НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
март–апрель 2025 Том 25 № 2 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
March–April 2025 Vol. 25 No 2 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2025-25-2-328-338

Verification of the formal requirements for the system behavior based on automaton objects

Fedor A. Novikov[1], Irina V. Afanasieva[2], Ludmila N. Fedorchenko[3]✉, Taisia A. Kharisova[4]

[1] Peter the Great St. Petersburg Polytechnic University, Saint Petersburg, 195251, Russian Federation
[2] Special Astrophysical Observatory of the Russian Academy of Sciences (SAO RAS), Nizhny Arkhyz, 369167, Russian Federation
[3] St. Petersburg Federal Research Center of the Russian Academy of Sciences, Saint Petersburg, 199178, Russian Federation
[4] Ioffe Institute, Saint Petersburg, 194021, Russian Federation

[1] fedornovikov51@gmail.com, https://orcid.org/0000-0003-4450-0173
[2] riv615@gmail.com, https://orcid.org/0000-0003-4225-4124
[3] lnf@iias.spb.su✉, https://orcid.org/0000-0002-4008-9316
[4] tais.harisova@mail.ru, https://orcid.org/0009-0002-3456-0471

Abstract

The paper studies the problems of automatic verification of the behavior of a reactive system in accordance with formalized requirements, i.e., automatic verification. The reactive system is described by interconnected automaton objects. Formalized requirements are written as conditioned regular expressions. In this case mathematically reliable verification of the system is possible. The proposed solution is based on the use of the CIAO (Cooperative Interaction of Automaton Objects) language to specify the interaction of automaton objects. This paper considers the third version of the language, CIAO v.3, which defines the means of describing automaton classes, the means of instantiating automaton objects, and linking these objects to the system using a link diagram. The requirements to be verified are specified using the so-called conditioned regular expressions constructed over a set of elementary actions and conditions defined in the system. A software tool has been developed that, using a link diagram, builds a semantic graph, i.e., a directed graph, where all paths from the initial nodes represent the execution protocols of the automata-based program, thereby specifying the semantics of the reactive system. Then, it is checked whether the automata-based program complies with the requirement defined by the conditioned regular expression. If a discrepancy is detected, the tool shows the place where the semantic graph exactly does not comply with the requirement. Algorithms have been developed that allow automatic verification of reactive systems with respect to the formalized requirements of a certain class. An example of a program is given that demonstrates elevator control in the CIAO v.3 language, and the constructed reactive system is verified for compliance with formally specified requirements. The purpose of the article is to demonstrate software implementation of the tool for automatic verification of a program in the CIAO v.3 language.

Keywords

automatic verification, conditioned regular expressions, behavior model, automata-based programming, state transition graph, UML, state machine diagram, concurrent behavior, software architecture, reactive system

For citation: Novikov F.A., Afanasieva I.V., Fedorchenko L.N., Kharisova T.A. Verification of the formal requirements for the system behavior based on automaton objects. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2025, vol. 25, no. 2, pp. 328–338. doi: 10.17586/2226-1494-2025-25-2-328-338

УДК 004.415.52, 004.434

Проверка соответствия поведения системы на основе автоматных объектов формальным требованиям

Федор Александрович Новиков[1], Ирина Викторовна Афанасьева[2], Людмила Николаевна Федорченко[3]✉, Таисия Анваровна Харисова[4]

[1] Санкт-Петербургский политехнический университет Петра Великого, Санкт-Петербург, 195251, Российская Федерация

[2] Специальная астрофизическая обсерватория Российской академии наук, Нижний Архыз, 369167, Российская Федерация

[3] Санкт-Петербургский Федеральный исследовательский центр Российской академии наук, Санкт-Петербург, 199178, Российская Федерация

[4] Физико-технический институт им. А.Ф. Иоффе Российской академии наук, Санкт-Петербург, 194021, Российская Федерация

[1] fedornovikov51@gmail.com, https://orcid.org/0000-0003-4450-0173

[2] riv615@gmail.com, https://orcid.org/0000-0003-4225-4124

[3] lnf@iias.spb.su✉, https://orcid.org/0000-0002-4008-9316

[4] tais.harisova@mail.ru, https://orcid.org/0009-0002-3456-0471

Аннотация

Введение. В работе исследованы проблемы автоматической проверки соответствия поведения реагирующей системы формализованным требованиям, а именно, автоматической верификации. Реагирующая система описывается в форме взаимосвязанных автоматных объектов. Формализованные требования записываются в виде обусловленных регулярных выражений. Показано, что в этом случае возможна математически достоверная верификация системы. **Метод.** Предложенное решение основано на применении языка спецификации взаимодействия автоматных объектов Cooperative Interaction of Automaton Objects (CIAO). В данной работе рассматривается третья версия языка, CIAO v.3, в котором определены средства описания автоматных классов, средства инстанциации автоматных объектов и связывания этих объектов в систему с помощью схемы связей. Верифицируемые требования заданы с помощью так называемых обусловленных регулярных выражений, построенных над множеством элементарных действий и условий, определенных в системе. Разработан программный инструмент, который, используя схему связей, строит семантический граф — ориентированный граф, в котором все пути из начальных узлов представляют протоколы выполнения автоматной программы, задавая тем самым семантику реагирующей системы. Выполнена проверка соответствия автоматной программы требованию, определяемому обусловленным регулярным выражением. В случае обнаружения несоответствия инструмент показывает место, где именно семантический граф не соответствует требованию. **Основные результаты.** Разработаны алгоритмы, позволяющие проводить автоматическую верификацию реагирующих систем относительно формализованных требований определенного класса. Рассмотрен пример программы, демонстрирующий управление работой лифта на языке CIAO v.3 и проведена верификация построенной реагирующей системы на соответствие формально заданным требованиям. **Обсуждение.** Продемонстрирована программная реализации инструмента автоматической верификации программы на языке CIAO v.3.

Ключевые слова

автоматическая верификация, обусловленные регулярные выражения, модель поведения, автоматное программирование, граф переходов состояний, унифицированный язык моделирования UML, диаграмма конечного автомата, параллельное поведение, архитектура программного обеспечения, реагирующая система

Ссылка для цитирования: Новиков Ф.А., Афанасьева И.В., Федорченко Л.Н., Харисова Т.А. Проверка соответствия поведения системы на основе автоматных объектов формальным требованиям // Научно-технический вестник информационных технологий, механики и оптики. 2025. Т. 25, № 2. С. 328–338 (на англ. яз.). doi: 10.17586/2226-1494-2025-25-2-328-338

Introduction and relationship to previous studies

The article discusses a special case of the general concept of verification and defines an automaton object, in comparison with other cases of using the word “automaton”.

