Fault Diagnosis in Extended Finite State Machines

Khaled El-Fakih[1], Svetlana Prokopenko[2],
Nina Yevtushenko[2], and Gregor v. Bochmann[3]

[1] Department of Computer Science, American University of Sharjah, UAE
kelfakih@aus.ac.ae
[2] Tomsk State University, Russia
{prokopenko,yevtushenko}@elefot.tsu.ru
[3] School of Information Technology and Engineering, University of Ottawa, Canada
bochmann@site.uottawa.ca

Abstract. In this paper, we propose a method for the derivation of an adaptive diagnostic test suite when the system specification and implementation are given in the form of an extended finite state machine. The method enables us to decide if it is possible to identify the faulty transitions in the system when faults have been detected in a system implementation. If this is possible, the method also returns test cases for locating the faulty transitions. An example is used to demonstrate the steps of the method.

1 Introduction

The purpose of diagnostic testing is to locate the differences between a specification and its implementation, when the implementation is found to be faulty. In the software domain where a system is represented as an FSM, some work has already been done on diagnostic testing [2][4]. However, no work has been reported for systems represented as Extended FSMs.

In [1] we considered the problem of the derivation of an adaptive diagnostic test suite for a system of two communicating FSMs. It is known that we can not always locate the difference between the system specification and its implementation, due to the fact that different faults can result in the same behavior of a system/implementation under test (SUT). In [1], we presented a method that enables us to decide if it is possible to locate a faulty component machine, and if this is possible then tests for locating the fault(s) are derived. In this paper, we use a similar approach to the diagnostic testing of a single Extended FSM (EFSM). We assume that either predicate, transfer or assignment faults may occur in an EFSM implementation. Moreover, none of these faults increases the number of states in the implementation of the system. Similar to [1], we present a method for the derivation of an adaptive diagnostic test suite that enables us to decide whether it is possible to identify the faulty transitions in the given system, and if this is possible then tests for locating the faulty transitions are derived. We assume that the specification domain of each variable is finite and therefore, an EFSM can be represented as a classical FSM. We use a non-deterministic FSM, called *Fault Function* (FF) [6], for the compact representation of transfer, predicate and assignment faulty transitions. The fault domain is the

D. Hogrefe and A. Wiles (Eds.): TestCom 2003, LNCS 2644, pp. 197–210, 2003.

union of sub-machines of three FFs, *FF-predicate*, *FF-transfer* and *FF-assignment*. In this paper, we present a new method how a FF can be reduced by deleting sub-machines that do not agree with the observed Input/Output behavior of an SUT. We also describe two strategies for the derivation of diagnostic tests that differentiate between different implementations without the explicit enumeration of sub-machines of the FFs. In order to reduce the number of superfluous or infeasible sub-machines of FFs that do not correspond to possible implementations we study how a faulty transition of an EFSM affects transitions of the corresponding FSM.

This paper is organized as follows. Section 2 includes necessary preliminaries needed for understanding the diagnostic problem introduced in Section 3 and solved in subsequent sections. Section 6 concludes the paper and includes some insights for future work.

2 Preliminaries

A non-deterministic finite state machine (FSM) is an initialized non-deterministic complete machine that can be formally defined as follows [7]. A *non-deterministic finite state machine A* is a 5-tuple $\langle S, I, O, h, s_0\rangle$, where S is a finite nonempty set of states with s_0 as the initial state; I and O are finite nonempty sets of inputs and outputs, and h: $S\times I \rightarrow 2^{S\times O}\backslash\emptyset$ is a behavior function where $2^{S\times O}$ is the set of all subsets of the set $S\times O$. The behavior function defines the possible transitions of the machine. Given present state s_i and input symbol i, each pair $(s_j,o)\in h(s_i,i)$ represents a possible transition to the next state s_j with the output o. This is also written as a transition of the form $s_i \xrightarrow{i/o} s_j$. If for each pair $si \in S\times I$ it holds that $|h(s,i)| = 1$ then FSM A is said to be *deterministic*. In the deterministic FSM A instead of behavior function h we use two functions, transition function δ: $S\times I \rightarrow S$ and output function λ: $S\times I \rightarrow O$. FSM $B = (S',I,O,g,s_0)$, $S'\subseteq S$, is a *sub-machine* of A if for each pair $si\in S'\times I$, it holds that $g(s,i) \subseteq h(s,i)$.

