ISSN 0146-4116, Automatic Control and Computer Sciences, 2011, Vol. 45, No. 7, pp. 380–389. © Allerton Press, Inc., 2011.
Original Russian Text © I.B. Burdonov, A.S. Kosachev, 2010, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2010, No. 4, pp. 27–40.

Safe Simulation Testing of Systems with Refusals and Destructions

I. B. Burdonov and A. S. Kosachev
Institute for System Programming, Russian Academy of Sciences, Moscow, 109004 Russia
e-mail: igor@ispras.ru, kos@ispras.ru

Received October 13, 2010

Abstract—This paper deals with conformance testing based on formal specifications. The concept of safe testing was earlier proposed by the authors for trace based conformance. This concept is propagated for the case of (weak) simulation based on a relation between the specification and implementation states. The theory of the safe simulation of systems with refusals and destructions is proposed. The problems of complete testing and sufficient conditions for the existence of a complete test suite are discussed. A practical algorithm of complete testing for restricted classes of specifications and implementations is described.

Keywords: formal model, model based testing, complete testing, simulation, system with refusals and destructions

DOI: 10.3103/S0146411611070042

1. INTRODUCTION

Conformity testing is an experimental test of the conformity of the implementation of the requirements given in the form of specification. This correspondence is determined by the semantics of the test interaction, which describes the possible test actions and observations of the possible response behavior implementation.

If the response to test impacts can be observed only as (external) actions performed by the implementation or lack of action (failure), the conformity is determined by observing the tracks—sequences of actions and failures. The specification describes the tracks that are allowed in implementation.

Testing that has a guaranteed finite waiting time of observation after the test's impact is called safe. There are two possible reasons for an infinite wait: divergence and unobservable failures. Divergence is infinite execution by implementation of internal (unobservable) actions. Unobserved failure is a lack of external actions carried out by implementation, which the test can not determine in a finite time.[1] In both cases, a test cannot continue testing or finish it, as it is unknown whether it should wait for observations or waiting is unavailing.

We also introduce a special nonadjustable test by the stimuli effect of implementation, which is called *destruction*. It models any undesirable behavior of a system, including its actual destruction. Testing in which there is no divergence, unobserved failures, and destruction is called safe.

A specification describes situations when the test impact should be safe to implement. Accordingly, conformity is based only on safe behavior of the implementation. The general theory of such conformity is developed by the authors of [1–3].

The literature also deals with conformities based on the correlation of the implementation and specification states (for a review, see [5]); such conformities are called simulations. Simulation requires correctness not only of the observed external behavior of the implementation but also correctness of its state changes. All the simulations discussed in the literature either do not take safety into account, while assuming there is no divergence and unobserved failures, or are designed for direct observation of the divergence and all failures. Also, they do not consider the possibility of destruction.

[1] For example, if the time interval between a rest input and a response is limited by a time-out condition, then reaching this time out when waiting for external actions means an observance of a refusal. When there are no such conditions, the refusal is unobservable.

The aim of this paper is to extend the general approach, including failures and destruction, to simulations. The specification defines not only the class of its conformal implementations but also a wider class of implementations that can be safely tested for conformance checking (the safety hypothesis).

The choice of the simulation is the most natural for conformity when the implementation states are available for observation. Testing, in which at any given time the current state of the implementation can be obtained, is called testing with an open state. The task of testing is the detection of errors in the implementation, which is understood as a mismatch between its behavior and the specification requirements. The testing is called complete if it allows one to detect any actual error while getting no "false" errors. This article considers complete testing with an open state of safe simulation.

This consideration is conducted both in general theoretical and practical terms. A theoretical full testing should detect any error in finite time, but in the absence of errors it can continue indefinitely. Endless testing causes are the infiniteness of the implementation and/or specifications, as well as unlimited nondeterminism of the implementation behavior. Under certain restrictions, it is possible to construct a complete test completing its work within a finite time. Such tests may be used in practice.

Sections 2–4 of the manuscript contain the basic propositions of the theory of conformity: the semantics of interaction and safe testing, the mathematical model of implementation and specification, the definition of simulation, and the hypothesis of safety and identification of safe simulation. The 5th section discusses the connection between safe simulation and trace conformity. The 6th section examines the completeness of the testing. The 7th chapter considers theoretical and the 8th section practical tests. Limitations on the implementation and specification enabling a full test in a finite time and an algorithm for such testing are defined. Section 9 provides an example of the verification of the simulation.

2. SEMANTICS OF SAFE COMMUNICATION

The semantics of interaction are formalized in terms of *external* actions and *buttons*. An action is a behavior of an implementation observed in response to external stimuli. A set of actions is called the action alphabet and is denoted by $\mathbf{L}$. The button is a subset $P \subseteq L$; pressing P simulates the impact of implementation, which reduces to the permission to perform any action from P. We can observe either action $a \in P$ or (for some keys) the absence of such action, which called the failure R. The semantics of the interaction are given by an alphabet $\mathbf{L}$ and two sets of buttons: one corresponding to the observation of failure, a family $\mathbf{R} \subseteq 2^{\mathbf{L}}$, and without such observations, a family $\mathbf{Q} \subseteq 2^{\mathbf{L}}$. It is assumed that $\mathbf{R} \cap \mathbf{Q} = \varnothing$ and $(\cup\mathbf{R}) \cup (\cup\mathbf{Q}) = \mathbf{L}$.

In the general case, it is unknown whether after pressing the $Q \in \mathbf{Q}$ button we should wait for observation $a \in Q$ or there will be no observation because of an unobserved failure Q. In the case of the proper interaction, such a button is pressed only if there is no failure in the implementation.

In addition to the external actions, an implementation can make internal (unobservable) actions denoted by τ. Such actions are always allowed. It is assumed that any finite sequence of any actions takes place within a finite time and an infinite sequence within an infinite time. An infinite sequence of τ actions ("looping") is called *divergence* and is denoted by Δ. Divergence is not dangerous by itself, but when implementation tries to exit it (by pressing any button) it is unknown whether we should wait for an observation or τ actions will take place infinitely. Therefore, at the correct interaction, buttons are pressed only if there is no divergence in the implementation.

We also introduce a special action not adjustable by buttons called *destruction* and denoted by γ. It simulates any undesirable behavior of a system, including its actual destruction. The semantics of destruction assumes that correct interaction should avoid it.

A correct interaction, in which there is no unobserved failures, attempts to exit from the divergence and destruction and is called safe.

