Correctness of Control Systems with Concurrency Behavior

Liudmila Cheremisinova
United Institute of Informatics Problems
of NAS of Belarus
Minsk, Belarus
cld@newman.bas-net.by

Dmitry Cheremisinov
United Institute of Informatics Problems
of NAS of Belarus
Minsk, Belarus
cher@newman.bas-net.by

Abstract. **The discussed problem is to verify whether a reactive control system design with concurrency behavior meets its specification. A model of the desired behavior is in the form of parallel automaton that describes concurrent control algorithms. It is proposed to generate test patterns in the process of simulating the design specification of a concurrent system, which includes an algorithm for the behavior of not only the system itself, but also the environment of the designed device.**

Keywords: **concurrent algorithm, hardware verification, test pattern, simulation, PRALU language**

I. INTRODUCTION

The paper deals with the problem of verification of digital systems with parallelism of behavior. In the functional verification phase, it is established whether the designed device implements the desired behavior, i.e. whether it works according to the requirements set in its specification. The interest in the problem is motivated by the fact that with the growth of the complexity of the designed control systems, the labor costs for their testing also grow. The testing and debugging phase accounts for up to 60 - 80% or more of the total cost of developing control systems.

The most common approach to verification is simulation testing, which requires a test system, which is a specialized software environment that solves three main tasks: generating a test sequence; verifying the correct behavior of the component under test and evaluating the completeness of testing relative to the original specification. A test is a sequence of sets of signal values applied to the input of a device under test and a set of expected signal values generated by it. The purpose of the verification test is to identify errors as a result of situations where the expected results do not coincide with the results of the device under test when the corresponding test sequence is submitted [1].

The quality of testing directly depends on the test sequences used. The methods for constructing test sequences traditionally used in testing practice are based on manual, random and directed test generation. Although such tests allow detecting a significant number of errors in the design, they do not give an estimate of the completeness of the coverage of the operation area of the device under test. In this sense, more effective is the verification of control systems based on model checking [2, 3]. In the case, a test sequence is generated based on a model describing the desired system behavior, which is specified by a device design specification in a certain language. Tests are built on the basis of the specification of the designed system in an algorithmic way; the responses of the device under test are compared with the expected values derived from the specification.

If the model is correct and the device under test must implement the specified behavior (and only it), then the successful passing of tests, generated appropriately based on this model, can serve as a sufficient guarantee of the system correct implementation. The description of the specification is assumed to be correct. The internal structure of an implementation under test can be viewed as a black box. The assessment of the completeness of testing is determined by the degree of coverage of the scenarios of the device operation, specified by the specification.

The verification problem is considered for the case of reactive systems [4]. The peculiarity of these systems (as opposed to systems of transformational type) lies in the continuous (and, in the general case, infinite) exchange of signals with the external environment to accomplish the task. The most popular model of reactive systems is the state machine [5, 6], which describes sequential behavior and is widely used to describe protocols

However, there are a number of systems in which the expressive means of finite state machines are insufficient. The most important property of such systems is the inherent parallelism of the processes occurring in them. The problem of model-based

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

verification for devices with concurrent behavior has not yet been sufficiently studied. One of the most studied types of models of such devices is a system consisting of simultaneously operating components, which is modeled as a network of finite state machines or labeled transition systems (LTS) [3]. Approaches to testing labeled transition systems were proposed [7], in which this problem was considered as checking the system for input-output conformance - ioco relation. The device under test meets the specification in respect to ioco relation if, after any sequence of inputs allowed by the specification, the observed responses of the device under test meet the values expected in the specification.

There are also known approaches to generating test sequences for systems using models of "true parallelism", where some actions are performed in parallel and in the same component, and when it is necessary to control the order of execution. These approaches are based on Petri nets. Test cases for such systems are generated on the basis of the Petri net reachability graph [8], according to which test cases are generated by traversing it [9–11], similarly to how it is done for the case of finite state machines [12]. Reachability graph-based verification is one of the most studied approaches to verifying systems with behavior parallelism. The disadvantage of this approach is the exponential growth of the size of the space of possible states of the system and, accordingly, the size of the reachability graph. As a result, the reachability graph faces the problem of an explosion in the number of states, which negatively affects the performance of testing complex systems.