The process of checking the compliance of the system behavior with the requirements imposed on it is referred to as *verification*[1]. Typically, such verification is performed by conducting a finite number of tests, which is called *testing*. Testing cannot ensure absolute reliability because, for most complex software systems, the set of possible behavior scenarios is infinite, and exhaustive testing is impossible. Reliability is desirable in all cases; however, in some cases, unreliability is unacceptable. Software with unacceptable failures is usually referred to as *responsible* (life-critical). Among the various methods for ensuring the reliability of responsible systems, *formal* verification methods for software are more promising than traditional testing methods. Verification, as a check for compliance with requirements, allows for various explanations and interpretations [1]. If the requirements are formalized and the system behavior is specified algorithmically, *automatic* verification is possible. We share the concepts outlined in [2] and believe that responsible software requires the use

[1] GOST R ISO/IEC 25010-2015. Information technology. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models. Moscow, Standartinform, 2015, 36 p. (in Russian)

of mathematical methods to verify software conformity to formal specifications.

Numerous formal verification methods coexist and are used that allow automation, and for the implementation of automatic verification; the choice of software design tools and software requirement specification tools is critically important.

In the most general case, when a general purpose programming language is used to construct programs, and a first-order predicate calculus language is used to define formal specifications, formal automatic verification of complex systems appears to be an insurmountably difficult or even practically insoluble task [3]. In addition, a remarkable study [4] presented examples of manual (non-automatic) verification of fairly complex algorithms. The constructions in this book are perfect in the sense that the postconditions considered there are complete, i.e., exhaustive specifications of programs. Therefore, programs verified in this way are indeed valid by construction. However, it is unlikely that Dijkstra's method can be fully automated at present — the "poles" of verification are too far apart: declarative specification and imperative program.

The next step towards verification automation was the Model Checking method. In this method, the requirement (specification) being verified is defined by a temporal logic formula, and it is not the program itself that is verified, but its model, the so-called Kripke automaton (Kripke structure) [5]. At present, this method is sufficiently advanced, supported by industrial tools and widely used, including in the discussed area of reactive systems [6]. Note that the success of the Model Checking method, in our opinion, is to some extent due to the convergence of the verification "poles": the Kripke structure is much more formalized than the program text in a programming language, and temporal logic formulas have significant expressive power when specifying processes that develop over time. The Model Checking method itself is impeccable, but two questions remain outside the scope of the method: how to ensure the completeness of specifications by means of temporal logic and how to ensure the adequacy of the program and its Kripke model.

The question of how to ensure the completeness of formal specifications remains open and nothing better than private ad hoc methods have been proposed so far. The question of the correspondence between the verified model and original program has received a satisfactory solution within the framework of the automata-based programming paradigm [7, 8]. The fact is that the automata-based programming paradigm, along with other advantages, has the property that the description of behavior in the form of interacting automata is unambiguously and formally translated, on the one hand, into general-purpose programming languages and, on the other hand, translated into the Kripke structure. Thus, if we begin developing a system by fixing the description of its behavior in the automata-based programming paradigm, the question of the adequacy of the model used and the program being verified is removed. Taking this circumstance into account, our constructions are based on the automata-based programming paradigm.

Another approach to ensuring the feasibility of automatic verification methods is to use a language (or more precisely, a family of languages) for constructing reactive systems, which we call Cooperative Interaction of Automata Objects (CIAO). The history of the language has undergone several stages of development. In particular, the first version of the CIAO language was used to create an automata-based method for determining domain-specific languages [9–11]. The second version of the CIAO language [12, 13] was used to implement a critical system for collecting and processing astrometric data [14]. In the process of using the language, a number of improvements were made, as a result of which it became possible to use it to verify low-level communication protocols [15] and to solve other problems [16]. The results obtained showed that the CIAO language can be successfully used for the automatic verification of device control programs [17, 18]. Subsequent research in this area led us to the need to clarify the behavior model that underlies the language and to improve the graphical notations.

As a result, the CIAO v.3 language [19] was developed for which automatic verification algorithms were implemented. Note that the CIAO v.3 language is designed in such a way that, based on a system of interacting automaton objects, it is possible to uniquely and efficiently (with linear complexity by the total number of transitions in the system) construct a semantic graph [17] in which all paths from the initial states define all possible system execution protocols. The so-called Conditioned Regular Expressions (CREs) [20] are used as a means of specifying formal specifications. In fact, CREs specify a pattern that must match the paths in the semantic graph. Checking that the specified elements in a given sequence occur in a certain relative order is a well-known task in syntactic analysis (pattern matching). Note that a sequence pattern is a weaker specification tool compared to temporal logic. Of course, specifying permissible sequences of actions is only a partial, not a complete description of system requirements. However, such requirements can be effectively verified automatically. Thus, it was possible to bring the "poles" of verification even closer together in comparison with the Model Checking method. The next important concept is the automaton object. We have taken this term from the lexicon of automata-based programming [19], but it should not be identified with the concept of an "automated object" [7]. Although the concept of an automaton object used here was developed on the basis of the concept of an automated object, the differences are significant, especially from the point of view of verification. There are three most important differences:

— firstly, automaton objects are endowed with internal memory in the form of local variables where it is possible to store information obtained in the parameters of events (input actions), calculate new information and transfer information to the outside in the parameters of effects (output actions). Thus, automaton objects fall out of the class of finite automata which have only finite internal memory;
— secondly, automaton objects are instances of automaton classes in the sense that objects are instances of classes in object-oriented programming. Automaton objects

receive the initial values of all variables, including the initial state during instantiation, and interact exclusively by accessing the provided interfaces. The fact that the internal behavior of an automaton object is specified by a state transition graph is not essential for verification and is not even mandatory. We prefer the automaton-based programming style described in the fundamental book [21], but we do not insist on its use;

— thirdly, the behavior of automaton objects is non-deterministic since they coexist in parallel, interact asynchronously, and are completely equal in rights. No central control automaton is assumed.

Cooperative interaction is the "zest" that ensures the feasibility of automatic verification in this case. Automaton objects are absolutely equal and interact with each other in accordance with the connection scheme [19] which connects them in the provided and required interfaces coordinated by types and kinds. An elementary act of interaction begins with the fact that in one automaton object (or in the external environment) an action (effect) is called which is transmitted through a connected pair of interfaces to another automaton object (or to the external environment) where the transmitted action is perceived as an event that triggers a transition. During the transition, the automaton object goes to another state (or remains in the same state) and, possibly, causes an effect. Calling an effect starts a new elementary interaction, and so on, the process continues. If there is no effect in the automaton object or if there is no transition in this state for the incoming event, then the event is lost and the process (thread) is terminated. In the proposed approach, the behavior of the system is a sequence (more precisely, a partially ordered set) of such elementary interactions. The proposed verification method allows us to check whether the behavior of the system corresponds to a given pattern or not.

