ELSEVIER

Computer Communications 19 (1996) 1236–1249

computer communications

Testing in context: framework and test derivation

Alexandre Petrenko[a], Nina Yevtushenko[b], Gregor v. Bochmann[a], Rachida Dssouli[a]

[a]*Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CP. 6128, Succ. Centre-Ville, Montréal (Québec) H3C 3J7, Canada*
[b]*Tomsk State University, 36 Lenin str., Tomsk, 634050, Russia*

Abstract

The paper addresses the problem of testing a component embedded within a given modular system. A context of the component represents the rest of the system and serves as its operational or testing environment. A framework for testing in context is presented based on the model of a system of communicating finite state machines. In particular, the problems of test executability and fault propagation in the presence of the context are identified and discussed. The proposed solution to these problems consists in computing so-called approximation of the specification in context, i.e. the FSM model of the component's properties that can be controlled and observed through the context. The approximation assures executability of tests and fault propagation through the context and serves as a base for test derivation. A conformance relation used for test derivation is shown to be the reduction relation between an implementation and the approximation of the given specification. This relation requires that the implementation produces a (sub)set of output sequences that can be produced by its specification in response to every input sequence. An approach to test generation for the reduction relation and deterministic implementations is also presented.

Keywords: Communication protocols; Conformance testing; Communicating FSMs

1. Introduction

There have been many research efforts on conformance test derivation for protocols based on their formal models and the black-box view of an implementation under test (IUT). According to the black-box testing strategy, a test suite is generated from, for example, an isolated FSM representing the protocol behavior. Testing an FSM implementation in isolation is basically the FSM equivalence problem. In this classical problem, we are given two machines with the same input alphabet: one is referred to as the *specification machine*, the other is referred to as the *implementation machine*. It is required to determine by testing whether the two are equivalent. Equivalent machines have the same input/output behavior which is completely controllable and observable in testing. Any discrepancy in their behaviors can be immediately detected provided that a proper test is applied.

In practice, however, an IUT is often tested through its environment or context. This is the case, for example, in the distributed test method of IS 9646. The IUT may also represent only an embedded part of a complex system under test which has some components that have been thoroughly tested in isolation (the embedded test method). In such situations, its input/output behavior is not directly controllable and observable. Intuitively, this means that not every aspect of the implementation behavior can be tested because it cannot be executed at all, or the context of the implementation tolerates certain faults. Deriving tests for an embedded FSM implementation significantly differs from that for an FSM in isolation. Testing in context is the core problem in assessing correctness of protocol implementations, hardware systems and object-oriented software. This is a difficult problem when it comes to fault coverage; not much work was done on systematic methods for deriving tests with a guaranteed fault coverage oriented towards testing in context.

In this paper we provide a basic framework to analyze the problems arising from testing in context. The framework is based on the model of a system of communicating finite state machines. The problems of test executability and fault propagation in the presence of the context are identified and discussed within the presented framework. The proposed solution to these problems consists in computing a so-called approximation of the specification in context, i.e. the FSM model of the component's properties that can be controlled and observed through the context. The idea is to reduce testing in context to testing in isolation, so tests with a guaranteed fault coverage can be derived from the approximations in the form of tests for the reduction relation between FSMs. An approach to test generation for the reduction relation and deterministic implementations is also presented.

This paper is structured as follows. In the next section, we define basic constructs used in the model of communicating FSMs. The following sections give our basic framework for analyzing and testing properties of an embedded component in the presence of the context. Finally, the conclusion relates our work with that in the area of system decomposition.

2. Communicating FSMs

2.1. Basic notions and definitions

A finite state machine (FSM), often simply called a machine throughout this paper, is an initialized (possibly nondeterministic) Mealy machine which can be formally defined as follows [1]. A *finite state machine* A is a 6-tuple (S, X, Y, h, s_0, D_A), where S is a set of n states with s_0 as the initial state; X – a finite set of input symbols; Y – a finite set of output symbols; D_A – a specification domain which is a subset of $S \times X$; and h – a behavior function $h: D_A \rightarrow \mathbb{P}(S \times Y)$, where $\mathbb{P}(S \times Y)$, is the powerset of $S \times Y$.

Let $\alpha = x_1, x_2 \ldots x_k \in X^*$, α is called an *acceptable input sequence for state* $s_i \in S$, if there exist k states $s_{i1}, s_{i2}, \ldots, s_{ik} \in S$ and an output sequence $\gamma = y_1 y_2 \ldots y_k \in Y^*$ such that there is a sequence of transitions $s_i - x_1/y_1 -> s_{i1} - x_2/y_2 -> s_{i2} - > \cdots - > s_{ik-1} - x_k/y_k - > s_{ik}$ in the machine. We use X_i^* to denote the set of all the acceptable input sequences for state s_i and X_A^* for state s_0, i.e. for A. The FSM A is an *observable* machine [2], if $|\{s' | (s', y) \in h(s, x)\}| \leq 1$ for all $(s, x) \in D_A$ and all $y \in Y$. This means, in observable machines, a state and an input/output (I/O) sequence uniquely determine the next state. In this paper, we consider only observable machines. The machine A becomes deterministic when $|h(s, x)| = 1$ for all $(s, x) \in D_A$. In a deterministic FSM, instead of the behavior function which is required for defining nondeterministic FSMs, we use two functions: the next state function δ, and the output function λ.

An FSM A is said to be *completely specified*, if $D_A = S \times X$. We will omit the specification domain D_A in the case of completely specified machines. If D_A is a proper subset of $S \times X$ then A is considered *partially specified*. An FSM can also be referred to as a complete or a partial machine. We will consider mainly partial machines with 'harmonized traces', i.e. machines with the following property [3]. If an observable FSM starting from its initial state can reach two different states when an input sequence is applied, then these two states accept the same subset of inputs.

We extend the behavior function of a partial machine with harmonized traces to a function on the set X_A^* of all acceptable input sequences containing the empty sequence ε, i.e. $h: S \times X_A^* \rightarrow \mathbb{P}(S \times Y^*)$. For convenience we use the same notation h for the extended function as well, since in the context of this paper such identification of these notations does not imply any contradiction. Assume $h(s, \varepsilon) = \{(s, \varepsilon)\}$ for all $s \in S$, and suppose that $h(s, \beta)$ is already specified. Then

$$h(s, \beta x) = \{s', \gamma y) \mid \exists s'' \in S[(s'', \gamma) \in h(s, \beta) \\ \& (s', y) \in h(s'', x)]\}.$$

The function h^1 is the next state function, while h^2 is the output function [1] of A, h^1 is the first and h^2 is the second projection of h, i.e.

$$h^1(s, x) = \{s' \mid \exists y \in Y[(s', y) \in h(s, x)]\},$$

$$h^2(s, x) = \{y \mid \exists s' \in S[(s', y) \in h(s, x)]\}.$$

Given sequences α, β over the same alphabet, we write $\alpha < \beta$ if α is an arbitrary prefix of β. A similar notation will be used to indicate that an I/O sequence α/δ is a prefix of another I/O sequence β/γ, i.e. we write $\alpha/\delta < \beta/\gamma$ if $\alpha < \beta$ and $\delta < \gamma$ (note that α and δ, β and γ are assumed to be of the same length, respectively). Clearly, the set X_A^* of acceptable input sequences of A is a prefix closed language.

A partial machine A with harmonized traces can often be treated as a special complete nondeterministic machine $\tilde{A}$ by treating its undefined transitions as 'don't care' transitions to a *trap* state [4]. Such transitions are labeled with an input not accepted by the current state of A and all outputs in Y. The trap state has looping transitions labeled with all inputs in X and all outputs in Y. Input sequences leading $\tilde{A}$ into the trap state are sequences not acceptable by A, they constitute the set $X^* \backslash X_A^*$. The machine $\tilde{A}$ is said to be a *completed form* of A.

We will be interested in machines which are initially connected. Given an FSM $A = (S, X, Y, h, s_0, D_A)$, A is said to be *initially connected* if $\forall s \in S \, \exists \alpha \in X_A^* \, (s \in h^1(s_0, \alpha))$.

The *equivalence* relation between two FSMs $A = (S, X, Y, h, s_0, D_A)$ and $B = (T, X, Y, H, t_0, D_B)$, written $A = B$, holds, iff

(i) $X_A^* = X_B^*$;
(ii) $\forall \alpha \in X_A^* (h^2(s_0, \alpha) = H^2(t_0, \alpha))$.

The equivalence relation between FSMs is sometimes called *trace equivalence*. The traces of a machine are those I/O sequences accepted by its initial state. Equivalent machines exhibit identical behaviors, i.e. they execute the same traces.

Given a set of input sequences $W \subseteq X_A^* \cap X_B^*$, assume that there exists an input sequence $\alpha \in W$ such that A and B produce different sets of output sequences when α is applied to their initial states. In this case, A and B are not equivalent machines (w.r.t. the set W), written as $A \neq_W B$. When the set W is not important, we use $A \neq B$ to denote nonequivalence of A and B.

We also need another relation, generalized from the classical equivalence relation, a so-called V-equivalence, where $V \subseteq X^*$. For A and B to be V-equivalent machines, written $A =_V B$, the two machines are required to produce the same set of output sequences whenever an input sequence from the set $V \subseteq X_A^* \cap X_B^*$ is applied to their initial states.

