3

FAULT DETECTION POWER OF A WIDELY USED TEST SUITE FOR A SYSTEM OF COMMUNICATING FSMS

Ana Cavalli*, Svetlana Prokopenko°, Nina Yevtushenko°

**Institut National des Télécommunications*

9 rue Charles Fourier

F-91011 Evry Cedex

France

Email: Ana.Cavalli@int-evry.fr

° *Tomsk State University*

36 Lenin av.

634050 Tomsk

Russia

Email: prokopenko.rff@elefot.tsu.ru, qel@asd.iao.tsc.ru

Abstract This paper studies the fault detection power of a widely used test suite, i.e., a test suite that traverses each possible transition of each component FSM (Finite State Machine) in the reference system. It is shown that such a test suite is complete, with respect to single output faults of a component under test, if the output of the component is accessible during a testing mode. Experiments have been performed showing that a test suite detecting single outputs faults of each component is good: 92 % of transfer and output faults of the composite FSM are detected.

Keywords: conformance testing, embedded testing, fault detection, communicatings FSMs

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35516-0_20

1. INTRODUCTION

One important aspect of automatic test generation is the derivation of tests for a complex system of a distinguished structure. The problem is known as gray-box testing [7]. A number of approaches have been developed for testing a system under an assumption that at most one component can be faulty. In this case, the system is divided into two parts: the context that is assumed to be fault-free and the component that should be tested. This problem is well known as embedded testing [7] or testing in context [17]. Some of the approaches proposed to solve the problem for a system of communicating Finite State Machines (FSMs) are heuristic and do not guarantee a complete fault coverage [9]. Other approaches [11,12,13,17,18,19,25,26] deliver complete test suites with respect to various fault domains, i.e. sets of possible implementations of the component under test.

In this paper, we consider a system of communicating FSMs and study fault detection power of a widely used test suite that should traverse each involved transition of each component FSM in the reference system. We formally show that the test suite detects all single output faults of a component when the output of the component under test is accessible during a testing mode. The performed experiments clearly show that a complete test suite w.r.t. single output faults of each component FSM usually also detects "almost all" transfer and output faults of the corresponding composite FSM, i.e. is good enough. There exist a number of methods for such test suite derivation [1,9]. Similar to the paper [19], our results are based on a so-called embedded equivalent that represents all faulty traces of the component FSM that can be detected at points of observation. However, in our case, component FSMs can be partially specified. To illustrate our approach we present the testing of a system composed by telephone services.

The rest of the paper is structured as follows. Section 2 presents some basic notions. Section 3 is devoted to the problem statement and explains how an embedded equivalent for partial FSMs can be derived. This section also the results of the performed experiments. We then present, in Section 4, a set of external input sequences that traverse each transition of a component under test in the reference system. We show that this set of external inputs is a complete test suite w.r.t. single output faults of the component FSM if the component's output is accessible during a testing mode. A procedure is proposed to perform the fault coverage evaluation of a given test suite.

Section 5 comprises an example of testing a system composed by telephone services. Finally, section 6 gives the conclusions of this work.

2. PRELIMINARIES

2.1 I/O Finite State Machines

An I/O finite state machine (often simply called an FSM or a machine throughout this paper) is initialized, possibly partially specified and a finite set of states with $s_0 \in S$ as the initial state, X is a finite nonempty set of inputs, Y is a finite nonempty set of outputs, and h is a behavior function mapping a specification domain $D_A \subseteq S\times X$ into $P(S\times Y)$ where $P(S\times Y)$ is the set of all nonempty subsets of the set $S\times Y$. The machine A is *deterministic* if $|h(s,x)|=1$ for all $(s,x)\in D_A$. In a deterministic FSM, the function h usually is replaced with two functions: next state function δ: $D_A \rightarrow S$ and output function λ: $D_A \rightarrow Y$. Deterministic I/O finite stste machine is denoted by 7-tuple $(S,X,Y,\delta,\lambda,D_A,s_0)$. The machine A is *complete* if $D_A=S\times X$; otherwise, the machine is *partial*. The machine $B=(T,X,Y,g,D_B,s_0)$ is called a *submachine* of A if $T\subseteq S$, $D_B=D_A$ and for all $(s,x)\in D_A$, $g(s,x)\subseteq h(s,x)$.

In the usual way, the function h is extended to a so-called *output function* h^y over an appropriate subset I_A of input sequences with results in the set of nonempty subsets of output sequences. The set I_A is the set of all input sequences where a behavior of the A is defined. The set $h^y(\alpha)$ comprises each output sequence that can be produced by the FSM when the sequence α is submitted. As usual, given input sequences $x_1...x_k$ and output sequence $y_1...y_k$, the sequence $x_1/y_1,...,x_k/y_k$ is called *a trace* of the A if $h^y(x_1...x_k)=y_1...y_k$.

Given two FSMs $A=(S,X,Y,h,D_A,s_0)$ and $B=(T,X,Y,g,D_B,t_0)$, FSM A is called a *trace reduction* of the FSM B, or simply a *reduction* of B, denoted $A\leq B$, if the set of traces of A is a subset of that of the FSM B, i.e. $I_A\subseteq I_B$ and for any input sequence $\alpha\in I_A$, $h^y(s_0, \alpha)\subseteq g^y(t_0, \alpha)$. If for some input sequence $\alpha\in I_A \cap I_B$ it holds that $h^y(s_0, \alpha)\not\subseteq g^y(t_0, \alpha)$ then the sequence α is said to *distinguish* the FSM A from B.

For the class of deterministic FSMs, the reduction relation coincides with a so-called quasi-equivalence relation [5]. Deterministic FSM $A=(S,X,Y,\delta,\lambda,D_A,s_0)$ is a reduction of deterministic FSM $B=(T,X,Y,\Delta,\Lambda,D_B,t_0)$ if and only if B is quasi-equivalent to A, i.e. $I_A\subseteq I_B$ and for any input sequence $\alpha\in I_A$, the output sequences of A and B to α coincide.

2.2 Fault Domain

One important aspect of high quality test generation is to specify an appropriate fault model. We further assume that a reference system and a system under test are modeled by deterministic FSMs specified over the same input and output alphabets. We refer to an FSM modeling the reference system as to a *reference* FSM while referring a *fault domain* to the set $\mathfrak{I}$ of FSMs modeling all possible systems under test. A FSM $B\in\mathfrak{I}$ is called an *implementation.* The B is called a *faulty* or a *nonconforming* implementation if the reference FSM is not a reduction of B; otherwise it is a *conforming* implementation.