Many researchers have considered the issues of verification of automata-based programs, which is reflected in various publications, from pioneering articles [22] to textbooks [23]. A natural question arises: why do we need another study and how does our work differ from the previous ones? A comparison is necessary. For comparison, we chose the article [24] in which the verification issue is covered as thoroughly and strictly as possible. In the approach [24], reactive systems defined by a hierarchy of finite automata are subject to verification. In the approach to verification that we are developing, the class of systems under consideration is the same: reactive systems. The methods for describing reactive systems differ somewhat in notation and the constructions used, but both methods are equally powerful since they are Turing complete [16]. The semantic graph that we introduced is also a state transition model, like the Kripke structure, but does not have a load in the form of temporal logic formulas. Conditioned regular expressions, which we use as a specification tool, are logically a weaker formalism than all temporal calculi. Thus, the approach we develop never goes beyond the limits outlined in the article [24]. The difference has another nature, and this difference is much more important than the differences in notation. The difference is in the motivation and goals of the study. In the case of verification of automata-based programs using the method [24], a method is chosen (model checking), an object is specified (an automata-based program), and a study is conducted to determine what can be obtained from this object using this method. Since the article [24] obtained the maximum of what can, in principle, be obtained, we can conclude that the study has achieved its goal, the problem is solved, and further manipulations are inappropriate. In our case, everything is the other way around. The goal is to achieve the ability to automatically guarantee error-free behavior of the system, the object is pointwise designated (a set of examples of reactive systems), and it is necessary to find a minimum of means that ensure the achievement of the goal. Note that the side effect of the minimality of the means used is, as a rule (in this case, this is true), the efficiency of the resulting algorithms and the scalability of the approach. In such a formulation of the problem, the study should not be declared complete, since new examples and new interpretations of infallibility appear in life — there is no limit to perfection.

This study demonstrated the capability of automatic verification of conformity to formal program specifications using the CIAO v.3 language.

Using the CIAO v.3 specification language

Let us consider an example of constructing a behavior specification for a fragment of an elevator control system which we started to consider in the article [16] and continued in the articles [17–20]. This example demonstrates the expressive power of the CIAO v.3 language and the benefits of introducing the concept of automata classes and a connection scheme. The fragment of the system under consideration consists of a control device and several actuators: a cabin, doors, and lighting. Technical details are omitted; our focus is on describing the interactions between the system components.

The behavior of the control device (the **Controller** class) is shown in Fig. 1, on the left.

Let us assume that there are four possible types of influence from the external environment on the system:

1. The user may want to use the system (press the door handle if the door has a handle, or press the door opening button, or show a QR code, etc). The system processes any such event as a provided command (**in** event).
2. The user may enter the elevator cabin if the doors are open and the light is on (**enter** event).
3. The user may want to stop using the system (press the door handle or press the door opening button from the inside, etc.). Such an event is designated as **out**.
4. The user may exit the elevator via open doors (**exit** event).

The actions performed during transitions from one state to another determine the behavior of the system. The reasons for transitions in the CIAO v.3 language are limited to two cases: a transition by a call event which is designated by calling the provided interface with the "command" stereotype, and a transition by a timer event which is designated by the **after** keyword (Fig. 1). We do not use changing or signaling events that are provided for in UML. In addition, the CIAO v.3 language does not use unmotivated (spontaneous) transitions upon completion [21].

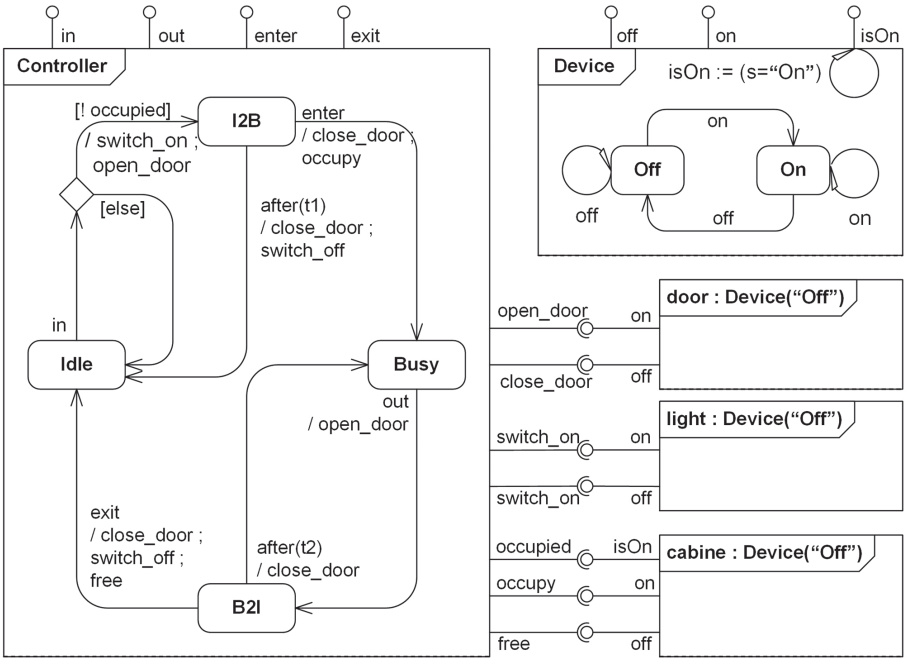


Fig. 1. Example description of an elevator control system fragment in the CIAO v.3 language

From the perspective of interaction description, it can be seen that the behavior of the actuators is very similar. In other words, each device is either 'on' and performs a function or is 'off' and does not perform a function. In addition, all devices can report what state they are 'in' at the moment: 'on' or 'off'. Therefore, in the CIAO v.3 language, it is sufficient to describe one automata class (the **Device** class, Fig. 1) which implements the behavior of the trigger [22], and then use three objects (the **door**, **light**, and **cabin** objects) in the connection scheme, linking their provided **on** and **off** interfaces with the required interfaces of the control system, as shown in Fig. 1.

In this example, we demonstrate the convenience and flexibility of the proposed connection scheme in the CIAO v.3 language. The proposed system involves only one instance of the **Controller** class and three instances of the **Device** class (Fig. 1). In such cases, the CIAO v.3 language allows us to treat a class with one instance as a "singleton" class [17], and not explicitly create a separate instance by specifying the instantiation parameters directly in the class description (initial state **Idle**).