Sometimes a behavior of an FSM is not defined for some pairs $si\in S\times I$. In this case, the transition is said to be *undefined* at state s under input i. An FSM where some transitions are undefined is called *partial*. If a partial FSM has a complete submachine, i.e. a submachine where each transition is defined, then the FSM has the largest complete submachine [3]. The latter can be obtained by iteratively deleting states where at least one transition is undefined. If the initial state has an undefined transition then the FSM has no complete submachine. Otherwise, the procedure is terminated when no state can be deleted.

As usual, function h can be extended to the set I^* of finite input sequences. Given state $s\in S$ and input sequence $\alpha=i_1i_2...i_k\in I^*$, output sequence $o_1o_2...o_k\in h(s,\alpha)$ if there exist states $s_1=s$, s_2, ... , s_k, s_{k+1} such that $(s_{j+1},o_j)\in h(s_j,i_j)$, $j = 1,\ldots,k$. Input/Output sequence $i_1o_1i_2o_2...i_ko_k$ is called a *trace* of A if $o_1o_2...o_k\in h(s,i_1i_2...i_k)$. We let the set $h^o(s,\alpha) = \{\gamma \mid \exists s'\in S\ [(s',\gamma)\in h(s,\alpha)]\}$ denote the *output projection* of h, while denoting $h^s(s,\alpha) = \{s' \mid \exists \gamma\in Y^*\ [(s',\gamma)\in h(s,\alpha)]\}$ the *state projection* of h. If for each prefix β of α the set $h^s(s,\beta)$ is a unique state then we say that the state $h^s(s,\alpha)$ is *deterministically reachable* from the initial state via the sequence α.

Given state s of FSM $A=(S,I,O,h,s_0)$ and state t of FSM $B=(T,I,O,g,t_0)$, states s and t are *equivalent*, written $s \cong t$, if for each input sequence $i_1 i_2 ... i_k \in I^*$, it holds that $h^o(s,i_1 i_2 ... i_k) = g^o(t,i_1 i_2 ... i_k)$. If states s and t are not equivalent then they are *distinguishable*, written $s \neq t$. A sequence $\alpha \in I^*$ such that $h(s, \alpha) \neq g(t, \alpha)$ is said to *distinguish* states s and t. States s_1 and s_2 of FSM A are said to be *separable* [7] if there exists an input sequence α such that the sets $h^o(s_1,\alpha)$ and $h^o(s_2,\alpha)$ are disjoint. Sequence α is called a *separating* sequence for the states s_1 and s_2.

FSMs $A = (S,I,O,h,s_0)$ and $B = (T,I,O,g,t_0)$ are *equivalent*, written $A \cong B$, if their sets of traces coincide. A sequence α that distinguishes the initial states of non-equivalent FSMs A and B is said to *distinguish* FSMs A and B.

Given a deterministic FSM $A = (S, I, O, \delta, \lambda, s_0)$, a non-deterministic FSM FF defined over the same input and output alphabets is called a *Fault Function* (FF) for A if it includes A as a sub-machine. Fault functions were introduced in [6] to represent in a compact way all implementations of a given FSM with a given type of error. In this case, the set of all deterministic sub-machines of FF is considered as a *fault domain* of A.