A finite set of finite input sequences of the reference machine RM is a *test suite* (w.r.t. $\mathfrak{I}$) if it detects at least a single nonconforming implementation. A test suite which detects each nonconforming implementation is called a *complete* test suite w.r.t. the fault domain $\mathfrak{I}$. Formally, given a reference FSM RM and a fault domain $\mathfrak{I}$, a test suite TS is complete w.r.t. $\mathfrak{I}$ if for each FSM $B\in\mathfrak{I}$ such that RM is not a reduction of B, the TS has a sequence distinguishing RM from B.

Thus, if a reference FSM is deterministic, a system under test is modeled by an FSM of the set $\mathfrak{I}$ and is not detected by a complete test suite w.r.t. $\mathfrak{I}$ then one concludes that the reference system is a reduction of the system under test, i.e. a behavior of the system under test coincides with that of the reference system under any input sequence where the behavior of the reference system is specified.

As it is claimed in [14,15,24], a complete test suite derived from a partial reference FSM RM is different from that derived from the reference FSM where each undefined transition of the RM is augmented as a looping transition with the *Null* output. In the latter case, during a testing mode one also checks the looping transitions that are never traversed during a working mode. By this reason, in this paper, we derive a test suite without augmenting partial FSMs. We now consider two widely used fault domains.

Given a deterministic reference FSM $RM=(S,X,Y,\delta,\lambda,D_A,s_0)$ with n states, a fault domain $\mathfrak{I}_{RM}$ has each FSM $B=(S,X,Y,\Delta,\Lambda,D_B,s_0)$ such that $D_B\supseteq D_{RM}$ and for each $(s,x)\in D_{RM}$, $\delta(s,x)=\Delta(s,x)$, i.e. only output faults may occur in an implementation. The fault domain $\mathfrak{I}_n$ comprises each complete FSM over alphabets X and Y with at most n states. In the former case, a transition tour of the reference FSM is a complete test suite [20]. In the latter case, also transfer faults are involved. Procedures for a complete test suite derivation w.r.t. the fault domain $\mathfrak{I}_n$ are well developed [2,4,10,14,16,21,22,23,24].

However, most of them only deal with a reduced reference FSM and deliver tests with length proportional to the product of number of transitions and number of states of the reference machine.

2.3 Composition of FSMs

We consider a system composed by two FSMs, as shown in Figure 1. We refer to symbols of alphabet X_1 and X_2 as to external inputs, to symbols of alphabets Y_1 and Y_2 as to external outputs while referring to symbols of alphabets Z and U as to internal actions. In fact, one of the alphabets X_1 or X_2 (Y_1 or Y_2) can be empty.

In order to define the composition of FSMs we do the following assumptions. The system has always a single message in transit, i.e. the environment submits the next input only when the system has produced an output to the previous input. Moreover, a component machine accepting an input may produce either an external or an internal output. If the component machines fall into infinite internal dialog when an appropriate external input sequence is submitted we say the system falls into live-lock under the input sequence and its behavior is not specified under the input sequence. If the behavior of one of component FSMs is not specified under a submitted input we also assume the system behavior is not specified.

Under the above assumptions we can derive the composite FSM *Context*◊*Spec* of the context FSM *Context* and the component FSM *Spec* using various algorithms [11,17]. Below we briefly sketch the algorithm from the paper [11].

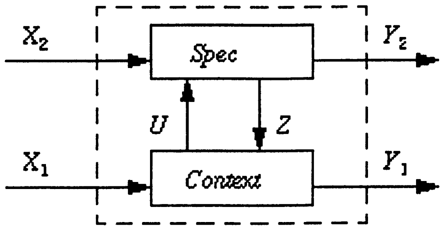

Figure 1. System model

Let the component FSMs *Context* and *Spec* have the sets Q and T of states, respectively. Then the composite FSM is a FSM over alphabets $X_1 \cup X_2$ and $Y_1 \cup Y_2$ with the state space that is a subset of $Q \times T$. States of the set $Q \times T$ are divided into stable and transient states. By definition, the initial state q_0t_0 is stable. Otherwise, the state is stable if it is reached after the system has produced an external output. We start from the initial state q_0t_0..

Given a stable state *qt* and input *x*, there is a transition labeled with *x*/*u* (*x*/*z*) from the initial state *qt* to a transient state *q't* (*qt'*) if *x*/*u* (*x*/*z*) takes the FSM *Context* (*Spec*) from the state *q* (*t*) to the state *q'* (*t'*). There is a transition labeled with *x*/*y* from the state *qt* to a stable state *q't* (*qt'*) if *x*/*y* takes the FSM *Context* (*Spec*) from the state *q* (*t*) to the state *q'* (*t'*).

Given a transient state *q't'* with an incoming transition labeled by *a*/*z* (*a*/*u*), a pair of internal actions *z*/*u* (*u*/*z*), and a transient state *q"t'* (*q't"*), there is a transition labeled with *z*/*u* (*u*/*z*) from the state *q't'* to the state *q"t'* (from the state *q't'* to *q't"*) if there is an outgoing transition from the state *q'* to the state *q"* under *z*/*u* in the *Context* (under *u*/*z* in the *Spec*). There is a transition labeled with *z*/*y* (*u*/*y*) from the state *q't'* to a stable state *q"t'* (from the state *q't'* to *q't"*) if there is a transition from the state *q'* to the state *q"* under *z*/*y* in the *Context* (from the state *t'* to the state *t"* under *u*/*y* in the *Spec*). The stable states cannot be merged with transient states. Two transient states with the same names are merged if they have an incoming transition labeled with a pair with the same output part.

If no stable state is reachable then a transition from the state *qt* under *x* is undefined. Otherwise, given the final stable state *q't'* with an incoming transition *x*/*y*, *z*/*y* or *u*/*y*, the composite FSM has a transition from the state *qt* to *q't'* under *x*/*y*. As mentioned above, no stable state is reachable if one of the following conditions holds: a) a transition of the context (or the component) at a current state is undefined under a submitted input; b) the system falls into live-lock, i.e. has a cycle labeled with internal actions.

The procedure is repeated unless all reachable stable states with possible external inputs are considered.

3. TEST SUITE DERIVATION FOR AN EMBEDDED COMPONENT W.R.T. OUTPUT FAULTS

We consider a reference system composed by two deterministic FSMs, as shown in Figure 1.