As stated in a previous article [19], the CIAO v.3 language exists in two forms: graphical (shown above in Fig. 1) and textual (Listing 1). When using the developed software tools for automatic verification, the textual form is still used, and the graphical form is an illustration of human perception.

Listing 1. Description of the elevator control system in the CIAO v.3 language

```
01 ciao Elevator // program Elevator
in CIAO language
02   class Controller // description
of the Controller class
03     events // provided commands
04       in
05       out
06       enter
07       exit
08     effects // required commands
09       open_door
10       close_door
11       switch_on
12       switch_off
13       occupy
14       free
15     conditions // required queries
16       occupied
17     states
18       Idle -> in[occupied] -> Idle
[else] / switch_on; open_door -> I2B
19       I2B -> after(t1) / close_
door; switch_off -> Idle
20       I2B -> enter / close_door;
occupy -> Busy
21       Busy -> out / open_door ->
B2I
22       B2I -> after(t2) / close_door
-> Busy
23       B2I -> exit / close_door;
switch_off; free -> Idle
24   class Device // description of
the Device class
25     events
26       off
27       on
28     assertions // provided queries
29       isOn
30     states // states and
transitions
31       Off -> on -> On
32       Off -> off -> Off
33       On -> on -> On
```

```
34       On -> off -> Off
35   scheme // connection scheme
36     objects // instantiation of
automaton objects
37       controller = new
Controller(Idle)
38       door = new Device(Off)
39       light = new Device(Off)
40       cabin = new Device(Off)
41     links // linked interfaces
42       controller.open_door <- door.
on
43       controller.close_door <-
door.off
44       controller.switch_on <-
light.on
45       controller.switch_off <-
light.off
46       controller.occupy <- cabin.on
47       controller.free <- cabin.off
48       controller.occupied <- cabin.
isOn
49     public // public interfaces
50       controller.in
51       controller.out
52       controller.enter
53       controller.exit
54 . // end of program
```

The example presented in this paper covers only a small part of elevator control problems. In particular, the reliability of the elevator control system can be increased by providing for the handling of exceptional situations or failures. For example, it is easy to make the doors not open if the light fails to turn on (the bulb burns out), and instead escalate the malfunction to the appropriate level (for example, send an emergency signal to the dispatcher of the building in which the elevator is installed). For simplicity, we exclude all such subtleties. The purpose of this example was to demonstrate the sufficiency of the CIAO v.3 language tools in the automatic verification of a reactive system.

Construction of a semantic graph based on the connection scheme

Having a connection scheme, it is possible to create a *semantic graph*, i.e., a directed graph in which all paths from the initial nodes represent all possible protocols for executing an automaton program; thus, it defines semantics.

There are various ways to create a semantic graph. In particular, in [17, 18], we placed the actions to be performed in the graph nodes and the conditions to be checked on the arcs. The software implementation showed that it is more convenient to place actions with the conditions on graph arcs, with one action or condition per arc. This slightly increases the number of nodes but simplifies the machine-processing algorithm of the graph.

In such agreements, the algorithm for constructing a semantic graph identifies initial events based on the analysis of the connection scheme, i.e., events whose arrival from an outside world can trigger the operation of the system, and then tracing all possible paths in the state transition graphs of automaton objects, using the connection diagram to move from one state transition graph to another. In fact, constructing a semantic graph is a *symbolic execution* of an automaton program [22–27].

In this case, the analysis of the connection diagram yields an unambiguous result. The system has four external interfaces (**in**, **out**, **enter**, **exit**, the remaining interfaces are connected), and of these, only the **in** event can be processed in the initial state; thus, there is a single initial node (it is indicated by the black circle with an arrow in Fig. 2, *a*, bottom), from which a transition is possible by the **in** event.

Next, in accordance with the state transition graph (Fig. 2, *a*, top), the guard condition [**occupied**] is checked. If the elevator is occupied, then service is impossible and a return to the initial state occurs; otherwise, actions **switch_on** and **open_door** are performed (Fig. 2, *a*, bottom).

Then, two possible scenarios are possible: either the passenger enters the elevator cabin (**enter** event) or the passenger does not dare to do so. Then, after time **t1**, the doors are closed, the light is released, and the system returns to the initial state. All other transitions in Fig. 1 are processed similarly, and the semantic graph is built automatically by tracing the paths in the state transition graphs and moving from one automaton object to another according to the connection scheme. The final result is shown in Fig. 2, *b*.

Conditioned regular expressions as requirement formalization

All possible sequences of actions performed during the operation of a reactive system determine the overall behavior of the entire system. Each sequence of actions is called an execution occurrence protocol. In this case, protocols may include both acceptable sequences satisfying the system requirements and unacceptable sequences that must be classified as erroneous. Our view is that formalized requirements (specifications) must be provided in the form of a description of acceptable sequences of events/actions. Regular expressions with guard conditions (we refer to them as CREs) can be used to mathematically describe such specifications.

Various versions of regular expression languages are currently in use and co-exist [28]. We used the regular expression language with the operations of union (notation $A|B$), concatenation (notation AB), and Tseitin iteration (notation $\{A\#B\}$) [29]. In addition to the binary operation of Tseitin iteration, defined by the Kleene iteration as $A\#B = A(BA)^*$, it is also possible to use the unary Kleene operation where the expression $A\#$ is equivalent to the expression AA^*, and the expression $\#A$ is equivalent to the expression A^*. Here A and B are arbitrary regular expressions.

A special feature of the CRE is the composition of the elementary languages used. Firstly, these are the names of the events and effects that appear on transitions in the semantic graph (Fig. 2, *b*). Secondly, it is the guard conditions which are written in square brackets. Thirdly, it is the symbol '*A*' (from the word any) which denotes any

