# 10

# Fault models for testing in context*

*A. Petrenko* [1], *N. Yevtushenko* [2], *and G. v. Bochmann* [3]

1 - *CRIM, Centre de Recherche Informatique de Montréal,*
*1801 Avenue McGill College, Montréal (Québec), H3A 2N4, Canada,*
*Phone: (514) 398-3054, Fax: (514) 398-1244, petrenko@crim.ca*
2 *Tomsk State University, 36 Lenin str., Tomsk, 634050, Russia,*
*yevtushenko@elephot.tsu.tomsk.su*
3 - *Université de Montréal,*
*CP. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7, Canada,*
*(514) 343-7535, Fax: (514) 343-5834, bochmann@iro.umontreal.ca*

**Abstract**

We focus in this paper on the problem of modeling faults located in a given component embedded within a composite system. The system is represented as two communicating FSMs, a component FSM and a context machine that models the remaining part of the system which is assumed to be correctly implemented. We elaborate various fault models for testing in context. The existing FSM-based methods are assessed for their applicability to derive tests complete w.r.t. the fault models appropriate for testing in context.

**Keywords**

conformance testing, embedded testing, communicating FSMs, fault models

## 1 INTRODUCTION

There have been many research efforts on conformance test derivation for protocols based on the FSM model and the black-box representation of an implementation under test (IUT). In this testing scenario, the fault model consists of a reference machine, fault domain and conformance relation. The reference machine is a given specification. The fault domain represents a finite set of possible implementations that is the set of mutant (faulty) machines. The conformance relation determines what is a conforming (or non-conforming) implementation. A test suite is to be considered as complete, w.r.t. a given fault model, i.e. having complete fault coverage, if it can detect nonconformance of any implementation from the predefined fault domain to the reference machine (Bochmann, Petrenko, and Yao, 1994). Testing an FSM implementation in isolation is usually the classical FSM equivalence problem. It is required to determine by testing if an implementation and the reference FSMs are equivalent. The fault domain is usually defined by an upper bound on the number of states in all potential implementations, since no assumptions are usually made about the internal structure of an IUT (Gill, 1962), (Petrenko and Bochmann, 1996).

In practice, however, an IUT is often embedded within a complex system under test, see, e.g. the embedded test method defined for conformance testing (Rayner, 1987). In this paper,

* This research was done when the second author was with Université de Montréal

we model a composite specification by two communicating FSMs. One FSM, called a *component* machine, represents the behavior of a certain component embedded within the system, while the other machine, called a *context* (machine), models the remaining part of the system. The context may be viewed as a lumped, i.e. composed machine of all components of the system, except the component at hand. The context of the component serves as its operational or testing environment (Heerink and Brinksma, 1995), (Dam, Kloosterman, and Kwast, 1991), (Petrenko, Yevtushenko, and Dssouli, 1994), (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996). We are required to test the component machine via the context which itself is assumed to be correctly implemented. The problem of testing an FSM in context is more complicated than that in isolation. The final goal is to provide systematic methods for deriving tests with fault coverage guarantee for a component at hand. To reach this goal, fault models adequate to implementation errors in the embedded component should first be developed. In this paper, we elaborate various fault models for testing in context. Based on these models, complete test suites can be derived once an appropriate method is devised. The existing FSM-based methods are assessed for their applicability to derive tests complete w.r.t. the fault models discussed in this paper.

The rest of this paper is structured as follows. Section 2 contains several notions related to the FSM model. Section 3 discusses fault models used for local testing of a single component for later use within a context. Section 4 presents an explicit fault model for testing in context. In Section 5, we analyze different fault models based on the composed machine of a given system. Fault models based on the so-called approximation of the component are considered in Section 6. In Section 7, we give a simple comparison of the proposed models.

## 2 PRELIMINARIES

### 2.1 Finite state machines

A finite state machine (FSM), often simply called a machine throughout this paper, is a completely specified initialized (possibly nondeterministic) Mealy machine which can be formally defined as follows. A *finite state machine* $A$ is a 5-tuple $(S, X, Y, h, s_0)$, where $S$ is a set of states with $s_0$ as the initial state; $X$ - a finite set of input symbols; $Y$ - a finite set of output symbols; and $h$ - a behavior function $h: S\times X \rightarrow \mathbb{P}(S\times Y)$, where $\mathbb{P}(S\times Y)$ is the powerset of $S\times Y$ (Starke, 1972). The machine $A$ becomes deterministic when $|h(s,x)|=1$ for all $(s,x)\in S\times X$. In a deterministic FSM, instead of the behavior function which is required for expressing a nondeterministic behavior, we use two functions: the next state function $\delta$, and the output function $\lambda$.

We extend the behavior function to a function on the set $X^*$ of all input sequences containing the empty sequence $\varepsilon$, i.e., $h: S\times X^* \rightarrow \mathbb{P}(S\times Y^*)$. For convenience we use the same notation $h$ for the extended function, as well. 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 of $A$, $h^1$ is the first and $h^2$ is the second projection of $h$, i.e., $h^1(s,\alpha) = \{ s' \mid \exists\ \beta \in Y^*\ [(s',\beta) \in h(s,\alpha)] \}$, $h^2(s,\alpha) = \{\beta \mid \exists\ s' \in S\ [(s',\beta) \in h(s,\alpha)] \}$ for all $\alpha\in X^*$.

Given two states $s$ of the FSM $A$ and $r$ of the FSM $B=(T, X, Y, H, t_0)$, and a set $V\subseteq X^*$; state $r$ is said to be a *V-reduction* of $s$, written $r\leq_V s$, if for all input sequences $\alpha \in V$ the condition $H^2(r,\alpha) \subseteq h^2(s,\alpha)$ holds; $r$ is not a reduction of $s$, $r\not\leq_V s$, if there exists an input sequence $\alpha \in V$ such that $H^2(r,\alpha) \subsetneq h^2(s,\alpha)$. States $s$, and $r$ are *V-equivalent* states, written $s\equiv_V r$, iff $s\leq_V r$ and $r\leq_V s$. On the class of deterministic machines, the above relations coincide.

We denote $\leq$ the $V$-reduction in the case where $V=X^*$, similarly, $\equiv$ denotes the equivalence relation.

Given two machines, $A$ and $B$, $B$ is a reduction of $A$, written $B \leq A$, if the initial state of $B$ is a reduction of the initial state of $A$. If $B \leq A$ and $B$ is deterministic then it is referred to as a $D$-reduction of $A$. We denote the set of all $D$-reductions of a given FSM $A$ with at most $m$ states as $D\text{-}red_m(A)$.

Similarly, the equivalence relation between machines is defined. $B \equiv A$, iff $B \leq A$ and $A \leq B$. The equivalence relation between FSMs is sometimes called trace equivalence. The traces of a machine are I/O sequences defined for its initial state. Equivalent machines exhibit identical behaviors, i.e. they execute the same traces. In this context, the reduction can be viewed as a trace preorder.

