ISSN 0361-7688, Programming and Computer Software, 2011, Vol. 37, No. 1, pp. 1–14. © Pleiades Publishing, Ltd., 2011.
Original Russian Text © I B. Bourdonov, A.S. Kossatchev, 2011, published in Programmirovanie, 2011, Vol. 37, No. 1.

RUBRIKA

Specification Completion for IOCO

I. B. Bourdonov and A. S. Kossatchev

Institute of System Programming, Russian Academy of Sciences,
ul. Solzhenitsyna 25, Moscow, 109004 Russia
e-mail: igor@ispras.ru, kos@ispras.ru

Received August 24, 2010

Abstract—The paper is devoted to the *ioco* relation, which determines conformance of an implementation to the specification. Problems related to nonreflexivity of the *ioco* relation, presence of nonconformal traces (which are lacking in any conformal implementation) in the specification, lack of the *ioco* preservation upon composition (composition of implementations conformal to their specifications may be not conformal to the composition of these specifications), and "false" errors when testing in a context are considered. To solve these problems, an algorithm of specification completion preserving *ioco* is proposed (the class of conformal implementations is preserved). The above-specified problems are lacking in the class of completed specifications.

DOI: 10.1134/S0361768811010014

1. INTRODUCTION

Correctness of a system under study is meant to be conformance of the system to given requirements. When speaking of conformance verification, one suggests that the system is mapped into an implementation model (implementation); requirements are mapped into a specification model (specification); and conformance, into a binary conformance relation. If requirements to the system are expressed in terms of its interaction with the external environment, then the conformance can be checked by means of testing, i.e., in the course of test experiments with the system under study.

One of the most well-known conformances is the ***ioco*** (Input-Output COnformance) relation introduced by J. Tretmans [12]. The interaction is meant to be exchange of discrete portions of information between the implementation and environment. The information transmitted from the environment to the implementation is called *stimulus* (input), and the information that the implementation sends to the environment is called *reaction* (output). It is assumed that the implementation cannot refuse to receive a stimulus (stimulus *blocking is impossible*); however, the implementation can stop to produce reactions. Note that such a *refusal constitutes* a special type of observation of implementation behavior, which is denoted by symbol δ. The result of a test experiment is an observation trace—a sequence of stimuli, reactions, and refusals δ.

The ***ioco*** relation is defined on the LTS (labeled transition system) models. The LTS model is based on the notions of a *state* and *transition* between states. There are three types of transitions: stimulus-driven transition, reaction-driven transition, and internal (unobservable) τ-transition.

The ***ioco*** relation suggests that the implementation is always ready to receive any stimulus (input-enabled), whereas no such constraint is imposed on the specification. This difference results in nonreflexivity of the ***ioco*** relation. As a result, the specification may have *nonconformal traces*, which never appear in a conformal implementation.

Moreover, an ***ioco*** implementation and ***ioco*** specification must not contain infinite chains of τ-transitions (*divergence*). This leads to certain difficulties in composition of such models, since divergence may appear as a result of composition.

The basic problem of the ***ioco*** relation is also related to composition. This is violation of monotonicity: composition does not preserve conformity in the sense that composition of implementations that are conformal to their specifications is nonconformal to the composition of these specifications. A special kind of the monotonicity problem is displayed when testing in a context, where the test interacts with the implementation via some context (for example, queues of stimuli and/or reactions) rather than directly. The composition of a conformal implementation with a context may occur nonconformal to the composition of the specification with this context. As a result, the test detects "false" errors, which never occur in the direct interaction of the test with the implementation.

The above-specified problems are usually resolved by means of one or another transformation of specifications, which is called *completion* [3, 6–8, 10, 13, 14]. The completed specifications are everywhere defined by stimuli, like the implementations, and the above-specified problems are lacking, or simplified, on the

class of such specifications. However, the essential disadvantage of the transformations used so far is that the ***ioco*** relation is not invariant with respect to them: completion can reduce conformance (some nonconformal implementations become conformal) and/or strengthen it (some conformal implementations become nonconformal) [1, Section 2.5.3].

The best transformation among them is the so-called *demonic completion* [13, 14]. It replaces stimulus blocking by the so-called *chaotic behavior* [4], which admits any observation traces. This completion does not strengthen conformance but can reduce it. A completion without these disadvantages was proposed in [7]. However, firstly, it is defined for a simpler relation ***ioconf*** rather than for ***ioco***. Secondly, it is not an LTS that is transformed: a set of its traces is subjected to the transformation. That is, an LTS with a completed set of traces is not constructed, which does not allow us to use this completion upon composition of LTS models.

The earlier proposed completions do not solve the problem of "false" errors when testing in a general context, which admits stimulus blocking. Note that such contexts are met in practice. For example, bounded queues of stimuli and reactions may be treated as LTSs with stimulus blocking (when the queue is full).

In this paper, we propose two completion transformations of LTS specifications that do not have the above-specified disadvantages and are applicable to bounded, but still sufficiently wide, classes of specifications. The first transformation is applied when testing in a context that is not everywhere defined by stimuli is not planned. The second, more complicated, transformation imposes more severe restrictions on the specification and can be applied in all cases.

In Section 2, we introduce basic LTS concepts and notation and define the ***ioco*** relation, LTS composition, and test generation for ***ioco***. In Section 3, the ***ioco*** relation is discussed in more detail, and examples of specifications and implementations are given to illustrate the problems under study. The first of the proposed completions of the LTS specifications is introduced in Section 4. In Section 5, it is applied to the examples from Section 3. The second transformation for testing in a general context is defined in Section 6.

2. LTS MODEL AND IOCO RELATION

2.1. Labeled Transition System (LTS)

The implementation and specification model used is the labeled transition system (LTS). It is defined as a set $S = \mathrm{LTS}(V_S, L, E_S, s_0)$, where V_S is a nonempty set of states, L is an alphabet of external actions, $E_S \subseteq V_S \times (L \cup \{\tau\}) \times V_S$ is a set of transitions, and $s_0 \in V_S$ is an initial set. The class of all LTSs is denoted as **LTS**. For $\mathbf{LTS_i} \subseteq \mathbf{LTS}$, the class of all LTSs in alphabet L belonging to $\mathbf{LTS_i}$ is denoted as $\mathbf{LTS_i}(L)$.

Single arrows will denote presence/absence of transitions:

$$s \xrightarrow{u} s' \triangleq (s, u, s') \in E_S,$$

$$s \xrightarrow{u} \triangleq \nexists s' \; s \xrightarrow{u} s'.$$

A sequence of external actions is called a trace. For a given state s, we consider traces beginning in this state and traces ending in this state.

Formally, for $z \in L$ and $\mathrm{s} = \langle z_1, \ldots, z_n \rangle \in L^*$,

$$\mathrm{s} \Rightarrow s' \triangleq s = s'$$

$$\vee \exists s_1, \ldots, s_n \; s = s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_n = s',$$

$$s \overset{\langle z \rangle}{\Rightarrow} s' \triangleq \exists s_1, s_2 \; s \Rightarrow s_1 \xrightarrow{z} s_2 \Rightarrow s',$$

$$s \overset{\sigma}{\Rightarrow} s' \triangleq \exists s_0, \ldots, s_n$$

$$s = s_0 \overset{\langle z_1 \rangle}{\Rightarrow} s_1 \ldots s_{n-1} \overset{\langle z_n \rangle}{\Rightarrow} s_n = s',$$

$$s \overset{\sigma}{\Rightarrow} \triangleq \exists s' \; s \overset{\sigma}{\Rightarrow} s',$$

$$s \; \textbf{\textit{after}} \; \sigma \triangleq \{ s' \mid s \overset{\sigma}{\Rightarrow} s' \}.$$

A state s is said to be *reachable* if there is a trace from the initial state to it. The set of reachable states of an LTS S is denoted as $\textbf{\textit{der}}(S) \triangleq \{ s \in V_S \mid \exists \sigma \; s_0 \overset{\sigma}{\Rightarrow} s \}$. A state s is said to be *terminal* if there are no transitions from it. A state s is *stable* if there are no τ-transitions from it: $\forall u \in L \cap \{\tau\} \; s \xrightarrow{u}$. A state s is said to be *divergent* (which is denoted as $s \uparrow$) if it is a beginning of an infinite chain of τ-transitions. Otherwise, state s is said to be *convergent*. An LTS is said to be strictly convergent if all reachable states in it are convergent. The class of all strictly convergent LTSs is denoted as $\mathbf{LTS_1}$.

2.2. ioco Semantics of Interaction

Semantics of the *ioco* relation assumes that the universe of external actions **L** is divided into universe of stimuli (input) **X** and universe of reactions (output) **Y**: $\mathbf{L} = \mathbf{X} \cap \mathbf{Y} \; \& \; \mathbf{X} \cap \mathbf{Y} = \varnothing$. For an alphabet $L \subseteq \mathbf{L}$ of an LTS, we denote $X = L \cap \mathbf{X}$ and $Y = L \cap \mathbf{Y}$. Then, $L = X \cup Y \; \& \; X \cap Y = \varnothing$. Interaction of an implementation in alphabet $L \subseteq \mathbf{L}$ with the environment reduces to a sequence of interactions of the following two types: (1) sending one specified stimulus $x \in X$ from the environment to the implementation and (2) receipt of an arbitrary reaction $y \in Y$ from the implementation by the environment. It is assumed that no *refusal* of receipt of a stimulus by the implementation (which is called stimulus *blocking*) is possible. At the same time, the implementation may refuse to produce reactions. Note that such a *refusal* is observable, and the corresponding observation is denoted by symbol δ.