The paper considers the problem of constructing a test system for verifying the circuit (or software) implementation of a control device with parallelism of behavior. As an example of such systems, control systems in industry, where it is necessary to take into account the parallelism present in control objects, can be mentioned. Within the framework of this system, the specification of the designed device is set in the PRALU language for describing parallel control algorithms [4]. The same language describes the behavior of an object controlled by a designed device. The control object is considered as a part of the test environment. The device implementation is viewed as a black box for which only inputs and outputs are available. Test sequences are formed on the basis of the described algorithms for the behavior of the device and the control object dynamically - in the process of simulating the control algorithm.

II. LANGUAGE TO SET SPECIFICATIONS FOR DESIGNING DEVICES WITH PARALLELISM OF BEHAVIOR

Concurrency in a specification arises for a variety of reasons. For example, it can be a multi-block system in which some actions are performed in parallel, but in different components. And finally, systems with "real parallelism", when some actions are performed in parallel and in the same component,

The problem of designing control devices is one of the most important in the automation of production processes in various industries. When solving the problem of implementing control devices, one has to deal with the parallelism present in control objects. The aim of such object control is to ensure the interacting components to work in parallel and asynchronously in a coordinated manner. The parallelism present in control objects is reflected in the functional model of digital devices that control these objects. It is also inherent in digital devices of this class that control actions and signals about the state of control objects are described by Boolean variables, and only a small percentage of all information is numerical. At present, networks of interacting finite state machines and languages based on Petri net are used as the language for setting the specification for the design of control devices.

To set the specification for the design of devices with parallelism of behavior, it is proposed to use parallel logic control algorithms, which are widely used in the design and testing of digital systems. One of such languages is the PRALU language [4] for describing logic control algorithms. Algorithms in the PRALU language are represented in the form of causal dependencies between events occurring in a technical system, the behavior of which is described in terms of binary variables: control actions and signals about the state of the control object are Boolean variables.

The main operations of the PRALU language waiting are acting operations. The waiting operation "$- k^{in}$" boils down to waiting for the moment in time when the conjunction k^{in} takes the value 1. The acting operation "$\rightarrow k^{out}$" is performed by assigning the variables that form the conjunction k^{out} to values that turn it into 1 In one of the interpretations of the acting operations in the language, it is assumed that all internal variables (if any in the description) and output (or control) variables retain their values until any of the acting operations changes them. The waiting and acting operations can be interpreted as polling the states of the sensors of the control object and issuing commands to the executive and signal equipment.

A control algorithm on PRALU is represented by an unordered set of sentences, each of which opens with a label and consists of one or several equally

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

labeled linear chains of language operations, ending with transition labels: "μ_i: $l_i \to \nu_i$", where l_i denotes some linear algorithm consisting of language operations; μ_i and ν_i are initial and final labels, which are non-empty subsets of elements from the set M = ={1, 2,…, m}, which can be interpreted as partial states (in the sense that they can exist simultaneously).

The order of execution of the chains in the process of a control algorithm implementation is determined by the start set N [4], its current values are $N_t \subset M$. Among the algorithm proposals, one is distinguished as the initial; its label is entered into the set N before the implementation of the algorithm.

In the process of a control algorithm implementation, the chains are started independently of each other. If at some moment in time for some chain "μ_i: $l_i \to \nu_i$" the condition $\mu_i \subseteq N_t$ is satisfied and the event k_i^{in} is realized, with the expectation of which the chain l_i begins, then it is started. In this case, N_t is replaced by $(N_t \setminus \mu_i) \cup \nu_i$, and after the end of the chain, the new state of N_t becomes equal to $(N_t \setminus \mu_i) \cup \nu_i$. The syntactically parallel algorithm is characterized by the presence of labels $|\mu_i| > 1$, $|\nu_i| > 1$. An alternative branching is provided by the constraint

$$(i \neq j) \wedge (\mu_i \cap \mu_j \neq \varnothing) \to (k_i^{in} \wedge k_j^{in} = 0).$$

As an example, we will give a parallel algorithm describing the cycle of the manipulator, which consists in moving it between the extreme positions recorded by the sensors r and l. Movements to the left and to the right are initiated by signals L and R, respectively. In the initial position, the manipulator is in position r and starts the working cycle after pressing s button. The manipulator is controlled by the buttons of the control panel "enable" - s and "disable" - e. These buttons can be pressed during the working cycle in any sequence, the manipulator reacts to them only in the position r: it continues the working cycle if the s button was pressed last, or stops if e.