Given an input alphabet $X$ and output alphabet $Y$, there exists a special nondeterministic FSM such that any machine, deterministic or not, is its reduction. In particular, consider a *chaos* machine $Ch(X,Y) = (\{p\}, X, Y, H, p)$, where $H(p,x) = \{p\} \times Y$ for all $x \in X$. Clearly, $A \leq Ch(X, Y)$ for all $A$ over the alphabets $X$ and $Y$.

An NFSM $B = (S', X, Y, h', s_0)$ is said to be a *submachine* of the NFSM $A = (S, X, Y, h, s_0)$ if $S' \subseteq S$ and $h'(s,x) \subseteq h(s,x)$ for all $(s,x) \in S' \times X$. Obviously, all submachines of $A$ are reductions of $A$, but the converse is not true. If a submachine of $A$ is deterministic then it is said to be a *D-submachine* of $A$. We denote *D-sub*$(A)$ the set of all $D$-submachines of $A$.

Given an input alphabet $X$, an output alphabet $Y$, and the number of states $m$, there exists a special nondeterministic FSM such that any machine, deterministic or not, with up to $m$ states defined over X and Y, is isomorphic to one of its submachines. In particular, consider another *chaos* machine $Ch_m(X,Y) = (P, X, Y, H, p_0)$, where $|P|=m$, $H(p,x) = P \times Y$ for all $(p,x) \in P \times X$. Clearly, any machine $A$ over the alphabets $X$ and $Y$ is isomorphic to a submachine of $Ch_m(X,Y)$ provided that the number of states of $A$ does not exceed $m$. The two types of chaos machines $Ch(X,Y)$ and $Ch_m(X,Y)$ are equivalent, both have the same set of reductions; however, all the submachines of the machine $Ch(X,Y)$ have only one state and submachines of $Ch_m(X,Y)$ may have up to $m$ states. We will be using both chaos machines to compactly represent either sets of reductions or sets of submachines whenever convenient.

### 2.2 Fault models

The equivalence and reduction relations serve as conformance relations between implementations and their FSM specifications for deriving test suites with guaranteed fault coverage.

We define a fault model as a triple $<A, \sim, \mathfrak{I}>$, where $A$ is a (reference) specification, a finite set $\mathfrak{I}$ is the fault domain that is a set of possible implementations (mutant machines) defined over the same input alphabet as the specification, and $\sim$ is a conformance relation, $\sim \in \{\equiv, \leq\}$.

A test suite w.r.t. a given fault model is a finite set $E$ of finite input sequences of the reference machine $A$. A test suite $E$ is said to be *complete* w.r.t. the fault model iff for all $B \in \mathfrak{I}$, $B \not\sim A$ implies $B \not\sim_E A$.

Let $\mathfrak{I}_m(X,Y)$ denote the universal set of all deterministic FSMs over input and output alphabets $X$ and $Y$ with at most $m$ states. The fault model $<A, \equiv, \mathfrak{I}_m(X,Y)>$ is a classical black-box fault model for testing deterministic FSM implementations in isolation (Gill, 1962). It reflects a black-box representation of an IUT and is used in a number of methods for deriving test suites that provide complete fault coverage, for example, (Vasilevski, 1973), (Chow, 1978), (Fujiwara et al, 1991).

### 2.3 The model of a system with an embedded component

Many compound systems are typically specified as a collection of communicating components. Assuming that the behavior of each component of a given system is known and can be described by an FSM, we can use a system of communicating machines as the model of the given system. The composition of two communicating FSMs, connected as shown in Figure 1a, is general

enough to discuss problems related to testing an embedded component. Further we assume for the sake of simplicity that the sets $X$, $U$, $Z$, and $Y$ of actions are pairwise disjoint.

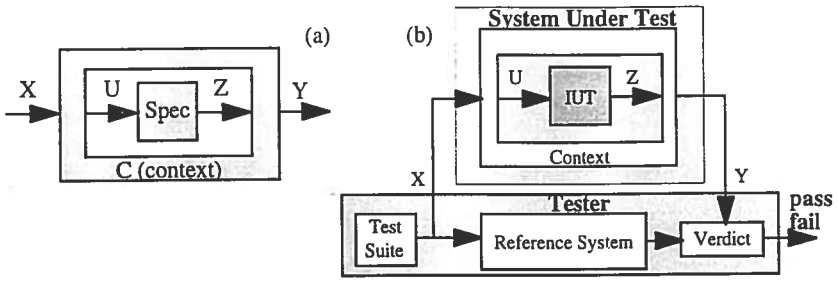


**Figure 1** Composition of two communicating FSMs (a) and test architecture (b).

One FSM, called a *component* machine *Spec*, represents the behavior of a certain component embedded within the system, while the other machine, called a *context* (machine) *C*, models the remaining part of the system. The context may be viewed as a lumped, i.e. composed machine of all components of the system, except the component *Spec*.

Two FSMs are communicating asynchronously via bounded input queues where actions are stored. Bounded queues are usually assumed in order to obtain a finite state model of the global system. In addition, we impose an I/O *ordering constraint* on a manner the environment interacts with the system. Specifically, a next external input $x$ is only submitted to the system after it has produced an external output $y$ in response to the previous input. In other words, the system at hand has a single message in transit. Under these assumptions, the number of global states of the system is finite. A global state consists of states of input queues and states of the component and context machines and can be represented in the form of a 2×2 matrix, where the first row contains states of each input queue and the second row contains current states of the two 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. Note that the number of stable states in the system is bounded by the product of the numbers of states of the two machines, whereas the number of the transient states may exceed that of stable states by the factor of the total number of internal and external inputs.

The collective behavior of the system of two communicating FSMs can be described by means of a *product machine* and *composed machine*. The former describes the behavior of all component machines in terms of all actions within the system, whereas the latter describes the observed behavior in terms of external inputs and outputs.

The product machine *Spec*×*C* is customary represented by a graph of global states, obtained by performing reachability computation (West, 1978), (Bochmann and Sunshine, 1980), (Merlin and Bochmann, 1983), (Brand and Zafiropulo,1983), (Luo, Bochmann, and Petrenko, 1994). It 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 *Spec*×*C*, a composed machine *Spec*∘*C* can be obtained. Here '∘' is a hiding operation on all internal actions in the product machine. However, *Spec*∘*C* becomes an FSM under certain assumptions only, viz. the product machine should have no livelocks. Hiding internal actions usually amounts to determinizing the LTS (an automaton), and coupling external inputs and external outputs into labels of transitions between stable states.

The feedback composition of FSMs shown in Figure 1a generalizes the cascaded composition of two machines considered in the previous work (Petrenko, Yevtushenko and Dssouli, 1994).

In particular, the context machine for the serial composition of two FSMs is transparent either to actions $X$ (the component is the head machine) or to actions $Z$ (the component is the tail machine).