Given two machines, it is always possible to determine a maximal set V of input sequences w.r.t. which they are equivalent. Consider as an example, two complete deterministic FSMs, $A = (S, X, Y, \delta, \lambda, s_0)$ and $B = (T, X, Y, \Delta, \Lambda, t_0)$. We define an automaton $A \cap B = (S \times T, X, \omega, s_0 t_0)$ as follows. $\omega(st, x) = s't'$ iff

(i) $\delta(s,x) = s'$ and $\Delta(t,x) = t'$;
(ii) $\lambda(s,x) = \Lambda(t,x)$.

Proposition 1. *The set $L(A \cap B)$ of acceptable input sequences for the initial state $s_0 t_0$ of the automaton $A \cap B$ is the maximal set of input sequences w.r.t. which the two machines A and B are equivalent, i.e. $A =_V B$ for all $V \subseteq L(A \cap B)$.*

In the case where $V = X_A^* = X_B^*$, A and B are equivalent machines.

Given the FSMs $A = (S, X, Y, h, s_0, D_A)$ and $B = (T, X, Y, H, t_0, D_B)$, B is said to be *quasi-equivalent* [1] *to* A, iff

(i) $X_A^* \subseteq X_B^*$;
(ii) $\forall \alpha \in X_A^* \; (H^2(t_0, \alpha) = h^2(s_0, \alpha))$;

otherwise, B is not quasi-equivalent to A.

B is said to be a *reduction* of A, $B \leq A$, iff

(i) $X_A^* \subseteq X_B^*$;
(ii) $\forall \alpha \in X_A^* \; (H^2(t_0, \alpha) \subseteq h^2(s_0, \alpha))$;

otherwise, B is not a reduction of A, $B \not\leq A$.

Similar to the V-equivalence, we will also use the reduction relation and its negation w.r.t. a given set V of input sequences, namely, $\leq_V$, $\not\leq_V$.

Note that the above relations are defined for machines over the same input alphabet. However, in some cases, we may need to compare two machines even when their input alphabets are not necessarily identical. In such a case, we assume that a machine ignores all input actions that are not in its input alphabet by producing a null output n while maintaining its current state. This assumption corresponds to a particular completeness assumption widely used in the context of protocol conformance testing [5]. Under this convention, input alphabets become identical, as required by the corresponding definitions.

The introduced relations serve as the conformance relations between implementations and their specifications for deriving test suites. We assume that all potential faults are represented by a finite set $\Im(X, Y)$ of 'mutant' completely specified FSMs of the specification machine *Spec* defined over the alphabets X and Y. The set $\Im(X, Y)$ is the fault model. If this set is a universal set of all machines with at most m states then we denote it $\Im_m$.

A test suite is a finite set of finite input sequences accepted by the FSM *Spec*. A test suite *TS* is said to be *complete* for *Spec* w.r.t. the equivalence relation in the class $\Im(X, Y)$ iff

for all $Imp \in \Im(X, Y)\, Imp \neq Spec$ implies $Imp \neq_{TS} Spec$.

A complete test suite guarantees full coverage of all faults within the defined fault model. Let $\Im(X, Y) = \Im_m$. Then a complete test suite in this class is called an m-complete test suite [6]. Similarly, a complete (m-complete) test suite for *Spec* w.r.t. quasi-equivalence and reduction relations can be defined. For test derivation approaches for FSMs, the reader is referred elsewhere [1, 5, 7, 8].

2.2. Behavior of communicating FSMs

Many complex systems are typically specified as a collection of communicating components. Assuming that the behavior of each component of a system under test is known and can be described by an FSM, a system of communicating machines serves as a model of the given system.

Let K be a collection of deterministic FSMs $M_i = (S_i, U_i, Z_i, \delta_i, \lambda_i, s_{0i})$, where $i = 1, \ldots, N$. For K to form a meaningful system, it must satisfy certain constraints. Specifically, external inputs X and external outputs Y should be distinct action sets to enable interactions between the overall system and its environment, i.e. $X \cap Y = \emptyset$. An external input x should be accepted by at least one machine, i.e. $X \subseteq (\cup_i U_i)$, but not be produced by any component machine, i.e. $X \cap (\cup_i Z_i) = \emptyset$. An external output y in turn, should be produced by at least one machine, i.e. $Y \subseteq (\cup_i Z_i)$, but ignored by every component machine, i.e. $Y \cap (\cup_i U_i) = \emptyset$. A component machine M_i should not communicate with itself, i.e. $U_i \cap Z_i = \emptyset$, but machines together should be able to accept all internal output actions produced, i.e. $(\cup_i U_i) \backslash X \supseteq (\cup_i Z_i) \backslash Y$. Given two machines M_i and M_j, they communicate if there exists an internal action in the intersection $Z_i \cap U_j$. In the context of testing, we assume that communications are performed asynchronously via bounded reliable channels and bounded input buffers; to simplify our discussions we further consider bounded input queues where actions are stored. Moreover, for the sake of our discussion in this paper, we assume that the system at hand has a single message in transit. The intention behind these limitations is to obtain a finite state model of the global system. The collection K with these properties is what we mean by a (finite) *system of communicating FSMs* (ComFSMs).

In order to be able to describe the external behavior of the given system of communicating machines by the FSM model as well, we impose an I/O *ordering constraint* on a manner the environment interacts with the system. Specifically, a next external input is only submitted to the system after it has produced an external output in response to the previous input. The external output might be a null output.

The joint behavior of a finite system of ComFSMs can be described by means of a finite *product machine* and finite *composed machine*, since the number of all possible global

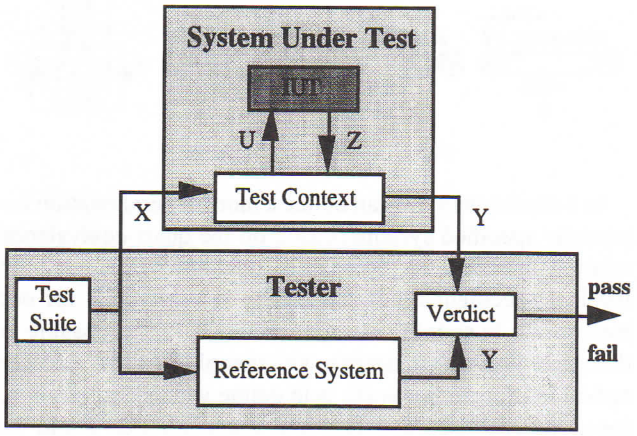

Fig. 1. A test architecture.

states of the system is limited. The former describes the joint behavior of component machines in terms of all actions within the system, whereas the latter describes the observed behavior in terms of external inputs x and outputs y.

The product machine $\Pi = M_1 \times \ldots \times M_N$ is customary represented by a global graph, obtained by performing reachability computation [9–13]. A global state consists of states of input queues and states of individual machines and can be represented in the form of a $2 \times N$ matrix

$$\begin{bmatrix} a_1 \ldots a_N \\ s_1 \ldots s_N \end{bmatrix},$$

where the first row contains states of each input queue and the second row contains current states of the machines. According to the I/O ordering constraint, global states fall into the two categories, *stable* and *transient* states. A stable state has empty input queues, and thus it is ready to accept an external input action. Accepting such an action, the system changes its current state from a stable to a transient state where it cannot accept any external action. The system returns to a stable state after it has produced an external output action or a sequence of them. Transitions between global states are labeled with an action causing a corresponding change of a global state. Note that the number of stable states in the system is bounded by the product of the numbers of states of the component machines, whereas that of the transient states depends in general on the size of queues and the total number of actions in the system. The product machine of a system of communicating machines can be deemed as a labeled transition system (LTS). The action set of this LTS is the union of all alphabets of the communicating machines.

Based on the product machine Π, a composed machine $M_1 \circ \cdots \circ M_N$ can be obtained. Here '$\circ$' is a hiding operation on all internal actions in the product machine. However, $M_1 \circ \cdots \circ M_N$ becomes an FSM under certain assumptions only, e.g. the product machine should have neither deadlocks nor livelocks. Hiding internal actions usually amounts to determinizing the LTS (an automation), and coupling external inputs and external outputs into labels of transitions between stable states.

3. A framework for testing in context

3.1. Conformance relation

Conformance testing is usually an experiment carried out by executing test cases of an appropriate test suite against a given system under test. We assume that the system under test consists of two parts, an implementation under test (IUT) and a *test context*, also simply called context. The context is the part of the system in which the IUT is embedded, and via which the IUT communicates with the tester. The context is in fact a lumped, i.e. composed machine of all components of the system, except the component under test. The context of the component serves as its operational or testing environment [14, 15] which does not require testing. In the case where the context is completely transparent (absent), testing in context reduces to the classical black-box testing. The three entities, the IUT, test context and tester, constitute the test architecture shown in Fig. 1.