3. LTS MODEL

The LTS (Labeled Transition System) model is used as a model of implementation and specification. It is defined as $\mathbf{S} = LTS(V_\mathrm{S}, \mathbf{L}, E_\mathrm{S}, s_0)$, where V_S is a nonempty set of states, $\mathbf{L}$ is an alphabet of external actions, $E_\mathrm{S} \subseteq V_\mathrm{S} \times (\mathbf{L} \cup \{\tau, \gamma\}) \times V_\mathrm{S}$ is a set of transitions, and $s_0 \in V_\mathrm{S}$ is the initial state. A transition from state s to state s' by the action of z is denoted as $s \xrightarrow{z} s'$. A trace is a chain of adjacent transitions: the first transition begins in the initial state, and every other starts at the end of the previous transition.

A state is *divergent* if it is an initial state for an infinite τ route. A condition is *stable* if there does not emerge τ and γ transitions. Fault $P \in \mathbf{R} \cup \mathbf{Q}$ is generated by a stable state from which there are no transitions for the actions of P.

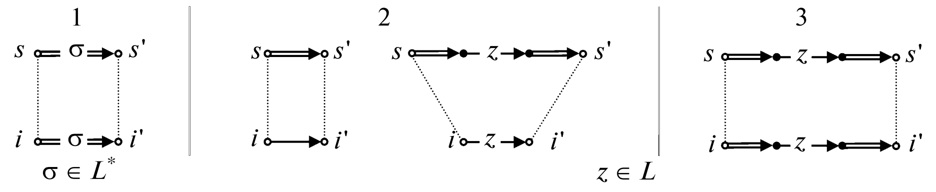

Fig. 1. Three definitions of weak simulation.

In order to determine the LTS, traces **S** (with failures from $\mathbf{R} \cup \mathbf{Q}$) to each of its stable states are added virtual loops $s \xrightarrow{P} s$ generated by marked failures and Δ loops in divergent states $s \xrightarrow{\Delta} s$. We then consider the routes that are not continued after the γ and Δ transitions; a trace is a sequence of markings on passages of the route with the omission of τ characters. Let us denote for $s, s' \in V_{\mathrm{S}}$, $u \in \mathbf{L} \cup \mathbf{R} \cup \mathbf{Q} \cup \{\gamma, \Delta\}$, $\sigma = \langle u_1, \ldots, u_n\rangle \in (\mathbf{L} \cup \mathbf{R} \cup \mathbf{Q} \cup \{\gamma, \Delta\})^*$:

$$s \Rightarrow s' \triangleq s = s' \vee \exists s_1, \ldots, s_n\ s = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} s_n = s',$$
$$s \overset{\langle u\rangle}{\Longrightarrow} s' \triangleq \exists s_1, s_2\ s \Rightarrow s_1 \xrightarrow{u} s_2 \Rightarrow s',$$
$$s \overset{\sigma}{\Rightarrow} s' \triangleq \exists s_1, \ldots, s_{n+1}\ s = s_1 \overset{\langle u_2\rangle}{\Longrightarrow} s_2 \ldots s_n \overset{\langle u_n\rangle}{\Longrightarrow} s_{n+1} = s',$$
$$s \overset{\sigma}{\Rightarrow} \triangleq \exists s'\ s \overset{\sigma}{\Rightarrow} s',$$
$$s = \sigma \not\Rightarrow \triangleq \nexists s'\ s \overset{\sigma}{\Rightarrow} s',$$
$$s\ \boldsymbol{after}\ \sigma \triangleq \{s' | s \overset{\sigma}{\Rightarrow} s'\}.$$

4. WEAK SIMULATION

Simulation requires that each observation u possible in an implementation state i with a poststate i' is possible in each corresponding specification state s and that the specifications for s and u contain the poststate s' corresponding to i'. Simulations differ from each other mostly by the relation to the observability of the internal actions (τ). In this article, we proceed from the basic assumption about the fundamental unobservability of τ actions: at the interaction, it is impossible to distinguish the presence and absence of τ actions, both before and after the external action. This corresponds to weak simulation, which is also known as observation simulation. Let us give three equivalent definitions of weak simulation (the first two belong to Milner [6, 7]):

$$\mathbf{I} \le^1_{ws} \mathbf{S} \triangleq \exists R \subseteq V_{\mathrm{I}} \times V_{\mathrm{S}} (i_0, s_0) \in R\ \&\ \forall (i, s) \in R \forall \sigma \in \mathbf{L}^* \forall i'$$
$$(i \overset{\sigma}{\Rightarrow}\ \Rightarrow \exists s \overset{\sigma}{\Rightarrow} s'\ \&\ (i', s') \in R)$$

(Fig. 1, left).

$$\mathbf{I} \le^2_{ws} \mathbf{S} \triangleq \exists R \subseteq V_{\mathrm{I}} \times V_{\mathrm{S}} (i_0, s_0) \in R\ \&\ \forall (i, s) \in R \forall u \in \mathbf{L} \forall i'$$
$$(i \xrightarrow{\tau} i' \Rightarrow \exists s' s \Rightarrow s'\ \&\ (i', s') \in R)$$
$$\&\ (i \xrightarrow{u} i' \Rightarrow \exists s' s \overset{u}{\Rightarrow} s'\ \&\ (i', s') \in R)$$

(Fig. 1, center).

$$\mathbf{I} \le^3_{ws} \mathbf{S} \triangleq \exists R \subseteq V_{\mathrm{I}} \times V_{\mathrm{S}} (i_0, s_0) \in R\ \&\ \forall (i, s) \in R \forall u \in \mathbf{L} \forall i'$$
$$(i \overset{\langle u\rangle}{\Longrightarrow} i' \Rightarrow \exists s' s \overset{\langle u\rangle}{\Longrightarrow} s'\ \&\ (i', s') \in R)$$

(Fig. 1, right).

The correspondence R, for which the conditions of weak simulation are satisfied, is called a *conformal correspondence*.

Failures. If by observation not only the external actions of **L** but also the observable failures of **R** are meant, the definition of weak simulation should be modified (changes as compared to $\le_{ws}^3$ are underlined):

$$\mathbf{I} \le_{ws}^4 \mathbf{S} \triangleq \exists R \subseteq V_\mathbf{I} \times V_\mathbf{S}(i_0, s_0) \in R \;\&\; \forall(i, s) \in R \forall u \in \mathbf{L} \underline{\cup \mathbf{R}} \forall i'$$
$$(i \xRightarrow{\langle u \rangle} i' \Rightarrow \exists s' s \xRightarrow{\langle u \rangle} s' \;\&\; (i', s') \in R)$$

In the class of implementations without observable failures, these correspondences match: $\le_{ws}^3 = \le_{ws}^4$.