**Example**. Consider an example system of deterministic context and component machines, shown in Figure 2. The initial states are $a$ and 1. The composed machine $Spec \circ C$ obtained from the product machine (not shown here) has initially five states (Figure 3a), among which states ($a2$) and ($a3$) are equivalent states. Merging these two states, we obtain the reduced form of the composed machine (Figure 3b).

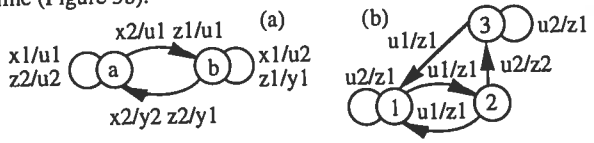


**Figure 2** The context $C$ (a) and component *Spec* (b) machines.

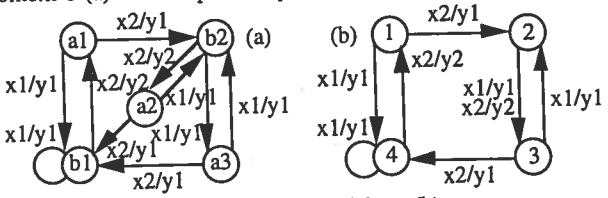


**Figure 3** The composed machine (a) and its reduced form (b).

We will use this system to illustrate our discussion on testing an embedded component.

## 3 LOCAL TESTING OF A COMPONENT MACHINE

In a number of situations, it is possible to assume that, for testing purposes, the component of interest can be tested locally, before being integrated with the context or, equivalently, by taking it out of a composite system. Such a local testing of implementations of a given component machine reduces to the classical black-box testing in isolation. In this paper, we assume that no fault in the embedded component can increase the number of states compared with its specification. A number of test derivation methods can directly be applied to local testing. In this section, we expose some hidden problems of local testing of the component which is intended to be used within the given context.

### 3.1 Testing for equivalence to the specification

Given a specification machine *Spec* and the set $\mathcal{I}_n(U,Z)$ of its possible implementation machines (the fault domain), the equivalence relation is a natural candidate for the conformance relation. Hereinafter $n$ is the number of states of *Spec*. We have thus the classical black-box fault model $<Spec, \equiv, \mathcal{I}_n(U,Z)>$, as in Section 2.2. Based on this fault model, a complete test suite can be derived by employing any known method.

**Example.** The machine *Spec* (Figure 2b) can completely be tested in isolation with seven test cases of a total length of 35, obtained by applying the Wp-method (Fujiwara et al, 1991). The test suite is complete w.r.t. the fault model $<Spec, \equiv, \mathcal{I}_3(U,Z)>$. It detects any machine with at most three states that is not equivalent to the FSM *Spec*.

When we derive a test suite complete w.r.t. the fault model $<Spec, \equiv, \mathcal{I}_n(U,Z)>$, we ignore the context which may not require that all the facets of the specification are implemented exactly as described. Intuitively, this is the case when certain features of the implemented component cannot be exercised at all (there is no need to test them) or the context is tolerable to certain faults. The conclusion is that the equivalence relation is too strong for the embedded machine

and there may exist a shorter complete test suite than that derived w.r.t. the black-box fault model $<Spec, \equiv, \mathfrak{I}_n(U,Z)>$. We use our example system shown in Figure 2 to illustrate the redundancy of tests based the above fault model.

**Example.** Consider the machine *Imp* in Figure 4a. The FSMs *Spec* and *Imp* are not equivalent. However, the machine $Imp \circ C$ is equivalent to $Spec \circ C$ (Figure 3), that is, embedded in the context, *Imp* provides the same externally visible behavior as *Spec*. One may view the FSM *Imp* as a mutant of *Spec* representing certain faults that are tolerated by the context machine. From this viewpoint, any complete test suite for the *Spec* in isolation may be redundant. A test case distinguishing such an *Imp* from the *Spec* can be deleted from any given test suite.

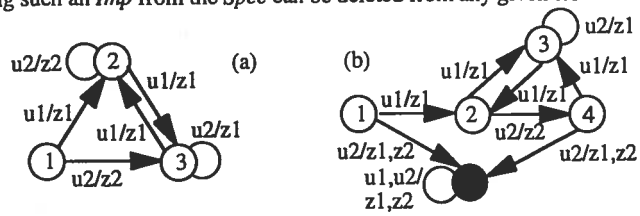


**Figure 4** A machine *Imp* non-equivalent to *Spec* (a) and the FSM *G* (b).

### 3.2 Testing for equivalence in context

To avoid this redundancy of tests we can refine the fault model by relaxing the conformance relation. The idea is to treat as conforming any implementation machine that combined with the given context leads to a composed machine equivalent to $Spec \circ C$.

We say that the *Imp* is a *conforming* implementation in a given context if and only if

(1) the composed machine $Imp \circ C$ exists and

(2) the two composed machines $Imp \circ C$ and $Spec \circ C$ are equivalent (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996).

Assume (1) holds, then the *equivalence in context*, written $Imp =_C Spec$, is defined as follows:

$Imp =_C Spec$ iff $Imp \circ C \equiv Spec \circ C$.

We write $Imp \neq_C Spec$ if *Imp* and *Spec* are not equivalent in the context *C*. Obviously, equivalent FSMs are also equivalent in any context, that is $Imp \equiv Spec$ implies $Imp =_C Spec$ for any context *C*, but the converse is not true.

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

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

In fact, all machines equivalent to *Spec* in context can be captured by a single machine. Specifically, we consider the equation $G \circ C = Spec \circ C$ with *G* being a free variable. The general solution to the equation is a nondeterministic FSM *G* (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996). Any deterministic reduction of the FSM *G* that together with the context gives the composed machine is equivalent to *Spec* in context, i.e. it is a conforming implementation machine.

This leads us to the following fault model. The reference machine is the machine *G*, the fault domain is the set $\mathfrak{I}_n(U,Z)$ and the conformance relation is the reduction relation. The fault model $<G, \leq, \mathfrak{I}_n(U,Z)>$ is, however, not accurate. There may exist a deterministic reduction of the machine *G* such that the corresponding product machine exhibits livelocks, i.e. no composed machine exists.

Assume that the fault domain $\mathfrak{I}_n(U,Z)$ has no machine of this kind. This is usually the case, for example, for compositions without feedback. Under this assumption, local testing provides complete fault coverage if tests complete w.r.t. the fault model $<G, \leq, \mathfrak{I}_n(U,Z)>$ are used. There exists a systematic method for deriving tests complete w.r.t. this fault model (Petrenko,

Yevtushenko, and Bochmann, 1996). A test suite complete w.r.t. $<G, \leq, \mathfrak{I}_n(U,Z)>$ avoids the redundancy of tests complete w.r.t. $<Spec, \equiv, \mathfrak{I}_n(U,Z)>$ by excluding test cases used in the latter to detect FSMs that are not equivalent to *Spec* in isolation, but equivalent in context.