The architecture can be considered as a detailed view of that in Ref. [16]. In particular, we further structure the tester, assuming that it implements a given test suite by executing test cases, i.e. external input sequences, simultaneously against both, the system under test and the specification, called a *reference system*. Test verdicts are produced by a part of tester called a *verdict machine*. In other words, the tester itself consists of three entities, test suite, reference system, and verdict machine. The verdict machine observes external output traces of the reference system and system under test and produces a verdict **pass** or **fail** for the pair of expected and obtained output traces. Verdict assignment reflects conformance requirements. For a completely specified deterministic specification, trace equivalence, i.e. the FSM equivalence, serves as a conformance relation. In this case, the system under test is required to implement the entire behavior described in the specification. The verdict machine is quite simple, it compares every pair of actions in the observed traces. If actions are identical then the machine produces the verdict **pass** which also indicates that a next test event can be executed. As soon as a discrepancy occurs, the machine produces the verdict **fail**, terminating further test execution (the tester may continue if fault diagnosis is desired, but this case falls out of the scope of this paper and is left for further study). Thus, the verdict machine has two states, an active one and state stop. We also require that the verdict machine produces the verdict **fail** if a system under test falls into livelock. A timeout mechanism can be used to detect such behavior.

We say that the IUT *passes* a test case (an external sequence) if the verdict **pass** is produced for all prefixes of the sequence. The IUT *fails* a test case (test suite) if the verdict **fail** is produced. An IUT is a *conforming* implementation if and only if the tester produces the verdict **pass** for all possible test cases.

To formalize the corresponding conformance relation between an implementation (IUT) and its specification we further assume that all the entities of the test architecture can be described by communicating FSMs. In particular, assume that C is a context FSM. Also, let *Imp* be an implementation FSM. The FSMs *Spec* and *Imp* are assumed to be defined over the input set U and output set Z. We denote the class of all FSMs over these alphabets by $\Phi(U,Z)$. $\Phi(U,Z) \ni Spec, Imp$. When an arbitrary machine A of this class is composed with the given context a livelock or deadlock may occur. As a result, the composition may not possess a composed FSM. Since our tester is equipped with a proper timer we further assume that such implementations can be detected during test execution and excluded from the set of possible implementations. The set of possible implementation machines, denoted by $\Im(U,Z)$ is then defined as follows. $\Im(U,Z) = \{A \mid A \in \Phi(U,Z) \text{ and } A \circ C$ is an FSM$\}$.

Given two FSMs *Spec* and C, a composed machine $RS = Spec \circ C$ is nothing else but the reference system. Similarly, a composed machine $IS = Imp \circ C$ models the system under test (implemented system). Outputs of the two composed machines are compared and a corresponding verdict is produced by the verdict machine. Assuming that the behavior of the reference system is completely specified, the *external equivalence* is defined as follows:

$$Imp =_C Spec \text{ iff } Imp \circ C = Spec \circ C.$$

$Imp =_C Spec$ is interpreted as 'implementation is *equivalent* to its specification in the context'; or we say '*Imp* is externally equivalent to *Spec*', for short. We also write $Imp \neq_C Spec$ if *Imp* and *Spec* are not externally equivalent.

Obviously, equivalent FSMs are also externally equivalent, regardless of the context C, that is

$$Imp = Spec \text{ implies } Imp =_C Spec,$$

but the converse is not true.

We will also use a weaker relation, the V-external equivalence for a given $V \subseteq X^*$, $V \neq \emptyset$. It is defined based on the V-equivalence of the composed machines:

$$Imp =_{C,V} Spec \text{ iff } Imp \circ C =_V Spec \circ C.$$

Based on the conformance relation, we can now formally define an m-complete test suite for the given specification in context.

Given the *Spec* and the context C, let TS be a test suite of external input sequences, $TS \subset X^*$. TS is said to be *complete* for *Spec* in the context C w.r.t. the class $\Im(U,Z)$ iff

$$\text{for all } Imp \in \Im(U,Z)\; Imp \neq_C Spec$$

$$\text{implies } Imp \neq_{C,TS} Spec.$$

An m-complete test suite in the context C is a complete test suite w.r.t. the class $\Im_m$.

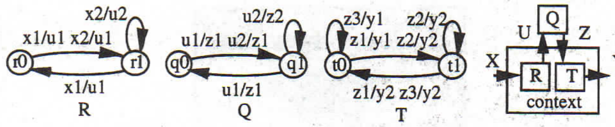

Fig. 2. A system of ComFSMs.

In a similar way, we can define a more general relation for partially specified systems, based on the quasi-equivalence relation between the composed machines. However, the level of specifiedness of the system is not crucial for our framework; to simplify our discussion we further assume in this paper that we are dealing with complete FSMs. Another option left for further study is to define yet another relation, based on the reduction relation. Such relation would be appropriate, for example, when the global system is nondeterministic. Hereinafter, we mainly consider reference systems consisting of complete deterministic machines.

3.2. Context and conforming behavior

Given a *Spec* and context C, our final objective is to have an m-complete test suite for *Spec*. The conformance relation to be tested is the external equivalence and unlike the case of testing an FSM in isolation, it is expressed in terms of actions that even do not explicitly belong to the alphabets of the specification. These actions should form the desired test suite. Testing in context significantly differs from that in isolation, and thus requires new approaches to deriving complete test suites.

Intuitively, a test has to be constructed in such a way that it excites a fault and propagates its effect through the context. The fault is a property of a non-conforming behavior of an implementation to be detected by executing the test. Then to distinguish conforming and non-conforming behaviors it is necessary to know first what constitutes the conforming behavior of an arbitrary IUT such that can be executed in the given context.

Consider an example system of ComFSMs consisting of three machines R, Q and T, as shown in Fig. 2. The set of external inputs is $X = \{x_1, x_2\}$, the set of external outputs is $Y = \{y_1, y_2\}$. We will use this system of ComFSMs to identify the problems arising from testing in context, assuming that the second machine Q of Fig. 2 is the specification machine *Spec*, and the remaining machines form the context C.

The corresponding product machine is shown in Fig. 3(a), and the composed machine *RS* extracted from it is in Fig. 3(b). Here a rather simplified version of the diagram

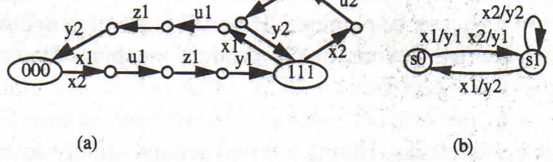

Fig. 3. The product (a) and composed (b) machines.

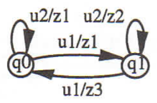

Fig. 4. A machine I externally equivalent to *Spec*.

for the product machine is presented; in particular, states of component machines related only to the two reachable stable global states with empty buffers are indicated.

A straightforward way to derive a complete test suite for *Spec* is to follow a black-box testing approach ignoring the fact that the context does not require any testing at all. Within this approach, the composed machine *RS* is the specification to derive tests from. Then one may apply, for example the W-method [17, 18] which gives an m-complete test suite, where m is an upper bound on the number of states in implementations of the specification machine *RS*. Assuming, for example, that no fault in implementations of *Spec* increases the number of states, we conclude that in the worst case m may reach eight, that is the product of the numbers of states of communicating machines. The composed machine *RS* (Fig. 2(b)) has just two reachable states out of eight possible states, but the remaining six states can become reachable should a fault occur. A rigorous analysis is needed to identify the exact number of reachable states in all mutant composed machines, assuming that faults are located in a given component and can increase its state number up to a certain integer. According to the W-method, the set of input sequences $VX^{m-n+1}W$ is an m-complete test suite. Here the set V is a state cover of RS, $X = \{x_1, x_2\}$, $m = 8$, $n = 2$, W is a characterization set of RS. In particular, $V = \{\varepsilon, x_1\}$, $X^{m-n+1} = \{\varepsilon, x_1, x_2\}^{[7]}$, $W = \{x_1\}$, where ε is the empty sequence. Concatenating these sets, we obtain a complete test suite with 192 test cases which total length is 1664 test events. Such a huge test suite is supposed to solely test one machine within the given system of three communicating machines, instead, it exhaustively tests the entire system. The problem is that black-box testing performs testing not only in context as required, but it also tests the context itself, which is not required. Clearly, a more sophisticated approach should tune tests for the given component only.

One may attempt to derive the required tests directly from the specification machine *Spec*. The problem now is that there may exist other machines that are not equivalent to *Spec* in isolation, but are externally equivalent. Consider our example. Fig. 4 shows an FSM I that is externally equivalent to *Spec* but not equivalent to it in isolation. The machine T produces the same outputs regardless of which of the two feeds its inputs. In other words, the FSM I composed with the context complies with the overall specification, i.e. with the composed machine *RS* (Fig. 3(b)). The specified system behavior would not change if one uses the FSM I as the specification for the component Q instead of *Spec*. To compute a test suite for testing in context, *Spec* alone is obviously not sufficient. All other machines externally equivalent to *Spec* have also to be taken into consideration.

Suppose one has somehow found all externally equivalent deterministic machines (within a certain limit of the number of states). Let they constitute a set $\Im_e \subseteq \Im(U, Z)$. A complete external test suite should then verify whether or not an arbitrary machine from a given class of implementations is equivalent to a machine from $\Im_e$. Note that in the extreme case, $\Im_e = \Im(U, Z)$, there is no need to test anything at all, as the context tolerates any behavior of the IUT.

Compared with the traditional testing of deterministic FSMs in isolation, which is essentially a two-machines equivalence problem, testing in context should not rely on a single deterministic specification machine.