Safety. The state s is called *safe* if from this state there does not emerge a γ track: $s = \gamma \not\Rightarrow$. In safe interactions, only safe states of implementation are passed. The button $P \in \mathbf{R} \cup \mathbf{Q}$ is called *safe in a safe state s* if it can be pressed at a safe interaction:

$$P \textit{\textbf{ safe }} s \triangleq s = \langle \gamma \rangle \not\Rightarrow \;\&\; s = \langle \Delta \rangle \not\Rightarrow$$
$$\& \; (P \in \mathbf{Q} \Rightarrow s = \langle P \rangle \not\Rightarrow) \;\&\; \forall z \in P \; s = \langle z, \gamma \rangle \not\Rightarrow.$$

The observation is called *safe* if it is allowed by a safe button. The condition is called *safely* achievable if it is accessible from the initial state by a sequence of pressings of safe buttons. The modification of a weak simulation with failures and safety is as follows (changes as compared to $\le_{ws}^4$ are underlined):

$$\mathbf{I} \le_{ws}^5 \mathbf{S} \triangleq \exists R \subseteq V_\mathbf{I} \times V_\mathbf{S}(s = \langle \gamma \rangle \not\Rightarrow \; \Rightarrow (i_0, s_0) \in R)$$
$$\& \; \forall(i, s) \in R \; \forall \underline{P \textit{\textbf{ safe }} i} \forall u \in P \cup \{P\} \forall i'$$
$$(\underline{P \textit{\textbf{ safe }} s \;\&}\; i \xRightarrow{\langle u \rangle} i' \Rightarrow \exists s' s \xRightarrow{\langle u \rangle} s' \;\&\; (i', s') \in R).$$

In the class of implementations and specifications in which all failures are observable, there is no divergence and destruction; these correspondences are the same: $\le_{ws}^4 = \le_{ws}^5$.

Safety hypothesis. Since a specification is given, it is possible to check the condition P ***safe*** s. The condition P ***safe*** i can be checked if the implementation is also known. Otherwise (in testing), we can judge the safety of the buttons in the implementation states only on the basis of some *safety hypothesis*. This hypothesis is based on the correspondence $H \subseteq V_\mathbf{I} \times V_\mathbf{S}$ of the states of the implementation and specification, and it is called the *H hypothesis*. It assumes (1) the safety of the initial state i_0 of the implementation if the initial state s_0 of the specification is safe, and (2) the safety of a button in the implementation state if it is safe at least in one corresponding by H specification state.

Let us define the correspondence H recursively. If the initial states are safe, then they, as well as any two states that are reachable from the initial states by a blank trace, correspond to each other. The states i' and s' correspond to each other if they are reachable from states i and s, corresponding, to each other by an observation u allowed by button P, which is safe in both states i and s. The correspondence H is a minimal correspondence generated by the following inference rules:

$$\forall i, i' \in V_\mathbf{I} \forall s, s' \in V_\mathbf{S} \forall P \in \mathbf{R} \cup \mathbf{Q} \forall u \in P \cup \{P\}$$
$$s_0 = \langle \gamma \rangle \not\Rightarrow \;\&\; i_0 = \langle \gamma \rangle \not\Rightarrow \;\&\; i_0 \Rightarrow i \;\&\; s_0 \Rightarrow s \vdash (i, s) \in H,$$
$$(i, s) \in H \;\&\; P \textit{\textbf{ safe }} i \;\&\; P \textit{\textbf{ safe }} s$$
$$\& \; i \xRightarrow{\langle u \rangle} i' \;\&\; s \xRightarrow{\langle u \rangle} s' \vdash (i', s') \in H.$$

Button P is called *H safe* in an implementation state i if it is safe in at least one corresponding to i specification state s:

$$P \; H\text{-}\textit{\textbf{safe}} \; i \triangleq \exists s (i, s) \in H \;\&\; P \textit{\textbf{ safe }} s.$$

We now give a formal definition of the H hypothesis:

$$\mathbf{I} \; H\text{-}\textit{\textbf{safe}} \; \mathbf{S} \triangleq (s_0 = \langle \gamma \rangle \not\Rightarrow \;\&\; i_0 = \langle \gamma \rangle \not\Rightarrow$$
$$\& \; \forall i \in V_\mathbf{I} \forall P \in \mathbf{R} \cup \mathbf{Q} (P \; H\text{-}\textit{\textbf{safe}} \; i \Rightarrow P \textit{\textbf{ safe }} i).$$

Secure simulation. By merging weak simulation and the H hypothesis on safety, we get *safe simulation*, which is denoted by *ss* (changes as compared to $\le_{ws}^5$ are underlined):

$$\begin{aligned}&\mathbf{I}\ \boldsymbol{ss}\ \mathbf{S} \triangleq \mathbf{I}\ \underline{H\text{-}\boldsymbol{safe}\ \mathbf{S}}\ \&\ \exists R \subseteq V_{\mathbf{I}} \times V_{\mathbf{S}}(\underline{s = \langle\gamma\rangle \not\Rightarrow\ \Rightarrow} (i_0, s_0) \in R)\\ &\quad \&\ \forall (i, s) \in R \forall P\ \boldsymbol{safe}\ i\ \forall u \in \mathbf{L} \cup \mathbf{R} \forall i'\\ &\quad (P\ \boldsymbol{safe}\ s\ \&\ i \overset{\langle u\rangle}{\Longrightarrow} i' \Rightarrow \exists s' s \overset{\langle u\rangle}{\Longrightarrow} s'\ \&\ (i', s') \in R).\end{aligned}$$

The relation *ss* is also transitive in the class of specifications that satisfy their own H hypothesis and reflexively, and thus it is a preorder.

If the implementation is given explicitly, we can analytically check both the H hypothesis and safe simulation. When the implementation is unknown, testing is required, and the H hypothesis becomes the precondition of the test's safety. If $s_0 \overset{\langle\gamma\rangle}{\Longrightarrow}$, then $H = \varnothing$ and safe testing is not possible, but is also not necessary, since any implementation is conformal (for any R). If $s_0 = \langle\gamma\rangle \not\Rightarrow$, the testing is essentially a verification of the *test condition* (the two bottom lines in the *ss* definition). Every button P H ***safe*** i is pressed, and the received observation u and poststate i are checked for the specification compliance: the observation u should be in every state of the specification s, which correspond to the state i by R, where P is a safe button, and among the poststates s' at least one must correspond to i' by R.