**Example.** For our working example the FSM $G$ representing the general solution to the equation is shown in Figure 4b. Here a black hole corresponds to a so-called trap state that produces all the outputs for all the inputs (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996). Intuitively, it indicates that any behavior of the component trapped to this state is conforming in context since it cannot be exercised. To derive a complete test suite w.r.t. the fault model $<G, \leq, \mathfrak{I}_3(U,Z)>$, we follow the approach (Petrenko, Yevtushenko, and Bochmann, 1996). The resulting tests contain five test cases with the total length of 28. Recall that the test suite complete w.r.t. the fault model $<Spec, \equiv, \mathfrak{I}_3(U,Z)>$ obtained in Section 3.1 has 35 test events. As expected, a less stringent conformance relation requires a shorter complete test suite.

In the case where the fault domain includes an implementation machine that falls into livelocks with the given context, local testing based on the above outlined approach is insufficient. To ensure the correct external system behavior, integration testing has yet to be performed. Another possible solution to the problem is to identify the implemented reduction of the FSM $G$ and to check whether or not the identified machine may have livelocks when it is combined with the given context.

### 3.3 Using local tests for testing in context

As discussed in the previous section, tests derived for local testing can ensure complete coverage of faults in the embedded component. The above tests are expressed in terms of the actions of internal inputs $U$ and outputs $Z$. We call them internal tests. A natural question arises on whether internal tests can be reused for testing in context, i.e. in the case where local testing itself is unfeasible. We have no access to the internal interface, so we need external tests, i.e. tests expressed in terms of external actions $X$ and $Y$. An internal test should be translated into an appropriate external test such that the internal test is applied to the component under test and moreover the effect of an internally detected fault is propagated through the context. The general answer to the question is negative.

**Example.** Consider the FSM $G$ of Figure 4b. Assume that the input $u_1$ is applied to the initial state 1. Any conforming implementation should produce the output $z_1$. Assume, however, that there is a fault, and an implementation produces the internal output $z_2$ instead of $z_1$. The context (Figure 2a) in the state $b$ outputs just $y_1$, no matter which internal signal comes from the component. The fault is latent, though. The context moves to a wrong state $a$ and in response to the next external input $x_2$ it eventually propagates an error on the external output. Such a latent fault is detected internally by a simple internal test case $u_1$, but can be externally observed when a longer external test case $x_2x_2$ is applied.

As this example shows, translation of test cases does not necessarily preserve the length of tests. Moreover, different implementations may require distinct external test cases to excite the same internal test, since the same external input sequence causes various sequences of internal actions for different implementation machines in the presence of feedback signals, unlike serial compositions of FSMs (Petrenko, Yevtushenko, and Dssouli, 1994). There is even no guarantee that the translation problem is decidable at all. It may well happen that no external test can excite a required internal test for an implementation in context.

The test translation problem is thus obscure and local tests are not easy to reuse for testing in context. There is a need for fault models tuned for testing in context.

## 4 EXPLICIT FAULT MODEL FOR TESTING IN CONTEXT

Testing in context is based on the test architecture shown in Figure 1b. It can be considered as a detailed view of the one described in (ISO, 1995). An IUT is the component that needs to be tested for conformance to its specification through the context. Further we assume that a given

composite specification consists of deterministic components, moreover their implementations are deterministic, as well. The verdict machine 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. We also require that the verdict machine produces the verdict **fail** when the system under test falls into livelock. A timeout mechanism can be used to detect such behavior (tests related to timed behavior are not considered here).

Based on the test architecture (Figure 1b), we now can define what constitutes a fault of the embedded component, i.e. a fault model for deriving complete test suites. Since our tester is equipped with a proper timer for detecting livelocks, we further assume that $\mathfrak{I}_n(U,Z)$ denotes the set of machines with at most $n$ states such that $Imp \circ C$ exists. Given two FSMs *Spec* and $C$, the composed machine $Spec \circ C$ is the reference system *RS*. Similarly, the composed machine $Imp \circ C = IS$ 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. Producing the verdict **fail**, the verdict machine indicates that the two composed machines *IS* and *RS* are not equivalent.

Similar to the case of an isolated FSM, we can use the fault model consisting of the reference machine *RS*, the equivalence relation as conformance relation and the fault domain $\{Imp \circ C \mid Imp \in \mathfrak{I}_n(U,Z)\} = \mathfrak{I}_n(U,Z) \circ C$, in other words, the triple $<RS, \cong, \mathfrak{I}_n(U,Z) \circ C>$. We call it the *explicit* fault model since all possible implemented systems are explicitly included within the fault domain $\mathfrak{I}_n(U,Z) \circ C$.

With respect to the fault domain $\mathfrak{I}_n(U,Z) \circ C$, there are two possible cases. In the first case, a given context is dummy and the fault domain $\mathfrak{I}_n(U,Z) \circ C$ is the universal set of possible deterministic FSMs over $X$ and $Y$ within a certain number of states. Testing in such a context is nothing more than black-box-based testing in isolation. In the second case, a given context is not trivial and the fault domain may be a proper subset of the universal set. We need to derive a test suite complete w.r.t. the fault model $<RS, \cong, \mathfrak{I}_n(U,Z) \circ C>$, however, we are unaware of any systematic method except for a brute force search. Nevertheless, it is worth to outline such a straightforward approach.

Assume that we could enumerate all machines in the set $\mathfrak{I}_n(U,Z)$ while not taking into account isomorphic machines. For each FSM we construct a composed machine. Comparing the result with the FSM $RS = Spec \circ C$, we can derive at least one input sequence that distinguishes them whenever they are not equivalent. The union of distinguishing sequences for all machines in $\mathfrak{I}_n(U,Z) \circ C$ gives a desired test suite. We may try to minimize it, since a single sequence can distinguish several composed machines from the FSM *RS*. The last problem reduces to a classical set cover problem (Johnson, 1974).

Because of its complexity, such a solution is feasible only for a small number of faults to be detected by testing in context, for example for single output faults of the embedded component. All the machines of the set $\mathfrak{I}_n(U,Z) \circ C$ are simply not possible to explicitly construct in a realistic situation. Since we have no other method for deriving tests complete w.r.t. the above fault model, we shall look for another fault models.

## 5 FAULT MODELS BASED ON THE COMPOSED MACHINE

### 5.1 Classical black-box fault model

The straightforward approach to the problem of deriving a test suite for an component embedded within a system, as discussed above, faces a potential explosion of the number of possible mutant composed machines. An attempt to avoid the necessity of enumerating all these machines could be made based on the fact that they constitute a subset of all machines over the alphabets $X$ and $Y$ with a certain number of states. An upper bound on the number of states in any composed

machine can be, in fact, established as follows. The product of the number of states of *Spec* and the number of states in the context gives the upper bound $m$ on the number of states in any composed machine. The fault domain becomes $\mathfrak{I}_m(X,Y) = D\text{-}sub(Ch_m(X,Y))$. Now any complete test suite w.r.t. the fault model $<RS, \cong, D\text{-}sub(Ch_m(X,Y))>$ is also complete w.r.t. the explicit fault model. The reason is that the fault domain $\mathfrak{I}_n(U,Z)\circ C$ is a subset of $\mathfrak{I}_m(X,Y)$. The latter fault model is used in a number of existing test derivation methods for an FSM in isolation, so test derivation is feasible. Such an approach has, however, several drawbacks.