A sequence $s_0 \xrightarrow{i_1/o_1} s_1, \ldots, s_{k-1} \xrightarrow{i_k/o_k} s_k$ of consecutive transitions is called a *path* of FSM A. The path is said to be *deterministic* if it has no transitions with different next states and/or outputs for the same state-input combination. In other words, for any $j,p < k$ it holds: if $s_j = s_p$ and $i_{j+1} = i_{p+1}$ then $o_{j+1} = o_{p+1}$ and $s_{j+1} = s_{p+1}$. In this paper, we are interested in deterministic sub-machines of a given FSM. Since a path of a deterministic sub-machine is always deterministic we consider only deterministic paths of a given FSM. Moreover, a deterministic sub-machine has a family of deterministic paths if and only if these paths are *deterministically compatible*, i.e. for any two paths of the family there are no transitions with different next states and/or outputs for the same state-input combination.

Given two FSMs $A = \langle S, I, O, h, s_0 \rangle$ and $B = \langle T, I, O, g, t_0 \rangle$, the intersection $A \cap B$ is the largest initially connected sub-machine of FSM $\langle S \times T, I, O, H, s_0 t_0 \rangle$ where for each pair (st, i), $st \in S \times T$, $i \in I$,

$H(st, i) = \{(s't', o) \mid (s', o) \in h(s, i), (t', o) \in g(s, i)\}$.

The function $H(st, i)$ is undefined if there is no $o \in O$ such that $(s', o) \in h(s, i)$ and $(t', o) \in g(s, i)$ for appropriate $s' \in S$ and $t' \in T$. The intersection represents the set of common output responses of FSMs A and B to each input sequence and generally, the intersection is a partial FSM. It is known [8] that if the intersection has no complete submachine then any two deterministic sub-machines of A and B are non-equivalent. Moreover, in this case, there exists a set of input sequences, a so-called *distinguishing set* [3], such that any two sub-machines of A and B have different sets of output responses to sequences of this set. In [3] an algorithm is given for determining the distinguishing set.

2.1 The EFSM Model

The EFSM model extends the FSM model with input and output parameters, context variables, operations and predicates defined over context variables and input parameters.

Formally [5], an extended finite state machine M is a pair (S,T) of a set of states S and a set of transitions T between states from S, such that each transition $t \in T$ is a tuple (s,x,P,op,y,up,s'), where:

$s,s' \in S$ are the initial and final states of a transition;

$x \in X$ is an input, where X is the set of inputs, and $D_{inp\text{-}x}$ is the set of possible input vectors, associated with the input x, i.e. each component of an input vector corresponds to an input parameter associated with x;

$y \in Y$ is output, where Y is the set of outputs, and $D_{out\text{-}y}$ is the set of possible output vectors, associated with the output y, i.e. each component of an output vector corresponds to an output parameter associated with y;

P, op, and up are functions, defined over input parameters, and context variables, namely:

P: $D_{inp\text{-}x} \times D_V \rightarrow \{\text{True, False}\}$ is a predicate, where D_V is a set of context vectors $\mathbf{v}$;
op: $D_{inp\text{-}x} \times D_V \rightarrow D_{out\text{-}y}$ is an output parameter function;
up: $D_{inp\text{-}x} \times D_V \rightarrow D_V$ is a context update function.

As in [5], we use the following definitions:

Given an input x and an input vector from $D_{inp\text{-}x}$, the pair of input x and vector from $D_{inp\text{-}x}$, is called a *parameterized input*. A sequence of parameterized inputs is called a *parameterized input sequence*. A context vector $\mathbf{v} \in D_V$ is called a *context* of M. A *configuration* of M is a pair of a state s and a context vector $\mathbf{v}$. Given a parameterized input sequence of an EFSM we can calculate the corresponding parameterized output sequence by simulating the behavior of the EFSM under the input sequence starting from the initial state and initial values of the context variables.

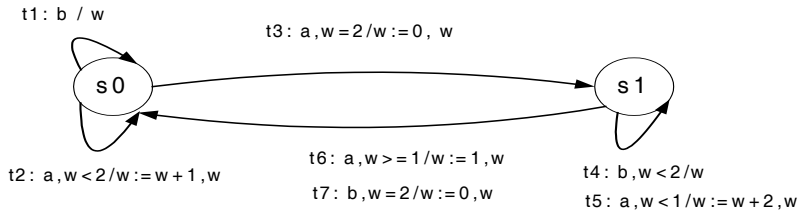

Fig. 1. The EFSM EM_1

Consider the EFSM in Fig 1. It has two states, two non-parameterized inputs a and b and a context variable w. The value of the variable w is an output of the machine. When an input a is applied to the machine at the state s_0 and w is equal to 2, the predicate of the transition t_3 is valid. The machine moves to state s_1, updates w according to action w:=0 and produces output 0.