3.1 Discussion about Faults

Our first step is to specify a type of faults that should be detected with a derived test suite, i.e. to define an appropriate fault domain. Usually two types of faults are considered, namely output and transfer faults. It is well known that a test suite detecting all transfer and output faults is a high-quality one. However, it can hardly be used in practical situations, for its length is proportional to the product of number of transitions and number of states of the reference FSM.

On the other hand, there are publications [see, for example, 3] where the authors claim that for a proper kind of FSMs length of a test suite detecting "almost all" faults is proportional to the number of states of the reference FSM, i.e. tests can be used in practical situations. The result is also confirmed with a widely distributed opinion that long tests are needed to detect a very low percent of proper faulty implementations which can be treated separately (if necessary). By this reason, we performed some experiments before selecting a fault domain.

Given an FSM, experiments have been performed in order to evaluate the fault coverage of a transition tour of the FSM w.r.t. transfer faults. Given a complete reference FSM $RM=(S,X,Y,\delta,\lambda,D_A,s_0)$ and a test suite, fault coverage of the test suite is calculated as ratio $(m/n)\cdot 100\%$. Here n is number of FSMs over alphabets X and Y with the state set S which are not equivalent to *RM*, while m is number of such FSMs which are detected with the given test suite, i.e. have an unexpected output sequence to an appropriate test case. Reference FSMs derived in pseudo-random way with up to 10 states have been studied. The average fault coverage of a transition tour is equal to 96%. In order to perform experiments with more complex reference FSMs we evaluated fault coverage of a transition tour w.r.t. 100 implementation FSMs also derived in pseudo-random way. The same result has been obtained.

We then considered a composition of two complete communicating FSMs. Given a complete test suite w.r.t. single output faults of the context *Context* and of the component *Spec*, we have evaluated its fault coverage w.r.t. output faults of the reference composite FSM $Context\Diamond Spec$. Our experiments show it is about 95%. In other words, given a test suite complete w.r.t. single output faults of the context *Context* and of the component *Spec*, the fault coverage w.r.t. transfer and output faults of the composite FSM is expected to be about 92%, i.e. such test is of a practical use.

In the following sections, we show a set of external input sequences which traverse each transition of a component under test in the reference composition is a complete test suite w.r.t. single output faults of the component when the component's output is accessible during a testing mode. Length of such test suite is proportional to number of transitions of the component FSM involved with a given context and there exist a number of methods [1,9] for such test suite derivation without constructing a composite FSM that usually has huge number of states.

3.2 Fault Assumptions and Test Architecture

1. We further assume that the context is correctly implemented.

2. Only single output faults are possible in the component under test. In other words, the next state function of a faulty component implementation *Imp* is an extension of that of the reference component *Spec*. Thus, the set $\mathfrak{R}_{Spec}$ of all possible component implementations is the set of possible extensions of all deterministic sub-machines of the FSM *Spec'* [8] that has the same next state function as *Spec* with each output for each transition.

3. An implementation *Imp* of the component under test is conforming if *Context◊Spec* is a reduction of *Context◊Imp* while the *Context* (the *Imp*) only produces internal output sequences where a behavior of the *Spec* (*Context*) is specified.

4. We have an access to the internal output of the implementation component *Imp*, i.e. can observe the internal output the component has been produced (Figure 2). However, we cannot control its internal inputs.

Formally speaking, fault domain of our interest is the set *Context◊*$\mathfrak{R}_{Spec}$ of all composite FSMs of the reference context FSM and a component FSM, possibly with a single output fault. However, the reference FSM now is slightly different from the *Context◊Spec*, because of the point of observation at the internal output of the component FSM during a testing mode. This new situation is illustrated by Figure 2.

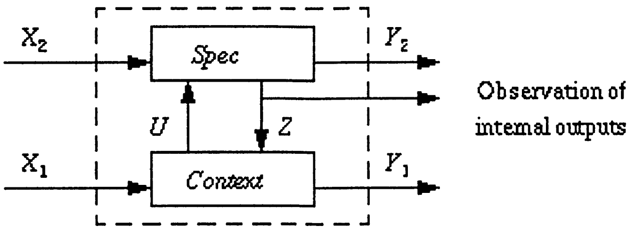

Figure 2. Test architecture

It is well known that if the reference FSM *Spec* is a reduction of an implementation component FSM *Imp* then the reference FSM *Context◊Spec* also is a reduction of the *Context◊Imp*. However, generally, a converse is not true. There can exist FSM *Imp* such that the machine *Context◊Spec* is a reduction of *Context◊Imp* while the reference component *Spec* is not a reduction of *Imp*. In the case, when machines *Context*, *Spec* and *Context◊Spec* are complete such machines *Spec* and *Imp* are called ***equivalent in the context*** while *Imp* also is called a ***conforming implementation in the context*** [18]. Thus, if one requires to have only reference internal outputs for the machine *Imp*, some conforming implementations can be rejected. The following example illustrates the situation.

Example. Consider FSMs *Context*, *Spec* and *Context◊Spec* in Figure 3 when the external input and output sets X_2 and Y_2 of the component *Spec* are empty. By direct inspection, one can assure that *Context◊Spec* is equivalent to the FSM *Context◊Imp* where the FSM *Imp* is shown in Figure 3d, i.e. the *Context◊Spec* is a reduction of the FSM *Context◊Imp*. Therefore, a conforming implementation system at its observation points can produce an unexpected output sequence $z_2z_1y_1$ when x_1 is submitted, instead of the reference z_1y_1.