First, the bound $m$ may not be tight, in other words, the question is whether there exists a faulty implementation machine *Imp* such that the composed machine $Imp\circ C$ has exactly $m$ states in its reduced form. It is well-known that the size of a test suite required to test a machine with $m$ states with respect to a specification machine with $k$ states ($k \leq m$) grows exponentially as $(m\text{-}k)$ increases (Vasilevski, 1973), (Chow, 1978), (Fujiwara et al, 1991). Any overestimation of the increase in the number of states due to faults results in a huge redundancy in the obtained test suite. We are unaware of any systematic method that can establish the least upper bound for the problem.

Second, the set of all machines with up to $m$ states includes machines that do not correspond to any system with the given context. We call such machines *unfeasible* for a given context. The context is a fault-free machine in any implemented system by our assumption. A complete test suite w.r.t. the above mentioned fault model even for a tight bound $m$, is redundant.

**Example.** To illustrate the results of treating the system as a black-box, we derive a complete test suite for the composed machine (Figure 3b) with four states w.r.t. all FSMs with at most six states ($m$=6). The Wp-method (Fujiwara et al, 1991) delivers a test suite containing 41 test cases of total length 294. Completeness of the obtained test suite means that it detects non-equivalence of any out of $(6\times2)^{(6\times2)} = 12^{12}$ possible machines with two inputs, two outputs and up to six states. These are exactly all submachines of the chaos machine $Ch_6(X,Y)$.

To decrease the size of a test suite required to test the component in context, unfeasible machines should somehow be excluded from the fault domain.

### 5.2 Deleting unfeasible machines

It is in fact possible to exclude at least some of the unfeasible machines from the fault domain represented by all $D$-submachines of the chaos machine $Ch_m(X,Y)$. The idea is to combine a given context machine $C$ with a compressed representation of all possible implementations of a component machine *Spec*. As mentioned in Section 2, the chaos machine $Ch(U,Z)$ contains all machines of the set $\mathfrak{I}_n(U,Z)$ as its $D$-reductions. Intuitively, this chaos machine can be seen as the loosest description of the behavior of an embedded component. Using such a compressed representation of the set $\mathfrak{I}_n(U,Z)$ of deterministic machines, we can obtain a composed machine that describes the external behaviors of all possible implementation machines with the given context. Unlike the straightforward method of Section 4, the necessity of processing deterministic machines one by one is now avoided. Specifically, we construct the product machine $Ch(U,Z)\circ C$ in a usual way. As discussed before, constructing the composed machine from the product machine we can neglect livelocks, as they are detected by the tester equipped with a timer. The resulting composed machine $Ch(U,Z)\circ C$ becomes nondeterministic and any feasible composed machine is its reduction. We use our working example to illustrate the idea of constructing such a nondeterministic composed machine.

**Example.** The chaos machine $Ch(U,Z)$ has a single state as shown in Figure 5a. The context machine (Figure 2a) has two states, the composed machine $Ch(U,Z)\circ C$ shown in Figure 5b has two states, as well.

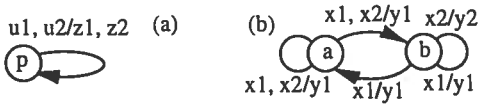


**Figure 5** The chaos machine $Ch(U, Z)$ (a) and the composed machine $Ch(U,Z)\circ C$ (b).

Intuitively, the machine $Ch(U,Z)\circ C$ describes the external behavior of any implemented system of the component machine *Spec* with the given context $C$, no matter how many states a component *Imp* has. The reference system $RS = Spec\circ C$ is a $D$-reduction of $Ch(U,Z)\circ C$. Speaking more formally, we have the following fact.

**Proposition 5.1.** For any implementation machine $Imp \in \mathfrak{I}_n(U,Z)$, the composed machine $Imp\circ C$ is a $D$-reduction of $Ch(U,Z)\circ C$.

Among all $D$-reductions of $Ch(U,Z)\circ C$ there are machines with more than $m$ states, where $m$ is the upper bound on the number of states in any implemented system. In our example, $m$=6. The chaos machine $Ch_6(X,Y)$ used in the previous section contains $12^{12}$ $D$-submachines. Now we estimate the number of $D$-reductions of $Ch(U,Z)\circ C$ (Figure 5b) with up to six states. Any $D$-reduction of this machine may have an outgoing transition to any of six states, for any particular state and any input, provided that an output equals to that of $Ch(U,Z)\circ C$. The total number of such machines is $6^{12}$. We conclude that in our example, approximately $12^{12} - 6^{12}$ unfeasible machines are removed when the fault domain is represented by the set $D\text{-}red_6(Ch(U,Z)\circ C)$ instead of $D\text{-}sub(Ch_6(X,Y))$.

We define the following fault model $\langle RS, \cong, D\text{-}red_m(Ch(U,Z)\circ C)\rangle$.

**Proposition 5.2.** A complete test suite w.r.t. the fault model $\langle RS, \cong, D\text{-}red_m(Ch(U,Z)\circ C)\rangle$ is a complete test suite w.r.t. the explicit fault model $\langle RS, \cong, \mathfrak{I}_n(U,Z)\circ C\rangle$.

In an extreme case, the two machines $Ch(U,Z)\circ C$ and $Ch_n(X,Y)$ become equivalent. Intuitively, it means that the context is dummy. In cases where $Ch(U,Z)\circ C$ and $Ch(X,Y)$ are not equivalent, the machine $Ch(U,Z)\circ C$ is 'less nondeterministic' and thus more precisely represents the fault domain than $Ch_m(X,Y)$.

The question arises now on how a test suite satisfying Proposition 5.2 can be constructed. This is an open problem, since in general, not much is yet known about complete test suites for nondeterministic machines. The solution of this problem may have various applications, however, in the context of this study, we refrain from further elaborating this approach because of the following reason.

Take our example system, there exist $(3\times2)^{(3\times2)} = 6^6$ possible component machines with two inputs, two outputs and up to three states. At the same time, the set $D\text{-}red_6(Ch(U,Z)\circ C)$ has $6^{12}$ machines. This means that the set $D\text{-}red_6(Ch(U,Z)\circ C)$ is not the best possible approximation of the fault domain, since it contains, along with the interesting machines, as much as $6^{12} - 6^6$ unfeasible FSMs that are superfluous and should be further excluded from the fault domain.