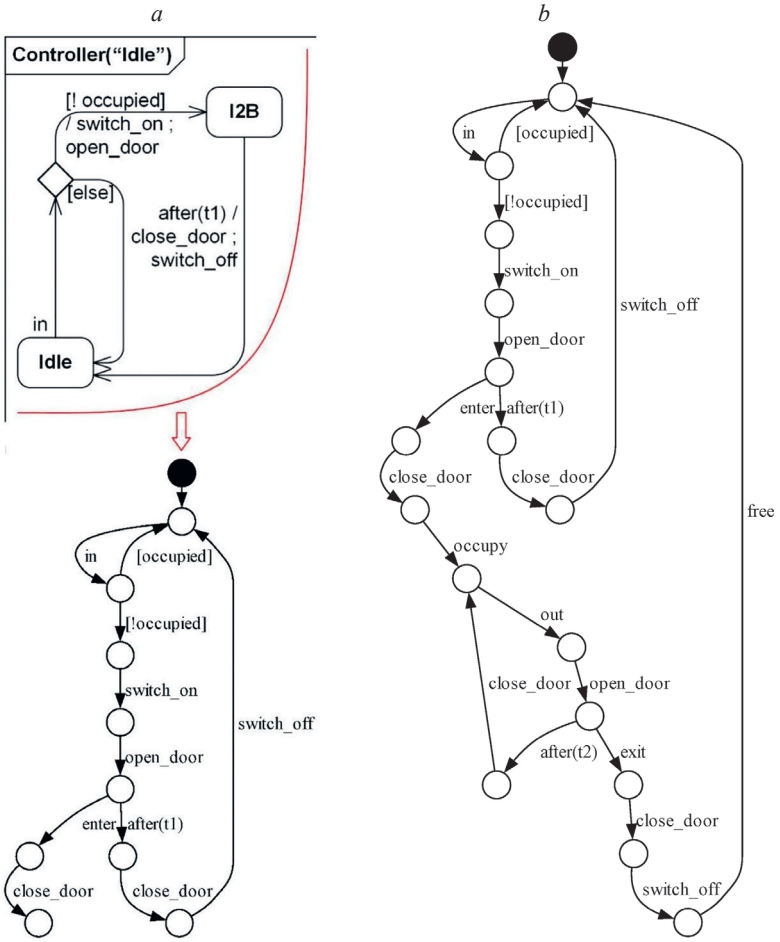


Fig. 2. Fragment of the transformation of the program state graph into a semantic graph (*a*) and a complete semantic graph constructed automatically (*b*)

event, effect, or guard condition other than those explicitly used in the given CRE.

The syntax rules of the CRE language are written in the same conventions as those used in the article [19]. The terminal symbols in the rules are underlined. Metasymbols and nonterminals are highlighted in bold. CRE syntax diagrams are presented in the Table.

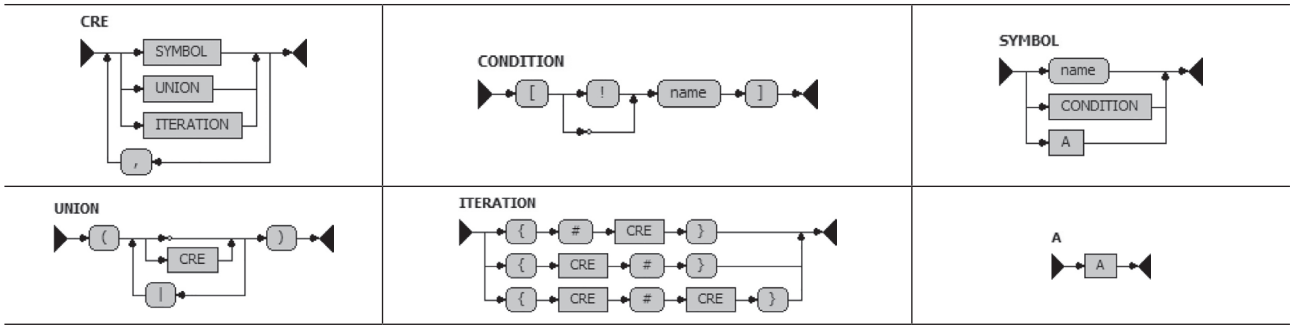
CRE : (SYMBOL | UNION | ITERATION) # ,.
UNION : (**CRE #** |).
ITERATION : { # **CRE** } | { **CRE** # } | { **CRE** # **CRE** } .
SYMBOL : name | **CONDITION** | **A** .
CONDITION : [! name | name] .
A : *A* .

For example, CRE
{#*A*},switch_on,{# *A* },open_door,{# *A* },close_door,{# *A* },switch_off,{# *A* }
describes the set of all sequences in which actions **switch_on**, **open_door**, **close_door**, and **switch_off** occur exactly once and strictly in the specified order.

Verification of the Automaton Objects Behavior

Verification, i.e., checking that the behavior of an automaton program corresponds to formal requirements, is performed automatically. However, the formalization of requirements, i.e., composing conditioned regular

Table. CRE syntax diagram



expressions describing acceptable sequences of actions and statements, is not performed automatically. This is a creative, trial-and-error method.

Here, we provide an example illustrating the capabilities of the proposed verification method. Let the elevator control system satisfy the natural informal requirement given below.

Informal requirement. When a passenger enters the elevator, i. e., he/she is in the elevator cabin, and exits the elevator, the light *must* be switched on. At all other times the light *may* be switched off.

This requirement must be *formalized*, i.e., the permissible sequences of actions performed to turn on and off the light, open, and close the doors must be written down as a regular expression. As an example, we provide a possible scenario for formalizing the requirement using the "successive approximation" method.

The first approximation is as follows: to enter the elevator and exit the elevator, the doors must be open. This is how the first version of formalization is obtained.

Formal requirement (option 1). The actions of turning on/off the light and opening/closing the doors must be performed once in the established order.

This requirement is input to the verification program as expressed in the sequence

$$\{\#A\}, \text{switch\_on}, \{\#A\}, \text{open\_door}, \{\#A\}, \text{close\_door}, \{\#A\}, \text{switch\_off}, \{\#A\}$$

Result: False, i.e., the requirement is not met (Fig. 3, *a*).

The door-opening action **open_door** is encountered after the door-closing action **close_door**. The transition along the arc **open_door** and all subsequent transitions are considered erroneous and are marked red in the semantic graph; the transition along the arc **switch_off** also does not satisfy the following requirement: the arc closes the cycle, and in this requirement all actions are performed once outside the cycle.

Second approximation: it is necessary to consider the cyclic nature of the process.

Formal requirement (option 2). The actions of turning on/off the light and opening/closing the doors must be performed cyclically in the established order:

$$\{\#(\{\#A\}, \text{switch\_on}, \{\#A\}, \text{open\_door}, \{\#A\}, \text{close\_door}, \{\#A\}, \text{switch\_off}, \{\#A\})\}$$

Result: False (Fig. 3, *b*). The door-opening action **open_door** is encountered after the door-closing action **close_door**; the transition along the arc **open_door** and all subsequent transitions are considered erroneous and are marked red in the semantic graph.

In the third attempt to formalize **formal requirement (option 1)**, the shortcomings of the first two were eliminated.