A state s is called *quiescent* and denoted as $\delta(s)$ if it is stable and has no reaction-driven transitions $\delta(s) \triangleq \forall u \in$

$Y \cup \{\tau\}\ s \xrightarrow{u}$. Refusal δ is observable in quiescent states of the implementation. In quiescent states of an LTS S, let us add loop transitions by symbol δ: $s \xrightarrow{\delta} s \stackrel{\Delta}{=} \delta(s)$. This will result in a new LTS $S' = \mathrm{LTS}(V_S, L_\delta, E_{S'}, s_0)$ where $\mathrm{Ld} = \mathrm{L} \ll \{\mathrm{d}\}$ and $E_{S'} = E_S \cup \{(s, \delta, s) | s \in V_S \,\&\, \delta(s)\}$. A trace of LTS S' will be referred to as an S-trace (*suspension trace*) of LTS S. Let us extend the double-arrow notation to the S-traces. The set of the S-traces of an LTS S is denoted as

$$\boldsymbol{Straces}(S) \stackrel{\Delta}{=} \{\delta \in L_\delta^* \,|\, s_0 \stackrel{\sigma}{\Rightarrow}\}.$$

An LTS in alphabet $X \cup Y$ is said to be *everywhere defined by stimuli* if, in each reachable stable state of the LTS, transitions by all stimuli are defined:

$$\forall s \in \boldsymbol{der}(S)(s \xrightarrow{\tau} \ \forall x \in \underline{X}\ s \xrightarrow{x}).$$

A *specification* for *ioco* is a strictly convergent LTS in alphabet $L \subseteq \mathbf{L}$. An *implementation* for *ioco* is a strictly convergent LTS in alphabet $L \subseteq \mathbf{L}$ that is everywhere defined by stimuli. The class of such LTSs is denoted as $\mathbf{LTS_2}$. For a strictly convergent LTS, the property of being everywhere defined by stimuli is equivalent to the existence of a trace by each stimulus in any reachable state: $\forall s \in \boldsymbol{der}(S) \forall x \in X\ s \stackrel{\langle x \rangle}{\Rightarrow}$. The property of an implementation of being strictly convergent and everywhere defined by stimuli is not verified upon testing; it is its precondition, i.e., a *hypothesis of implementation*.

2.3. Definition of the ioco Relation

The set of observations that can be obtained upon receiving reactions of an LTS S in alphabet $X \cup Y$ that is in a state $a \in V_S$ or in a state from the set $A \subseteq V_S$ is denoted as

$$\boldsymbol{out}(a) \stackrel{\Delta}{=} \{z \in Y_\delta | a \stackrel{\langle z \rangle}{\Rightarrow}\}, \text{ where } Y_\beta = Y \cup \{\delta\},$$

$$\boldsymbol{out}(A) \stackrel{\Delta}{=} \cup\{\boldsymbol{out}(a) | a \in A\}.$$

The *ioco* relation requires that, upon receiving reactions from an implementation I, only those observations can be obtained that are possible in specification S in the same situation, i.e., after observation of the same S-trace. Formally,

$$\forall L \subseteq \mathbf{L}\ \forall I \in \mathbf{LTS_2}(L)\ \forall S \in \mathbf{LTS_1}(L)$$

$$I\ \boldsymbol{ioco}\ S \stackrel{\Delta}{=} \forall \sigma \in \boldsymbol{Straces}(S)$$
$$\boldsymbol{out}(i_0\ \boldsymbol{after}\ \sigma) \subseteq \boldsymbol{out}(s_0\ \boldsymbol{after}\ \sigma).$$

2.4. Parallel Composition of LTSs

On the universe of external actions $\mathbf{L}$, we define involution "_" as an operation that, with each external action, associates the *opposite* action, reaction with stimulus and stimulus with reaction: $\forall x \in \mathbf{X}\ \underline{x} \in \mathbf{Y}$ & $\forall y \in \mathbf{Y}\ \underline{y} \in \mathbf{X}$ & $\forall z \in \mathbf{L} \underline{\underline{z}} = z$. Let us extend the involution "_" to a set A of external actions: $\underline{A} \stackrel{\Delta}{=} \{\underline{a} | a \in A\}$.

Parallel execution of two interacting LTSs defined in alphabets $A \subseteq \mathbf{L}$ and $B \subseteq \mathbf{L}$ is an execution where transitions by opposite actions z and $\underline{z}$, where $z \in A$ and $\underline{z} \in B$, are executed synchronously, i.e., simultaneously in both LTSs, such that, in the composition, this becomes a τ-transition. The other transitions are executed asynchronously, i.e., in one of the LTSs, while preserving the state of another LTS. This corresponds to the definition of parallel composition in CLS (Calculus of Communicating Systems) [9].

For alphabets $A \subseteq \mathbf{L}$ and $B \subseteq \mathbf{L}$, we define composition of the alphabets as follows: $A \between B \stackrel{\Delta}{=} (A \setminus \underline{B}) \cup (B \setminus \underline{A})$. Composition of two LTSs $I = \mathrm{LTS}(V_I, A, E_I, i_0)$ and $T = \mathrm{LTS}(V_T, B, E_T, t_0)$ is defined as the LTS $S = \mathrm{LTS}(V_I \times V_T, A \between B, E_S, i_0 t_0)$, where E_S is the least set generated by the following inference rules: $\forall z\ \forall i \in V_I$ $\forall t \in V_T$,

(1) $z \in (A \cup \{\tau\}) \setminus \underline{B}$ & $i \xrightarrow{z} i' \vdash it \xrightarrow{z} i't$,

(2) $z \in (B \cup \{\tau\}) \setminus \underline{A}$ & $t \xrightarrow{z} t' \vdash it \xrightarrow{z} i't$,

(3) $z \in A \cap \underline{B}$ & $i \xrightarrow{z} i'$ & $t \xrightarrow{\underline{z}} t' \vdash it \xrightarrow{z} i't'$.

Note that, in parallel composition in CSP [5], a synchronous transition corresponds to a pair of transitions in the LTS operands by one and the same symbol. Therefore, after the composition, an additional application of operator ***Hide*** [13] is required, which turns such compositional transitions into τ-transitions.

2.5. Testing and Test Generation for ioco

Testing is meant to be parallel execution of the LTS implementation in alphabet $L \subseteq \mathbf{L}$ and LTS test. The test plays role of the implementation environment; therefore, it is defined in the opposite alphabet $\underline{L}$. The composition of the implementation and test contains only τ-transitions, and parallel execution of the implementation and test is execution of the chain of τ-transitions in their composition. To observe a refusal δ arising in the implementation, a special τ-transition is introduced in the LTS test. It is designed for resolving deadlocks: in the composition of the implementation and test, it is executed as a τ-transition if and only if all other transitions are impossible. Symbol θ is added to the test alphabet. In the definition of the composition of LTS implementation $I = \mathrm{LTS}(V_I, L, E_I, i_0)$ and LTS test $T = \mathrm{LTS}(V_T, \underline{L} \cup \{\theta\}, E_T, t_0)$, the following fourth rule is added: $\forall i \subseteq V_I\ \forall t \in V_T$,

(4) $i \xrightarrow{\tau}$ & $t \xrightarrow{\tau}$ & $(\forall z \in L\ i \xrightarrow{z} \vee t \xrightarrow{z})$
& $t \xrightarrow{\theta} t' \vdash it \xrightarrow{\tau} it'$.

An LTS test must contain only two terminal states, which correspond to verdicts **pass** and **fail**. Testing terminates when the composition of the implementation and test occurs in the terminal state *it*. If the test state t is terminal (a verdict is rendered), then the compositional state is also terminal. In order that the converse be true (i.e., testing always terminates with a verdict), it is required that, for each terminal state *it* of the composition, the state t of the test be terminal. A deadlock does not arise in a composition if the test state is unstable or at least one transition by stimulus sending (i.e., by a symbol $\underline{x}$, where $x \in X$) is defined in it, since the implementation is everywhere defined by stimuli. In the ***ioco*** semantics, in addition to the external actions, only refusal δ is observable. Therefore, in order that a test could be applied to any implementation in alphabet L, we require that, in any reachable nonterminal stable state t of the test that has no transitions by stimulus sending, transitions by receipt of all reactions from the implementation and the θ-transition be defined:

$$\forall t \in \boldsymbol{der}(T)\ \forall u \in \underline{L} \cup \{\tau, \theta\}\ t \xrightarrow{u} \vee \exists x \in \underline{X} \cup \{\tau\}\ t \xrightarrow{x} \vee\ \forall y \in \underline{Y} \cup \{\theta\}\ t \xrightarrow{y}.$$

In order that testing always terminate in a finite time, the composition of the implementation and test should not contain infinite chains of transitions. In order that this hold for any implementation in alphabet L, all transition chains in the test must be finite (in particular, the test must be strictly convergent). In other words, the graph of the LTS test is a finite acyclic graph the sinks of which are states corresponding to verdicts ***pass*** and ***fail***.

Under the above-specified restrictions on the test, the implementation *passes* the test if, for any reachable terminal state *it*, the state t in the composition of the implementation and the test is associated with verdict ***pass***. The implementation passes a set of tests if it passes each test from the set. A test set is said to be *sound* if any conformal implementation passes it. A set of tests is *exhaustive* if any nonconformal implementation does not pass it. A set of tests is complete if it is sound and exhaustive.

Usually, tests are additionally required to satisfy the constraint of lack of "redundant indeterminism": in each reachable nonterminal state t, either one stimulus is sent ($\exists x \in \underline{X}\ t \xrightarrow{x}\ \&\ \forall u \neq x\ t \not\xrightarrow{u}$) or all reactions are received and refusal δ is detected: $\forall y \in \underline{Y} \cup \{\theta\}\ t \xrightarrow{y}\ \&\ \forall u \in \underline{X} \cup \{\tau\}\ t \not\xrightarrow{u}$. Under these constraints, a complete set of tests exists, while they do not reduce testing power.

In particular, a set of all *primitive* tests is complete. Such a test is constructed by one nonempty S-trace σ of specification S. States of an LTS test are prefixes of the S-trace σ different from the trace itself and some their extensions by one symbol; the initial state is an empty S-trace; and transitions are defined by the following inference rules: $\forall i = 1...n$, where n is the length of trace σ, $\forall z \in Y_\delta$,

$$i < n \vdash \sigma[1...i-1] \xrightarrow{\sigma(i)} \sigma[1...i],$$

$$\sigma(i) \in Y_\delta\ \&\ \sigma[1...i] \,.\, \langle z \rangle \in \boldsymbol{Straces}(S)\ \&\ \&\ (z \neq \sigma(i) \vee i = n) \vdash \sigma[1...i-1] \xrightarrow{z} \boldsymbol{pass},$$

$$\sigma(i) \in Y_\delta\ \&\ \sigma[1...i] \,.\, \langle z \rangle \notin \boldsymbol{Straces}(S) \vdash \sigma[1...i-1] \xrightarrow{z} \boldsymbol{fail},$$

where $\sigma[1..j]$ is a prefix of the S-trace σ containing the first j symbols and $\underline{\delta} \triangleq \theta$.