### 5.3 Further refinement of the fault domain

It is known that the set of $D$-submachines of an FSM often is a proper subset of the set of its D-reductions with the same number of states (Petrenko, Yevtushrenko, Lebedev, and Das, 1993). It is therefore worth to use $D$-submachines instead of $D$-reductions. The idea is then to replace the chaos machine with a single state by another chaos machine with several states. It is known that every FSM of $\mathfrak{I}_n(U,Z)$ is isomorphic to a $D$-submachine of the chaos machine $Ch_n(U,Z)$.

Similar to the approach of the previous section, we combine the nondeterministic FSM $Ch_n(U,Z)$ and the deterministic context machine $C$ into the composed machine $Ch_n(U,Z)\circ C$.

**Example.** The chaos machine has three states, as the component machine *Spec* (Figure 2b). The composed machine should have six states, as the context has two states. Table 1 gives the state table of $Ch_n(U,Z)\circ C$. Bold symbols in the table label transitions of the composed machine *RS* shown in Figure 3a. Faults cannot alter outputs, but can change the tail state of nine transitions, as shown in this table.

Intuitively, the machine $Ch_n(U,Z)\circ C$ describes all possible deviations in the behavior of the composed machine due to potential faults in any implementation of the machine *Spec* with the given context $C$. Speaking more formally, we have the following fact.

**Table 1** The nondeterministic composed machine $Ch_n(U,Z)\circ C$.

| input | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $x_1$ | **3/$y_1$** 1,2,4,5,6 | **5/$y_1$** 1,2,3,4,6 | **3/$y_1$** 1,2,4,5,6 | **2/$y_1$** 1,3,4,5,6 | **2/$y_1$** 1,3,4,5,6 | **6/$y_1$** 1,2,3,4,5 |
| $x_2$ | **2/$y_1$** 1,3,4,5,6 | **4/$y_2$** | **1/$y_2$** | **3/$y_1$** 1,2,4,5,6 | **3/$y_1$** 1,2,4,5,6 | **5/$y_2$** |

**Proposition 5.3.** For any implementation machine $Imp \in \mathfrak{I}_n(U,Z)$, the composed machine $Imp\circ C$ is isomorphic to a $D$-submachine of $Ch_n(U,Z)\circ C$.

The machines $Ch(U,Z)\circ C$ and $Ch_n(U,Z)\circ C$ are equivalent but have different sets of submachines. If the set of $D$-submachines of $Ch_n(U,Z)\circ C$ is a proper subset of the set of its $D$-reductions up to $m$ states then the set $D\text{-}sub(Ch_n(U,Z)\circ C)$ contains fewer unfeasible machines than the set $D\text{-}red_m(Ch(U,Z)\circ C$.

**Example.** The FSM $Ch_3(U,Z)\circ C$ (Table 1) has $6^9$ $D$-submachines, since there are nine cells in the state table each of which contains six different transitions. Recall that the fault domain based on the machine $Ch(U,Z)\circ C$ contains $6^{12}$ machines. Thus, approximately, $6^{12} - 6^9$ machines are further excluded as unfeasible.

Based on the obtained fault domain, the fault model can now be defined as $<RS, \equiv, D\text{-}sub(Ch_n(U,Z)\circ C)>$.

**Proposition 5.4.** A complete test suite w.r.t. the fault model $<RS, \equiv, D\text{-}sub(Ch_n(U,Z)\circ C)>$ is a complete test suite w.r.t. the explicit fault model $<RS, \equiv, \mathfrak{I}_n(U,Z)\circ C>$.

To derive a complete test suite w.r.t. this fault model we can now apply the FF-method (fault-function-based method) developed in (Petrenko and Yevtushenko, 1992), since the nondeterministic machine $Ch_n(U,Z)\circ C$ containing the reference machine $RS$ can be interpreted as a fault function of the FSM $RS$. The method is proven to deliver a complete test suite.

**Example.** We use the FF-method to derive a complete test suite for the composed machine $RS$ (Figure 3) and the fault function in Table 1. The resulting complete test suite w.r.t. the fault model with the fault domain $D\text{-}sub(Ch_3(U,Z)\circ C)$ has 32 test cases of the total length 235. Recall that the test suite complete w.r.t. the fault model considered in Section 5.1 has 294 test events. A more accurate description of the fault domain decreases the length of a necessary test suite.

Unfortunately, the machine $Ch_n(U,Z)\circ C$ may still contain some $D$-submachines that do not correspond to any composed machine with a faulty component machine. In our example, the set $\mathfrak{I}_3(U,Z)$ contains $6^6$ possible machines. It indicates that in the worst-case situation, the machine $Ch_n(U,Z)\circ C$ has $6^9 - 6^6$ unfeasible $D$-submachines. A complete test suite for testing in context may yet be redundant.

### 5.4 Discussion

Summarizing this section, we note that based on the composed machine $RS = Spec\circ C$, four different fault models can be devised. All of them have the same reference machine, the same conformance relation and only differ in their fault domains. Specifically, we have

$$D\text{-}sub(Ch_m(X,Y)) \supseteq D\text{-}red_m(Ch(U,Z)\circ C) \supseteq D\text{-}sub(Ch_n(U,Z)\circ C) \supseteq \mathfrak{I}_n(U,Z)\circ C.$$

In this case, we say that one fault model refines the other if strong inclusion between their fault domains holds. For a non-trivial context, the fault model based on $D\text{-}sub(Ch_n(U,Z)\circ C)$ is the best approximation of the explicit fault model among the three considered so far. Moreover, it seems to be nearly impossible to exclude all the unfeasible machines from any fault domain with a reference machine defined only over external input and output alphabets.

## 6 FAULT MODELS BASED ON THE APPROXIMATION OF THE COMPONENT

We wish to come up with a fault model such that the reference specification represents component's properties controlled and observed through the context and the fault domain is the set $\mathfrak{I}_n(U,Z)$ of all possible implementation machines of the component *Spec* augmented by an external inputs and an output *null*. In other words, we make the additional assumption that the FSM *Spec* as well as all its possible implementation machines ignores all external inputs $x$ by producing the *null* output while maintaining its current state. This is similar to a particular completeness assumption widely used in the context of protocol conformance testing. We refer to an *Imp* augmented in this way as to an $X$-augmentation of *Imp* and denote it as $Imp^a$. The set of all augmented machines of $\mathfrak{I}_n(U,Z)$ is denoted by $\mathfrak{I}_n^a(U,Z)$. The idea of determining a corresponding reference specification is to find the loosest behavior of any embedded component keeping the information on the correctness of external outputs produced by a given context in response to every external input sequence. Depending on the current stable state and external input $x$, the context and a component at hand may involve in various interactions generating an internal I/O sequence before an external output is produced. All sequences which can be produced by any *Imp* conforming to *Spec* w.r.t. $x$ at the current stable state result in the same expected external output $y$. Once internal sequences leading to a wrong external output are discarded, it is possible to exclude actions $Y$ from our description. The loosest description of the conforming behavior of a component in context w.r.t. all possible external input sequences is called the *approximation* of the specification machine *Spec* in context $C$ (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996). It is in fact, a nondeterministic machine defined over the input alphabet $X \cup U$, the output alphabet $Z \cup \{null, fail\}$. We use $[[Spec]]_C$ to denote the approximation of the specification.