Formal requirement (option 3). The actions of turning on/off the light and opening/closing the doors must

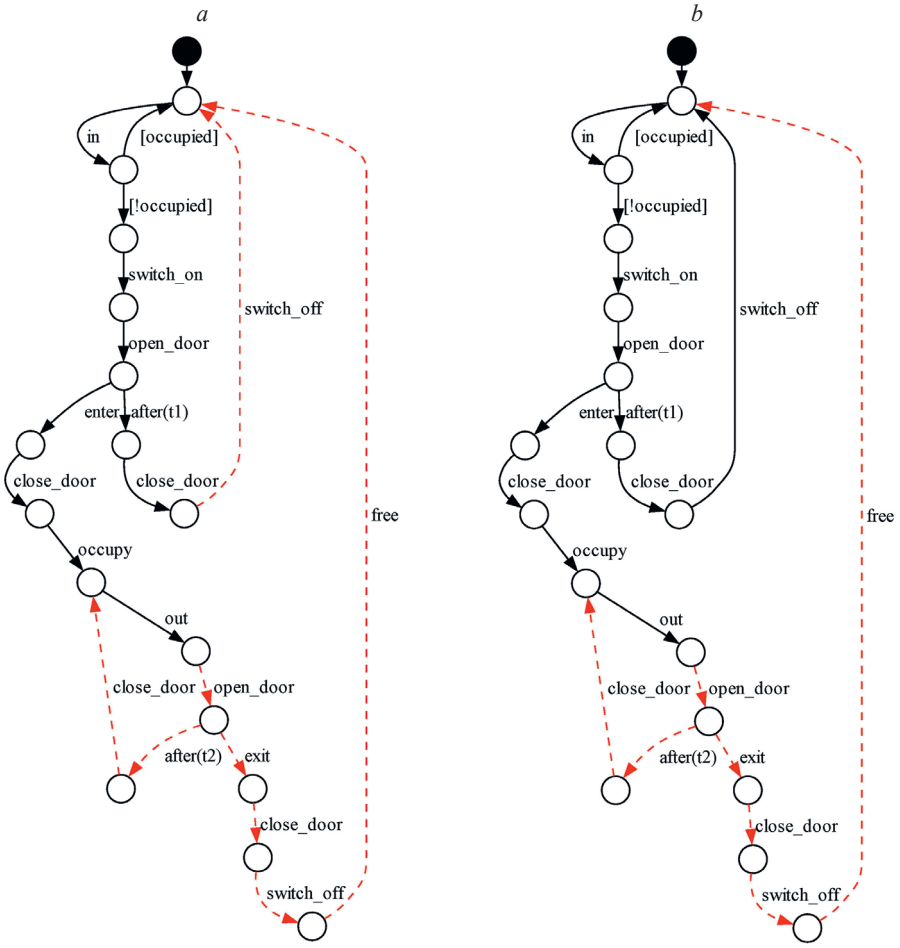


Fig. 3. Verification result of requirement: option 1 (*a*) and option 2 (*b*)

be performed cyclically in the established order, while between the actions of turning on/off the light there may be an arbitrary number of actions of opening/closing the doors (but not less than one).

{#({#A },switch_on,{#A },{({#A },open_door,{#A },
close_door,{#A })#},{#A },switch_off,{#A })}

Result: True.

In other words, the fulfillment of the formal requirement is verified: all sequences defined by the semantic graph satisfy the regular expression.

How adequate is the formalization? This is a legitimate and a difficult question. Informal inductive reasoning shows that if a passenger was outside the elevator cabin at the initial moment, he/she would enter the elevator, stay in the cabin, and exit the elevator only when the light is on, and the immutable requirement "the light must be on" is fulfilled.

A passenger may act in bad faith: call the elevator but do not enter it and leave for another business. In this case, the light is turned on for some time (namely, **t1**) in vain. After the timeout, the light is turned off; therefore, the optional requirement "the light may be turned off" is fulfilled.

Discussion and prospects for further research

The proposed automatic verification method was tested on several examples and demonstrated good applicability. In future research, we plan to expand the scope of the proposed method and the CIAO v.3 language in at least the following five directions.

1. The concept of a connection scheme between interacting automaton objects allows for both statically and *dynamic* interactions between automaton objects. This capability is especially important for reactive systems whose behavior changes according to environmental conditions.
2. Automaton objects allow for unlimited structural *decomposition* while maintaining object-oriented encapsulation. The concept of private interfaces allows us to detail both the connection scheme and the formal specification of behavior at nested levels. This capability is necessary for verifying complex reactive systems which often consist of many relatively independent subsystems.
3. The provided queries can be generalized so that they deliver results not only of Boolean types, but also of any other enumerated types. In terms of conventional programming languages, this means using not only branching by condition (the "if" operator) but also branching by value (the "switch" operator). This feature is very useful in various discrete control systems. It is so useful that it carries the main semantic load of the book [21].
4. The capabilities of the link diagram can be expanded so that the provided command interface of one automaton object can be linked to the required command interfaces of several other automaton objects at once. This feature allows us to exploit the competitive advantages of the client-server architecture within the framework of automaton programming.
5. It is also possible to link the required command interface of a single automaton object with the provided command interfaces of other automaton objects. This implies the concurrent launch of several interacting processes. It is difficult to overestimate this feature if the execution of the responsive system is assumed to be on multiprocessor hardware.

These priority improvements listed above do not exhaust the potential capabilities of the proposed model. The provided interfaces can be linked with the required interfaces of the same state machine, and the automaton object sends events to itself. Note that the capabilities of the model are not limited. The proposed behavior model is essentially asynchronous, parallel, and nondeterministic. It is quite easy to obtain a conventional sequential deterministic computation scheme: simply pull the automaton objects into a chain, linking the required interfaces of the previous object with the provided interfaces of the next object. However, if sequential execution is not required, the proposed model enables the construction of natural parallel schemes that are effective on modern platforms.