3. PROBLEMS RELATED TO THE *IOCO* RELATION

There are four basic problems associated with the ***ioco*** relation: (1) conformance nonreflexivity, (2) nonconformal S-traces of the specification, (3) conformance nonmonotonicity (under which we mean nonconservation of conformance upon composition), and (4) nonconservation of conformance when testing in a context.

3.1. Nonreflexivity of ioco

Since the implementation must be everywhere defined by stimuli and the specification in some states may block (not receive) certain stimuli, such a specification is nonconformal to itself with respect to ***ioco*** by definition. Nonreflexivity means that the specification cannot be understood as an implementation conformal to it. This is supposed to be a disadvantage because contradicts an intuitive idea of a specification as a set of requirements that can be implemented "as is." Such a specification cannot be used as a prototype of an implementation: if we write the implementation as an explicit expression of the specification, then we obtain a nonconformal implementation.

3.2. Nonconformal S-traces of Specifications

A *conformal* S-trace (for specification S) is an S-trace that is available in at least one conformal implementation. In the nonreflexivity problem, we separate a special case where the specification S itself contains nonconformal S-traces.

Consider an example of two specifications S_1 and S_2 shown in Fig. 1. In specification S_1, the stimulus is blocked in state 4 after S-trace $\langle \delta, x, \delta \rangle$. Since implementations are everywhere defined by stimuli, in any implementation containing the S-trace $\langle \delta, x, \delta \rangle$, this S-trace can be extended by stimulus x. If an implementation contains S-trace $\langle \delta, x, \delta, x \rangle$, then it also contains S-traces $\langle \delta, x, x \rangle$, $\langle x, \delta, x \rangle$, and $\langle x, x \rangle$. These three S-traces are available in the specification,

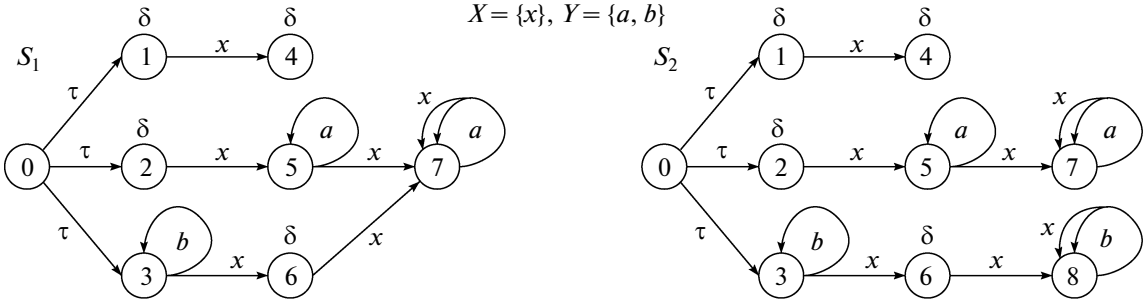

Fig. 1. S-trace $\langle \delta, x, \delta \rangle$ is conformal on the left and is nonconformal on the right.

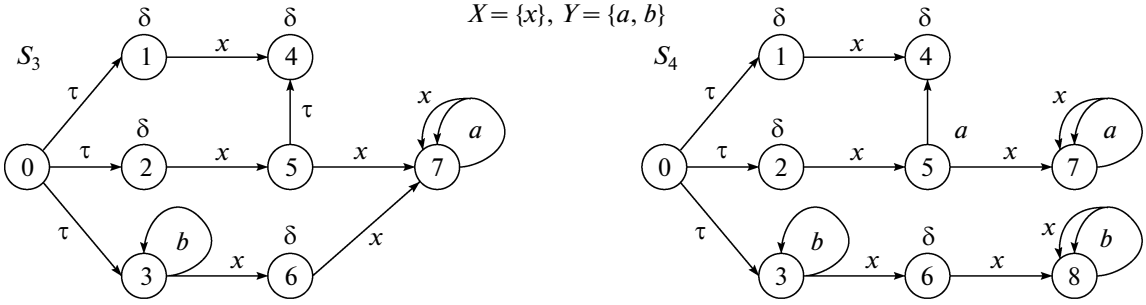

Fig. 2. S-trace $\langle \delta \rangle$.

and each of them can be extended in the specification only by reaction a. Moreover, all these three S-traces in the set of S-traces of the specification are extended by the same traces (to state 7). In order to obtain an LTS completion S_1' that is equivalent and conformal to specification S_1, it is sufficient to add transition $4 \xrightarrow{x} 7$ to the specification.

Similarly, for specification S_2, the implementation containing the S-trace $\langle \delta, x, \delta \rangle$ contains also S-traces $\langle \delta, x, \delta, x \rangle$, $\langle \delta, x, x \rangle$, $\langle x, \delta, x \rangle$, and $\langle x, x \rangle$. The last three S-traces are also available in specification S_2 and are extended in it by only reactions a, only reactions b, and both reactions a and b, respectively. Thus, in contrast to specification S_1, after these three S-traces, there is no common observation upon receiving reactions by the test (common reaction or δ). Hence, the S-trace $\langle \delta, x, \delta \rangle$ cannot be met in a conformal implementation. In order to obtain an LTS completion that is equivalent and conformal to specification S_2, it is required to delete the nonconformal S-trace $\langle \delta, x, \delta \rangle$ (for example, by deleting transition $0 \xrightarrow{\tau} 1$) rather than to add S-traces like in the case of specification S_1.

The presence of "redundant" nonconformal S-traces in a specification is not good because they generate extra tests (in particular, they result in generation of primitive tests). Consider specifications S_3 and S_4 in Fig. 2, which are modifications of specifications S_1 and S_2 from Fig. 1: in order to avoid refusal δ in state 5, instead of the transition by action a, the τ-transition is used. In specification S_3, like in S_1, all S-traces are conformal, and, to complete the specification, it is sufficient to add transition $4 \xrightarrow{x} 7$. In specification S_4, like in S_2, the S-trace $\langle \delta, x, \delta \rangle$ is not conformal. However, unlike in the case of S_2, the conformance requires that, after the S-trace $\langle \delta, x \rangle$, there were no reactions a and b. Hence, a conformal implementation cannot contain the S-trace $\langle \delta, x \rangle$. If a conformal implementation contained S-trace $\langle \delta \rangle$, it would contain a nonconformal S-trace $\langle \delta, x \rangle$ as well, since the implementation is everywhere defined by stimuli. Hence, a conformal implementation does not contain S-trace $\langle \delta \rangle$. To complete specification S_4, it is required to delete the nonconformal S-trace $\langle \delta \rangle$ by deleting, for example, transitions $0 \xrightarrow{\tau} 1$ and $0 \xrightarrow{\tau} 2$.

The example in Fig. 3 is even more amazing. Specifications S_5 and S_6 are modifications of specifications S_3 and S_4 from Fig. 2: in order to avoid refusal δ in state 3, instead of the transition by action b, the τ-transition is used. In specification S_5, like in S_3, all S-traces are conformal, and, to complete the specification, it is sufficient to add transition $4 \xrightarrow{x} 7$. In specification S_6, like in S_4, the S-trace $\langle \delta \rangle$ is not conformal. However,

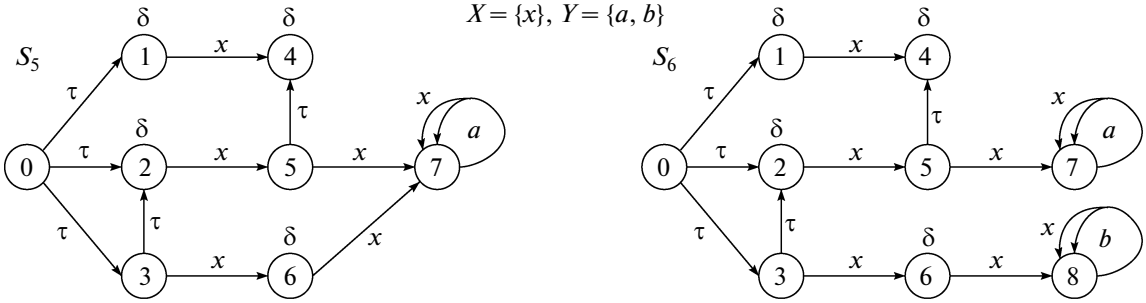

Fig. 3. All S-traces on the left are conformal and all S-traces on the right are not conformal.

unlike in the case of S_4, the conformance requires that, after an empty S-trace, there were no reactions a and b as well. Hence, a conformal implementation cannot contain an empty S-trace. Since any implementation contains an empty S-trace, there are no conformal implementations for specification S_6. All S-traces of specification S_6 are nonconformal.

These examples demonstrate how useful analysis of nonconformal S-traces can be. The complete set of primitive tests for specifications S_1–S_6 is infinite, since the number of S-traces is infinite (owing to the loop in state 7). Hence, complete testing is endless in the general case. Testing for specification S_4 can be terminated with verdict ***fail*** immediately after obtaining S-trace $\langle\delta\rangle$. Hence, there is no need to generate tests by all extensions of the S-trace $\langle\delta\rangle$ (the number of which is infinite) available in the specification. That is, we delete an infinite number of tests from the complete set. For specification S_6, testing is not required at all.

3.3. Nonmonotonicity of ioco

Conformance is said to be *monotone* with respect to composition of LTSs if composition of implementations that are conformal to their specifications is conformal to the composition of these specifications. Monotonicity is an important property: if conformance is monotone, to check "correctness" of a compound compositional system, it is sufficient to verify conformance of the system components to their specifications. In this case, the system turns out conformal to the composition of the component specifications, which may be viewed as a specification of the system. If a specification of the system is given, then we can verify decomposition of system requirements, i.e., check conformance of the system specification and specifications of its components. To this end, it is sufficient to check that the composition of the component specifications is conformal to the available system specification. Composition of the component specifications is the strongest specification (i.e., one that imposes maximal requirements) among all specifications conformal to the component specifications [1, 2].

Unfortunately, the ***ioco*** relation is not monotone [1, 2, 13, 14]. An example is presented in Fig. 4. Here, in the composition of conformal implementations $I_A \between I_B$, there is an observation δ in the very beginning (after empty S-trace), whereas this is not true for the composition of the specifications $S_A \between S_B$.