Based on the set of conforming implementations one could further try to compute the required test suite by first deriving internal test suites from every machine of $\Im_e$ and then translating them into external ones such that all internal tests are indeed executed (excited). However, this approach fails. And not because the number of machines to be checked for the external equivalence to *Spec* is huge even within a limited number of states. The real problem is that certain internal tests have no appropriate external test at all. They are not executable in the given context. Take our example. The specification machine *Spec* is completely specified, so its test suite may well contain for example an internal test u_2u_1. However, a direct analysis of the FSM R of the context shows that starting from the initial state, the sequence u_2u_1 can never be excited as the initial state produces only u_1 in response to both x_1 and x_2.

3.3. Test executability

The problem of test executability originates from the fact that the machines of the set $\Im_e$ including *Spec* itself, are specified in isolation and give no information on which of their input sequences can be excited when they are composed with the context. Under our convention, each of these machines is also defined over the external input actions x, specifically, it maintains its current state and produces the null output n in response to any x in all states. Intuitively, to facilitate translation of internal tests into external ones, every machine of the set $\Im_e$ should be unrolled (while preserving its behavior) in such a way that a mapping of external input sequences into internal input sequences becomes an explicit part of its behavior. This motivates the following definition.

Let $C = (T, X \cup Z, U \cup Y, \Delta, \Lambda, t_0)$ be the context, where the output function Λ consists of the two mappings, Λ^u and $\Lambda^y (\Lambda = \Lambda^u \cup \Lambda^y)$. $\Lambda^u : T \times (X \cup Z) \rightarrow U$. $\Lambda^y : T \times (X \cup Z) \rightarrow Y$.

Definition 1. *Let $A = (Q, U, Z, \omega, \lambda, q_0)$ be a deterministic FSM such that $A =_C Spec$. An FSM over the inputs $X \cup U$, outputs $Z \cup \{n\}$ is a (maximal)* ***conforming part*** *of A, if for each of its I/O sequences α/γ there exists a sequence $x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$, where $\beta_i \in U^*$, $\delta_i \in Z^*$, $i = 1, \ldots, k$ such that $\alpha/\gamma < x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$, and the following conditions hold for all $i = 1, \ldots, k$:*

$$\Lambda^u(t_0, x_1\delta_1 \ldots x_i\delta_i) = \beta_1 \ldots \beta_i, \tag{1}$$

$$\lambda(q_0, \beta_1 \ldots \beta_i) = \delta_1 \ldots \delta_i. \tag{2}$$

The conforming part of A is further denoted by $\bar{A}$. Informally, the machine $\bar{A}$ exhibits the maximal part of the behavior of A that can be executed together with the context C. The FSM A is assumed to be externally equivalent to *Spec*, thus the composed machine $A \circ C$ is the reference system *RS*. The condition (1) requires that the internal I/O sequence $\delta_1 \ldots \delta_k / \beta_1 \ldots \beta_k$ can be executed by the given context when the external sequence $x_1 \ldots x_k$ is fed to the system. Eq. (2) says that the internal sequence $\beta_1 \ldots \beta_k / \delta_1 \ldots \delta_k$ should also be executed by the given FSM A. Input traces of $\bar{A}$ describe the required mapping of external input sequences into internal input sequences.

To devise a method for computing the conforming part of an arbitrary machine externally equivalent to *Spec*, we consider the *Spec* itself and its corresponding product machine $Spec \times C$. It contains all possible execution sequences in response to all external input sequences. The FSM $\overline{\text{Spec}}$ is obtained from it by first projecting out external output actions y and then coupling external input actions x with the null output n and internal input actions u with corresponding internal output actions z.

Fig. 5 shows the conforming part $\overline{\text{Spec}}$ of the specification machine *Spec* extracted from the product machine shown in Fig. 3(a). The state depicted in bold is the initial state. According to the I/O ordering constraint, states of $\overline{\text{Spec}}$ fall into two categories: states accepting external inputs x and states accepting internal inputs u. The former accept every external input, since the composed machine is completely specified. The latter accept just a single u, since the context is deterministic.

A partially specified machine $\overline{\text{Spec}}$ contains I/O sequences of the specification machine which can be executed with the given context C. At the same time, we have also the complete characterization of the complementary part of the behavior of the specification which is not executable with the given context and therefore is redundant. In fact, consider the completed form of $\overline{\text{Spec}}$, as explained earlier, once all 'don't care' transitions are replaced by transitions to the trap state, all sequences leading to its trap state are not acceptable sequences of $\overline{\text{Spec}}$, i.e. they are not executable.

Such a characterization can be used for various purposes, for example, to simplify the design of *Spec* if desired. In our example, take the transition of the initial state of *Spec* labeled with u_2/z_1. It follows from $\overline{\text{Spec}}$ that no external tester can ever test this transition. There is no need to implement such redundant transitions.

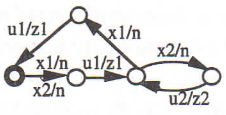

Fig. 5. The conforming part $\overline{\text{Spec}}$ of *Spec*.

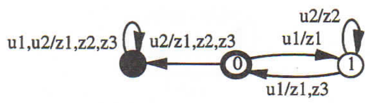

Fig. 6. The FSM E containing all conforming implementations of *Spec*.

Thus, to compute a conforming part of an FSM A externally equivalent to *Spec* simply amounts to composing it with the context into a product machine, projecting out external irrelevant actions, and coupling input and output actions, hereby containing its executable part. This is due to the fact that its composed machine is nothing more than the composed machine *RS*. The conforming part $\bar{A}$ and its completed form $\bar{A}$ enjoy an important property.

Proposition 2. *Let* $A =_C Spec$ *and* $B \leq \bar{A}$, $B \in \Im(U, Z)$. *Then* $B =_C Spec$.

Several completely specified machines can be reductions of the completed form of $\bar{A}$; these machines have a common conforming part, i.e. the machine $\bar{A}$ itself. By definition, the machine $\bar{A}$ contains only executable sequences, therefore any of its input sequences gives a test which can be executed by the context C composed with A.

Given the set $\Im_e$ of all externally equivalent (deterministic) machines, one can separately compute their conforming parts in order to eventually come up with external tests. However, Proposition 2 suggests that there could be a 'maximal' machine containing all other externally equivalent machines. In fact, all these machines can be captured by a single machine. Specifically, the set $\Im_e$ is the set of all deterministic reductions of the single nondeterministic FSM E that is the most general solution to the equation $E \circ C = Spec \circ C$ with E being a free variable.

Proposition 3. $Ip =_C Spec$ iff $Imp \leq E$.

For a cascaded composition of FSMs, we refer elsewhere [3, 19–21]; synchronous systems with feedbacks were considered [22]. For more references, see the concluding section of this paper. Corollary 3, given later in this paper, implies the existence of such a solution for asynchronously communicating FSMs considered in this paper.

In our running example, the FSM E is shown in Fig. 6. Here a 'black hole' represents the trap state.

Proposition 3 gives a precise characterization of faults tolerated by and hence undetectable with the given context. Deterministic reductions of the FSM E that are not equivalent to the FSM *Spec* correspond to the implementations that would not conform to *Spec* in isolation, but when tested through the context, become conforming to *Spec*.

Note that in an extreme case, where $\Im_e = \Im(U, Z)$, the machine E becomes a *chaos* machine $Ch = (\{p\}, U, Z, H, p)$, where $H(p, u) = \{(p, z) \mid z \in Z\}$ for all $u \in U$. The trap state in Fig. 6 represents such a chaos machine.

As explained above, to ensure executability of tests we should be interested not in externally equivalent machines, but in their conforming parts. Now that we have their aggregated characterization in the form of the single FSM E, we could compute the required parts simultaneously for all machines of the set $\Im_e$. But, first we have to generalize Definition 1 to cover nondeterministic machines externally equivalent to *Spec*.

Definition 2. *Let $A = (Q, U, Z, h, q_0)$ be an FSM such that $A =_C Spec$. An FSM over the inputs $X \cup U$, outputs $Z \cup \{n\}$ is a conforming part of A, if for each of its I/O sequences α/γ there exists a sequence $x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$, where $\beta_i \in U^*$, $\delta_i \in Z^*$, $i = 1, \ldots, k$ such that $\alpha/\gamma < x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$, and the following conditions hold for all $i = 1, \ldots, k$:*

$$\Lambda^u(t_0, x_1\delta_1 \ldots x_i\delta_i) = \beta_1 \ldots \beta_i,$$

$$\delta_1 \ldots \delta_i \in h^2(q_0, \beta_1 \ldots \beta_i).$$

Similar to the deterministic case, the conforming part $\bar{E}$ of the FSM E can be computed from a product machine $E \times C$. It is defined over sequences of internal inputs executable with the given context, interleaved with corresponding external inputs. Therefore, the external projection of an input sequence $x_1\beta_1 \ldots x_k\beta_k$ is exactly an external test $x_1 \ldots x_k$ which excites an internal test $\beta_1 \ldots \beta_k$ that is the internal projection of an input sequence $x_1\beta_1 \ldots x_k\beta_k$. The only difference from the deterministic case lies in the fact that an external sequence $x_1 \ldots x_k$ may induce several sequences.