For the class of specifications without unobservable faults, divergence, and destruction, we have H ***safe*** $\cap$ $\leq^5_{ws} =$ ***ss***, and, in the safe implementations subdomain, $\leq^5_{ws} = ss$.

For a conformal by *ss* correspondence R, the correspondence $R \cap H$ is also conformal. We can reformulate the definition of safe simulation as follows:

$$\begin{aligned}&\mathbf{I}\ \boldsymbol{ss}\ \mathbf{S} \triangleq \mathbf{I}\ H\text{-}\boldsymbol{safe}\ \mathbf{S}\ \&\ \exists R \subseteq H(s = \underline{\langle\gamma\rangle \Rightarrow \not\Rightarrow (i_0}, s_0) \in R)\\ &\quad \&\ \forall (i, s) \in R \forall P\ \boldsymbol{safe}\ s\ \forall u \in P \cup \{P\} \forall i'\\ &\quad (i \overset{\langle u\rangle}{\Longrightarrow} i' \Rightarrow \exists s' s \overset{\langle u\rangle}{\Longrightarrow} s'\ \&\ (i', s') \in R).\end{aligned}$$

We can restrict to such correspondences R that are included in H. The union of conformal by ***ss*** correspondences is conformal, which gives two natural conformal correspondences: R_1, the union of all conformal correspondences, and $R_2 = R_1 \cap H$.

5. RELATION OF SIMULATION AND TRACE CONFORMITY

In the trace theory of conformity, the safety hypothesis is based on traces of implementation and specification [1–3] and does not require state correspondence. We reproduce here the basic definitions of this theory. For *LTS*, the set $\mathbf{S}$ of its traces with faults of $\mathbf{R} \cup \mathbf{Q}$ is denoted by $\mathbf{T}(\mathbf{S})$. For implementation $\mathbf{I}$, a relation ***safe*** of the button $P \in \mathbf{R} \cap \mathbf{Q}$ after the trace $\sigma \in \mathbf{T}(\mathbf{I})$ is determined:

$$\begin{aligned}&P\ \boldsymbol{safe\ in}\ \mathbf{I}\ \boldsymbol{after}\ \sigma \triangleq (P \in \mathbf{R} \vee \sigma\bullet\langle P\rangle \notin \mathbf{T}(\mathbf{I})\\ &\quad \&\ \forall z \in P\ \sigma\bullet\langle z, \gamma\rangle \notin \mathbf{T}(\mathbf{I})\ \&\ \sigma\bullet\langle\Delta\rangle \notin \mathbf{T}(\mathbf{I}).\end{aligned}$$

Obviously, if the button is safe by ***safe in*** the trace (P ***safe in*** $\mathbf{I}$ ***after*** σ), then it is safe in each state after this trace: $\forall i \in$ ($\mathbf{I}$ ***after*** σ) P ***safe*** i.

For an implementation, the relation ***safe by*** of the buttons safety after traces is ambiguous: it is any relation satisfying three requirements: $\forall \sigma \in \mathbf{T}(\mathbf{S}) \forall R \in \mathbf{R}$; $\forall z \in \mathbf{L}$; $\forall Q \in \mathbf{Q}$

(1) P ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Leftrightarrow \forall u \in R\ \sigma\bullet\langle u, \gamma\rangle \notin \mathbf{T}(\mathbf{S})\ \&\ \sigma\bullet\langle\Delta\rangle \notin \mathbf{T}(\mathbf{S})$;

(2) $\sigma\bullet\langle z\rangle \in \mathbf{T}(\mathbf{S})\ \&\ T \in \mathbf{R} \cup \mathbf{Q}\ z \in T\ \&\ \forall u \in T \sigma\bullet\langle u, \gamma\rangle \notin \mathbf{T}(\mathbf{S})$
$\&\ \sigma\bullet\langle\Delta\rangle \notin \mathbf{T}(\mathbf{S}) \Rightarrow \exists P \in \mathbf{R} \cup \mathbf{Q}\ z \in P\ \&$ P ***safe by*** $\mathbf{S}$ ***after*** σ;

(3) Q ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Leftrightarrow \exists v \in Q\ \sigma\bullet\langle v\rangle \in \mathbf{T}(\mathbf{S})$
$\&\ \forall u \in Q\ \sigma\bullet\langle u, \gamma\rangle \notin \mathbf{T}(\mathbf{S})\ \&\ \sigma\bullet\langle\Delta\rangle \notin \mathbf{T}(\mathbf{S})$.

We assume that, along with a specification, the relation ***safe by*** is given, and it complies with these three requirements.

The trace σ of a specification $\mathbf{S}$ is called safe if the specification does not contain a trace $\langle\gamma\rangle$, and the route σ does not end with divergence and destruction, and each symbol u occurring in it is an external action or an $\mathbf{R}$ refusal allowed by a button that is safe after immediately preceding this character trace prefix. Let us denote the set of safe paths:

$$\boldsymbol{SafeBy}(\mathbf{S}) \triangleq \{\sigma \in \mathbf{T}(\mathbf{S}) | \langle\gamma\rangle \notin \mathbf{T}(\mathbf{S}) \;\&\; \forall u \in \mathbf{L} \cup \mathbf{R}$$
$$(\sigma = \mu \bullet \langle u \rangle \bullet \lambda \Rightarrow \exists P \in \mathbf{R} \cup \mathbf{Q}\; P\; \boldsymbol{safe\ by}\; \mathbf{S}\; \boldsymbol{after}\; \mu \;\&\; u \in P \cup \{P\})\}.$$

For the buttons $R \in \mathbf{R}$, the relations ***safe by*** and ***safe in*** are the same. Therefore, if a button $R \in \mathbf{R}$ is safe by ***safe by*** after some trace σ (R ***safe by*** **S** ***after*** σ), then it is safe in every state after this route: $\forall s \in$ (**S** ***after*** σ) R ***safe*** s. However, a button $Q \in \mathbf{Q}$ that is safe by ***safe*** after the trace can be unsafe in some (but not all) states $s \in$ (**S** ***after*** σ).