### 6.1 Construction of approximation of the embedded component in context

The method for constructing the approximation is based on the test architecture shown in Figure 1b. Assume that an IUT is replaced by a chaos machine that does whatever any feasible implementation can. The first verdict **fail** produced by the verdict machine *Ver* in response to a given external input sequence simultaneously applied to the system under test containing the chaos machine and to the reference system indicates that a certain output produced by the chaos machine is an error. The current external input $x$ is paired with the *null* output when there exists at least one machine *Imp* externally equivalent to *Spec* with respect to the accepted input sequence. An internal input $u$ to the component should then be paired with all internal outputs $z$ that do not force the context to produce a wrong external output. By doing this we can obtain the loosest description of the conforming behavior of the component in context in response to each external input sequence. When an external input sequence becomes longer, fewer implementations remain conforming, i.e. externally equivalent to *Spec* with respect to this sequence. To handle this situation, the current input $x$ should be paired with a designated output signal *fail* informing us that an expected output can no longer be produced. It means that a conforming behavior of any possible implementation is no longer possible. Intuitively, an external input sequence resulting in the output *fail* can later be used as a test that reveals certain faulty implementations.

We note that some internal I/O sequences cannot be excited at the input of any implementation machine in response to a given external input sequence. Nonexecutable sequences correspond in a sense to 'unfeasible' machines. The trap state, as in Section 3.2, accepts all nonexecutable sequences. Separating executable internal I/O sequences from nonexecutable ones we eventually tune our fault model to the explicit fault model.

Now we demonstrate how the approximation can be constructed. Given a specification machine *Spec* and a context machine $C$, the reference system is $RS = Spec \circ C$. Let *Ch* be a chaos machine with a single state in alphabet $U$ and $Z$. We construct first a product machine $Ch \times C \times RS$

$\times Ver$. The machine $[[Spec]]_C$ is then derived from the product machine in alphabets $X \cup U$ and $Z \cup \{null, fail\}$. In other words, constructing $[[Spec]]_C$ we hide in $Ch \times C \times RS \times Ver$ all actions $Y$ and verdicts **pass**. The chaos machine is nondeterministic, therefore for a particular input $u$ it produces all possible outputs $z$. Some of them result in the verdict **fail**, while others result in **pass**. The former actions are deleted and the input $u$ is paired with the remained actions. A current external input $x$ is coupled with *fail* only when all internal actions result in **fail**. The input $x$ is coupled with the output *null* otherwise. We illustrate the constructions on our working example system. For more details, the reader is referred to (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996).

**Example.** To obtain the approximation $[[Spec]]_C$ of the FSM *Spec* in context $C$ (Figure 2) we have to first construct the product machine $Ch \times C \times RS \times Ver$. Figure 6 shows the approximation $[[Spec]]_C$ obtained from the product machine. Here a black hole is a trap state that has incoming transitions from all states labeled with an input not specified at a particular state with corresponding outputs: $u_1$ or $u_2$ with outputs $z_1$, $z_2$, or $x_1$, $x_2$ with the output $n$ (*null*). These transitions are not shown in this figure. As an example, the initial state 1 has additional transitions to the trap state labeled with $u_1/z_1,z_2$ and $u_2/z_1,z_2$, and state 2 has three transitions labeled with $x_1/n$, $x_2/n$ and $u_2/z_1,z_2$. Stable states 1, 5, 6, 8, 9, 13, 14 are depicted in bold circles, thick lines represent their outgoing transitions.

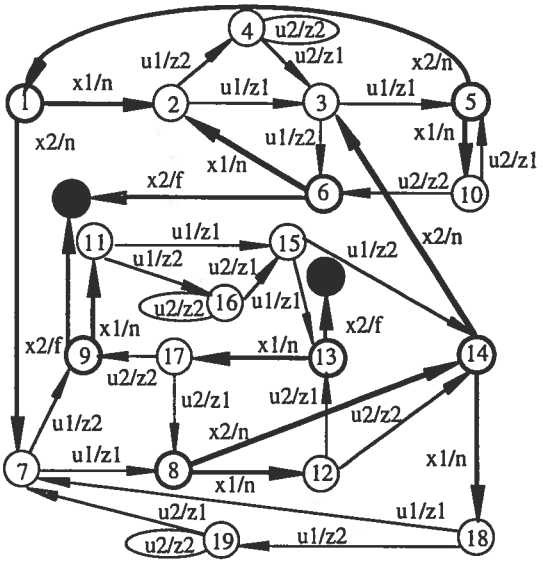


**Figure 6** The approximation $[[Spec]]_C$.

The approximation in context enjoys the following nice properties.

**Proposition 6.1.** Given the approximation $[[Spec]]_C$ of the specification *Spec* in context $C$, we have $Imp =_C Spec$ iff $Imp^a \leq [[Spec]]_C$ for all $Imp \in \mathfrak{I}_n(U,Z)$.

The proof is given in (Petrenko, Yevtushenko, Bochmann, and Dssouli, 1996).

Moreover, for every implementation machine that is not a reduction of the approximation, there exists an external input sequence that distinguishes the augmented implementation from the specification in the given context. For a given $\alpha \in (X \cup U)^*$ we denote $\alpha^x$ projection of $\alpha$ onto the alphabet $X$ ($X$-projection).

**Proposition 6.2.** Given an implementation machine $Imp \in \mathfrak{I}_n(U,Z)$ and the approximation $[[Spec]]_C$ of the specification *Spec* in context $C$, there exists a sequence $\alpha \in (X \cup U)^*$ such that $Imp^a \not\leq_\alpha [[Spec]]_C$ iff $Imp \neq_{C,\alpha^x} Spec$.

For the proof we refer again to (Petrenko, Yevtushenko, Bochmann and Dssouli, 1996).

### 6.2 Fault models

Proposition 6.2. indicates that the approximation in context of the specification can serve as a proper characterization of faults of the component in context. This leads us to propose the fault model $<[[Spec]]_C, \leq, \mathfrak{I}_n^a(U,Z)>$. The fault model seems to be precise since Proposition 6.2 establishes both, necessary and sufficient conditions.

**Proposition 6.3.** The $X$-projection of a complete test suite w.r.t. the fault model $<[[Spec]]_C, \leq, \mathfrak{I}_n^a(U, Z)>$ is a complete test suite w.r.t. the explicit fault model $<RS, \cong, \mathfrak{I}_n(U,Z) \circ C>$.