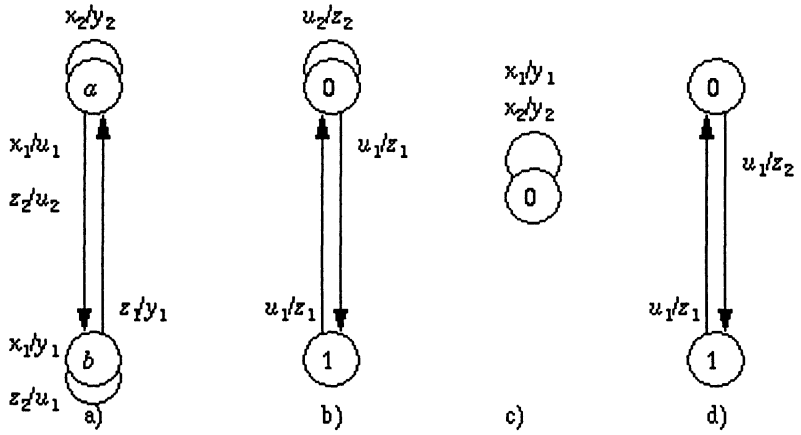

Figure 3. FSMs *Context* (a), *Spec* (b), *Context◊Spec* (c) and FSM *Imp* (d)

More rigorous analysis is necessary to determine which internal outputs are allowed for an implementation component FSM when an external input is submitted. In the case when FSMs *Context*, *Spec* and *Context◊Spec* are complete, this analysis can be performed based on the embedded equivalent [19] of the component FSM *Spec* in the given context. An FSM *Imp* is a conforming implementation of the *Spec* in the given context if and only if the FSM *Context◊Imp* is complete and the FSM *Imp* is a reduction of the embedded equivalent.

A complete test suite w.r.t. the fault domain *Context*$\Diamond\mathfrak{R}_{Spec}$ can be derived by a procedure proposed in [19]. We first derive a complete test suite *TS* w.r.t. the fault domain $\mathfrak{R}_{Spec}$ from the embedded equivalent. An internal test is then translated into a external test suite. We cannot use this approach directly for FSMs *Context* and *Spec* that can be partially specified. Moreover, we are interested only in single output faults; thus, a simpler procedure for a complete test suite derivation can be expected.

3.3 Trace Detecting FSM

Given an external input sequence α such that the behavior of the reference FSM *Context*$\Diamond$*Spec* is defined under α, a trace over alphabets $X_2 \cup U$ and $Y_2 \cup Z$ is said to be *detectable with* α [25] if any implementation system with the component *Imp* having this trace has an unexpected behavior when α is submitted, i.e. either the output response of the implementation system to α is different from that of the reference system or the behavior of the implementation system is not defined under α. A trace is said to be *detectable* if it is detectable with an appropriate external input sequence. Due to definition, the behavior of the implementation system is not defined if the system falls into live-lock when α is submitted or the behavior of the context is undefined under a current input. An observation point at the internal output of the component enables to detect these faults.

In this section, we derive a so-called *Trace Detecting FSM* (*TDF*). We show that the set of traces of a nonconforming implementation component FSM intersects the set of traces of the labeling paths which have no cycles and go from the initial state to a designated *Fail* state in the *TDF*. As usual, we further refer to a path that has no cycles as to a *simple* path.

In fact, the *TDF* is a particular case of an embedded equivalent when the FSMs *Context*, *Spec* and *Context*$\Diamond$*Spec* can be partial, and the test architecture allows to observe the internal output of the component under test, and we are interested only in output faults. By this reason, to derive the FSM *TDF*, we use the machine *Spec'* where *Spec'* has the same next state function as the *Spec* and all possible outputs for each defined transition, instead of the chaos machine [17,19]. The latter together with observation of the internal outputs of the component under test allows to determine internal traces implying live-locks as well as to simplify a procedure for a complete test suite derivation.

To determine all the detectable traces we first derive a FSM *F* that represents all possible composition traces and recognizes those of them that induce an unexpected behavior of the composition by a designated state *Fail*. The FSM is derived as a composition of the context and the component FSM *Spec'*. We use the FSM *Context◊Spec* to recognize traces with an unexpected external output projection. FSM *F* then is projected onto alphabets of the component *Spec* by a subset construction replacing each subset having the state *Fail* by a designated state *Fail* without outgoing transitions.

Let the FSMs *Context*, *Spec* and *Context◊Spec* have the state sets Q, T and S, respectively. The state space of the FSM F is a subset of $Q \times T \times S$. States of the set $Q \times T \times S$ are divided into stable and transient states. By definition, the initial state $q_0t_0s_0$ is stable. Otherwise, the state is stable if it is the fail-state or has an incoming transition with an external output. The stable states cannot be merged with transient states. Two transient states with the same names are merged if they have an incoming transition labeled with a pair with the same output part. We start from the initial state $q_0t_0s_0$. Then we apply the following procedure:

1. Given a stable state qts and input x, there is a transition labeled with x/u (x/z) from the initial state qts to a transient state $q'ts$ ($qt's$) if x/u (x/z) takes the FSM *Context* (*Spec'*) from the state q (t) to the state q' (t'). There is a transition labeled with x/y from the stable state qts to a stable state $q'ts'$ ($qt's'$) if x/y takes the FSM *Context* (*Spec'*) from the state q (t) to the state q' (t') while x/y takes the FSM *Context◊Spec* from the state s to the state s'. If the output of *Context◊Spec* at the state s to x is defined and is different from y, we specify a transition from the state qts to the designated state *Fail* labeled with x/y. The reason is if a component has a trace with the corresponding projection then an unexpected external output will be produced when an appropriate external input sequence is submitted to the implementation system.

2. Given a transient state qts with an incoming transition labeled by a/z (a/u) and a pair z/u (u/z), there is a transition from the state qts to the designated state *Fail* labeled with z/u (u/z) if one of the following conditions holds:

a) z/u (u/z) provides a cycle labeled with pairs of internal actions, i.e. when a component implementation FSM has a trace with the corresponding projection the composition falls into live-lock;

b) the context (the component) is undefined at the state q' (t') under input z (u), i.e. the context (the component) does not expect a submitted input at the current state.

If none of the above conditions holds then there is a transition labeled with *z*/*u* (*u*/*z*) from the state *qts* to a transient state *q'ts* (from the state *qts* to *qt's*) where *q'* is a successor of *q* under *z*/*u* in the *Context* (*t'* is a successor of *t* under *u*/*z* in the *Spec'*).

3. There is a transition labeled with *z*/*y* (*u*/*y*) from the transient state *qts* to a stable state *q'ts'* (from the state *qts* to *qt's'*) if there is a transition from the state *q* to the state *q'* under *z*/*y* in the *Context* (from the state *t* to the state *t'* under *u*/*y* in the *Spec'*) and *x*/*y* takes the FSM *Context* (*Spec'*) from the state *s* to the state *s'*. If the output of the *Context◊Spec* to *x* at the state *s* is defined and is different from *y*, we specify a transition from the state *qts* under *z*/*y* (*u*/*y*) as a transition to the designated state *Fail*.