The trace hypothesis of safety is defined as follows:

$$\mathbf{I}\; \boldsymbol{safe\ for}\; \mathbf{S} \triangleq (\langle\gamma\rangle \notin \mathbf{T}(\mathbf{S}) \Rightarrow \langle\gamma\rangle \notin \mathbf{T}(\mathbf{I})$$
$$\&\; \forall \sigma \in \boldsymbol{SafeBy}(\mathbf{S}) \cap \mathbf{T}(\mathbf{I}) \forall P \in \mathbf{R} \cup \mathbf{Q}$$
$$(P\; \boldsymbol{safe\ by}\; \mathbf{S}\; \boldsymbol{after}\; \sigma \Rightarrow P\; \boldsymbol{safe\ in}\; \mathbf{I}\; \boldsymbol{after}\; \sigma).$$

The trace conformity is defined as

$$\mathbf{I}\; \boldsymbol{saco}\; \mathbf{S} \triangleq \mathbf{I}\; \boldsymbol{safe\ for}\; \mathbf{S} \;\&\; \forall \sigma \in \boldsymbol{SafeBy}(\mathbf{S}) \cap \mathbf{T}(\mathbf{I})$$
$$\forall P\; \boldsymbol{safe\ by}\; \mathbf{S}\; \boldsymbol{after}\; \sigma\; \forall i \in (\mathbf{I}\; \boldsymbol{after}\; \sigma) \forall u \in P \cup \{P\}$$
$$(i \overset{\langle u \rangle}{\Longrightarrow} \Rightarrow \exists s \in (\mathbf{S}\; \boldsymbol{after}\; \sigma) s \overset{\langle u \rangle}{\Longrightarrow}).$$

Stimulation can be considered with a trace hypothesis of safety, which is defined as follows (the changes as compared to ***ss*** are underlined):

$$\mathbf{I}\; \boldsymbol{sst}\; \mathbf{S} \triangleq \mathbf{I}(\underline{\boldsymbol{safe\ for}\ }\mathbf{S} \;\&\; \exists R \subseteq V_{\mathbf{I}} \times V_{\mathbf{S}}(s_0 = \langle\gamma\rangle \not\Rightarrow \Rightarrow (i_0, s_0) \in R)$$
$$\&\; \forall (i, s) \in R\; \underline{\forall \sigma \in \boldsymbol{SafeBy}(\mathbf{S}) \cap \mathbf{T}(\mathbf{I}) \forall P\; \boldsymbol{safe\ by}\; \mathbf{S}\; \boldsymbol{after}\; \sigma}$$
$$\forall u \in P \cup \{P\} \forall i'$$
$$(P\; \boldsymbol{safe}\; s \;\&\; i \in (\mathbf{I}\; \boldsymbol{after}\; \sigma) \;\&\; i \overset{\langle u \rangle}{\Longrightarrow} i' \Rightarrow \exists s' s \overset{\langle u \rangle}{\Longrightarrow} s' \;\&\; (i', s') \in R).$$

Simulation with an H hypothesis of safety imposes stronger requirements on the implementation than the simulation with the hypothesis of trace safety. The simulation with the hypothesis of trace safety, in turn, imposes stronger requirements on the implementation than the trace conformity with the same hypothesis about safety.

As a result, we have the following relations of trace conformity and safe simulations with different hypotheses about safety: ***ss*** $\subset$ ***sst*** $\subset$ ***saco***. Strengthening the requirements for the implementation starts as an imposition of additional requirements for compliance of states of implementation and the specification while maintaining the same trace safety hypothesis ***sst*** $\subset$ ***saco***. Since the safety hypothesis is the same, the number of safe button pressings and, thus, the number of tests is the same. The only difference is the amount of analytical checkups after the tests. This is followed by the strengthening the hypothesis of safety with ***ss*** $\subset$ ***sst***, which leads to an increase in the safe button pressings and, thus, to a greater number of checkups.

6. TEST COMPLETENESS

For testing, it is considered that the system under investigation is unknown, but it is *assumed* that it has a model (of a given class, in the article it is LTS), which, when being interacted with, behaves just like the real system. In the literature, this is called a *test hypothesis* [4]. Therefore, the real system satisfies the requirements formalizable in the specification if and only if its model (implementation) is conformal with the specification.

For simulation, the operation of an implementation state request is used: in the beginning of the test and after each observation (open state testing).

A test is an instruction where each item describes either a required restart of the system[2] or a test action (button) and, depending on the received observations and poststate, the next item or a verdict (***pass*** or ***fail***). The implementation *passes* the test if the testing always (for any manifestation of nondeterminism) does not end with the verdict ***fail***. A test is *significant* if each of its conformal implementations passes,

[2] A test including a restart is equivalent to a set of tests, which is usually considered.

exhaustive if each of its nonconforming implementations does not pass, and complete if it is significant and exhaustive.

The aim is to generate *complete test* specifications, which clearly would determine conformity or nonconformity of any implementation (which may be of a certain class).

In order to be complete, the simulation test ***ss*** is required for every state reachable in the safe testing of an implementation state i and every H-safe button P to verify each implementation observation $u \in P \cup \{P\}$ and poststate i'. It is assumed that any pair (u, i') can be reached in a finite number of pressings of P from the state i. It is also assumed that any state $i'_0 \in I_0 = (i_0\ \textbf{\textit{after}}\ \langle\rangle)$ can be reached in a finite number of restarts. This is called the *global testing* hypothesis.

7. THEORETICAL TESTING

If $s_0 \overset{\langle\gamma\rangle}{\Longrightarrow}$, then all the implementations are conformal and testing is not required. If $s_0 = \langle\gamma\rangle \not\Rightarrow$, we define a minimal set N of *nonconformal state* pairs (i, s) generated by the following inference rules: $\forall(i, s) \in H\ \forall P\ H\textbf{\textit{-safe}}\ i\ \forall u \in P \cup \{P\}$:

1. $i \overset{\langle u\rangle}{\Longrightarrow} \ \&\ P\ \textbf{\textit{safe}}\ s\ \&\ s = \langle u\rangle \not\Rightarrow\ |\!-\ (i, s) \in N$;

2. $i \overset{\langle u\rangle}{\Longrightarrow} i'\ \&\ P\ \textbf{\textit{safe}}\ s\ \&\ \{i'\} \times (s\ \textbf{\textit{after}}\ \langle u\rangle) \subseteq N\ |\!-\ (i, s) \in N$.