3.4. Nonconservation of ioco upon Testing in a Context

A special case of nonconservation of the ***ioco*** relation is testing in a context [11, 13, 14], when the test and implementation interact through an intermediate environment (*context*) rather than directly. Testing in a context may be viewed as testing of a system consisting of two components, one of which is the implementation and another is a fixed context. In contrast to the general case of composition, it is assumed that the context is known, has no errors, and does not need testing. The conformance nonconservation problem consists in that testing in a context can detect "false" errors, which are not detected when the test interacts with the implementation directly, without a context.

Figure 5 shows an example for implementation I and specification S (in alphabet A) and context Q_x (in alphabet B), which is an unbounded queue of stimuli. Since the queue accumulates stimuli sent to the implementation, there appears an opportunity to send two stimuli x one after another when testing by specification S (composition of the specification with the context is everywhere defined by stimuli). If reactions are received after this, then, according to specification S, two reactions y are to be successively observed. At the same time, although implementation I is conformal, a "false" error will be detected when testing in a context: the implementation may first receive two stimuli x, then produce only one reaction y, after which δ is observed.

4. SPECIFICATION COMPLETION

Solutions of the four basic problems of the ***ioco*** relation mentioned in the previous section can be sought based on the specification equivalence. Two

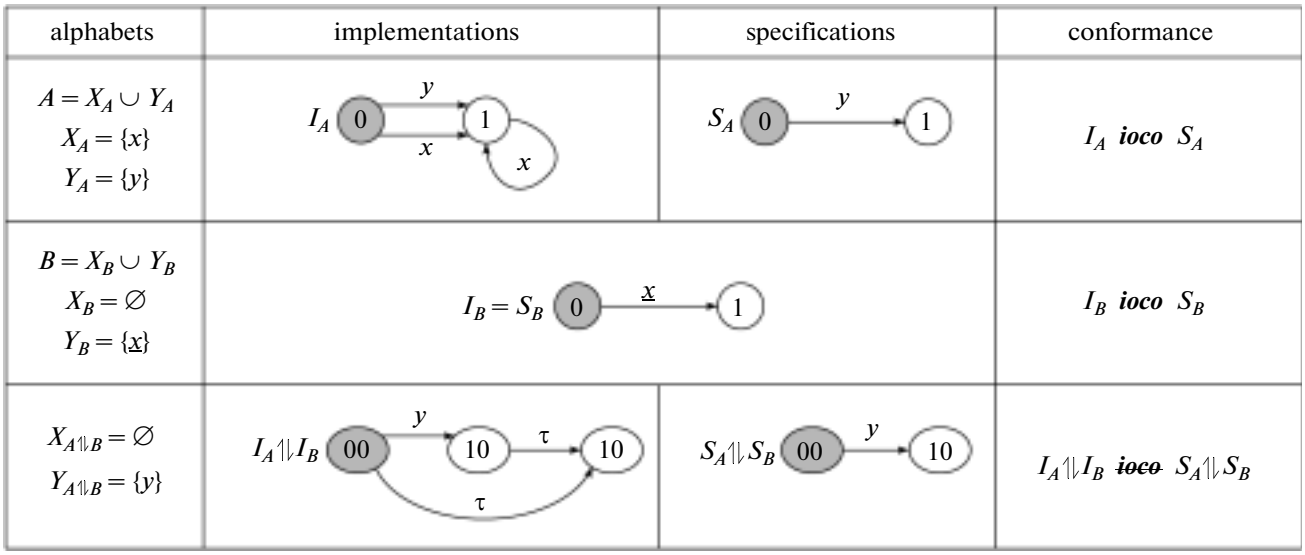

Fig. 4. Nonmonotonicity of the ***ioco*** relation.

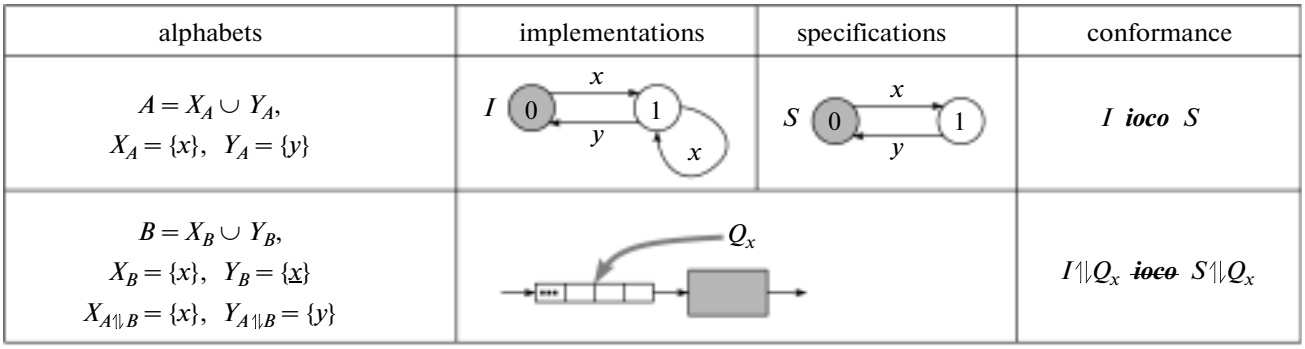

Fig. 5. Nonconservation of ***ioco*** in testing in a context.

specifications S and S' are equivalent if they define one and the same class of conformal implementations:

$$S \sim_{ioco} S' \stackrel{\Delta}{=} \{I | I \textbf{ ioco } S\} = \{I | I \textbf{ ioco } S'\}.$$

Completion is a transformation $\boldsymbol{C}$ that, for each specification S, constructs an equivalent specification $\boldsymbol{C}(S)$ for which these four problems of the ***ioco*** relation do not come to existence.

These transformations were proposed by the authors of this paper for the conformance relation $\textbf{\textit{ioco}}_{\beta\gamma\delta}$ in [1] and, for a wider class of conformances ***saco***, in [2]. Here, we optimize these transformations for the ***ioco*** relation, in particular, for the case where the specification is finite (finite number of transitions) and the transformation can be performed algorithmically for a finite time. Relation $\textbf{\textit{ioco}}_{\beta\gamma\delta}$ admits observation of not only absence of reactions δ but also of stimulus blocking. Therefore, the domain of $\textbf{\textit{ioco}}_{\beta\gamma\delta}$ is wider than that of ***ioco***. However, on the class $\textbf{LTS}_2$ of everywhere defined by stimuli specifications, these relations coincide [1, Section 2.5.3, Definition of the ***ioco*** relation]. Therefore, the results obtained in [1] are valid for the ***ioco*** relation if specifications are confined to class $\textbf{LTS}_2$.

Nonreflexivity of *ioco* and nonconformal S-traces of specifications. On class $\textbf{LTS}_2$, the $\textbf{\textit{ioco}}_{\beta\gamma\delta}$ relation is equivalent to the S-trace nesting: $\forall L \subseteq \textbf{L}\ \forall I,\ S \in \textbf{LTS}_2(L)\ I\ \textbf{\textit{ioso}}_{\beta\gamma\delta} S = \textbf{\textit{Straces}}(I) \subseteq \textbf{\textit{Straces}}(S)$ [1, Section 2.2.7, Proposition 24]. Hence, on the class of specifications $\textbf{LTS}_2$, the ***ioco*** relation is also equivalent to the S-trace nesting (see also [14, Lemma A4]). Then, it immediately follows that, on the class of specifications $\textbf{LTS}_2$, the ***ioco*** relation is reflexive. As a result, all S-traces of such specifications are conformal. Therefore, it makes sense to seek completion of specifications in class $\textbf{LTS}_2$.

Nonmonotinicity of *ioco*. The ***ioco*** relation assumes strict convergence of the specifications (class $\textbf{LTS}_1$) and that the implementations are everywhere defined by stimuli (class $\textbf{LTS}_2$).

Composition of everywhere defined by stimuli LTSs is also everywhere defined by stimuli. Indeed, consider composition $I \pitchfork T$, where $I \in \mathbf{LTS_2}(A)$ and $T \in \mathbf{LTS_2}(B)$. By the composition rules, stability of the compositional state it implies that states i and t in the LTS operands I and T, respectively, are stable. Therefore, for each stimulus $x \in X_A \backslash \underline{Y}_B$, from the fact that LTS I is everywhere defined by stimuli it follows that $i \xrightarrow{x}$, which implies $it \xrightarrow{x}$. Similarly, for each stimulus $x \in X_B \backslash \underline{Y}_A$, from the fact that LTS T is everywhere defined by stimuli, it follows that $t \xrightarrow{x}$, which implies $it \xrightarrow{x}$.

However, composition of strictly convergent LTSs may not occur strictly convergent. For example, let $I \in \mathbf{LTS_2}(A)$ and $T \in \mathbf{LTS_2}(B)$, and suppose that, for $z \in X_A \cap \underline{Y}_B$, there are loop transitions $i \xrightarrow{z} i$ in I and $t \xrightarrow{z} t$ in T. Then, there is a loop τ-transition $it \xrightarrow{\tau} it$ in composition $I \pitchfork T$; i.e., state it is divergent, and, if it is reachable, the composition is not strictly divergent.

Therefore, conformance monotonicity is meant in the following conditional sense: if composition of specifications $S_1 \pitchfork S_2$ and composition of implementations $I_1 \pitchfork I_2$ that are ***ioco***-conformal to them are strictly convergent, then $I_1 \pitchfork I_2$ ***ioco*** $S_1 \pitchfork S_2$. If this specification holds on some class of specifications, then this property is referred to as *conditional monotonicity* of the ***ioco*** relation on this class of specifications. Conditional monotonicity of ***ioco*** on class $\mathbf{LTS_2}$ was proved in [14, Theorem 9].[1]

However, while strict convergence of the specification composition can be verified (since the specifications are explicitly defined), the LTS models of the implementations may be unknown, and verification of strict convergence of the composition of the implementations may occur problematic. Therefore, it is important to determine additional restrictions on the specifications under fulfillment of which strict convergence of the specification composition implies strict convergence of the composition of the implementations. These restrictions determine a subclass of class $\mathbf{LTS_2}$. In [14], this problem was not solved. *Unconditional monotonicity* of the ***ioco*** relation on such a subclass of specifications is defined to be the following property: if composition of specifications $S_1 \pitchfork S_2$ from the given subclass is strictly convergent, then composition of the ***ioco***-conformal implementations $I_1 \pitchfork I_2$ is also strictly convergent and $I_1 \pitchfork I_2$ ***ioco*** $S_1 \pitchfork S_2$.

