanova, K.T. (2015). Using models of event-driven non-deterministic automata for the formal description of parallel algorithms of logical control. *Bulletin of NAS RK, 4,* 48-63.

Volchikhin, V.I., Vashkevich, N.P., & Biktashev, R.A. (2013). Models of event-driven non-deterministic automata for representing control algorithms for interacting processes in multiprocessor computing systems based on the use of the monitor mechanism. *Proceedings of higher educational institutions. Volga region. Technical science, 2*(26), 5-14.