Description of the control algorithm in the PRALU language, defined on the sets {s, e, r, l} and {L, R} of input (or conditional) and output (control) variables:

RUNNING_CYCLE(s, e, r, l / R, L)

1: $\to$ 2.3

2: $- g \to L - l \to \bar{L} \to R - r \to \bar{R} \to 2$

3: $- s \to g - e \to \bar{g} \to 3$

The control algorithm is cyclical: once started, it can function indefinitely. Input variables of the algorithm are the variables s, e, r, l, these variables fix the state of the environment. Output variables L, R initiate movement to the left and to the right. In addition, there is one more internal variable g, introduced to remember the fact of pressing the s or e button during the operating cycle of the manipulator: g = 1, if s button was pressed last, and g = 0, if e.

III. SIMULATION OF REACTIVE SYSTEMS WITH PARALLELISM OF BEHAVIOR

In [4], it was proposed to characterize digital devices by the type of algorithmic description. Devices, the model of which are classical algorithms (scheduling algorithms), belong to the type of transformation. Their purpose is to compute some result from the original data through a finite sequence of steps. Examples of such systems are processors, programming language compilers, web servers.

The purpose of a reactive system [4] is to interact with the environment. The behavior of a reactive system is set by a control algorithm. The functioning of reactive systems, ideally, never ends. It follows that the algorithm of a reactive system is not an algorithm in the sense of the classical theory of algorithms (there is no sign of a finite number of steps). In current literature, control algorithms are called communication protocols. Nevertheless, to formalize these algorithms, one can use the same approach as for transformation systems - description by specifying a formal language and an abstract computational mechanism. Examples of such devices are controllers of computer peripheral devices connected to a common bus, embedded systems and equipment control devices. Recently, the term "reactive system" began to be used to designate software systems in which data streams are processed asynchronously, the volume of which is not predetermined [13].

Traditionally, a protocol has been modeled as a set of interacting processes, where each process is described as an extended state machine that has a finite number of states. In modern verification systems, the interaction of processes is represented as communication, in which the acts of communication are transactions through common data structures called channels. The transaction-level model (TLM) is a performance-enhancing tool (up to 1000 times faster than RTL). The most popular and widely used language for simulating TLM level is SystemC (IEEE 1666 standard), which is an extension of the C ++ language. An executable program that results from compiling a SystemC model with any ANSI-compliant C ++ compiler implements a simulator with integrated simulation controls. Concurrency in SystemC has the semantics of interleaving the operations of sequential processes. SystemC concurrent processes are threads that are scheduled for sequential execution by the native scheduler SystemC based on cooperative multitasking cooperative multitasking.

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

The multithreading model has significant non-determinism, and SystemC processes are specially organized to eliminate this non-determinism. Atomic operations of processes, which are transactions, are linearizable. Linearizability is a property of a program in which the result of any parallel execution of operations is equivalent to some of their sequential execution [14]. For any other thread, the execution of the linearizable operation is instantaneous: the operation has either not started, or it has completed.

Synchronization of TLM processes at the transaction level is carried out by a barrier mechanism [15]. The barrier is the points in the source code where each process must pause and wait for all processes in the group to reach the barrier. In the SystemC TLM model, barrier points are set by calls to the wait (.) function.

IV. SIMULATION OF DEESCRIPTIONS OF REACTIVE SYSTEMS IN PRALU LANGUAGE

Algorithms in the PRALU language can be interpreted by the TLM model; this requires a refinement of the semantics of waiting and acting operations. The essence of the refinement is to extend the definition of the partial order of the implementation of operations, given by the original parallel algorithm, to a linear order. An interleaving parallelism model is used, in which concurrency is understood as the ability to order operations in an arbitrary way. Waiting and acting operations are considered as compositions of some elementary operations. In this interpretation, algorithms on PRALU have the property of linearizability, i.e. the result of parallel execution of the algorithm is equivalent to some sequential execution of atomic operations. Transactions in algorithms on PRALU are represented by waiting and acting operations that have a common variable and describe the interaction event [16].

The data structure in the TLM model of the PRALU algorithm is a vector of variable values, the components of which are pairs representing the current and planned values for each variable. Access to the vector components of variables is carried out through the operation of setting the planned value of the algorithm variable and the operation of checking the value of the conditional variable. The implementation of a waiting operation for an algorithm on the PRALU consists of the sequential execution of the operations of algorithm suspending and checking the values of the variables in the vector of current values, the acting operation consists of performing the operations of setting the planned values of the variables.