We will show that, for such a subclass of specifications, we can take the subclass of class $\mathbf{LTS_2}$ consisting of locally finitely branching LTSs [1, 2]. The latter are LTSs in each reachable state of which a finite number of transitions by each stimulus, including τ, are defined. The class of such LTSs is denoted by $\mathbf{LTS_3}$. It is proved in [1, Section 4.3.1, Proposition 116] that, on class $\mathbf{LTS_3}$, strict convergence of specification composition $S_1 \pitchfork S_2$ implies strict convergence of composition of $\mathbf{ioco}_{\beta\gamma\delta}$-conformal implementations $I_1 \pitchfork I_2$ and monotonicity of relation $\boldsymbol{ioco}_{\beta\gamma\delta}$: $I_1 \pitchfork I_2$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $S_1 \pitchfork S_2$. Since $\mathbf{LTS_3} \subseteq \mathbf{LTS_2}$, this assertion is true for the ***ioco*** relation as well.

Nonconservation of *ioco* when testing in a context. Similar results are obtained when context Q belongs to the class of implementations $\mathbf{LTS_2}$.

Conditional conservation of ***ioco***. On the class of specifications $\mathbf{LTS_2}$, under the condition of strict convergence of both composition of specifications with a context and composition of ***ioco***-conformal implementations with the context, the ***ioco*** relation is conserved upon testing in a context. Indeed, by [14, Lemma A.4], on the class of specifications $\mathbf{LTS_2}$, ***ioco*** is equivalent to the S-trace nesting and, hence, is reflexive: Q ***ioco*** Q. Hence, it follows from [14, Theorem 9] that $I \pitchfork Q$ ***ioco*** $S \pitchfork Q$.

Unconditional conservation of ***ioco***. It is proved in [1, Section 4.3.1, Proposition 116] that, on the class of specifications $\mathbf{LTS_3}$, strict convergence of composition of specifications with context $S \pitchfork Q$ implies strict convergence of composition of the $\boldsymbol{ioco}_{\beta\gamma\delta}$-conformal specification with the context $I \pitchfork Q$ and conservation of the $\mathbf{ioco}_{\beta\gamma\delta}$ relation upon testing in context: $I \pitchfork Q$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $S \pitchfork Q$. Since $\mathbf{LTS_3} \subseteq \mathbf{LTS_2}$, this assertion is also true for the ***ioco*** relation.

Solution of the problem of the ioco nonconservation upon testing in a general context (from class **LTS** rather than from class $\mathbf{LTS_2}$) is considered in Section 6.

In the next subsections, we define transformation $\boldsymbol{C}$: $\mathbf{LTS_1} \longrightarrow \mathbf{LTS_3}$ and show that, for a finite specification, completion can be performed algorithmically in a finite time.

In the course of completion, first of all, we will get rid of stimulus blockings in the specifications. If there are S-traces in S that are not extended by certain stimuli, $\boldsymbol{C}(S)$ should not possess this property. However, not all S-traces of this kind are preserved upon completion (with addition of lacking stimuli after them); i.e., the relation $\boldsymbol{Straces}(S) \subseteq \boldsymbol{Straces}(\boldsymbol{C}(S))$ does not necessarily hold. If specification S contains nonconformal S-traces, they are deleted (of course, with all their extensions). We define completion $\boldsymbol{C}$ as successive execution of two transformations. First, we get rid of stimulus blockings (transformation $\boldsymbol{B}$); then, nonconformal S-traces are deleted (transformation $\boldsymbol{D}$).

4.1. Transformation B

To get rid of stimulus blockings, we have to add "new" S-traces as extensions of the "old" S-traces

[1] In [14], the CSP composition with operator ***Hide***, rather than the CCS composition of LTSs, is used.

(which were available in the original specification) by those stimuli that were not used for extending traces in the original specification. These new traces should also be extended by stimuli and so on. Simultaneously, each added trace has to be extended by reactions and/or stationarity such that this extension does not change conformance. If this is impossible, the trace is announced to be nonconformal.

If an S-trace σ is added from which the "old" S-traces can be obtained by deleting certain occurrences of refusal δ, then requirements to the implementation imposed by the specification are determined based on just these "old" traces. If there are no "old" traces, the specification does not impose any requirements to the implementation after trace σ.

Let us formally define set $\boldsymbol{d}(\sigma)$ of the S-traces obtained from σ by deleting symbols δ as the least set of the S-traces generated by the following inference rules: $\forall \sigma, \mu, \lambda$,

$$\vdash \sigma \in \boldsymbol{d}(\sigma),$$

$$\mu \cdot \langle \delta \rangle \cdot \lambda \in \boldsymbol{d}(\sigma) \qquad \vdash \mu \cdot \lambda \in \boldsymbol{d}(\sigma).$$

A completed LTS specification $\boldsymbol{B}(S)$ is defined in [1] as a specification the states of which are S-traces (both "old" and "new" ones) and transitions have the form $\sigma \xrightarrow{z} \sigma \cdot \langle z \rangle$, where $z \in L$ and $\sigma \xrightarrow{\tau} \sigma \cdot \langle \delta \rangle$. The number of such S-traces in the general case is infinite even if we confine ourselves (from practical considerations) to only finite LTS specifications (with a finite number of transitions). On the other hand, when constructing states of the LTS $\boldsymbol{B}(S)$, we may not distinguish the S-traces terminating in one and the same set of states in S, since they have identical extensions. Therefore, a state of the LTS $\boldsymbol{B}(S)$ may correspond to a set σ_0 ***after*** σ of states of the LTS S rather than to the S-trace σ. However, this can be done only for the "old" S-traces $\sigma \in \boldsymbol{Straces}(S)$. We use another approach, which is suitable for both "old" and "new" S-traces, defining a state of the LTS $\boldsymbol{B}(S)$ to be family $\boldsymbol{P}(\sigma)$ of sets of states of the LTS S after all "old" traces from $\boldsymbol{d}(\sigma)$. State $\boldsymbol{P}(\sigma)$ will be one of the states of $\boldsymbol{B}(S)$ where the S-trace terminates. Formally,

$$\boldsymbol{P}(\sigma) = \{s_0 \textbf{ \textit{after}\ } \sigma' = \varnothing | \sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\}.$$

Note that all sets of states contained in family $\boldsymbol{P}(\sigma)$ are not empty (as sets of states after the S-traces available in the LTS): $\forall p \in \boldsymbol{P}(\sigma)\ p \neq \varnothing$, in particular, $\boldsymbol{P}(\sigma) \neq \{\varnothing\}$. The family $\boldsymbol{P}(\sigma)$ itself is not empty if and only if $\boldsymbol{d}(\sigma) \cap \boldsymbol{Srtaces}(S) \neq \varnothing$. Empty set $\boldsymbol{P}(\sigma)$ corresponds to all those "new" S-traces σ after which the specification does not impose any requirements to the implementation.

First, consider the case $\boldsymbol{P}(\sigma) \neq \varnothing$, i.e., $\boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S) \neq \varnothing$.

The S-trace σ is extended in $\boldsymbol{B}(S)$ by reaction $y \in Y$ if and only if each "old" S-trace $\sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)$ is also extended by this reaction in $\boldsymbol{Straces}(S)$. If this were not the case in the original specification S (i.e., if $\exists \sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S) \sigma' \cdot \langle y \rangle \notin \boldsymbol{Straces}(S)$), we would weaken conformance by adding S-trace $\sigma \cdot \langle y \rangle$ and, hence, by also adding S-trace $\sigma \cdot \langle y \rangle$: after S-trace σ', reaction y would be forbidden, whereas, now, it is permitted. Therefore, transition $\boldsymbol{P}(\sigma) \xrightarrow{y} \boldsymbol{P}(\sigma \cdot \langle y \rangle)$ is performed in the only case where $\forall \sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S) \sigma' \cdot \langle y \rangle \in \boldsymbol{Straces}(S)$. Note that, in this case, $\boldsymbol{P}(\sigma \cdot \langle y \rangle) \neq \varnothing$.

Similarly, S-trace σ is extended in $\boldsymbol{B}(S)$ by refusal δ only if each "old" S-trace $\sigma' \in \boldsymbol{d}(s) \cup \boldsymbol{Straces}(S)$is also extended by δ in $\boldsymbol{Straces}(S)$. If this were not the case in the original specification S (i.e., if $\exists \sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\ \sigma' \cdot \langle \delta \rangle \notin \boldsymbol{Straces}(S)$), we would weaken conformance by adding S-trace $\sigma \cdot \langle \delta \rangle$ and, hence, by also adding S-trace $\sigma' \cdot \langle \delta \rangle$: after S-trace σ', observation δ would be forbidden, whereas, now, it is permitted. Since transition by δ in the LTS is virtual, instead of transition $\boldsymbol{P}(\sigma) \xrightarrow{\delta} \boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$, we perform transition $\boldsymbol{P}(\sigma) \xrightarrow{\tau} \boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$ if and only if $\forall \sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\ \sigma' \cdot \langle \delta \rangle \in \boldsymbol{Straces}(S)$ and $\boldsymbol{P}(\sigma) \neq \boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$. In state $\boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$, transitions by reactions and τ-transitions are not carried out; i.e., this is a stationary state. Note that, in this case, $\boldsymbol{P}(\sigma \cdot \langle \delta \rangle) \neq \varnothing$.

Since $\boldsymbol{B}(S)$ must be everywhere defined by stimuli, in each state $\boldsymbol{P}(\sigma)$ and for each stimulus $x \in X$, transition $\boldsymbol{P}(\sigma) \xrightarrow{x} \boldsymbol{P}(\sigma \cdot \langle x \rangle)$ is performed.