Definition [5]: An EFSM is said to be:

- *Consistent* if for each transition t with input x, every element in $D_{inp\text{-}x} \times D_V$ evaluates exactly one predicate to true among all predicates guarding the different transitions with the same start state and input x; in other words, the predicates are mutually exclusive and their disjunction evaluates to true.
- *Completely specified* if for each pair $(s, x) \in S \times X$, there exists one transition leaving state s with input x.

In this paper, we consider a class of specification and implementation EFSMs that are consistent and completely specified. A behavior of such EFSM is defined under each parameterized input sequence. Moreover, for each parameterized input sequence there exists a single parameterized output sequence that is produced by the EFSM for the given input sequence. Two EFSMs are said to be equivalent if their parameterized output responses to each parameterized input sequence coincide.

2.2 Unfolding a Given EFSM to an Equivalent FSM and a Fault Model for EFSMs

When the specification domain of each context variable and input parameter is finite an EFSM can be unfolded to an equivalent FSM by simulating its behavior with respect to all possible values of context variables and input vectors. The equivalence relation means the set of traces of the FSM coincides with the set of parameterized traces of the EFSM. Given a state s of EFSM A, a context vector $\mathbf{v}$, an input x and vector $\boldsymbol{\rho}$ of input parameters, we derive the transition from configuration $s\mathbf{v}$ under input $x\boldsymbol{\rho}$ in the corresponding FSM. We first determine the outgoing transition (s,x,P,op,y,up,s') from state s where the predicate P is true for input vector $\boldsymbol{\rho}$ and context vector $\mathbf{v}$, update the context vector to the vector $\mathbf{v}'$ according to the assignment up of this transition, determine the parameterized output $y\boldsymbol{\omega}$ and add the transition $(s\mathbf{v},x\boldsymbol{\rho},y\boldsymbol{\omega},s'\mathbf{v}')$ to the set of transitions of the FSM. The obtained FSM has the same number of states as the number of different configurations $(s,\mathbf{v})$ of the EFSM that are reachable from the initial state. It is known that the simulation can lead to a state explosion problem.

As an example, consider the EFSM EM_1 presented in Fig. 1. At state s_0 two inputs can be applied to the machine; these inputs a and b are not parameterized. The variable w is a context variable and its domain is equal to $\{0,1,2\}$. When a is applied to the machine and the value of context variable w is equal to 0, the machine updates w according to an assignment $w:=w+1$ and moves from the initial configuration $(s_0,0)$ to the configuration $(s_0,1)$ and produces the output $w=1$, because in our example, an output coincides with the value of the context variable w. So, the corresponding FSM has a transition labeled with $a/1$ from the state $(s_0,0)$ to the state $(s_0,1)$. The FSM M_1 that corresponds to the EM_1 of Fig. 1 is shown in a tabular form in Fig. 2.

	$(s_0;0)$	**$(s_0;1)$**	**$(s_0;2)$**	**$(s_1;0)$**	**$(s_1;2)$**	**$(s_1;1)$**
a	**$(s_0;1)/1$**	**$(s_0;2)/2$**	**$(s_1;0)/0$**	**$(s_1;2)/2$**	**$(s_0;1)/1$**	**$(s_0;1)/1$**
b	**$(s_0;0)/0$**	**$(s_0;1)/1$**	**$(s_0;2)/2$**	**$(s_1;0)/0$**	**$(s_0;0)/0$**	**$(s_0;0)/0$**

Fig. 2. The FSM M_1 that corresponds to the EFSM EM_1 of Fig. 1

In our example, a reference behavior at the state $(s_1,1)$ is presented in Fig. 2. However, by direct inspection, one can see that this configuration is not reachable from the initial state. We include the configuration in the description, since it becomes reachable if a fault occurs in the EFSM.