When describing the scheduling procedures for computations associated with the linear ordering of partially ordered operations, the concept of a branch is traditionally used as a set of sequential subprocesses starting with a given operation. A sequential subprocess is usually called the maximum chain of operations of a process that are in a direct sequence. A branch is a dynamic object generated by the operation of its formation and destroyed by the operation of its termination.

Synchronization of concurrent chains of the PRALU algorithm is carried out using a barrier mechanism. Barrier points are set by the operation to suspend execution of branches. The data structure of the synchronization barrier is represented in the memory by the queue QR of branches ready for execution and the queue QW of waiting branches. The operation of forming a branch consists in entering its first operation into the QR. The meaning of the branch termination operation is clear from its name: the branch is removed from the QR.

The fundamental point in simulating algorithms on PRALU is the agreement on the duration of the execution of the operations of the language; in particular, it concerns the acting operations. This convention significantly affects the degree of conformity of signal changes produced by the emulator and appearing at the outputs of the circuit implementation. The implementation (as well as simulation) of PRALU algorithms is performed under some assumption about the duration of the execution of the language operations. The most natural assumption is that all operations (and, in particular, acting operations) have the same duration.

One of the ways to increase the speed of computations is to assume that operations have zero duration. In this case, calculations of operations of one branch are performed until their continuation requires a change in the states of conditional variables. This means that branches will only be suspended while waiting operations are in progress. For a hardware implementation, it is more natural to assume the same, but not zero, duration of the execution of acting operations. In this case, branch execution is suspended after acting operations.

When simulating the control algorithm, branches are sequentially extracted from the QR and executed until suspended. The execution of branch G is suspended if^

1) if its initial fragment "$- k^{in} \rightarrow k^{out}$" cannot be executed on the set of current values of algorithm variables;

2) if its initial fragment "$- k^{in} \rightarrow k^{out}$" is already executed.

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

In the first case, the branch *G* is transferred to the queue QW. In the second case, a new branch is entered into the queue QW, starting with the operation that should be executed next in the branch *G*. The barrier is reached when the queue becomes empty. When the barrier is reached, the following processes are started:

1) transferring elements from the queue QW into the queue QR (QW becomes empty);

2) entering the next values of variables as planned values (if the system is not closed);

3) sending the planned variables values to the current ones for each component of the vector of variables. Then the first operation from the queue QR is started.

Reaching the barrier fixes the clock cycles of the emulator, and the changes in the values of the variables (marked in the vector of the planned values) correspond to the changes in the signal values at the outputs of the circuit implementation of the control system when the signal values corresponding to the values in the vector of the planned values are fed to its inputs. Thus, the verification process of a control algorithm can be performed in two ways:

1) dynamically in the process of debugging the control algorithm;

2) on the test sequence obtained after simulation of the control algorithm.

The transformation of the description of the algorithm in the PRALU language into the TLM model is carried out by translating it into expressions of the intermediate procedural language, which is carried out by substitutions of compositions from elementary operations instead of the operations of the PRALU language [16]. In the PRALU TLM model, there is no need to explicitly specify the barrier points (unlike the SystemC model). The barrier is formed automatically during the broadcast. Synchronizing processes takes longer than computing, especially in distributed computing. The operations of forming, terminating and suspending branches are related to the overhead of organizing computations.

Barrier synchronization is considered to be quite memory and runtime expensive mechanism. However, in the TLM model in the PRALU language, each of the operations of the barrier mechanism can be implemented in modern microprocessors with one command, including the suspension, which is analogous to the wait (.) function in SystemC. A separate scheduler is not required.

V. METHODOLOGY OF CONSTRUCTING A TEST SYSTEM BASED ON THE PRALU LANGUAGE

Let us demonstrate the process of constructing a test sequence in the process of simulation and debugging the specification for the design of a control device given in PRALU language. As an example, consider the above algorithm RUNNING_CYCLE(*s*, *e*, *r*, *l* / *R*, *L*) of manipulator operation.