Now, it may occur that $\boldsymbol{P}(\sigma \cdot \langle x \rangle) = \varnothing$; i.e., each S-trace $\sigma' \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)$ cannot be extended in S by stimulus x. This means that, after each such an S-trace σ', specification S does not impose any constraints on the implementation, admitting chaotic behavior (i.e., permitting any possible S-traces). Then, in $\boldsymbol{B}(S)$ after S-trace σ (i.e., in state $\varnothing$), we should also permit chaotic behavior. A state s in which chaotic behavior is permitted is called *demonic*: $\forall \sigma \in L_\delta^*, s \overset{\sigma}{\Rightarrow}$. Demonic state $\varnothing$ can be implemented as shown in Fig. 6.

A difficulty arises when a "new" S-trace σ is obtained (as an extension of an "old" nonconformal S-trace) in the course of transformation B that cannot be extended by a reaction or refusal δ without changing conformance. Such an S-state is associated with stable state $\boldsymbol{P}(\sigma)$. In this state, we should not define (without changing conformance) transitions by reactions; at the same time, this state should not be stationary. Such a state is called a *nonconformal* state. To formally express nonconformance of a state, we define in it a loop transition by special fictitious reaction *error* $\in Y$. This reaction is added to the alphabet of the LTS $\boldsymbol{B}(S)$.

Let us define transformation $\boldsymbol{B}(S)$ in strict terms. Let an alphabet of stimuli and reactions $L = X \cup Y$,

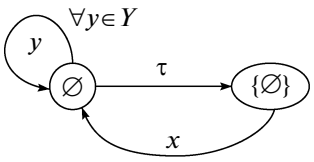

Fig. 6. Demonic state ∅.

where $X \subseteq \mathbf{X}$, $Y \subseteq Y$, and specification $S = \mathrm{LTS}(V, L, E, s_0)$ be given.

For family P of sets of states, we define operator ***after*** in such a way that, for any S-trace σ such that $\boldsymbol{P}(\sigma) = P$ and any observation u, the following equality holds: $\boldsymbol{P}(\sigma \cdot \langle u \rangle) = P$ ***after*** u. For $z \in L, p \in \mathcal{P}(\mathcal{V})$, and $\mathcal{P} \in \mathcal{P}(\mathcal{P}(\mathcal{V}))$, we have

$$p \textbf{ after } z \stackrel{\Delta}{=} \cup\{s \textbf{ after } \langle z\rangle \,|s \in p\},$$

$$\mathrm{P} \textbf{ after } z \stackrel{\Delta}{=} \{p \textbf{ after } z \neq \varnothing | p \in P\},$$

$$p \textbf{ after } \delta \stackrel{\Delta}{=} \cup\{s \textbf{ after } \langle \delta\rangle \,|s \in p\},$$

$$\mathrm{P} \textbf{ after } \delta \stackrel{\Delta}{=} P \cup \{p \textbf{ after } \langle \delta\rangle \, z \neq \varnothing | p \in P\}.$$

Note that $(P\,(\textbf{after}\ \sigma)\ \textbf{after}\ \sigma = P \textbf{ after } \delta, \varnothing \textbf{ after } z = \varnothing \textbf{ after } \delta = \varnothing)$ always holds.

Then, $\boldsymbol{B}(S) \stackrel{\Delta}{=} \mathrm{LTS}(V_1, L_1, E_1, s_1)$, where $V_1 = \mathcal{P}(\mathcal{P}(\mathcal{V}))$, $L_1 = L \cup \{error\}$, $error \in \boldsymbol{Y} \backslash Y$, $s_1 = \{s_0 \textbf{ after } \epsilon\}$, and E_1 is the least set generated by the following inference rules: $\forall P \in V_1$, $\forall x \in X$, and $\forall y \in Y$,

(1) $P \neq \varnothing$ & $\forall p \in P\, p \textbf{ after } y \neq \varnothing$

$$\vdash P \xrightarrow{y} P \textbf{ after } y,$$

(2) $P \neq \varnothing\ \forall p \in P\, p \textbf{ after } \delta \neq \varnothing$ &

$$\& \ P \neq P \textbf{ after } \delta \vdash P \xrightarrow{\tau} P \textbf{ after } \delta,$$

(3) $P \neq \varnothing$ & $P \textbf{ conf} \vdash P \xrightarrow{x} P \textbf{ after } x$,

(4) $P \neq \varnothing$ & $\neg(P \textbf{ conf}) \vdash P \xrightarrow{error} P$,

(5) $$\vdash \varnothing \xrightarrow{y} \varnothing,$$

(6) $$\vdash \varnothing \xrightarrow{\tau} \{\varnothing\},$$

(7) $$\vdash \{\varnothing\} \xrightarrow{x} \varnothing,$$

where $P \textbf{ conf} \stackrel{\Delta}{=} \exists u \in Y_\delta\ \forall p \in P\, p \textbf{ after } u \neq \varnothing$.

4.2. Transformation D

On the second stage, we get rid of nonconformal S-traces in the LTS $\boldsymbol{B}(S)$. First of all, the nonconformal S-traces are those that terminate in the nonconformal state P in which transition $P \xrightarrow{error} P$ was defined according to the fourth inference rule for $\boldsymbol{B}$.

Next, any state P for which at least one of the following three conditions holds is declared to be nonconformal:

(1) State P is stable, and a transition by stimulus $P \xrightarrow{x} P'$ leading to a nonconformal state P' is defined in it. Since not more than one transition by stimulus is defined in each state of the LTS $\boldsymbol{B}(S)$, after deletion of transition $P \xrightarrow{x} P'$, there arises blocking of stimulus x in state P.

(2) State P is stable, at least one transition by reaction is defined in it, and all transitions by reactions in state P lead to nonconformal states. After deletion of all transitions by reactions from state P, there arises refusal δ in this state, which was lacking before this (conformance nonconservation).

(3) In state P, transitions by reactions are not defined, but there is τ-transition $P \xrightarrow{\tau} P'$ leading to a nonconformal state P'. Since not more than one τ-transition is defined in each state of the LTS $\boldsymbol{B}(S)$, after deletion of transition $P \xrightarrow{\tau} P'$, there arises refusal δ in state P; therefore, the S-traces terminating in state P are nonconformally extended by refusal δ, as it was before.

This procedure of declaring nonconformal states is repeated until it is possible. Formally, the set $W(\boldsymbol{B}(S))$ of nonconformal states of the LTS $\boldsymbol{B}(S) = \mathrm{LTS}(V_1, L_1, E_1, s_1)$ is the minimal set generated by the following inference rules: $\forall P \in V_{S1}$,

$$P \xrightarrow{error} P \quad \vdash P \in W(\boldsymbol{B}(S)),$$

$$P \xrightarrow{\tau} \ \& \ \exists x \in X \exists P'\ P \xrightarrow{x} P' \ \& \ P' \in W(\boldsymbol{B}(S)) \vdash P \in W(\mathbf{B}(S)),$$

$$P \xrightarrow{\tau} \ \& \ \exists y \in Y\, P \xrightarrow{y} \ \& \ \forall y \in Y \forall P'\ (P \xrightarrow{y} P' \Rightarrow P' \in W(\boldsymbol{B}(S)) \vdash P \in W(\boldsymbol{B}(S)),$$

$$\forall y \in Y\, P \xrightarrow{y} \ \& \ \exists P'\ P \xrightarrow{\tau} P' \ \& \ P' \in W(\boldsymbol{B}(S)) \vdash P \in W(\boldsymbol{B}(S)).$$

If it turned out that $s_1 \in W(\boldsymbol{B}(S))$, then all S-traces of the LTS $\boldsymbol{B}(S)$ are nonconformal. Such a specification has no conformal implementations.

Otherwise, we delete from the LTS $\boldsymbol{B}(S)$ all nonconformal states and all transitions beginning or terminating in such states.

4.3. Transformation C

Let us define the final completion as

$$\boldsymbol{C}(S) \stackrel{\Delta}{=} \boldsymbol{D}(\boldsymbol{B}(S)) \stackrel{\Delta}{=} \mathrm{LTS}(V_2, L_2, E_2, s_2),$$

where $V_2 = V_1(\boldsymbol{B}(S))$, $L_2 = L_1 \backslash \{error\} = L$, $s_2 = s_1 = \{s_0 \textbf{ after } \epsilon\}$, $E_2 = \{(P, u, P') \in E_1 | P \in V_2 \ \& \ P' \in V_2\}$.

By construction, LTS $\boldsymbol{C}(S)$ is strictly convergent, everywhere defined by stimuli, and locally finitely branching, i.e., belongs to class $\mathbf{LTS_3}$. It is proved in [1, Proposition 50] that it is ***ioco***-equivalent to S, $\boldsymbol{C}(S) \sim_{ioco} S$. A similar assertion is proved in [2, Theorem 22]. As noted above (by [1, Section 4.3.1, Proposition 116]), based on this, all four above-specified

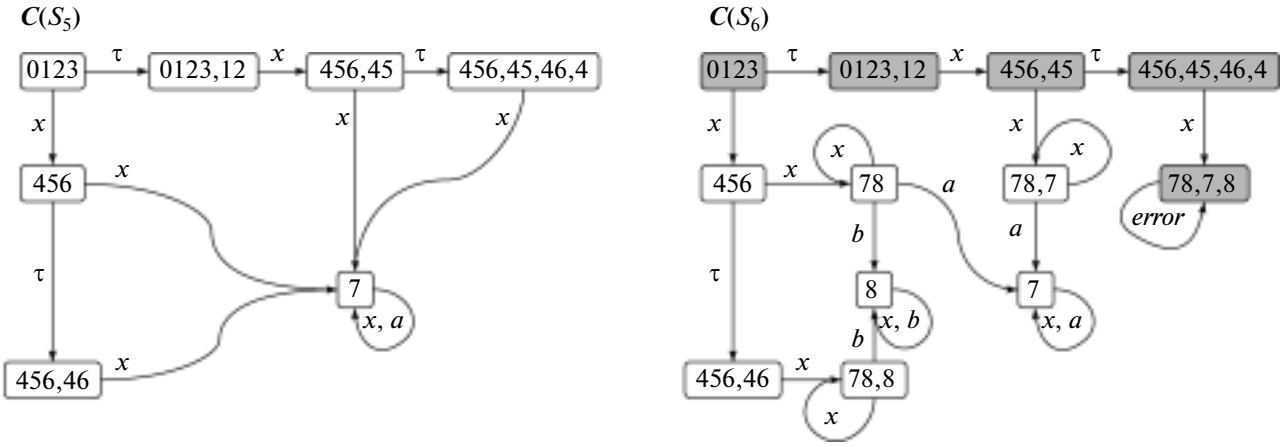

Fig. 7. Completion of specifications S_5 and S_6 (Fig. 3).

problems of the ***ioco*** relation are solved on the class of completed specifications. Monotonicity is meant as unconditional monotonicity, and, to preserve conformance when testing in a context, it is assumed that the context belongs to $\mathbf{LTS_2}$.