Once a test suite complete w.r.t. the fault model $<[[Spec]]_C, \leq, \mathfrak{I}_n^a(U,Z)>$ is obtained, it can easily be converted into a corresponding external test suite by making the $X$-projection of every test sequence. Unfortunately, the problem of deriving tests complete w.r.t. this fault model remains open.

**Example.** The test suite derived for our working example based on the fault model $<[[Spec]]_C, \leq, \mathfrak{I}_3^a(U, Z)>$ has 11 test cases of length 67. The machine *Spec* (Figure 2b), if tested in isolation, would require seven test cases of length of 35. This test suite is shorter than that for testing in context since several external input sequences may be needed to deliver a single internal test case to different implementations and to propagate an internally detected fault to the external output, as we have demonstrated in the Section 3.

Replacing the fault domain $\mathfrak{I}_n^a(U,Z)$ by the universal set $\mathfrak{I}_n(X \cup U,Z)$ of deterministic machines over the alphabets $X \cup U$ and $Z$, we can obtain the fault model $<[[Spec]]_C, \leq, \mathfrak{I}_n(X \cup U,Z)>$. Clearly, $\mathfrak{I}_n(X \cup U,Z) \supset \mathfrak{I}_n^a(U,Z)$. For this fault model there exists a suitable test derivation technique (Petrenko, Yevtushenko, and Bochmann, 1996).

**Example.** The test suite derived for our working example based on the above fault model has 21 test cases of length 120.

Unlike $\mathfrak{I}_n^a(U,Z)$, the fault domain $\mathfrak{I}_n(X \cup U,Z)$ contains unfeasible machines, therefore a test suite complete w.r.t. $<[[Spec]]_C, \leq, \mathfrak{I}_n(X \cup U,Z)>$ may be redundant, as it is in our example.

## 7 COMPARING FAULT MODELS

The following models of faults in an embedded component are proposed in this paper:

- $<RS, \cong, \mathfrak{I}_n(U,Z) \circ C>$, the explicit fault model;
- $<RS, \cong, D\text{-}sub(Ch_m(X,Y))>$ that is based on a black-box representation of an implemented system;
- $<RS, \cong, D\text{-}red_m(Ch(U,Z) \circ C)>$, based on deterministic reductions of the chaos machine with a single state;
- $<RS, \cong, D\text{-}sub(Ch_n(U,Z) \circ C\ )>$, based on deterministic submachines of the chaos machine with $n$ states;
- $<[[Spec]]_C, \leq, \mathfrak{I}_n^a(U, Z)>$, based on the approximation in context of the specification *Spec* and the fault domain $\mathfrak{I}_n^a(U,Z)$ of all $X$-augmented implementations of the component;

- $<[[Spec]]_C, \leq, \mathfrak{I}_n(X \cup U,Z)>$, using the universal set $\mathfrak{I}_n(X \cup U,Z)$ as the fault domain.

**Table 2** Characteristics of the fault models.

| RS | conf. relat. | fault domain | any method? | accuracy | TS length |
|---|---|---|---|---|---|
| $RS$ | $\cong$ | $\mathfrak{I}_n(U,Z) \circ C$ | – | + | 67 |
| $RS$ | $\equiv$ | $D\text{-}sub(Ch_m(X,Y))$ | + | – | 294 |
| $RS$ | $\cong$ | $D\text{-}red_m(Ch(U,Z) \circ C)$ | – | – | ? |
| $RS$ | $\equiv$ | $D\text{-}sub(Ch_n(U,Z) \circ C)$ | + | – | 235 |
| $[[Spec]]_C$ | $\leq$ | $\mathfrak{I}_n^a(U, Z)$ | – | + | 67 |
| $[[Spec]]_C$ | $\leq$ | $\mathfrak{I}_n(X \cup U,Z)$ | + | – | 120 |

## 8 CONCLUSION

We have considered in this paper the problem of modeling faults located in a given component embedded within a composite system. Various fault models appropriate for test derivation in context have been elaborated. All of them rely on the assumption that faults do not increase the number of states of the specification but can easily be adjusted to the case when the number of states in any implementation of the embedded component machine exceeds that of the specification. The existing FSM-based methods can be applied to derive test suites complete w.r.t. some fault models. Devising fault models appropriate for testing in context is the first step towards systematic methods for deriving test suite with guaranteed fault coverage. A lot of work remains to be done in this direction.

**Acknowledgments.** This work was partly supported by the NSERC strategic grant and by the Russian Found for Fundamental Research.

## 9 REFERENCES

Bochmann, v.G., Das, A., Dssouli, R., Dubuc, M., Ghedamsi, A., and Luo, G. (1992) Fault models in testing, in *IFIP Transactions Protocol Test Systems IV*, North-Holland.

Bochmann, v.G., Petrenko, A, and Yao, M. (1994) Fault coverage of tests based on finite state models, in *Protocol Test Systems VII*, Chapman & Hall.

Bochmann, v.G., and Sunshine, C.A. (1980) Formal methods in communication protocol design. *IEEE Trans. on Comm.*, **28**, 624-31.

Brand, D., and Zafiropulo, P. (1983) On communicating finite state machines. *Journal of ACM*, **30**, **2**, 323-42.

Chow, T.S. (1978) Testing software design modeled by finite-state machines. *IEEE Trans. on Soft. Eng.*, **SE-4**, **3**, 178-87.

Dam, v.H., Kloosterman, H., and Kwast, E. (1992) Test derivation for standardized test methods, in *IFIP Transactions Protocol Test Systems IV*, North-Holland.

Gill, A. (1962) *Introduction to the theory of finite-state machines*. McGraw-Hill, NY.

Fujiwara, S., Bochmann, v.G., Khenbek, F., Amalou, M., Ghedamsi A. (1991) Test selection based on finite state machine model. *IEEE Trans. on Soft. Eng.*, **SE-17**, **6**, 591-603.

Heerink, L. and Brinksma, E. (1995) Validation in context, in *15th IFIP International Symposium on Protocol Specification, Testing, and Verification*, Chapman & Hall.

ISO/IEC JTC1/SC21 WG7. (February 1995) *Formal methods in conformance testing*, Working Draft Project 1.21.54. ISO.

Johnson, D.S. (1974) Approximation algorithms for combinational problems. *Journal Comput. Syst. Sci.*, **9**, 256-78.

# Formal Description Techniques IX

## Theory, application and tools

## IFIP TC6 / 6.1 International Conference on Formal Description Techniques IX/Protocol Specification, Testing and Verification XVI, Kaiserslautern, Germany, 8–11 October 1996

Edited by

**Reinhard Gotzhein**

and

**Jan Bredereke**

*Department of Computer Science*
*University of Kaiserslautern*
*Germany*

Published by Chapman & Hall on behalf of the
International Federation for Information Processing (IFIP)

CHAPMAN & HALL
London · Weinheim · New York · Tokyo · Melbourne · Madras