Fig. 7 shows the conforming part $\bar{E}$ for our example. Comparing Figs. 6 and 7, one may now conclude that a certain part of the FSM E is not executable, namely the transition to a black hole followed by a universal behavior. The FSM $\bar{E}$ can be deemed as an aggregated conforming part of all implementations externally equivalent to *Spec*. Compared with the conforming part of the specification *Spec*, the FSM $\bar{E}$ has an additional transition labeled with u_1/z_3.

Combining Propositions 2 and 3, we immediately have the following characterization of the external equivalence, based on the completed form $\tilde{E}$ of the machine $\bar{E}$.

Proposition 4. *$Imp =_C Spec$* **iff** *$Imp \leq \tilde{E}$.*

3.4. Fault propagation

Since all conforming implementations, and only they, are reductions of the FSM $\tilde{E}$, and given that a test derived from an acceptable sequence is executable, one may now attempt deriving a test suite from such a machine.

To examine such an approach, we assume for simplicity that the class of implementations includes implementation machines with single output faults. In our example, there exist five such implementation machines. We wish to derive a test suite for these faults. Consider the FSM E (Fig. 6), the transition labeled with u_1/z_1 may have two types of output faults, z_2 and z_3; the transition labeled with u_2/z_2 may also have two, z_1 and z_3; and the one labeled with $u_1/z_1, z_3$ just a single fault, z_2. A transition tour of the FSM $\bar{E}$ gives a test suite that can distinguish these faulty implementations from the specification $\bar{E}$. In particular, the sequence $x_1u_1x_2u_2x_1u_1$ is a test sequence which traverses all transitions of $\bar{E}$ (Fig. 7). According to $\bar{E}$, any conforming implementation should produce either $nz_1nz_2nz_1$ or $nz_1nz_2nz_3$ in response to this input sequence. However, consider an implementation machine A with a single output fault u_1/z_3 (instead of u_1/z_1) shown in Fig. 8. $A \neq_C Spec$.

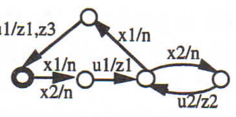

Fig. 7. The conforming part of all conforming implementations of *Spec*.

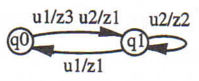

Fig. 8. A nonconforming implementation A of *Spec*.

This machine produces the output sequence $nz_3nz_2nz_1$ in response to $x_1u_1x_2u_2x_1u_1$. It is not a valid internal reaction according to the FSM $\bar{E}$ (Fig. 7). Unfortunately, when the external projection $x_1x_2x_1$ of the sequence $x_1u_1x_2u_2x_1u_1$ is executed against the system as a whole, the fault u_1/z_3 remains undetected. In fact, the context accepts the internal output sequence $z_3z_2z_1$ and produces the external output sequence $y_1y_2y_2$ of the composed machine *RS* (Fig. 3(b)). The fault is detected internally, but not externally. The shortest transition tour does not give a test suite complete in the class of single output faults.

The fault would have been detected if another external test, e.g. x_1x_1 were executed. This test cannot be obtained from the shortest transition tour of the FSM $\bar{E}$. The conclusion is that even though the FSM $\bar{E}$ is a conforming part of all possible conforming implementations and provides for the executability of tests, it does not guarantee that a given test internally detecting a fault shows its presence on the external output (fault is not propagated to the external output). The problem of fault propagation would, of course, disappear if we could observe behavior of any embedded implementation.

Proposition 5. *Let A be an implementation machine such that it is not a reduction of $\tilde{E}$ w.r.t. an input sequence $x_1\beta_1 \ldots x_k\beta_k$. Then there exists a prefix γ of the sequence $\beta_1 \ldots \beta_k$ such that A is not a reduction of E w.r.t. γ and the sequence γ is executed on input of A when the external sequence $x_1 \ldots x_k$ is applied to the context.*

This means that the sequence $x_1 \ldots x_k$ is an external test which detects the nonconforming implementation A assuming that its behavior is externally observed.

Proof. Let $x_1\beta_1 \ldots x_t\beta_t$ be a longest prefix of $x_1\beta_1 \ldots x_k\beta_k$ such that A is not a reduction of $\tilde{E}$ w.r.t. $x_1\beta_1 \ldots x_t\beta_t$ and $\delta_1 \ldots \delta_i$ be the output sequence produced by A in response to $\beta_1 \ldots \beta_i$ for $i = 1, \ldots, t$, thus $n\delta_1 \ldots n\delta_t$ does not belong to the set of output sequences of $\tilde{E}$ in response to $x_1\beta_1 \dots x_t\beta_t$. Executing the sequence $x_1\beta_1 \dots x_k\beta_t/n\delta_1 \dots n\delta_t$ the FSM $\tilde{E}$ cannot enter the trap state, since otherwise $x_1\beta_1 \dots x_t\beta_t$ is not the longest prefix with the above property. By this reason, the sequence $x_1\beta_1 \dots x_{t-1}\beta_{t-1}/n\delta_1 \dots n\delta_{t-1}$ is an I/O sequence of $\bar{E}$, and according to the definition of $\tilde{E}$, the sequence $x_1\beta_1 \dots x_{t-1}\beta_{t-1}x_t/n\delta_1 \dots n\delta_{t-1}n$ is an I/O sequence of $\tilde{E}$. Therefore $\beta_t \neq \varepsilon$.

Suppose then that $\beta_t = \beta_{t'}w$, $w \in U$. The sequence $x_1\beta_1 \dots x_t\beta_t/n\delta_1 \dots n\delta_t$ does not take the FSM $\tilde{E}$ into the trap state, hence the sequence $x_1\beta_1 \dots x_t\beta_{t'}/n\delta_1 \dots \delta_{t'}$ is an I/O sequence of $\bar{E}$, i.e. $\Lambda^u(t_0, x_1\delta_1 \dots x_i\delta_i) = \beta_1 \dots \beta_i, i = 1, \dots, t-1$ and

(a) $\Lambda^u(t_0, x_1\delta_1 \dots x_t\delta_{t'}) = \beta_1 \dots \beta_{t'}$ or
(b) $\Lambda^u(t_0, x_1\delta_1 \dots x_t\delta_{t'}) = \beta_1 \dots \beta_{t'}w', w' \in U$.

If (a) holds or $w' \neq w$ in (b) then w is not an acceptable input in the state q' which $\bar{E}$ enters executing the I/O sequence $x_1\beta_1 \dots x_t\beta_{t'}/n\delta_1 \dots n\delta_{t'}$. Then $\tilde{E}$ moves into the trap state from state q' with the input w, i.e. any continuation of $n\delta_1 \dots n\delta_{t'}$, in particular, $n\delta_1 \dots n\delta_t$ belongs to the set of output reactions of $\tilde{E}$ to the input sequence of $x_1\delta_1 \dots x_t\delta_t$. Thus, $\Lambda^u(t_0, x_1\delta_1 \dots x_t\delta_{t'}) = \beta_1 \dots \beta_t$ and the sequence $\beta_1 \dots \beta_t$ is executed on input of A when the external sequence $x_1 \dots x_t$ is applied to the context.

The implementation machine A is not a reduction of E w.r.t. the sequence $\beta_1 \dots \beta_t$, otherwise the sequence $x_1\beta_1 \dots x_t\beta_t/n\delta_1 \dots n\delta_t$ is an I/O sequence of $\bar{E}$ and thus that of $\tilde{E}$ due to Definition 2 and the fact that $\Lambda^u(t_0, x_1\delta_1 \dots x_t\delta_{t'}) = \beta_1 \dots \beta_t$. □

Thus, a nonconforming implementation can be detected by executing a test suite derived for the FSM $\tilde{E}$ w.r.t. the reduction relation provided that the behavior of the IUT is externally observed. Testing in context with observers may have useful applications [23, 24]. Our objective here, however, is to solve a more general case where no internal observer is allowed.

The problem of the conforming part $\bar{E}$ is that it comprises conforming parts of all implementations externally equivalent to *Spec*, but it lacks conforming parts of implementations which are V-externally equivalent to *Spec*, where $V \neq X^*$. As a result, the fact that an implementation is not a reduction of the machine $\tilde{E}$ w.r.t. a certain input sequence does not necessarily imply that the implementation is not externally equivalent to *Spec* w.r.t. to its external projection. In the case of the implementation A (Fig. 8), $A \neq_C Spec$, and it is not a reduction of $\tilde{E}$, but A is externally equivalent to *Spec* w.r.t. the sequence $x_1x_2x_1$. Such a conforming part of all machines should be taken into account, as well.

Therefore we formally define what constitutes the conforming part of a deterministic FSM A, such that $A =_{C,V} Spec$, where $V = L((A \circ C) \cap (Spec \circ C)) \subseteq X^*$. In fact, this is a more general version of Definition 1.