In the PRALU language, one can describe the functioning of the system as a whole, including not only setting the control algorithm, but also describing the behavior of its environment (as an object of control). This makes it possible to simplify the simulation of the control algorithm, since in this case it is sufficient to change the values of only those variables, the change in the values of which is not fixed in these two algorithms (they are external variables for the system). In our case these are the variables *s* and *e*. The description of the behavior of the environment is as follows:

OC (*R*, *L* / *r*, *l*/)

$1: -L \to \bar{r} \to l - R \to \bar{l} \to r \to 1$

The above algorithms on PRALU describe the functioning of the manipulator system as a whole; changes are recorded not only in the internal variable *g*, but also in the values of other variables, except for the variables *s* and *e*. Before starting the manipulator, the variables have the following values: $s = e = 0$, $r = 1$, $l = g = R = L = 0$, $r = 1$, set by the vector 0010000.

Let us demonstrate the process of simulation of the described manipulator functioning for the case of a synchronous implementation of the control algorithm. For this case, 10 branches are allocated in the algorithms:

$1: -L \to \bar{r}\,|_4 \to l\,|_5 - R \to \bar{l}\,|_6 \to r \to 1$

$2: -g \to L\,|_7 - l \to \bar{L}\,|_8 \to R\,|_9 - r \to \bar{R} \to 2$

$3: -s \to g\,|_{10} - e \to \bar{g} \to 3$

Table I shows the cycles of simulation of the PRALU manipulator algorithms. Each of the subtables corresponds to one simulation cycle spawned by the reached barrier B_i. The columns of each of the subtables specify the numbers of the branches G_j, selected from the queue QR at each cycle; the numbers of the branches in the queues QR and QW; the vector of current values of the algorithm variables at the corresponding step C_i, and vectors P_i of the planned values of variables after considering the branches.

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

The first row of the subtable shows the states of QR, QW, C_i and P_i at the beginning of the corresponding cycle. When passing from the i-th cycle to the (i+1)-th we set the values to be: $QR_{i+1} = QW_i$, $C_{i+1} = C_{i+1} = P_i$. The vector T_i generates with its components corresponding to the conditional and internal variables the test action, and the vector C_i obtained after the execution of the cycle generates a reference response to this action, set by the components corresponding to the control variables.

TABLE I SIMULATION STEPS

Simulation steps	Branches G_j	Queues		Variable values	
		QR$_i$	QW$_i$	T_i	P_i
B_1		1,2,3	∅	**1**0 100 00	00 100 00
	1	2,3	1		00 100 00
	2	3	1,2		00 100 00
	3	∅	1,2,10		00 10**1** 00
B_2		1,2,10	∅	00 101 00	00 101 00
	1	2,10	1		00 101 00
	2	10	1,7		00 101 0**1**
	10	∅	1,7,10		00 101 0**1**
B_3		1,7,10		00 101 01	00 101 01
	1	7,	4		00 **0**01 01
	7	10	4,7		00 **0**01 01
	10	∅	4,7,10		00 **0**01 01
B_4		4,7,10	∅	00 001 01	00 001 01
	4	7,10	5		00 0**1**1 01
	7	10	5,7		00 0**1**1 01
	10	∅	5,7,10		00 0**1**1 01
B_5		5,7,10	∅	00 011 01	00 011 01
	5	7,10	5		00 011 01
	7	10	5,8		00 011 0**0**
	10	∅	5,8,10		00 011 0**0**
B_6		5,8,10	∅	00 011 00	00 011 00
	5	8,10	5		00 011 00
	8	10	5,9		00 011 **1**0
	10	∅	5,9,10		00 011 **1**0
B_7		5,9,10	∅	**1**0 011 10	00 011 10
	5	9,10	6		00 0**0**1 10
	9	10	6,9		00 0**0**1 10
	10	∅	6,9,3		00 0**00** 10
B_8		6,9,3	∅	0**1** 001 10	00 001 10
	6	9,10	1		00 **1**01 10
	9	10	1,9		00 **1**01 10
	3	∅	1,9,3		00 **1**00 10
B_9		1,9,3	∅	00 100 10	00 100 10
	1	9,3	1		00 100 10
	9	3	1,2		00 100 **0**0
	3		1,2,3		00 100 **0**0
B_{10}		1,2,3	∅	00 100 00	00 100 00

The required tests are represented by pairs of vectors: a five-component vector of the test pattern, the components of which correspond to the values of the variables *s*, *e*, *r*, *l*, *g*, and a three-component vector of reactions, the components of which correspond to the values of the variables *g*, *R*, *L*.