4. The procedure is repeated unless all reachable stable states with possible external inputs are considered. Moreover, since there is an observation point at the internal output of the component under test, given a state of the FSM such that all its outgoing transitions result in *Fail*, we replace the state with the *Fail* state.

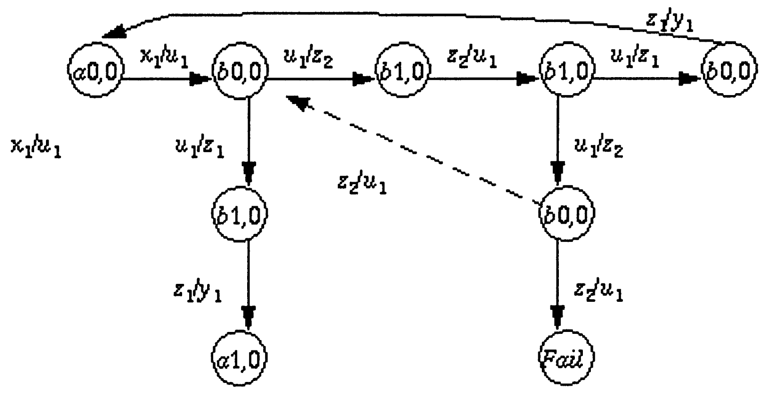

Figure 4. A fragment of the FSM *F*

Example. Consider FSMs *Context*, *Spec* and *Context◊Spec* shown in Figure 3. We derive the machine *Spec'* by adding each output to each defined transition of the *Spec*. Consider the initial stable state (*a*0,0). There is a transition under x_2/y_2 from the state to the same stable state and a transition under x_1/u_1 from the state to a transient state (*b*0,0). There are two transitions from the state (*b*0,0): a transition under u_1/z_1 to the transient state (*b*1,0) and a transition under u_1/z_2 to the transient state (*b*1,0). The latter two transient states cannot be merged since they have incoming transitions labeled by pairs with different output parts. There is a transition from the former state (*b*1,0) to a stable state (*a*1,0) under z_1/y_1 while there is a

transition from the latter state $(b1,0)$ to a transient state $(b1,0)$ under z_2/u_1. There is a transition under u_1/z_1 from the latter transient state $(b1,0)$ to a transient state $(b0,0)$ where is a transition to a stable state $(a0,0)$ under z_1/y_1. A transition under u_1/z_2 from the latter transient state $(b1,0)$ provides a cycle labeled with internal pairs, i.e. there is a transition to the *Fail* state under u_1/z_2. This fragment of FSM F is shown in Figure 4. In the same way, the fragment of the FSM is derived for a stable state $(a1,0)$.
By construction, the FSM F has complete in about formation traces of the *Spec'* that induce an unexpected behavior of the composition, i.e. the following statement holds.

Proposition 3.1. Given the component *Spec*, let *Spec'* denote the machine with the same next state function and each possible output for each transition. A trace of the machine *Spec'* is detectable if and only if it has a prefix β/γsuch that the FSM F has a trace with the corresponding projection β/γresulting in the *Fail* state.

Thus, our next step is to project the obtained FSM F onto the set of pairs $\{\alpha/\beta\text{: } a\in X_2\cup U,\ \beta\in Y_2\cup Z\}$. We perform this using a subset construction [HoUl79]. Two states of the FSM are merged into a single state if there exist traces to these states with the same projection over the component alphabets $X_2\cup U$ and $Y_2\cup Z$. Each subset comprising a designated state *Fail* is replaced with the state *Fail* without outgoing transitions. If for some state, all its outgoing transitions result in *Fail* then we replace the state with the *Fail* state. Denote *TDF* the obtained FSM and call it *trace detecting FSM.*

Theorem 3.2. Given the trace detecting FSM *TDF* derived by the above procedure and an implementation component FSM *Imp* that is a submachine of *Spec'*, the reference FSM *Context◊Spec* is not a reduction of *Context◊Imp* if and only if the machine *Imp* has a trace that labels a simple path in the *TDF* from the initial state to the *Fail* state.

Proof. If part is a corollary to Proposition 3.1.

Only if part. Suppose now that we have a nonconforming submachine *Imp* of *Spec'*, i.e. the reference FSM *Context◊Spec* is not a reduction of the *Context◊Imp*. Due to Proposition 3.1, the FSM F has a trace resulting in the *Fail* state such that its projection onto alphabets of the FSM is a trace $a_1/b_1 \ldots a_k/b_k$ taking the FSM *TDF* from the initial to the *Fail* state. Moreover, suppose that the trace labels some path in the FSM *TDF* from the initial state to the *Fail* state that has a cycle. In other words, we have S_0 - $a_1/b_1 \rightarrow$ S_1- $a_2/b_2 \rightarrow S_2$ ... - $a_k/b_k \rightarrow$ *Fail*, along with $S_i=S_j$ for some $i<j$, $i,j=0,\ldots,k-1$. States of the *TDF* are subsets of state of the FSM F. Each state of the set S_0 is of a kind qt_0s, i.e. has t_0 as the component part of the state. When

projected, the component part of each state of the subset S_1 is a successor of the state under the I/O pair a_1/b_1 in the *Spec'* and so on. Thus, the FSM *Imp* has also a trace

t_0 - a_1/b_1 $\rightarrow$... t_i- a_{j+1}/b_{j+1} $\rightarrow$ t_{j+1} ... - a_k/b_k..

Therefore, the *TDF* has a trace

S_0 - a_1/b_1 $\rightarrow$... S_i- a_{j+1}/b_{j+1} $\rightarrow$ S_{j+1} ... - a_k/b_k $\rightarrow$ *Fail*,

that labels a path resulting in the *Fail* state and has no above cycle, i.e. FSM *Imp* has a trace that labels a simple path resulting in the *Fail* state.

❑

Example. The FSM *TDF* for FSMs *Context* and *Spec* in Figure 3 is shown in Figure 5.

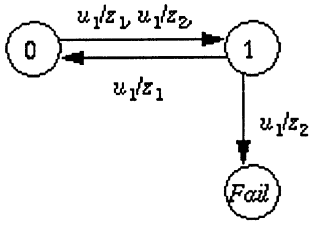

Figure 5. FSM *TDF*

4. TEST SUITE DERIVATION

4.1 Test Suite Derivation w.r.t. Output Faults

As a corollary to Theorem 3.2, we can establish sufficient conditions for a test suite to be complete w.r.t. the fault domain *Context*$\Diamond\Re_{Spec}$.