Definition 3. *Let $C = (T, X \cup Z, U \cup Y, \Delta, \Lambda, t_0)$ be the context, $A = (Q, U, Z, \omega, \lambda, q_0)$ be an arbitrary deterministic FSM such that $A =_{C,V} Spec$ where $V = L((A \circ C) \cap (Spec \circ C))$. An FSM $\bar{A}$ over the inputs $X \cup U$, outputs $Z \cup \{n\}$ is the* ***conforming part*** *of A, if for each of its I/O sequences α/γ there exists a sequence $x_1\beta_1 \dots x_k\beta_k/n\delta_1 \dots n\delta_k$, where $x_1 \dots x_k \in V$, $\beta_i \in U^*$, $\delta_i \in Z^*, i = 1, \dots, k$ such that $\alpha/\gamma < x_1\beta_1 \dots x_k\beta_k/n\delta_1 \dots n\delta_k$, and the following conditions hold for all $i = 1, \dots, k$:*

$$\Lambda^u(t_0, x_1\delta_1 \dots x_i\delta_i) = \beta_1 \dots \beta_i$$

$$\lambda(q_0, \beta_1 \dots \beta_i) = \delta_1 \dots \delta_i.$$

To compute $\bar{A}$ we may proceed in the following manner. First, given A, the maximal set V of external input sequences such that $A =_{C,V} Spec$ can be computed according to Proposition 1. We have $V = L((A \circ C) \cap (Spec \circ C))$. Then the behavior of $\bar{A}$ caused by all sequences in V can be obtained using the procedure for computing the conforming part of an externally equivalent machine. However, sequences causing a nonconforming behavior should be distinguished from unexecutable ones. By construction, $\bar{A}$ has two types of states: states accepting external inputs and states accepting internal inputs. Assume $\bar{A}$ enters a state accepting external inputs in response to an input sequence $x_1\beta_1 \dots x_k\beta_k$, where $x_1 \dots x_k \in V$, and $x_1 \dots x_k x_{k+1} \notin V$. Such a situation corresponds now to a functional error between the two composed machines, $A \circ C$ and $Spec \circ C$. To distinguish sequences causing nonconforming behavior from unexecutable ones we introduce a designated output **fail** into the completed form $\tilde{A}$ and $\bar{A}$ and replace the corresponding 'don't care' transition with the transition to the trap state. The transition is labeled with a pair x_{k+1}/**fail**. 'Don't care' transitions related to unexecutable sequences in the completed form $\tilde{A}$ of $\bar{A}$ are treated as explained earlier.

Hereafter, by the 'conforming part' of A we mean either $\bar{A}$ or $\tilde{A}$ and distinguish them only by the notation. To simplify the graphical representation of the completed form we will not explicitly present transitions leading to the trap state, except for transitions with **fail**.

It is possible to merge both steps into a single-step procedure. In particular, we again compose the given machine A and the context C into a product machine. However, to compare the external behavior of the obtained system with that of the reference system, the composed machine *RS* and the verdict machine *Ver* should also be included into the product machine. To ensure that V-equivalence holds between the two machines, the product machine $A \times C \times RS \times Ver$ should produce the verdict **pass** for a given input sequence. The verdict **fail** indicates that the corresponding sequence distinguishes $A \circ C$ from $Spec \circ C$. Stated in other words, the machine A is placed in the test system shown in Fig. 1. Each external input sequence is fed to the system until the verdict **fail** is produced. At this point, the verdict machines stops, and we are no longer interested in the behavior of A, as it became non-conforming. Note that the verdict **fail** is never produced whenever we are given a machine of the set $\Im_e$.

The conforming part is constructed from the obtained product machine similar to the above considered cases. We consider a path initiated by x, if it leads to the verdict **pass** then x is coupled with n and internal input actions u of the path are coupled with corresponding internal output actions z. Otherwise, the x is paired with **fail** and the path is discarded.

Fig. 9 illustrates the construction of the conforming part of the machine A shown in Fig. 8. Note that actions y are not shown here to simplify the graph. The state where the verdict machine stops is presented by a black square.

The conforming part of A is shown in Fig. 10, where a black hole represents the trap state. A is not externally equivalent to *Spec* w.r.t. the sequence x_1x_1, so the machine $\tilde{A}$ has a state which is reached by applying x_1 and which outputs **fail** in response to x_1.

Several important properties of the conforming part of an FSM, related to the problem of fault propagation, can now be established.

Proposition 6. *Let A be an implementation machine such that $A =_{C,V} Spec$, where $V = L((A \circ C) \cap (Spec \circ C))$, and a sequence $x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$ of A be not an I/O sequence of its conforming part $\tilde{A}$. Then $x_1 \ldots x_k \notin V$.*

Proof. Assume that the output sequence produced by A in response to the input sequence $x_1\beta_1 \ldots x_k\beta_k$ does not belong to the set of output responses of the completed form $\tilde{A}$ to this sequence. According to the definition of the completed form, it is possible iff there exists $j \leq k$ such that the input sequence $x_1 \ldots x_j \notin V$. Therefore, $x_1 \ldots x_k \notin V$. □

Proposition 7. *Given $\tilde{A}$, the conforming part of an implementation machine $A \in \Im(U,Z)$, there exists an input sequence $x_1\beta_1 \ldots x_k\beta_k$, where $\beta_i \notin U^*$, such that A is not a reduction of $\tilde{A}$ w.r.t. $x_1\beta_1 \ldots x_k\beta_k$ iff A is not externally equivalent to Spec w.r.t. $x_1 \ldots x_k$.*

Proof. First part immediately follows from Proposition 6.

Second part. We assume that $A \circ C$ and $Spec \circ C$ are not equivalent w.r.t. $x_1 \ldots x_k$ while they are equivalent w.r.t. $x_1 \ldots x_{k-1}$. In other words, $x_1 \ldots x_{k-1} \in V$, and $x_1 \ldots x_k \notin V$. Also let $x_1\beta_1 \ldots x_{k-1}\beta_{k-1}/n\delta_1 \ldots n\delta_{k-1}$ be the corresponding executable sequence of A with the context C. By Definition 3, the sequence $x_1\beta_1 \ldots x_{k-1}\beta_{k-1}$ is an acceptable sequence of $\tilde{A}$, and by definition of the completed form $\tilde{A}$, there exists a single output sequence $n\delta_1 \ldots n\delta_{k-1}$ **fail** in the completed form caused by the input sequence $x_1\beta_1 \ldots x_k\beta_{k-1}x_k$. Then the sequence $x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$ cannot be an I/O sequence of $\tilde{A}$. Thus, A is not a reduction of $\tilde{A}$ w.r.t. $x_1\beta_1 \ldots x_k\beta_k$. □

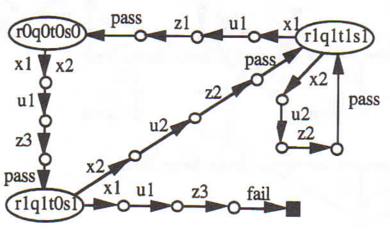

Fig. 9. Computing the conforming part of A (Fig. 8).

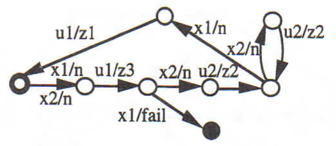

Fig. 10. The conforming part of A.

Thus, the conforming part of the given implementation machine distinguishes three types of input sequences: (a) executable sequences which constitute the conforming behavior of the given machine; (b) executable sequences which constitute the nonconforming behavior (they produce the output **fail**); and (c) unexecutable sequences. In other words, we have determined the mapping of external input sequences into internal sequences which propagate all faults of an arbitrary implementation to the external output once the implementation is not externally equivalent to its specification. The mapping is an inherent property of the conforming part offering a way to cope with the fault propagation problem for the implementation machine at hand. Then it remains to find such a mapping for all possible implementations. In fact, we can avoid the necessity of processing deterministic machines one by one. The idea is the same as capturing conforming parts of machines externally equivalent to *Spec* by the means of a nondeterministic machine.

Similarly, we determine the conforming part of an arbitrary nondeterministic FSM which might not be externally equivalent to *Spec*. Our intention is to eventually consider the replacement of *Spec* by a chaos machine $Ch = (\{p\}, U, Z, H)$, where $H(p,u) = \{(p,z) | z \in Z\}$ for all $u \in U$. Clearly, $A \leq Ch$ for all $A \in \Im(U,Z)$, thus Ch represents all possible implementation machines. This results in the following generalization of the previous definitions to cover nondeterministic machines:

Definition 4. *Let $A = (Q,U,Z,h,q_0)$ be an FSM such that $A =_{C,V} Spec$, where $V = L((A \circ C) \cap (Spec \circ C))$. An FSM over the inputs $X \cup U$, outputs $Z \cup \{n\}$ is the conforming part $\tilde{A}$ of A if for each of its I/O sequences α/γ three exists a sequence $x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$, where $x_1 \ldots x_k \in V$, $\beta_i \in U^*$, $\delta_i \in Z^*$, $i = 1, \ldots, k$ such that $\alpha/\gamma < x_1\beta_1 \ldots x_k\beta_k/n\delta_1 \ldots n\delta_k$, and the following conditions hold for all $i = 1, \ldots, k$:*

$$\Lambda^u(t_0, x_1\delta_1 \ldots x_i\delta_i) = \beta_1 \ldots \beta_i,$$

$$\delta_1 \ldots \delta_i \in h^2(q_0, \beta_1 \ldots \beta_i).$$

The conforming part of the chaos machine can be seen as the loosest description of the behavior of an embedded machine that can be controlled and observed through the context. It establishes boundaries for a conforming behavior of any possible implementation of *Spec* in the

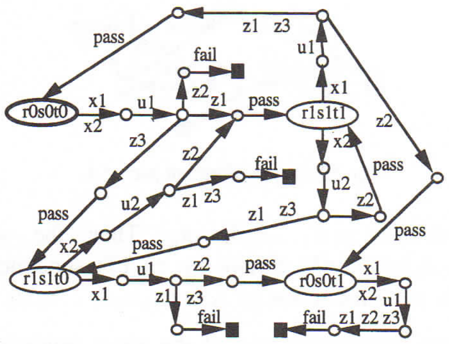

Fig. 11. Computing a conforming part of the chaos machine.

given context. Its unacceptable (unexecutable) sequences reflect unnecessary properties which the component may possess or not. These properties cannot be assessed by testing the component in the given context. The completed form of $\overline{Ch}$ is called the *approximation* of the specification in the context and is denoted [[*Spec*]].