For the simulation fragment, shown in Table 1, the following test sequence is obtained for verifying the control device from the initial state:

10100 / 100; 00101 / 101; 00001 / 101; 00011 / 100;

00011 / 110; 10011 / 010; 01001 / 010; 00100 / 000.

Here the first part of each test sets a test pattern and the second part shows the expected responses of the device under test after feeding it by the test pattern.

VI. CONCLUSION

The behavior of an embedded control device is highly dependent on the object it controls and the environment in which it operates. Simulation of the control device for verification purposes should be carried out in the area of its planned operation. Using the PRALU language to describe control algorithms makes it possible to specify the behavior of the control system as a whole. Currently, there is software support for design automation and debugging of control systems based on PRALU, which includes simulation tools and synthesizers of the PRALU language in the hardware model in the Verilog and C languages [16].

REFERENCES

[1] A. Kamkin, M. Chupilko, “Obzor sovremennykh tekhnologiy imitatsionnoy verifikatsii apparatury” (Overview of modern technologies for simulation verification of equipment), Programmirovaniye, 2011, № 3, pp. 42–49.

[2] L. Hoffman, “Talking Model-Checking Technology”, Communications of the ACM, vol. 51, 2008, № 07/08, pp. 110–112.

[3] Tretmans J. Model based testing with labelled transition systems. Formal Methods and Testing: Lecture Notes in Computer Science (Springer), vol. 4949, 2008, pp. 1–38.

[4] A. D. Zakrevskij, “Parallel'nye algoritmy logicheskogo upravlenija” (Parallel Logic Control Algorithms), Minsk: Institut tehnicheskoj kibernetiki Nacional'noj akademii nauk Belarusi, 1999, 202 p. (in Russian).

[5] D. Lee, M. Yannakakis, “Principles and methods of testing finite state machine – a survey”, Proceedings of the IEEE, vol. 84, 1996, № 8, pp. 1090–1123.

[6] B. Kanso, O. Chebaro, “Compositional testing for FSM-based models”, International Journal of Software Engineering & Applications (IJSEA), vol. 5, 2014, № 3, pp. 9–23.

[7] H. Ponce de Leon, S.H. Delphine Longuet, “Model-based Testing for Concurrent Systems with Labeled Event Structures”, Software Testing, Verification & Reliability, vol. 24, 2014, № 7, pp. 558–590.

[8] H. Watanabe, T. Kudoh, “Test Suite Generation Methods for Concurrent Systems based on Coloured Petri Nets”, Proceedings of the 2nd Asia-Pacific Software Engineering Conference, 1995, pp. 242–251.

[9] U. Farooq, C. P. Lam, H. Li, “Towards Automated Test Sequence Generation”, Proceedings of the 19th Australian Conference on Software Engineering, 2008, pp. 441–450.

[10] H. Zhu, X. D. He, “A Methodology of Testing High-level Petri Nets”, Information and Software Technology, vol. 44, 2002, pp. 473–489.

[11] J. Liu, X. Ye1, J. Zhou, X. Song, “I/O Conformance Test Generation with Colored Petri Nets”, Applied Mathematics and Information Sciences, vol. 6, 2014, № 6, pp. 2695–2704.

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.

[12] I. B. Burdonov, A. S. Kosachev, V. V. Kuljamin, “Neizbytochnye algoritmy obhoda orientirovannyh grafov. Determinirovannyj sluchaj” (Irredandant algorithms for traversal of directed graphs, The determinate case), Programmirovanie, 2003, № 5, pp. 11–30 (in Russian).

[13] N. Halbwachs, “Synchronous Programming of Reactive Systems”, Springer-Verlag, 2010, 192 p.

[14] M. P. Herlihy, J. M. Wing, “Linearizability: A Correctness Condition for Concurrent Objects”, ACM Trans. Program. Language Systems, 1990, pp. 463–492.

[15] Y. Solihin, “Fundamentals of Parallel Multicore Architecture”, CRC Press, 2015, 494 p.

[16] D. I. Cheremisinov, “Proektirovanie i analiz parallelizma v processah i programmah” (Design and the Analysis of Parallelism in Processes and Programs), Minsk: Belaruskaja navuka Publ, 2011, 300 p. (in Russian).

Pattern Recognition and Information Processing (PRIP'2021) : Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. – Minsk : UIIP NASB, 2021. – 246 p. – ISBN 978-985-7198-07-8.