Proposition 4.1. Given a test suite, the test suite is complete w.r.t. the fault domain *Context*$\Diamond\Re_{Spec}$ if it detects each trace labeling a simple path in the *TDF* from the initial state to the *Fail* state.

Example. Due to proposition 4.1, we obtain a complete test suite $\{x_1x_1\}$ w.r.t. the fault domain *Context*$\Diamond\Re_{Spec}$ after translation of the set of internal traces $\{u_1/z_1,u_1/z_2;\ u_1/z_2,u_1/z_2\}$ which label simple paths in the *TDF* (Figure 5) from the initial state to the *Fail* state.

Thus, coming back to procedure proposed in [19] for a complete test suite derivation we now do not need to derive a complete test suite from a nondeterministic FSM *TDF*. An internal test can be derived as the set of all traces of the labeled simple paths in the FSM from the initial state to the *Fail* state. However, we notice that the condition of Proposition 4.1 is not necessary, i.e. a shorter complete internal test may exist. We now establish a necessary and sufficient condition for a test suite detecting all single output faults of the component FSM.

4.2 Test Suite Derivation w.r.t. Single Output Faults

A single output fault t - $a/b \longrightarrow t'$ of the component FSM is *detectable* if there exists a simple path in the *TDF* from the initial state to the *Fail* state such that the path traverses the transition t - $a/b \longrightarrow t'$, while all other transitions traversed by the path are transitions of the reference component FSM *Spec*. We say that the path, or the trace labeling the path, *captures* the fault t - $a/b \longrightarrow t'$. The trace detecting FSM can be essentially simplified if we are interested only in single output faults of the component FSM. To represent such faults we use the FSM $Spec_{sof}$. The next state function of the FSM coincides with that of the *Spec*; output b is in the set of outputs to input a at the state t if and only if the single output fault t - $a/b \longrightarrow t'$ is not detectable.

Proposition 4.2. Given an implementation $Imp \in \Re_{Spec}$, *Imp* is a conforming implementation if and only if *Imp* is an extension of some sub-machine of $Spec_{sof}$.

Therefore, to detect an implementation component FSM with a single output fault we need an external case that induces a trace traversing the corresponding transition in the component FSM. A procedure for deriving an internal test suite complete w.r.t. single output faults of the component FSM is almost transparent. A complete test suite is a transition tour traversing each detectable transition of the FSM *Spec'*. The obtained internal test then is translated into external test suite.

Proposition 4.3. An external test suite is complete w.r.t. single output faults of the component FSM if and only if the set of corresponding traces of the component FSM in the reference system traverses each detectable transition.

Proof. Only if part is a corollary of Proposition 4.2.

If part. Let $Imp \in \Re_{Spec}$, t - $a/b \longrightarrow t'$ be a faulty detectable transition and αx be the shortest prefix of a test case such that the corresponding traces of the component FSM in the reference system is tailed by t - $a/b \longrightarrow t'$. All other transitions of the trace induced by α are reference transitions; by this reason, the global states of the systems $Context \Diamond Imp$ and $Context \Diamond Spec$ after

α coincide. Thus, the external input induces the faulty transition t - $a/b \rightarrow t'$ in the component implementation *Imp*, i.e. an internal (or external) output sequence of the *Imp* is not in the set of output sequences of $Spec_{sof}$. Therefore, αx detects the nonconforming implementation *Imp*.

□

Here we notice that in general case, a test suite traversing each transition of a component under test does not detect its all output faults in the component under test. We can only guarantee that it detects each single output fault. However, if there is the reference output response of the component under test to each test case then there are no output faults in the component.

Given a test suite, an implementation component FSM $Imp \in \Re_{Spec}$ is a conforming implementation if and only if the output sequence at the component's outputs to each test case is in the set of corresponding output sequences of the FSM $Spec_{sof}$. Thus, a test suite derived by procedure proposed in [9] is complete w.r.t. single output faults of the component FSM when there is an observation point at the internal output of a component under test. The same conclusion can be drawn about the Hit-or-Jump method in [1]. However, both methods do not use a trace detecting FSM; a verdict "pass" is only drawn in the case when the output sequence at the component's outputs coincides with that of the reference component *Spec*. By this reason, some conforming implementations can be rejected by a test suite delivered by the above methods.

4.3 Fault Coverage Evaluation

Given a test suite *TS*, we can calculate its fault coverage w.r.t. single output faults. As usual, the fault coverage is calculated as the ratio m/n multiplied 100% where m is number of single output faults detected with the test suite *TS* while n is number of all such detectable faults. Number n can be calculated as the product $l|Y_2 \cup Z|$-p where l is number of defined transitions of the FSM *Spec* which can be involved with the given context [1] while p is number of single output faults over these transitions which are not detectable. In the case of complete FSMs, an implementation with such fault is equivalent to the specification component FSM *Spec* in the given context [18].Number n is calculated in the same way: m=$k|Y_2 \cup Z|$-q, where k is number of transitions of the component FSM traversed with the test suite in the reference system while q is number of single output faults over these transitions which are not detectable.

Example. Consider the reference component FSM *Spec* (Figure 3). There are two single output faults, namely 0 - $u_1/z_2 \rightarrow 1$ and 1 - $u_1/z_2 \rightarrow 0$ over transitions involved with the given context. The fault 0 - $u_1/z_2 \rightarrow 1$ is not detectable since there is no trace in the FSM *TDF* (Figure 5) from the initial state to the *Fail* state such that the trace traverses this transition while all other transitions traversed by the trace are transitions of the reference component *Spec*. An implementation component FSM *Imp* with the fault is shown in Figure 3d. The system of the context and the FSM *Imp* is a conforming implementation. To detect an implementation component FSM with the fault 1 - $u_1/z_2 \rightarrow 0$ an external test case should induce an internal trace with a prefix u_1/z_1, u_1/z_2. By direct inspection, one can assure the test case x_1x_1 possesses the feature. We also notice that a test case x_1 does not detect any nonconforming implementation w.r.t. single output faults of the component FSM.

5. TELEPHONE SERVICES EXAMPLE

We demonstrate our approach by testing a system composed by telephone services: the Basic Call Services (BCS), Call Forward Unconditional (CFU) and Original Call Screening (OSC). The approach is illustrated by testing OCS in the context of the BCS and CFU. The I/O FSMs modeling the services (Figure 6) have been obtained from SDL specifications of the system [13].