The conformity $\mathbf{I}\ \textbf{\textit{ss}}\ \mathbf{S}$ is equivalent to the condition $(i_0, s_0) \notin N$. Inference rules define the output graph, whose points are the pairs $(i, s) \in N$ obtained using the first (1-point) or second (2-point) rule of inference. The labeled arc $(i, s) \xrightarrow{(u,\, i')} (i', s')$ corresponds to inference rule 2 for $s' \in (s\ \textbf{\textit{after}}\ \langle u\rangle)$.

In the output graph, there is a tree of routes (perhaps not unique), which we call an output tree. Each route of the tree begins in (i_0, s_0), the root of the tree is an empty route, and the leaves are routes ending in 1-points. The route of the tree ending in a 2-point in the tree is continued by those and only those arcs that are labeled by the same label (u, i'), which corresponds to the second rule of inference.

If $(i_0, s_0) \in N$ but all the output trees are infinite, then no test can determine nonconformity in a finite time. Therefore, upon the class of all H-safe implementations, there may exist only meaningful tests. One such test is complete for the subclass of implementations in which there is either no inference tree (the implementation is conformal) or a finite inference tree (the implementation is nonconformal). In addition to the global test, the enumerability of a set $S_0 = (s_0\ \textbf{\textit{after}}\ \langle\rangle)$ is required; the set of buttons $P(s) = \{P \in \mathbf{R} \cup \mathbf{Q}\ |\ P\ \textbf{\textit{safe}}\ s\}$, which is safely achievable in each state s; and the set of poststates $S(s, u) = (s\ \textbf{\textit{after}}\ \langle u\rangle)$ for each safe in s observation u.

An inference tree is finite if and only if it has finite branching, which is equivalent to the finiteness of each $s\ \textbf{\textit{after}}\ \langle u\rangle$. It is sufficient that, in every safely achievable state s, the specifications of the following are finite:

(1) the number of transitions for each safe action, and

(2) the set $s\ \textbf{\textit{after}}\ \langle\rangle$.

8. PRACTICAL TESTING

In practice, it is necessary that the test comes to an end after a finite time. A full test detects an error in a finite time, but, in the absence of errors, can run indefinitely if we do not imply special restrictions. The following restrictions are enough. (1) The number of buttons is *finite* and each button is *resolvable* with respect to the alphabet of actions. (2) The specification is *finite*: it has a finite number of states and transitions. (3) "Part" $\mathbf{I}'$ of the implementation $\mathbf{I}$, which is "passed" in a safe test, is *finite* and t-nondetermined: all the possible pairs (observation, poststate) can be obtained not only in a finite number of button pressings in this state, as in the global testing, but in not more than t pressings. At $t = 1$, the implementation is determined.

The finiteness conditions allow one to check the implementation $\mathbf{I}$ first, that is, to build $\mathbf{I}'$ and then to conduct the verification. The transition $i \xrightarrow{u} i'$ is added to $\mathbf{I}'$ when, after a survey of the state i, the P button is pressed; the observation u is obtained; and the poststate, which is now the state i', is queried again. This button P will be called the *control* button and denoted by $P(i \xrightarrow{u} i')$. Note that we add not only the transitions by external actions but also the virtual transitions from the observable failures. If P is an $\mathbf{R}$ button and $u = P$, then the added transition $i \xrightarrow{P} i'$ means that the implementation includes a virtual loop

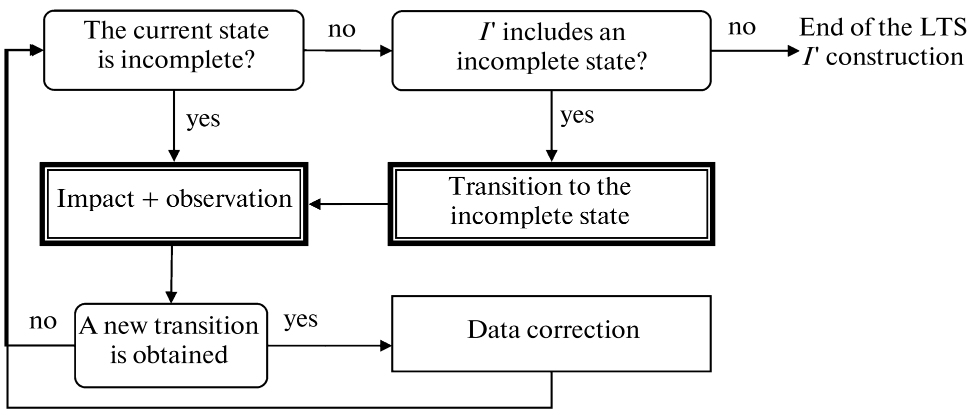
The current state is incomplete?
no
I' includes an incomplete state?
no
End of the LTS I' construction
yes
yes
Impact + observation
Transition to the incomplete state
no
A new transition is obtained
yes
Data correction

Fig. 2. Scheme of the algorithm of the implementation investigation.

on the failure $i \xrightarrow{P} i$ if $i = i'$, or the implementation has a τ path from i to i' and a virtual loop on the failure $i' \xrightarrow{P} i'$.

For each passed state i, there is stored a set of $H(i)$, which is the set of specification states corresponding to it by H that will grow and the set of safe buttons $P(i) = \cup\{P(s) \mid s \in H(i)\}$. For each button $P \in P(i)$ there is a counter $C(i, P)$, which stores the number of pressings of P in the state i. The button $P \in P(i)$ is *complete* in the state i if $C(i, P) = t$. The state i is complete only if all the buttons of $P(i)$ are complete.

In the beginning of the test after the state query $i'_0 \in I_0$ we have

$$H(i'_0) = S_0,\ P(i'_0) = \cup\{P(s) \mid s \in S_0\},\ C(i'_0, P) = 0 \text{ for each button } P \in P(i'_0).$$

First we check the completeness of the current state i from **I**' (Fig. 2). If the state is incomplete, then there is an incomplete button $P \in P(i)$. Then, we press the button P and get a transition $i \xrightarrow{u} i'$, and the poststate i' becomes the new current state. The counter gets incremented: $c(P, i) := c(p, i)+1$. If the transition $i \xrightarrow{u} i'$ is new, we add it to **I**', remember the button P as control $P(i \xrightarrow{u} i')$, and update $H(i') := H(i') \cup S(s, i)$ for each $s \in H(i)$ if P ***safe s*** is true.