It is known that if an EFSM is consistent and completely specified, the corresponding FSM is complete and deterministic. Two EFSMs are equivalent iff their corresponding FSMs are equivalent.

In this paper, we consider a fault model based on transfer, predicate, and assignment faults of a consistent and completely specified EFSM. Consider a transition t=(s,x,P,op,y,up,$s´$) of an implementation EFSM EM. Transition t has a transfer fault if its final state is different from that specified by the reference EFSM, i.e. an implementation machine has a transition (s,x,P,op,y,up,$s´´$) instead of t. Moreover, t has a predicate fault, if the predicate of the transition in the implementation EFSM is different from P, i.e. an implementation EFSM has the transition (s,x,Q,op,y,up,$s´$) instead of t, where $Q \neq P$, i.e. Q and P return different results for some value(s) of input and context vectors. Transition t has an assignment fault if it has an action other than that specified by the reference EFSM. That is after the execution of the wrong transition, the context and/or output vector will have a value different than expected by the reference assignment statement. We note that an implementation with a predicate fault is in general not consistent, unless certain assumptions can be made about the implementation, as explained in subsequent sections.

3 Fault Diagnosis for EFSMs

Let *EM-RS* be an EFSM representing the specification of the given system while *EM´* is its implementation. We denote *M-RS* the corresponding FSM. Both specification and implementation EFSMs are complete and consistent. We also assume that the implementation belongs to the finite set of machines $E\Re$ = {EM_0=*EM-RS*, EM_1..., $EM_.$}. If the implementation at hand (SUT) say *EM´* does not pass a given test suite *TS*, then our objective is to recognize *EM´* (or identify the set of faulty implementations that are equivalent to *EM´*).

In order to solve the given problem, we work at the FSM level, rather than at the EFSM level. That is, we unfold the given EFSM specification *EM-RS* and obtain an FSM *M-RS*. Due to our assumptions, the implementation at hand is a deterministic complete FSM of the finite set $\Re$ = {M_0=*M-RS*, M_1..., $M_.$}, where each FSM M of $\Re$ corresponds to an EFSM EM of $E\Re$. The set $\Re$ is further called the *fault domain* of *RS*.

A set of input sequences is called a *diagnostic test suite* $Diag_{TS}$ w.r.t. the given fault domain $\Re$ if for each two non-equivalent machines of $\Re$, there exists a sequence that distinguishes them. The set $Diag_{TS}$ can be considered as a distinguishing set of the fault domain $\Re$. When the test cases of $Diag_{TS}$ are applied to the SUT, we can always identify the SUT up to the subset of equivalent machines. The problem is that the number of machines in the fault domain and correspondingly the size of the $Diag_{TS}$ are usually huge. However, to identify a machine of the set $\Re$, it is enough to use its identifier that is a subset of a distinguishing set, i.e. usually is much shorter. Moreover, since we do not know the implementation at hand we have to derive the identifier only on-the-fly, i.e. by the use of an adaptive experiment with the SUT. In this case, the set of diagnostic test cases is not given a priori but is incrementally derived throughout the experiment.

In this paper we use non-deterministic FSMs, called *Fault Functions* (FFs), for the compact representation of the fault domain. A detailed description of these FFs is given in the following subsections.

Let *M-RS* = (S, I, O, δ, λ, s_0) be a complete deterministic specification FSM of a given EFSM and FF is a fault function of *M-RS*. As a consequence of the compact representation, a FF is known to contain *infeasible sub-machines* that do not correspond to possible faulty implementations. In order to reduce the number of infeasible sub-machines we define three FFs for different types of the considered faults instead of a unique FF. Moreover, since a single fault of a given EFSM usually implies multiple faults in the corresponding FSM, for each fault of the EFSM, we determine the set of corresponding transitions of the FSM affected by the fault and we obtain a corresponding partition of input-state combinations of FF. The partition is also used to reduce the number of infeasible sub-machines. We denote *FF-Transfer*, *FF-Predicate*, and *FF-Assignment* these three FFs that we consider for the compact representation of transfer, predicate, and assignment faults. The fault domain is the union of all the deterministic feasible sub-machines of these FFs.