References

1. Kulyamin V.V. *Software Verification Methods*. Moscow, ISP RAS, 2008, 111 p. (in Russian)
2. Shalyto A.A. Validation of state machine specifications. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2023, vol. 23, no. 2, pp. 436–438. (in Russian). https://doi.org/10.17586/2226-1494-2023-23-2-436-438
3. Lavrov S.S. *Programming. Mathematical Foundations, Tools, Theory*. St. Petersburg, BHV Petersburg Publ., 2001, 320 p. (in Russian)
4. Dijkstra E.W. *A Discipline of Programming*. Prentice Hall, 1976, 217 p.
5. Karpov Iu.G. *Model Checking. Verification of Parallel and Distributed Software Systems*. St. Petersburg, BHV-Petersburg, 2010, 560 p. (in Russian)
6. Vinogradov R.A., Kuz'min E.V., Sokolov V.A. Verification of automata programs using CPN/Tools. *Modelirovanie i analiz informacionnyh sistem*, 2006, vol. 13, no. 2, pp. 4–15. (in Russian)
7. Shalyto A. Automata-based programming paradigm. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2008, vol. 53, pp. 3–23. (in Russian)

Литература

1. Кулямин В.В. Методы верификации программного обеспечения. М.: ИСП РАН, 2008. 111 с.
2. Шалыто А.А. Валидация автоматных спецификаций // Научно-технический вестник информационных технологий, механики и оптики. 2023. Т. 23. № 2. С. 436–438. https://doi.org/10.17586/2226-1494-2023-23-2-436-438
3. Лавров С.С. Программирование. Математические основы, средства, теория. СПб.: БХВ-Петербург, 2001. 320 с.
4. Dijkstra E.W. A Discipline of Programming. Prentice Hall, 1976. 217 p.
5. Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных систем. СПб.: БХВ-Петербург, 2010. 560 с.
6. Виноградов Р.А., Кузьмин Е.В., Соколов В.А. Верификация автоматных программ средствами CPN/Tools // Моделирование и анализ информационных систем. 2006. Т. 13. № 2. С. 4–15.
7. Шалыто А.А. Парадигма автоматного программирования // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2008. № 53. С. 3–23.

8. Polikarpova N.I., Shalyto A.A. *Automata-Based Programming*. St. Petersburg, Piter Publ., 2011, 176 p. (in Russian)
9. Novikov F.A., Tikhonova U.N. An Automata Based Method for Domain Specific Languages Definition. Part 1. *Information and Control Systems*, 2009, no. 6 (43), pp. 34–40. (in Russian)
10. Novikov F.A., Tikhonova U.N. An Automata Based Method for Domain Specific Languages Definition. Part 2. *Information and Control Systems*, 2010, no. 2 (45), pp. 31–37. (in Russian)
11. Novikov F.A., Tikhonova U.N. An Automata Based Method for Domain Specific Languages Definition. Part 3. *Information and Control Systems*, 2010, no. 3 (46), pp. 29–37. (in Russian)
12. Novikov F., Fedorchenko L., Vorobiev V., Fatkieva R., Levonevskiy D. Attribute-based approach of defining the secure behavior of automata objects. *Proc. of the 10th International Conference on Security of Information and Networks (SIN'17)*, 2017, pp. 67–72. https://doi.org/10.1145/3136825.3136887
13. Novikov F.A., Afanasyeva I.V. Cooperative interaction of automata objects. *Information and control systems*, 2016, no. 6 (85), pp. 50–64. (in Russian). https://doi.org/10.15217/issn1684-8853.2016.6.50
14. Afanasieva I.V. Data acquisition and control system for high-performance large-area CCD Systems. *Astrophysical Bulletin*, 2015, vol. 70, no. 2, pp. 232–237. https://doi.org/10.1134/S1990341315020108
15. Levonevskiy D., Novikov F., Fedorchenko L., Afanasieva I. Verification of internet protocol properties using cooperating automaton objects. *Proc. of the 12th International Conference on Security of Information and Networks (SIN'19)*, 2019, pp. 1–4. https://doi.org/10.1145/3357613.3357639
16. Afanasyeva I.V., Novikov F.A., Fedorchenko L.N. Methodology for constructing event-driven software systems using the CIAO specification language. *SPIIRAS Proceedings*, 2020, vol. 19, no. 3, pp. 481–514. (in Russian). https://doi.org/10.15622/sp.2020.19.3.1
17. Afanasieva I.V., Novikov F.A., Fedorchenko L.N. Verification of event-driven software systems using the specification language of cooperating automata objects. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2023, vol. 23, no. 4, pp. 750–756. https://doi.org/10.17586/2226-1494-2023-23-4-750-756
18. Novikov F.A., Afanasyeva I.V., Fedorchenko L.N., Kharisova T.A. Application of conditional regular expressions in problems of verification of control automata programs. *14th All-Russian conference on control problems. Collection of Scientific Papers*. Moscow, IPU RAS, 2024, pp. 2651–2655. (in Russian)
19. Novikov F.A., Afanasieva I.V., Fedorchenko L.N., Kharisova T.A. Specification language for automatabased objects cooperation. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2024, vol. 24, no. 6, pp. 1035–1043. https://doi.org/10.17586/2226-1494-2024-24-6-1035-1043
20. Novikov F.A., Afanasyeva I.V., Fedorchenko L.N., Kharisova T.A. Application of Conditional Regular Expressions in Verification Problems of Control Automata Programs. *Proc. of Materials of the All-Russian Scientific Conference "PhysMech Science Week"*, St. Petersburg, POLYTECH-PRESS, 2024, pp. 167–170. (in Russian)
21. Novikov F.A., Ivanov D.Iu. *UML Modeling. Theory, Practice, Video Course*. St. Petersburg, Professional'naja literature Publ., 2010, 640 p. (in Russian)
22. Shalyto A.A. *Switch-Technology. Algorithmization and Programming of the Logical Control Problems*. St. Petersburg, Nauka Publ., 1998, 617 p. (in Russian)
23. Velder S.E., Shalyto A.A. Methods of verification of models of automata-based programs. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2008, vol. 53, pp. 123–137. (in Russian)
24. Velder S.E., Lukin M.A., Shalyto A.A., Iaminov B.R. *Verification of Automata-Based Programs*. St. Petersburg, Nauka Publ., 2011, 244 p. (in Russian)
25. Kuzmin E.V., Sokolov V.A. Modeling, specification, and verification of automaton programs. *Programming and Computer Software*, 2008, vol. 34, no. 1, pp. 27–43. https://doi.org/10.1134/s0361768808010040
26. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994, 416 p.
27. Gerasimov A.Y. Directed dynamic symbolic execution for static analysis warnings confirmation. *Programming and Computer Software*, 2018, vol. 44, no. 5, pp. 316–323. https://doi.org/10.1134/S036176881805002X

8. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. СПб.: Питер, 2010. 176 с.
9. Новиков Ф.А., Тихонова У.Н. Автоматный метод определения проблемно-ориентированных языков. Ч. 1 // Информационно-управляющие системы. 2009. № 6 (43). С. 34–40.
10. Новиков Ф.А., Тихонова У.Н. Автоматный метод определения проблемно-ориентированных языков. Ч. 2 // Информационно-управляющие системы. 2010. № 2 (45). С. 31–37.
11. Новиков Ф.А., Тихонова У.Н. Автоматный метод определения проблемно-ориентированных языков. Ч. 3 // Информационно-управляющие системы. 2010. № 3(46). С. 29–37.
12. Novikov F., Fedorchenko L., Vorobiev V., Fatkieva R., Levonevskiy D. Attribute-based approach of defining the secure behavior of automata objects // Proc. of the 10th International Conference on Security of Information and Networks (SIN'17). 2017. P. 67–72. https://doi.org/10.1145/3136825.3136887
13. Новиков Ф.А., Афанасьева И.В. Кооперативное взаимодействие автоматных объектов // Информационно-управляющие системы. 2016. № 6 (85). С. 50–64. https://doi.org/10.15217/issn1684-8853.2016.6.50
14. Afanasieva I.V. Data acquisition and control system for high-performance large-area CCD Systems // Astrophysical Bulletin. 2015. V. 70. N 2. P. 232–237. https://doi.org/10.1134/S1990341315020108
15. Levonevskiy D., Novikov F., Fedorchenko L., Afanasieva I. Verification of internet protocol properties using cooperating automaton objects // Proc. of the 12th International Conference on Security of Information and Networks (SIN'19). 2019. P. 1–4. https://doi.org/10.1145/3357613.3357639
16. Афанасьева И.В., Новиков Ф.А., Федорченко Л.Н. Методика построения событийно-управляемых программных систем с использованием языка спецификации CIAO // Труды СПИИРАН. 2020. Т. 19. № 3. С. 481–514. https://doi.org/10.15622/sp.2020.19.3.1
17. Afanasieva I.V., Novikov F.A., Fedorchenko L.N. Verification of event-driven software systems using the specification language of cooperating automata objects // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2023. V. 23. N 4. P. 750–756. https://doi.org/10.17586/2226-1494-2023-23-4-750-756
18. Новиков Ф.А., Афанасьева И.В., Федорченко Л.Н., Харисова Т.А. Применение условных регулярных выражений в задачах верификации управляющих автоматных программ. Сборник научных трудов XIV Всероссийского совещания по проблемам управления. М.: ИПУ РАН, 2024. С. 2651–2655.
19. Novikov F.A., Afanasieva I.V., Fedorchenko L.N., Kharisova T.A. Specification language for automatabased objects cooperation // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2024. V. 24. N 6. P. 1035–1043. https://doi.org/10.17586/2226-1494-2024-24-6-1035-1043
20. Новиков Ф.А., Афанасьева И.В., Федорченко Л.Н., Харисова Т.А. Применение условных регулярных выражений в задачах верификации управляющих автоматных программ. Сборник материалов Всероссийской научной конференции "Неделя науки ФизМех". СПб.: ПОЛИТЕХ-ПРЕСС, 2024. С. 167–170.
21. Новиков Ф.А., Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. СПб.: Профессиональная литература, 2010. 640 с.
22. Шалыто А.А. Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 617 с.
23. Вельдер С.Э., Шалыто А.А. Методы верификации моделей автоматных программ // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2008. № 53. С. 123–137.
24. Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р. Верификация автоматных программ. СПб.: Наука, 2011. 244 с.
25. Кузьмин Е.В., Соколов В.А. Моделирование, спецификация и верификация «автоматных» программ // Программирование. 2008. Т. 34. № 1. С. 38–60.
26. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Паттерны объектно-ориентированного проектирования. СПб.: Питер, 2020. 448 с.
27. Герасимов А.Ю. Направленное динамическое символьное исполнение программ для подтверждения ошибок в программах // Программирование. 2018. № 5. С. 31–42. https://doi.org/10.31857/S013234740001213-9
28. Friedl J.E.F. Mastering Regular Expressions: Understand Your Data and Be More. O'Reilly Media, Inc., 2006. 544 p.
29. Fedorchenko L., Baranov S. Equivalent transformations and regularization in context-free grammars // Cybernetics and

28. Friedl J.E.F. *Mastering Regular Expressions: Understand Your Data and Be More*. O'Reilly Media, Inc., 2006, 544 p.
29. Fedorchenko L., Baranov S. Equivalent transformations and regularization in context-free grammars. *Cybernetics and Information Technologies*, 2015, vol. 14, no. 4, pp. 29–44. https://doi.org/10.1515/cait-2014-0003

Authors

Fedor A. Novikov — D.Sc., Senior Researcher, Professor, Peter the Great St. Petersburg Polytechnic University (SPbPU), Saint Petersburg, 195251, Russian Federation; sc 16441904500, https://orcid.org/0000-0003-4450-0173, fedornovikov51@gmail.com

Irina V. Afanasieva — PhD, Head of Laboratory, Special Astrophysical Observatory of the Russian Academy of Sciences (SAO RAS), Nizhny Arkhyz, 369167, Russian Federation, sc 57210431774, https://orcid.org/0000-0003-4225-4124, riv615@gmail.com

Ludmila N. Fedorchenko — PhD, Senior Researcher, St. Petersburg Federal Research Center of the Russian Academy of Sciences, Saint Petersburg, 199178, Russian Federation, sc 36561350100, https://orcid.org/0000-0002-4008-9316, lnf@iias.spb.su

Taisia A. Kharisova — Engineer, Ioffe Institute, Saint Petersburg, 194021, Russian Federation, https://orcid.org/0009-0002-3456-0471, tais.harisova@mail.ru

Received 25.12.2024
Approved after reviewing 27.02.2025
Accepted 30.03.2025

Information Technologies. 2015. V. 14. N 4. P. 29–44. https://doi.org/10.1515/cait-2014-0003

Авторы

Новиков Федор Александрович — доктор технических наук, старший научный сотрудник, профессор, Санкт-Петербургский политехнический университет Петра Великого, Санкт-Петербург, 195251, Российская Федерация, sc 16441904500, https://orcid.org/0000-0003-4450-0173, fedornovikov51@gmail.com

Афанасьева Ирина Викторовна — кандидат технических наук, заведующий лабораторией, Специальная астрофизическая обсерватория Российской академии наук, Нижний Архыз, 369167, Российская Федерация, sc 57210431774, https://orcid.org/0000-0003-4225-4124, riv615@gmail.com

Федорченко Людмила Николаевна — кандидат технических наук, старший научный сотрудник, Санкт-Петербургский Федеральный исследовательский центр Российской академии наук, Санкт-Петербург, 199178, Российская Федерация, sc 36561350100, https://orcid.org/0000-0002-4008-9316, lnf@iias.spb.su

Харисова Таисия Анваровна — инженер, Физико-технический институт им. А.Ф. Иоффе Российской академии наук, Санкт-Петербург, 194021, Российская Федерация, https://orcid.org/0009-0002-3456-0471, tais.harisova@mail.ru

Статья поступила в редакцию 25.12.2024
Одобрена после рецензирования 27.02.2025
Принята к печати 30.03.2025