When a new state s is added to the $H(i)$, we adjust $P(i) := P(i) \cup P(s)$ and update $H(i') := H(i') \cup S(s, u)$ for the previous transition $i \xrightarrow{u} i'$ provided that $P(i \xrightarrow{u} i')$ ***safe s*** while marking the newly added states. This recursive procedure is repeated as long as possible. The finiteness conditions guarantee that the procedure ends after a finite number of steps.

If the current state i is complete, then we move to any incomplete state (if there are no such states, the construction of **I**' is finished). To do this, select from LTS **I**' a forest of trees covering all the states and oriented towards their roots, which are all incomplete states. With every transition $i \xrightarrow{u} i'$ of the forest, we associate a control button $A(i) = P(i \xrightarrow{u} i')$. We move by pressing the button $A(i)$ in each current state i. Because of nondeterminism, we may move not to i' but to a different state i'', where we press the button $A(i'')$. The transition to an incomplete state is guaranteed by the t nondeterminism of the implementation.

The checkup of the implementation ends in a finite time. The number of test inputs is equal to $O(bt^n)$ for $t > 1$, $O(bn^2)$ for $t = 1$, and the amount of computations is $O(bnt^n) + O(bn^m) + O(mk)$ for $t > 1$. For $t = 1$, the first compound is replaced by $O(bn^3)$, where b is the number of buttons, n is the number of implementation states, $m = O(bnt)$ is the number of implementation transitions, and k is the number of specification states.

The verification of the simulation after the construction of LTS **I**' tries to build a conformal correspondence R_2, while giving the verdict ***pass*** if such correspondence exists and is built or the verdict ***fail*** if such correspondence is impossible. First, a bipartite graph is constructed. Vertices of type 1 are pairs (i, s), where i is a state from **I**', and $s \in H(i)$ is a state from **S**. Vertices of type 2 are pairs $(i \xrightarrow{u} i', s)$, where $i \xrightarrow{u} i'$ is a transition from **I**', and $s \in H(i)$. At each vertex of type 2 is an arc of type 1 $(i, s) \longrightarrow (i \xrightarrow{u} i', s)$.

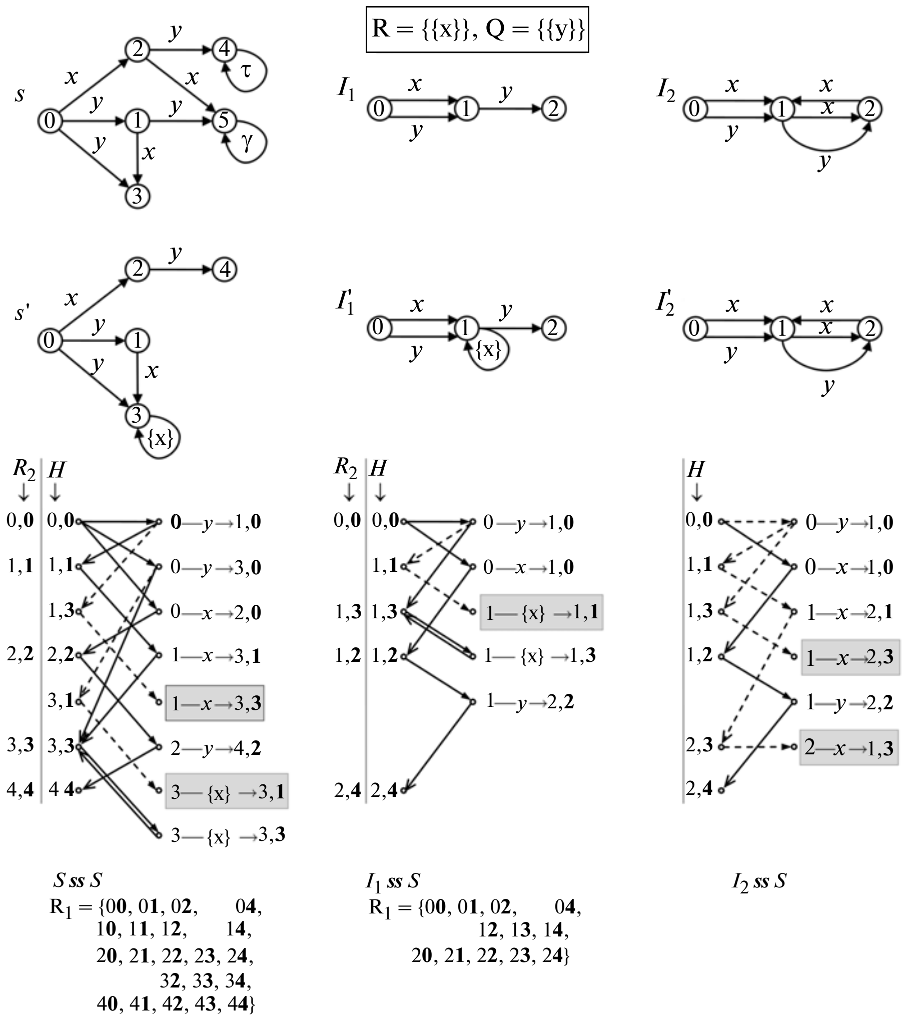
R = {{x}}, Q = {{y}}
S
I1
I2
S'
I'1
I'2
R2
H
S ss S
R1 = {00, 01, 02, 04, 10, 11, 12, 14, 20, 21, 22, 23, 24, 32, 33, 34, 40, 41, 42, 43, 44}
I1 ss S
R1 = {00, 01, 02, 04, 12, 13, 14, 20, 21, 22, 23, 24}
I2 ss S

Fig. 3. Example of simulation verification.

An arc of type 2 $(i \xrightarrow{u} i', s) \longrightarrow (i', s')$ is built if $s' \in S(s, u)$. At the same time, a list of terminal nodes of type 2 is built.

After constructing a bipartite graph, each terminal node v_2 of type 2 is removed along with an arc $v_1 \longrightarrow v_2$, the initial vertex v_1 of the arc, and each ark that ends in it $v_1' \longrightarrow v_1$. At the same time, for $v_1 = (i, s)$, the state s is removed from the set $H(i)$. These operations are repeated until a vertex of type 1 (i_0', s_0), where $i_0' \in I_0'$, gets removed or until all the terminal nodes of the second type are removed. In the first case, the algorithm terminates with the verdict ***fail***, since $(i_0', s_0) \in N$ implies that $(i_0, s_0) \in N$; in the second case, the verdict is ***pass***. In the second case, the correspondence $R_2 = \{(i, s) \mid i \in V_I' \,\&_s \in H(i)\}$ gets built. If at the construction of a bipartite graph each $H(i)$ is replaced by the set of all states of the specification, the algorithm will build the largest conformal correspondence R_1.