If an LTS specification is finite (the number of transitions is finite), then the completed specification $\boldsymbol{C}(S)$ is finite if there is no need in a demonic state, i.e., inference rules 5–7 are not applied. Obviously, transformation $\boldsymbol{C}$ in this case is carried out algorithmically for a finite time. The implementation of a demonic state is finite if and only if the alphabet is finite, since, in state $\varnothing$, transitions by all reactions are defined (inference rule 5) and, in state $\{\varnothing\}$, transitions by all stimuli are defined (inference rule 7). Therefore, if a demonic state is available, transformation $\boldsymbol{C}$ is carried out algorithmically for a finite time only for a finite alphabet. For finite LTSs, their composition is finite and is carried out algorithmically for a finite time. Verification of strict convergence for a finite LTS is also performed for a finite time (it is checked whether τ-loops are absent).

5. EXAMPLES OF SPECIFICATION COMPLETION

Consider completions of the specifications from the examples given in Section 3.

1 Resolving nonreflexivity problem of the ***ioco*** relation and problem of nonconformal S-traces in specifications. Figure 7 shows completions of specifications S_5 and S_6 (Fig. 3). Nonconformal states are highlighted in grey.

Resolving problem of nonmonotonicity of the ***ioco*** relation. Figure 8 shows completions of specifications S_A and S_B (Fig. 4) and composition of the completed specifications.

Resolving problem of conformance conservation upon testing in a context from class $\mathbf{LTS_2}$. Figure 9 shows completions of specification S (Fig. 5).

6. TESTING IN A GENERAL CONTEXT

It was shown in the previous section that the ***ioco*** relation for specifications from class $\mathbf{LTS_3}$ is certainly conserved when testing in a context from class $\mathbf{LTS_2}$. However, the requirement of being everywhere defined by stimuli is violated for some practically important contexts. For example, bounded queues of stimuli and reactions can be treated as LTSs with stimulus blocking (when the queue is full). Here, we consider the case of unconditional conservation of ***ioco*** for a general context and specifications from class $\mathbf{LTS_3}$ (in particular, those obtained by means of completion $\boldsymbol{C}$).

In order to apply ***ioco*** after composition, it is required that composition $I \between Q$ of a conformal implementation I with a context Q belong to class $\mathbf{LTS_2}$ and composition $S \between Q$ of specification S with the same context Q, to class $\mathbf{LTS_1}$. Composition $I \between Q$ for a general context Q (in contrast to the case of a context from class $\mathbf{LTS_2}$) may contain stimulus blockings and, hence, not to belong to class $\mathbf{LTS_2}$. Moreover, such an implementation always exists for a specification $S \in \mathbf{LTS_2}$ if composition $S \between Q$ contains stimulus blockings: in view of reflexivity of ***ioco***, such an implementation is the specification S itself. Therefore, for specifications from some subclass of class $\mathbf{LTS_3}$, unconditional conservation of the relation means that, if $S \between Q \in \mathbf{LTS_2}$, then $I \between Q \in \mathbf{LTS_2}$ and $I \between Q$ ***ioco*** $S \between Q$.

In [1, Chapter 4.1], transformation $T_{\beta\delta}$ was suggested to resolve problem of conservation of conformance of $\boldsymbol{ioco}_{\beta\gamma\delta}$ when testing in a general context (class LTS). This transformation is applied to locally finitely branching LTSs in an alphabet with a finite number of reactions. Intersection of the class of such LTSs with class $\mathbf{LTS_3}$ is denoted as $\mathbf{LTS_4}$.

Since relations $\boldsymbol{ioco}_{\beta\gamma\delta}$ and ***ioco*** coincide on the class of specifications $\mathbf{LTS_2}$ and $\mathbf{LTS_4} \subseteq \mathbf{LTS_3} \subseteq \mathbf{LTS_2}$, we may take advantage of this transformation for the ***ioco*** relation and specifications from class $\mathbf{LTS_4}$. Here,

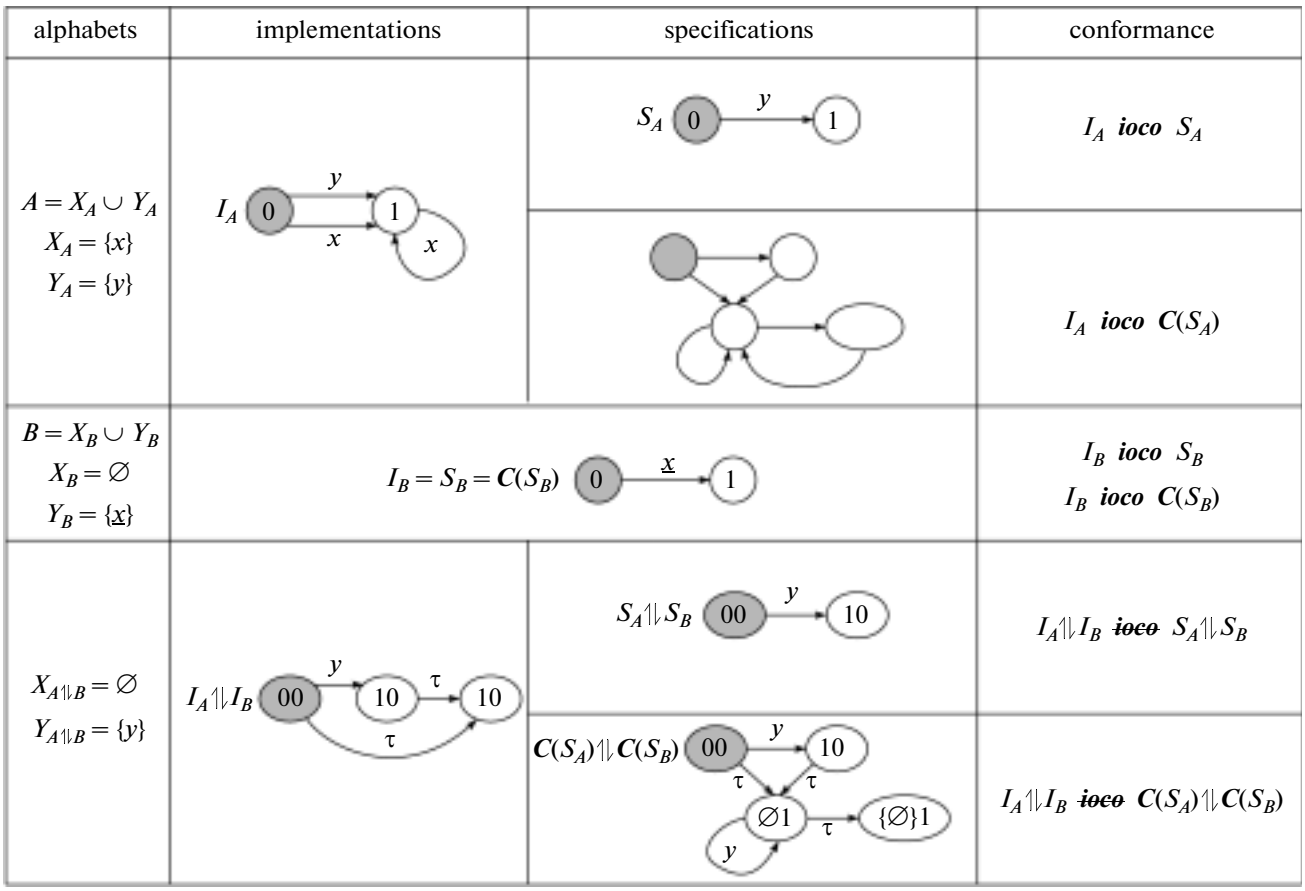

Fig. 8. Completion of specifications S_A and S_B (Fig. 4).

we describe its modified variant—transformation T: $\mathbf{LTS_4} \longrightarrow \mathbf{LTS_4}$—designed for specifications from class $\mathbf{LTS_4}$, in which stimulus blockings are absent. For LTS $S \in \mathbf{LTS_4}$, states of the LTS $T_{\beta\delta}(S)$ are constructed based on the sets of states after the traces of the LTS S. At the same time, as noted above, we may not distinguish traces that terminate in one and the same set of states, since they have identical extensions. Therefore, states of the LTS $T(S)$ are constructed simply based on the sets of the LTS S, not taking into account what traces terminate in one or another set.

Let us define transformation T in strict terms. Let an alphabet of stimuli and reactions $L = X \cup Y$, where $X \subseteq Y$ and $Y \subseteq \mathbf{Y}$, and specification $S = \mathrm{LTS}(V, L, E, s_0) \in LTS_4$ be given.

Then, $T(S) \triangleq LTS(V_T, L, E_T, s_T)$, where $V_T = \mathcal{P}(\mathcal{V}) \cup (\mathcal{P}(\mathcal{V}) \times \mathcal{Y})$, $s_T = \{s_0$ ***after*** $\epsilon\}$, and E_T is the least set generated by the following inference rules:

$\in \mathcal{P}(\mathcal{V})\ \forall \S \in \mathcal{X}\ \mathcal{X}\ \forall \dagger \in \mathcal{Y}$,

(1) p *after* $x \neq \varnothing \xrightarrow{x} p$ *after* x;

(2) p *after* $y \neq \varnothing \xrightarrow{\tau} (p, y) \xrightarrow{y} p$ *after* y

(3) p *after* $x \neq \varnothing$ & p *after* $y \neq \varnothing$

$$\vdash (p, y) \xrightarrow{x} p \text{ after } x;$$

(4) p *after* $\delta \neq \varnothing$ & p *after* $\delta \neq p$

$$\xrightarrow{\tau} p \text{ after } \delta.$$

By construction, for $S \in \mathbf{LTS_4}$, we have $T(S) \in \mathbf{LTS_4}$.

First, let us prove invariance of ***ioco*** under transformation T: $\forall S \in \mathbf{LTS_4}$, $T(S) \sim_{ioco} S$.