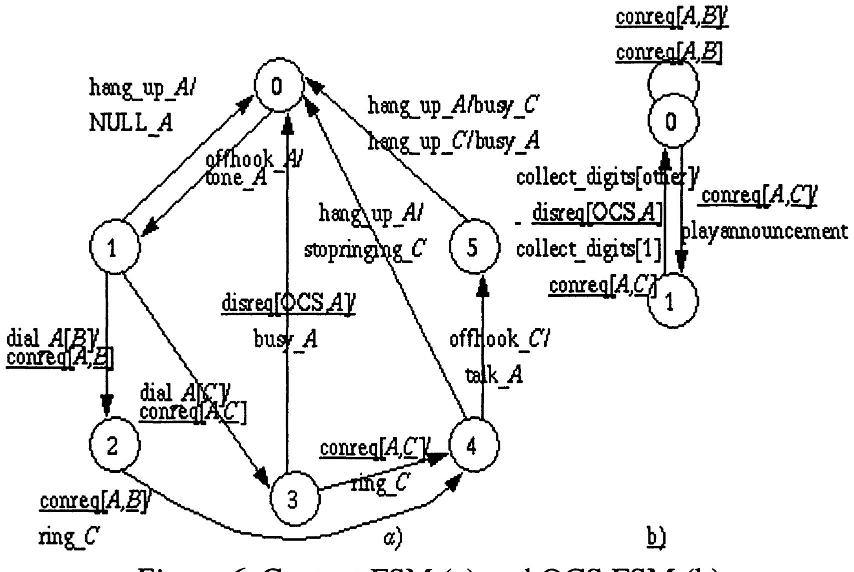

Figure 6. Context FSM (a) and OCS FSM (b)

Figure 7 presents the corresponding composite FSM while Figure 8 shows the trace detecting FSM.

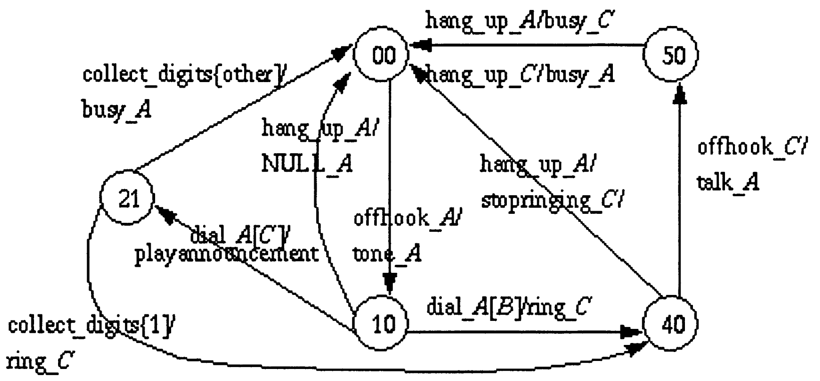

Figure 7. The composite FSM

By direct inspection, one can assure each single output fault is detectable, i.e. there are 12 detectable single output faults. Given a test suite *TS*={off_hook_*A*, dial_[*A*,*B*]; off_hook_*A*, dial [*A*,*C*], collect_digits[other]}, we determine the set *T* of all single output faults detected with the test suite *TS*. In our example, *T*=
{0 - conreq_[*A*,*B*]/conreq_[*A*,*C*] → 0;
0 - conreq_[*A*,*B*]/disreq_[OCS,*A*] → 0;
0 - conreq_[*A*,*B*]/play_announcement → 0;
0 - conreq_[*A*,*C*]/conreq_[*A*,*B*] → 1;
0 - conreq_[*A*,*C*]/conreq_[*A*,*B*] → 1;
0 - conreq_[*A*,*C*]/disreq_[OCS,*A*] → 1;
1 - collect_digits[other]/conreq_[*A*,*B*] → 0;
1 - collect_digits[other]/conreq_[*A*,*B*] → 0;
1 - collect_digits[other]/play_announcement → 0}.
Thus, three faults remain undetectable with the test suite, i.e. the fault coverage *Φ*(*TS*) of the *TS* w.r.t. single output faults is equal to (12/15)100%=80%.

Consider a single output fault 1 - collect_digits[1]/conreq_[*A*,*B*] → 0 undetected with the *TS*. By direct inspection of Figure 6 (a), one can assure that to induce the trace traversing the transition 1 - collect_digits[1]/conreq_[*A*,*B*] → 0 in an implementation OSC, the context must enter the state 1, i.e. the external sequence off_hook_A, dial [*A*,C] should be applied. At this point we

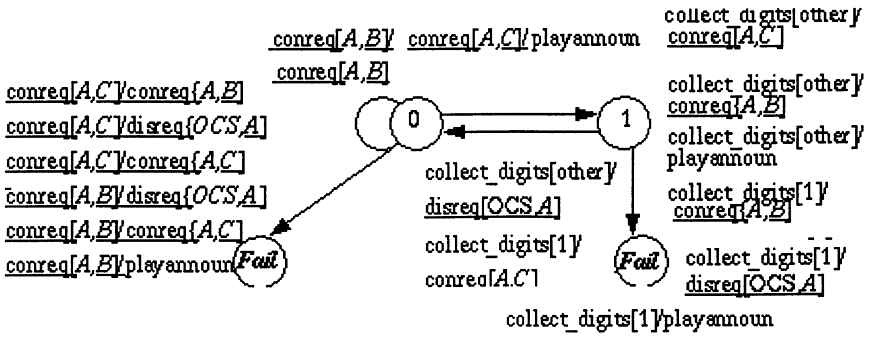

Figure 8. FSM *TDF*

have the context at the state 3, the OCS at the state 1. The external input sequence collect_digits[1] then induces the desired transition if the implementation OSC has such a transition. However, the context does not expect the input conreq [*A*,*B*] at the current state 3, i.e. it is a faulty output. Thus, a test case off_hook_*A*, dial_[*A*,*C*], collect_digits[1] detects the single output fault 1 - collect_digits[1]/conreq [*A*,*B*] → 0.

By direct inspection, one can assure the same test case also detects single output faults 1 - collect_digits[1]/disreq [OCS,*A*] → 0 and 1 - collect_digits[1]/play_announcement → 0, i.e. the fault coverage of the test suite with the extra test case is 100%. Therefore, the set {off_hook_*A*, dial_[*A*,*B*]; off_hook_*A*, dial [*A*,*C*], collect_digits[other]; off_hook_*A*, dial_[*A*,*C*], collect_digits[1]} is complete w.r.t. single output faults. Here we notice that a complete test suite not necessary traverses each transition of the reference composed FSM *Context◊Spec*, as it happens in the above example.