Computing the approximation, we again construct the product machine $Ch \times C \times RS \times Ver$, as in the previous cases. Combining global states we may now omit states of the chaos machine, as it has just a single state. The only difference from the above considered cases is that the output **fail** is coupled with an input x only when no path leads to the verdict **pass**. Except of this, we apply the technique explained above. Fig. 11 illustrates the computations for our example specification.

Fig. 12 shows the approximation [[*Spec*]]. Here we have deliberately depicted states in two different shapes to highlight their origins from Fig. 11. Comparing the approximation [[*Spec*]] with the conforming part $\bar{E}$ of all externally equivalent machines (Fig. 7) one can see that $\bar{E}$ is a submachine of [[*Spec*]]. The reason is simple, the chaos machine can do whatever any deterministic or nondeterministic machine can, i.e $A \le Ch$ for all $A \in \Im(U, Z)$. Moreover, the conforming part $\overline{Ch}$ of the chaos machine contains all I/O sequences of $\bar{A}$ for all A in $\Im(U, Z)$. This observation suggests that the approximation possesses the property of the conforming part of an arbitrary implementation machine, as stated in Proposition 7.

Theorem 1. *An implementation machine $A \in \Im(U, Z)$ is not externally equivalent to Spec w.r.t. an input sequence $x_1 \ldots x_k$ iff there exists an input sequence $x_1\beta_1 \ldots x_k\beta_k$ such that A is not a reduction of [[Spec]] w.r.t. $x_1\beta_1 \ldots x_k\beta_k$.*

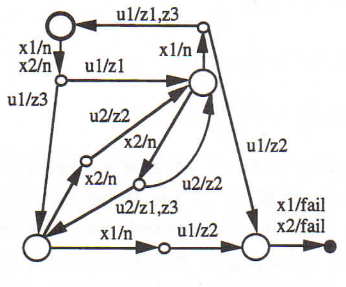

Fig. 12. The approximation [[*Spec*]].

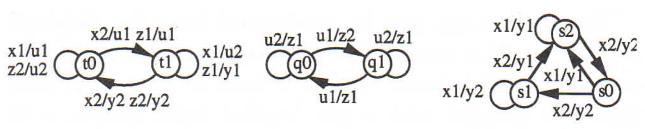

Fig. 13. The FSMs C, *Spec* and *Spec* $\circ C$.

Corollary 1. *Let $A \in \Im(U, Z)$. Then $A =_C Spec$ iff $A \le$ [[Spec]].*

Corollary 2. *An external part of an m-complete test suite for [[Spec]] w.r.t. the reduction relation is an m-complete test suite for Spec in the context C.*

Corollary 3. *There exists an FSM E such that any implementation externally equivalent to Spec is a reduction of FSM E.*

Thus, we have established that the approximation of the specification in context can serve as a proper characterization of the behavior of a component in context. Based on the approximation, test suite development for testing in context can be performed like for testing in isolation. It is sufficient first to derive a complete test suite for the approximation w.r.t. the reduction relation and second to delete internal inputs out of obtained tests leaving external inputs controlled by the tester. The outlined approach relies on the existence of a suitable test derivation technique for a nondeterministic machine w.r.t. the reduction relation. This is the subject of the next section.

Before we switch to this subject, we would like to present another example which shows that the approximation of a component at hand can be computed systematically even in the case of a system with a complete topology allowing multiple interactions between component machines. In fact, the system of ComFSMs (Fig. 2) used throughout our discussion was a simplified snapshot of a complex communications taking place between an embedded component and its context. We have intentionally divided the context into two separate machines. Test delivery and fault propagation is hampered by the two machines independently. This helped us to identify the problems arising from testing in context. Clearly, in more realistic situations, executability of tests is also affected by faults. We present here yet another tiny system of two ComFSMs. Fig. 13 shows the system and its composed machine. The process

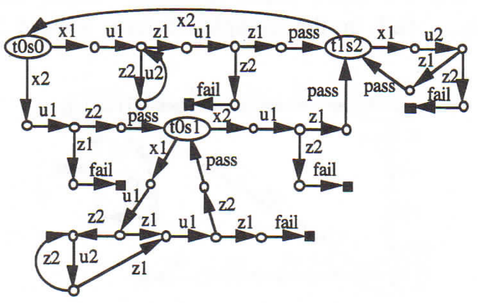

Fig. 14. Computing [[*Spec*]].

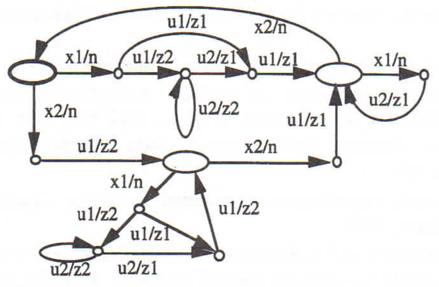

Fig. 15. The approximation [[*Spec*]].

of computing the approximation is illustrated in Fig. 14, and finally Fig. 15 presents the obtained approximation.

4. Test suite derivation for nondeterministic FSMs

As follows from the results of the previous section, an external test suite for an embedded component with respect to trace equivalence for external behavior can be obtained from a test suite derived for a nondeterministic FSM based on the reduction conformance relation.

Let $A = (S, X, Y, h, s_0)$ be an initially connected observable (possibly nondeterministic) FSM; $\Im_m$ be a set of deterministic FSMs with the input alphabet X, the output alphabet Y, and with at most m states. The FSM A is the specification machine and the set $\Im_m$ represents all possible deterministic implementations. A conforming implementation is a deterministic reduction of A. The definition of an m-complete test suite is as follows.

A test suite *TS* is said to be *m-complete* for A w.r.t. the reduction relation iff

for all $B \in \Im_m \not\leq Spec$ implies $B \not\leq_{TS} Spec$.

We introduce several auxiliary notions. Given $A = (S, X, Y, h, s_0)$ if there exists an input sequence α_i such that $h^1(s_0, \alpha) = \{s_i\}$ then s_i is said to be deterministically reachable, written *D-reachable*, in A. A is said to be *D-connected* if each of its states is D-reachable. In particular, every initially connected deterministic machine is D-connected.

We define a D-reachable state cover set V for the given FSM A in the following way. For each D-reachable state s_i in S we select an input sequence α_i which uniquely brings A from the initial state into s_i. V is the set of selected sequences for all D-reachable states.

A sequence β_{ij} *separates* states s_i and s_j if sets of output responses of A in the states s_i and s_j to β_{ij} do not intersect. In this case, states s_i and s_j are said to be *separable*. Assume such a sequence is found for each pair of separable states. The set W of these sequences is said to be a *characterization* set of A; a subset W_i of sequences in W separating state s_i from any other state separable from s_i is said to be a *state identifier* of the state s_i.

Theorem 2. *Let the given observable FSM A be D-connected and have all states pairwise separable. Then the set* VX^{m-n+1} *W is an m-complete test suite for A w.r.t. the reduction relation [25].*

The above proposition characterizes a subclass of observable FSMs for which an m-complete test suite w.r.t. the reduction relation can be derived in a systematic way similar to that for the class of deterministic FSMs. The cardinal difference lies in characterization sets and state identifiers. Separable states are not equivalent, however, nonequivalent states are not necessarily separable. As our preliminary results [3, 25] show, the approach based on concepts of separable and D-reachable states can be generalized to arbitrary observable machines, however, because of space limitation we will report on this generalization in a separate paper.

We also note that approximations of specifications constitute a special subclass of machines with distinct properties. In particular, the input alphabet comprises external and internal inputs; the former are required for external tests, the latter will be eventually dropped out. Outputs are not directly observed during test execution. Then certain tests can be functionally more powerful than the others in the sense that they induce fewer 'permissible' internal output sequences of the component under test. Such a redundancy of tests could be identified and removed. Another feature of approximation is the possible existence of the designated output **fail**. It is exists, it indicates what behavior an implementation should not exhibit to comply with the specification. As a result, the approximation may have certain states which do not correspond to any state in any conforming implementation. We conjecture that state identification should be accordingly adjusted. Several examples at hand, such as the machine in Fig. 12 show that a complete test suite can be reduced with no loss of its fault coverage. Detailed elaboration of heuristics for improving the test derivation technique based on specifics of approximation machines is our current research topic.