Indeed, by [1, Section 4.1.2, Proposition 97], $T_\beta\delta(S)$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ S & S' $\boldsymbol{ioco}_{\beta\gamma\delta}$ $T_{\beta\delta}(S)$, which, for $S \in \mathbf{LTS_4}$, is equivalent to T(S) $\boldsymbol{ioso}_{\beta\gamma\delta}$ S & S $\boldsymbol{ioco}_{\beta\gamma\delta}$ $T(S)$. By virtue of transitivity of relation $\boldsymbol{ioco}_{\beta\gamma\delta}$ [1, Section 2.2.7, Proposition 20], this implies $\forall I \in \mathbf{LTS}$ I $\boldsymbol{ioco}_{\beta\gamma\delta}$ $S \Leftrightarrow I$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $T(S)$. For $S \in \mathbf{LTS_4}$, we have $T(S) \in \mathbf{LTS_4}$. Since relations $\boldsymbol{ioco}_{\beta\gamma\delta}$ and **ioco** coincide on the class of specifications $\mathbf{LTS_4} \subseteq \mathbf{LTS_2}$, we have $\forall I \in \mathbf{LTS}$ I ***ioco*** $S \Leftrightarrow I$ ***ioco*** $T(S)$, which implies that $T(S) \sim_{ioco} S$.

Now, let us prove unconditional conservation of ***ioco*** when testing in a general context for T-transformed specifications:

$\forall I \in \mathbf{LTS_2}\ \forall S \in \mathbf{LTS_4}\ \forall Q \in \mathbf{LTS}$ I ***ioco*** S

& $T(S) \pitchfork Q \in \mathbf{LTS_2} \Rightarrow I \pitchfork Q$ ***ioco*** $T(S) \pitchfork Q$.

Indeed, since relations $\boldsymbol{ioco}_{\beta\delta}$ and ***ioco*** coincide on the class of specifications $\mathbf{LTS_4} \subseteq \mathbf{LTS_2}$, I ***ioco*** S

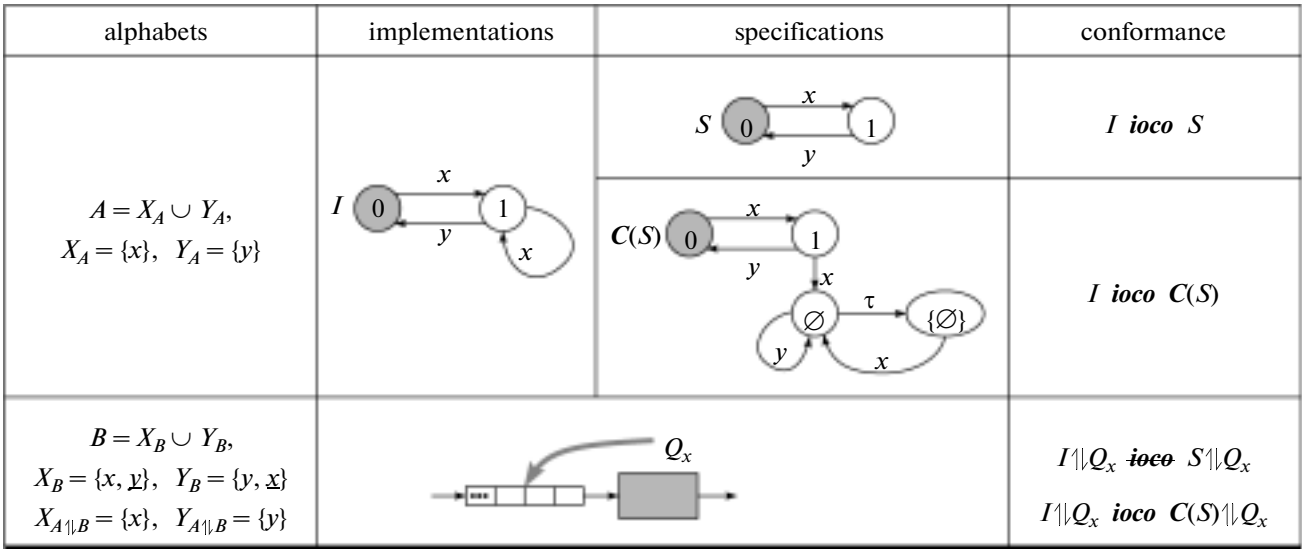

Fig. 9. Completion of specification S (Fig. 5).

implies I ***ioco***$_{\beta\gamma\delta}$ S. By [1, Section 4.1.2, Proposition 97], this implies $I \wr Q$ ***ioco***$_{\beta\gamma\delta}$ $T_{\beta\delta}(S) \wr Q$, which, for $S \in \mathbf{LTS_4}$, is equivalent to $I \wr Q$ ***ioco***$_{\beta\gamma\delta}$ $T(S) \wr Q$. Since relations ***ioco***$_{\beta\gamma\delta}$ and ***ioco*** coincide on the class of specifications $\mathbf{LTS_2}$, we have $T(S) \wr Q \in \mathbf{LTS_2}$.

The final transformation of completion for testing in a general context is performed as successive transformations $\boldsymbol{C}$ and $\boldsymbol{T}$ for any strictly convergent LTS specifications in an alphabet with a finite number of reactions.

If an LTS specification S is finite (has finite number of transitions), then specification $T(S)$ is finite. Obviously, in this case, transformation T is performed algorithmically for a finite time. Composition of a finite LTS specification with the context is finite and can be performed algorithmically for a finite time if the LTS of the context is finite. For a finite composition, strict convergence and the property of being everywhere defined by stimuli are verified for a finite time.

7. CONCLUSIONS

In the paper, transformation $\boldsymbol{C}$ of completion of LTS specifications for ***ioco*** conformance is suggested. The completed specification is equivalent to the original specification in the sense of conservation of the class of conformal implementations; i.e., the ***ioco*** relation is invariant with respect to completion. Transformation $\boldsymbol{C}$ resolves problem of the ***ioco*** reflexivity and deletes nonconformal S-traces from the original specification.

The problem of conformance monotonicity upon composition has also been addressed: if composition of completed specifications is strictly convergent, then composition of any implementations conformal to these specifications is conformal to it.

Transformation $\boldsymbol{C}$ solves also problem of conformance nonconservation when testing in a context ("false" errors): if the context and implementation are everywhere defined by stimuli and composition of the completed specification with the context is strictly convergent, then composition of any conformal implementation with this context is conformal to the former composition.

For a finite LTS specification in a finite alphabet, specification $\boldsymbol{C}(S)$ is finite, and transformation $\boldsymbol{C}$ is algorithmically performed for a finite time. Composition of finite LTSs is finite and is algorithmically performed for a finite time. Strict convergence test for a composition is also performed for a finite time.

For a general context (where stimulus blockings are allowed), transformation T is further applied to specification $\boldsymbol{C}(S)$. It preserves the class of conformal implementations and, if the number of reactions in the alphabet is finite, guarantees the following: if composition of specification $T(\boldsymbol{C}(S))$ with the context is strictly convergent and everywhere defined by stimuli, then composition of any conformal implementation with this context is conformal to it.

For a finite specification $\boldsymbol{C}(S)$ in an alphabet with a finite number of reactions, specification $T(\boldsymbol{C}(S))$ is finite, and transformation T is algorithmically performed for a finite time. Composition of a finite LTS specification with a finite context is finite and is algorithmically performed for a finite time. Check of strict convergence and everywhere definiteness by stimuli for a finite composition is algorithmically performed for a finite time.

REFERENCES

1. Bourdonov, I.B., Kossatchev, A.S., and Kuliamin, V.V., *Teoriya sootvetstviya dlya sistem s blokirovkami i razrusheniem* (Conformance Theory for Systems with Blockings and Destruction), Moscow: Nauka, 2008.
2. Bourdonov, I.B., Conformance Theory for Functional Testing of Software Systems Based on Formal Models, *Doctoral (Math.) Dissertation*, Moscow: Institute of System Programming, Russian Academy of Sciences, 2008.
3. von Bochmann, G. and Petrenko, A., Protocol Testing: Review of Methods and Relevance for Software Testing, *Proc. of the 1994 Int. Symp. on Software Testing and Analysis*, Seattle, 1994, pp. 109–124.
4. Brookes, S.D., Hoare, C.A.R., and Roscoe, A.W., A Theory of Communicating Sequential Processes, *J. Association Computing Machinery*, 1984, vol. 31, pp. 560–599.
5. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall, 1985.
6. Huo, J.L. and Peterenko, A., On Testing Partially Specified IOTS through Lossless Queues, *Lecture Notes in Computer Science* (Proc. of TestCom 2004), Berlin: Springer, 2004, vol. 2978, pp. 76–94.
7. Jard, C., Je'ron, T., Tanguy, L., and Viho, C., Remote Testing Can Be as Powerful as Local Testing, *Formal Methods for Protocol Engineering and Distributed Systems* (FORTE XII/PSTV XIX'99, Beijing, 1999), Wu, J., Chanson, S., and Gao, Q., Eds., Beijing, 1999, pp. 25–40.
8. Lee, D. and Yannakakis, M., Principles and Methods of Testing Finite State Machines: A Survey, *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, Berlin: IEEE Computer Society, 1996.
9. Milner, R., *Communication and Concurrency*, Prentice-Hall, 1989.
10. De Nicola, R. and Segala, R., A Process Algebraic View of Input/Output Automata, *Theor. Comput. Sci.*, 1995, vol. 138, pp. 391–423.
11. Revised Working Draft on "Framework Formal Methods in Conformance Testing," JTC1/SC21/WG1/ PROJECT 54/1//ISO Interim Meeting/ITU, Paris 1995.
12. Tretmans, J., Test Generation with Inputs, Outputs and Repetitive Quiescence Software-Concepts and Tools, 1996, Vol. 17, Issue 3.
12. van der Biji, M., Rensink, A., and Tretmans, J., Compositional Testing with ioco, *Lecture Notes in Computer Science* (Third Int. Workshop "Formal Approaches to Software Testing," Montreal, October 2003), Berlin: Springer, 2003, pp. 86–100.
13. van der Biji, M., Rensink, A., and Tretmans, J., Component Based Testing with ioco, *Technical Report TR-CTIT-03-34*, Univ. of Twente, 2003.

SPELL: 1. nonreflexivity