Thus, if at observation points of a system under test we observe the reference output sequence tone_*A*, conreq [*A*,*B*], ring_*C* to the off_hook_*A*, dial_[*A*,*B*]; the reference output sequence tone_*A*, play_announcement, disreq [*OCS*,*A*], busy_*A* to the off_hook_*A*, dial_[*A*,*C*], collect_digits[other]; while the reference output sequence tone_*A*, play_announcement, conreq [*A*,*C*], ring_*C* to the off_hook_*A*, dial_[*A*,*C*], collect_digits[1], then the system under test has the reference system as its reduction. Otherwise, the system under test is a nonconforming implementation.

6. CONCLUSION

In this paper, we studied fault detection power of a test suite widely used in practice for a system of communicating Finite State Machines (FSMs). The test suite traverses each involved transition of each component FSM in the reference system. We have shown that the test suite is complete w.r.t. single output faults of a component FSM when the output of the component is accessible during a testing mode. The performed computer experiments have shown that a test suite w.r.t. single output faults of each component FSM usually also detects "almost all" transfer and output faults in the corresponding composite FSM. We illustrated our approach by testing a system composed by telephone services.

Some of the reviewers did very interesting remarks to the work presented here: for our assumption that the output of the component is accessible during testing, the reviewer signalled that there are real protocols where outputs are not observable, in particular, when an unit implements several layers it is not possible to observe such outputs. Our answer is that if different layers are implemented by an unit, we will consider this unit as a component. On the other hand, our experience on the test of services on a CORBA platform shows that it is possible to observe exchanged messages between software components.

References

[1] Cavalli, A., Lee, D., Rinderknecht, C., and Zaïdi, F. *Hit-or-Jump: An algorithm for embedded testing with applications to IN services.* Proceedings of Joint Inter. Conf. FORTE/PSTV99, pp: 41-58.

[2] Chow, T.S. (1978). *Test software design modeled by finite state machines.* IEEE Transactions on Software Engineering, 4(3): 178-187.

[3] David, R., and Wagner K. *Analysis of detection probability.* IEEE Transactions on Computers, 39(10): 1284-1291.

[4] Fujiwara, S., v.Bochmann,G., Khendek, F., Amalou, M., and Ghendamsi, A. (1991). *Test selection based on finite state models.* IEEE Transactions on Software Engineering, 17(6): 591-603.

[5] Gill, A. *Introduction to automata theory.* Mc Graw-Hill, NY, 1962.

[6] Hopkroft, J.E. and Ulman, J.D. (1979). *Introduction to automata theory, languages and computation.* Addison-Welsey, NY.

[7] Information technology. (1991) *Open systems interaction. Conformance testing methodology and framework.* International standard IS-9646.

[8] Koufareva, I., Petrenko, A., Yevtushenko, N. *Test generation driven by user-defined faults.* Proceedings of 12th IWTCS, pp: 215-236.

[9] Lee, D., Sabnani, K.K., Kristol, D.M., and Paul, S. (1996). *Conformance testing of protocols specified as communicating finite state machines - a guided random walk based approach.* IEEE Transactions on Communications, 44(5): 631-640.

[10] Lee, D., and Yannakakis, M. (1996). *Principles and methods of testing finite state machines, a survey.* IEEE Transactions, 84(8):1090-1123

[11] Lima, L.P., and Cavalli, A.R. (1997). *A pragmatic approach to generating test sequences for embedded systems.* Proceedings of 10th IWTCS, pp: 125-140.

[12] Lima, L.P., and Cavalli, A.R. (1997). *Application of embedded testing methods to service validation.* Second IEEE Intern. Conf. on Formal Engineering methods.

[13] Lima, L.P. (1998). *A pragmatic method to generate test sequences for embedded systems.* Ph.D.Thesis. Institute National des telecommunications, Evry, France.

[14] Petrenko, A. *Checking experiments with protocol machines.* Proceedings of 4th IWTCS, 1991, pp: 83-94.

[15] Petrenko A., Yevtushenko, N., and Dssouli, R. (1994). *FSM based strategies for testing communicating FSMs.* Proceedings of 7th IWTCS, pp:181-196.

[16] Petrenko A., Yevtushenko, N. and v.Bochmann, G. (1996). *Testing deterministic implementations from their nondeterministic specifications.* Proceedings of 9th IWTCS, pp: 125-140.

[17] Petrenko A., Yevtushenko, N., and v. Bochmann, G. (1996). *Fault models for testing in context.* Proceedings of Joint Inter. Conf. FORTE/PSTV96, pp: 125-140.

[18] Petrenko, A., Yevtushenko, N., v. Bochmann, G., and Dssouli, R. (1996). *Testing in context: Framework and test derivation.* Computer Communications, 19: 125-140.

[19] Petrenko, A., and Yevtushenko, N. (1997). *Testing faults in embedded components.* Proceedings of 10th IWTCS, pp: 125-140.

[20] B.Sarikaya and G.v.Bochmann. *Synchronization and Specification issues in protocol testing.* IEEE Transactions on Communications, Vol. COM-32, April 1984, pp. 389-395.

[21] Vasilevsky, M.P. (1973). *Failure diagnosis of automata.* Cybernetics, (4): 653-665.

[22] Vuong, S.T., Chan, W.W.L., and Ito, M.R. (1989). *The UIO-method for protocol test sequence generation.* In IFIP TC6 Second Inter. Workshop on Protocol Test Systems, pp. 161-175.

[23] Yannakakis, M., and Lee, D. (1995). *Testing finite state machines: fault detection.* Journal of Computer and System Sciences, (50): 209-237.

[24] Yevtushenko, N., Petrenko, N. *On fault detection power of checking experiments with automata.* Automatic Control and Computer Science. Allerton Press, N.Y., 1989, Vol. 23, No.3, pp: 3-7.

[25] Yevtushenko, N., Cavalli, A.R., and Lima, L.P. (1998). *Test minimization for testing in context.* Proceedings of 11th IWTCS, pp: 127-145.

[26] Yevtushenko, N., Cavalli, A.R., and Anido, R. (1999). *Test suite minimization for embedded nondeterministic finite state machines.* Proceedings of 12th IWTCS, pp: 237-250.