The amount of calculations for the verification is $O(mk^2)$.

9. EXAMPLE OF VERIFICATION FOR SAFE SIMULATION

Figure 3 shows an example of a simulation verification. The semantics contain both **R** and **Q** buttons. The **S** specification includes all the hazards: unobserved failure $\{y\}$ (in state 3), divergence (in state 4), and destruction (in state 5). The only manifestation of nondeterminism in the specification are two transitions

$0 \xrightarrow{y} 1$ and $0 \xrightarrow{y} 3$. However, in these states, the same buttons ($\{x\}$) are safe, and the specification satisfies its H hypothesis. Since the relation *ss* is reflexive on the class of specifications that meet their H hypotheses, the specification **S** is conformal to itself.

In addition to the conformal implementation **S**, there are given examples of another conformal implementation $\mathbf{I}_1$ and a nonconformal implementation $\mathbf{I}_2$. After investigating the implementations, LTS **S**', $\mathbf{I}'_1$, and $\mathbf{I}'_2$ will be built. They are distinguished by the presence of explicit (as opposed to virtual) transitions loops by the refusal $\{x\}$. In addition, in **S**', there are no transitions, which are unsafe in the specification **S**; respectively, state 5 is unreachable in **S**', since it is unreachable in **S** by safe routes.

For each of these three implementations, below are bipartite graphs to verify the compliance R_2 with the corresponding H. The state specifications are given in bold. As an example, let us consider the construction of a bipartite graph for the implementation $\mathbf{I}'_1$. Since the initial states of the implementation and the specification always H match, $(0, \mathbf{0}) \in H$, and the pair $(0, \mathbf{0})$ is the vertex of type 1. Since in implementation transitions $0 \xrightarrow{y} 1$ and $0 \xrightarrow{x} 1$ were observed, the arcs $(0, \mathbf{0}) \longrightarrow (0 \xrightarrow{y} 1, \mathbf{0})$ $(0, \mathbf{0}) \longrightarrow (0 \xrightarrow{x} 1, 0)$ are added to a bipartite graph. Since the implementation state **0** has two transitions $0 \xrightarrow{y} \mathbf{1}$ and $0 \xrightarrow{y} \mathbf{3}$, arca $(0 \xrightarrow{y} 1, \mathbf{0}) \longrightarrow (1, \mathbf{1})$ and $(0 \xrightarrow{y} 1, \mathbf{0}) \longrightarrow (1, \mathbf{3})$ are added to the bipartite graph. Similarly, since the specification state **0** has the transition $\mathbf{0} \xrightarrow{x} \mathbf{2}$, the arc $(0 \xrightarrow{x} 1, \mathbf{0}) \longrightarrow (1, \mathbf{2})$ is added to the bipartite graph. Since the implementation has a transition from refusal $1 \xrightarrow{\{x\}} 1$, the arc $(1, \mathbf{1}) \longrightarrow (1 \xrightarrow{\{x\}} 1, \mathbf{1})$ is added to the bipartite graph. And so on.

After constructing the bipartite graph, we analyze the terminal nodes of type 2 (marked in gray). There is one such vertex in the implementation $\mathbf{I}'_1$: $(1 \xrightarrow{\{x\}} 1, \mathbf{1})$. It is terminal, as in the specification there is no corresponding (for the refusal x) transition from state **1**. Consequently, the pair of states $(1, \mathbf{1})$ can not belong to a conformal correspondence. However, to build $R_2 \subseteq H$, we need to remove the (unique) incoming arc $(1, \mathbf{1}) \longrightarrow (1 \xrightarrow{\{x\}} 1, \mathbf{1})$ of a bipartite graph, the vertex of type 1 $(1, \mathbf{1})$, and all its incoming arcs (in this case, one arc $(0 \xrightarrow{y} 1, \mathbf{0}) \longrightarrow (1, \mathbf{1})$. Arcs that are subjected to removal are marked by dotted lines. The set of vertices of type 1 left after the completion of the removal process form the conformal correspondence R_2 if it includes a mandatory pair of initial states $(0, \mathbf{0})$. This is the case for implementations **S**' and $\mathbf{I}'_1$ but not for implementation $\mathbf{I}'_2$.

For conformal realizations, the **S**' and $\mathbf{I}'_1$ correspondences R_1 are given without the bipartite graphs, which are constructed similarly but at the initial stage, while the vertices of type 1 are all pairs of states of implementation and specifications.

ACKNOWLEDGMENTS

This study was supported by the Russian Foundation for Basic Research, project no. 10-07-00147-a.

REFERENCES

1. Burdonov, I.B., Kosachev, A.S., and Kulyamin, V.V., Formalization of Test Experiments, *Program. Comput. Software* 2007, vol. 33, no. 5, pp. 239–260.

2. Burdonov, I.B., Theory of Conformity for Functional Testing of Programming Systems Based on Formal Models, *Doctoral (Phys.–Math.) Dissertation*, Moscow, 2008, Available from: http://www.ispras.ru/redverst/publications/tr-01-2007.pdf

4. Bernot, G., Testing against Formal Specifications: A Theoretical View, *Proc.4th Int. Joint Conf. on Theory and Practice of Software Development (TAPSOFT'91)*, Abramsky, S. and Maibaum, T.S.E., Eds., Brighton, 1991, vol. 2, pp. 99–119; *Lect. Notes Comput. Sci.*, 1991, vol. 494, pp. 99–119.

5. van Glabbeek, R.J., The Linear Time—Branching Time Spectrum II. The Semantics of Sequential Processes with Silent Moves, *Proc. 4th Int. Conf. on Concurrency Theory, (CONCUR'93)*, Best, E., Ed., Hildesheim, Germany, 1993; *Lect. Notes Compt. Sci.*, 1993, vol. 715, pp. 66–81.

6. Milner, R., Lectures on a Calculus for Communicating Systems, *Proc. Semin. on Concurrency*, *Lect. Notes Compt. Sci.*, 1986, vol. 197, pp. 197–220.

7. Milner, R., *Communication and Concurrency*, New York: Prentice-Hall, 1989.