Finally, to complete our running example, we give an m-complete test suite for the approximation in Fig. 12. We assume that faults do not increase the number of states in the embedded component, i.e. $m = 2$. Following the approach of this section and applying heuristics mentioned above, we have obtained the following external test suite: $\{x_2x_1x_2x_1x_1;\ x_2x_1x_2x_2x_1;\ x_2x_2x_1x_1x_1x_1;\ x_2x_2x_2x_1\}$. The total length is 20. Note that the complete test suite with 192 test cases which total length is 1664 was derived for the same component and the same assumption on its faults, following a black-box testing approach.

5. Conclusion

A basic framework for testing in context has been given. The framework is based on the model of a system of communicating finite state machines. The problems of test executability and fault propagation in the presence of the context are identified and discussed within the presented framework. The proposed solution to these problems consists in computing a so-called approximation of the specification in context, i.e. the FSM model of the component's properties that can be controlled and observed through the context. The idea is to reduce testing in context to testing in isolation, so tests with a guaranteed fault coverage can be derived from the approximations in the form of tests for the reduction relation between FSMs.

As our discussion shows, the problem of testing in context has much in common with that of solving compositional equations. In fact, the problem of computing the most general solution E to the equation $E \circ C = Spec \circ C$ has been studied extensively. Solving such equations is a core problem in various areas: top-down design, decomposition, module validation, protocol derivation, hardware optimization, and others. In particular, the problem was addressed in terms of process algebras and the LTS model [11, 26–29]. In term of FSMs, it was explicitly or implicitly tackled in a number of papers [3, 19–22, 30–36]. The solution gives the weakest requirements imposed on components and corresponds to what is called 'the maximal set of permissible behaviors for an FSM' in the context of hardware optimization, however, here mainly synchronously communicating machines are considered. As we have demonstrated in this paper, the problem of testing in context is close to the above problem, however it cannot be reduced to it.

The framework presented in this paper also gives the most general solution to the above equation, however it does so in a different way by imposing an additional requirement on the solution to ensure executability of the tests needed to achieve complete fault coverage in implementations. It seems that by lifting this requirement the most general solution can also be obtained from our, so-called approximation. However, this requires further investigation.

Acknowledgements

This research was supported by the HP-NSERC-CITI Industrial Research Chair on Communication Protocols at the Université de Montréal, an NSERC strategic research grant, and by the Russian Found for Fundamental Research. The first author wants to thank Ed Brinksma, Pim Kars and Lex Heerink (University of Twente) for interesting discussions.

References

[1] A. Petrenko, G. Bochmann and R. Dssouli, Conformance relations and test derivation, IFIP Trans. Protocol Test Systems VI (Proc. IFIP TC6 Fifth Int. Workshop on Protocol Test Systems 1993), North-Holland, 1994, pp. 157–178.

[2] P.H. Starke, Abstract Automata, North-Holland/American Elsevier, 1972.

[3] A. Petrenko, N. Yevtushenko, A. Lebedev and A. Das, Nondeterministic state machines in protocol conformance testing, IFIP Trans. Protocol Test Systems VI (Proc. IFIP TC6 Fifth Int. Workshop on Protocol Test Systems 1993), North-Holland, 1994, pp. 363–378.

[4] S.H. Unger, Asynchronous Sequential Switching Circuits, Wiley-Interscience, 1969.

[5] G. v Bochmann and A. Petrenko, Protocol testing: review of methods and relevance for software testing, ISSTA'94 ACM Int. Symposium on Software Testing and Analysis, 1994, pp. 109–124.

[6] G. v Bochmann, A. Petrenko and M. Yao, Fault coverage of tests based on finite state models, Protocol Test Systems VII (Proc. IFIP WG 6.1 Int. Workshop on Protocol Test Systems, Chapman & Hall, 1995, pp. 55–78.

[7] A. Petrenko and G. v Bochmann, On fault coverage of tests for finite state specifications, Computer Networks & ISDN Systems, 1996.

[8] D.P. Sidhu and T.K. Leung, Formal methods for protocol testing: a detailed study, IEEE Trans. Softw. Eng., SE-15(4) (1989) 413–426.

[9] C.H. West, An automated technique of communication protocols validation, IEEE Trans. Comm., 26 (1978) 1271–1275.

[10] G. v Bochmann and C.A. Sunshine, Formal methods in communication protocol design, IEEE Trans. Comm., 28 (1980) 624–631.

[11] P. Merlin and G. v Bochmann, On the construction of submodule specifications and communications protocols, ACM Trans. Programming Languages and Systems, 5(1) (1983) 1–25.

[12] G. Luo, G. v Bochmann and A. Petrenko, Test selection based on communicating nondeterministic finite state machines using a generalized Wp-method, IEEE Trans. Softw. Eng., SE-20(2) (1994) 149–162.

[13] D. Brand and P. Zafiropulo, On communicating finite state machines, J. ACM, 30(2) (1983) 323–342.

[14] L. Heerink and E. Brinksma, Validation in context, Proc. 15th IFIP WG6.1 Int. Symposium on Protocol Specification, Testing, and Verification, Chapman & Hall, 1995, pp. 221–236.

[15] H. v Dam, H. Kloosterman and E. Kwast, Test derivation for standardized test methods, IFIP Trans. Protocol Test Systems IV (Proc. IFIP TC6 Fourth Int. Workshop on Protocol Test Systems 1991, 1992, pp. 69–82.

[16] ISO/IEC JTC1/SC21 WG7, Formal methods in conformance testing, working draft, Project 1.21.54. ISO, February 1995.

[17] M.P. Vasilevski, Failure diagnosis of automata, Cybernetics, Plenum, New York, No 4, 1973, pp. 653–665.

[18] T.S. Chow, Testing software design modeled by finite-state machines, IEEE Trans. Softw. Eng., SE-4(3) (1978) 178–187.

[19] J. Kim and M. Newborn, The simplification of sequential machines with input restrictions, IEEE Trans. Computers, C-21(12) (1972) 1440–1443.

[20] N. Yevtushenko and A. Matrosova, On one approach to automata networks checking sequences construction, Automatic Control and Computer Sciences, Allerton Press, New York, Vol 25 No 2, 1991, p. 3–7.

[21] A. Petrenko, N. Yevtushenko and R. Dssouli, Testing strategies for communicating fsms, Protocol Test Systems VII (Proc. IFIP WG 6.1 Int. Workshop on Protocol Test Systems 1994, Chapman & Hall, 1995, pp. 193–208.

[22] Y. Watanabe and R.K. Brayton, The maximal set of permissible behaviors for fsm networks, Proc. IEEE.ACM Int. Conf. on Computer-Aided Design, 1993, pp. 316–320.

[23] R. Dssouli and G. v Bochmann, Conformance testing with multiple observers, Proc. IFIP 6th Int. Symposium on Protocol Specification, Testing, and Verification, 1986.

[24] D. Lee, K. Sabnani, D. Krispot, S. Paul and M. Uyar, Conformance testing of protocols specified as communicating fsms, INFOCOM, 1993, pp. 115–127.

[25] A. Petrenko, N. Yevtushenko and G. v Bochmann, Experiments on

nondeterministic systems for the reduction relation, Technical Report 932, Université de Montréal, 1994.

[26] J. Parrow, Submodule construction as equation solving in ccs, Theoretical Computer Science, 68 (1989) 175–202.

[27] K.G. Larsen and L. Xinxin, Compositionality through an operational semantics in contexts, Lecture Notes in Computer Science, 443 (1990) 526–539.

[28] H. Qin and P. Lewis, Factorization of finite state machines under strong and observational equivalences, Formal Aspects of Computing (1991) 285–307.

[29] K. Drira, P. Azema, B. Soulas and A.M. Chemali, Testability of a communicating system through an environment, Proc. TAPSOFT, 1993, pp. 329–341.

[30] P. Das and D.E. Farmer, Fault-detection experiments for parallel-decomposable sequential machines, IEEE Trans. Computers, C-24(11) (1975) 1104–1109.

[31] S. Devadas, Approaches to multi-level sequential logic synthesis, Proc. 26th Design Automation Conf., Las Vegas, 1989, pp. 270–276.

[32] J.-K. Rho, G. Hachtel and F. Somenzi, Don't care sequences and the optimization of interacting finite state machines, Proc. IEEE Int. Conf. on Computer Aided Design, Santa Clara, 1991, pp. 414–421.

[33] A. Petrenko and N. Yevtushenko, Test suite generation for a fsm with a given type of implementation errors, IFIP Trans. Protocol Specification, Testing, and Verification XII (Proc. IFIP TC6 12th Int. Symposium on Protocol Specification, Testing, and Verification, 1992, pp. 229–243.

[34] H.Y. Wang and R.K. Brayton, Input don't care sequences in fsm networks, Proc. IEEE/ACM Int. Conf. Computer-Aided Design, 1993, pp. 321–328.

[35] Y. Watanabe and R.K. Brayton, State minimization of pseudo non-deterministic fsm's, Proc. Euro. Design and Test Conference, 1994, pp. 184–191.

[36] M. Damiani, Nondeterministic finite state machines and sequential don't cares, Proc. Euro. Design and Test Conf., 1994, 192–198.