3.1 An Overview of the Diagnostic Approach

Let *RS* be the specification FSM obtained by unfolding the given EFSM, while *TS* is a conformance test suite. If the SUT of the given system produces unexpected output responses to the test suite *TS*, then the SUT is not equivalent to *RS*, i.e. the SUT is a faulty implementation. Our objective is to determine what machine of the fault domain is equivalent to the SUT.

First we derive the three FFs of the specification system, *FF-Predicate*, *FF-Transfer*, and *FF-Assignment*. Here we notice that we use the unfolding procedure only once. The FFs are derived explicitly from the obtained specification FSM based on the types of faults. The set of all deterministic sub-machines of a *FF* includes each implementation FSM that corresponds to the specification EFSM with predicate, transfer, or assignment faults.

Since our implementation system is deterministic we do not take into account non-deterministic paths of FFs. Similar to [1], we first remove from a FF a behavior that does not agree with the observed outputs to the applied test suite *TS*. In Section 4, we describe a novel algorithm that removes from a non-deterministic FSM those sub-machines whose output responses to the test suite do not agree with those obtained by applying the test suite *TS* to the SUT.

Our diagnostic algorithm works under the assumption that the SUT has either predicate, transfer or assignment faults. If the SUT is equivalent to a deterministic sub-machine of a single FF, then there exist corresponding predicate, transfer, or assignment faults, and we try to locate them. However, if the SUT is equivalent to the deterministic sub-machine of two different FFs, or two sub-machines of the same FF, then the faulty machine cannot be uniquely identified. This is due to the fact that there are different possible faults that cause the same observable behavior of the SUT. However, in this case, if needed, we can determine the subset of these possible faults. If none of the above cases occurs, we conclude that the implementation has faults that are not captured by the considered fault model.

In order to draw one of the above conclusions, we should have test cases such that by observing the output responses of the SUT to these test cases, we can distinguish the SUT from other sub-machines of the FFs. If the FFs after deleting sub-machines with the output responses which do not agree with those observed with the confor-

mance test suite, have no equivalent deterministic sub-machines then there exists a distinguishing set of input sequences such that, given the set of output responses to these input sequences, we always can determine a single FF such that the machine under test is a sub-machine of the FF. Otherwise, we generate for the two FFs a set of sequences that distinguish some deterministic sub-machines of these FFs, then we apply these sequences to the SUT and we reduce the FFs based on the observed behavior of the SUT. We repeat the latter process as much as possible. Afterwards, we try to further reduce the remaining FFs, by repeating a process similar to the above, but for each FF alone. Here we note that a single fault in an EFSM can imply several faulty transitions in the corresponding FSM. To derive the FFs we consider each transition of the specification EFSM, insert a corresponding fault and determine the transitions of the corresponding FSM affected by the fault.

A submachine of the *FF-Transfer* (*FF-Predicate* or *FF-Assignment*) is said to be *feasible* if it corresponds to an EFSM that can be obtained from the specification EFSM through transfer (predicate or assignment) faults. In this paper, we define appropriate properties that restrict the fault domain of each FF, and thus can be used to reduce the number of the infeasible sub-machines. For this purpose, for each possible single transfer (assignment or predicate) fault in the EFSM, we determine the cluster of its corresponding faults in the FSM. The clusters obtained, for a particular FF, form a partition of its transitions.

3.2 *FF-Transfer*, *FF-Predicate*, and *FF-Assignment* and Their Properties

A transition of an implementation EFSM *EM* has a *transfer fault* if its final state is different from that specified by the reference EFSM. Consider a transition $t=(s,x,P,op,y,up,s')$ of *EM* under input x from state s to the tail state s'. When a transfer fault occurs we have a wrong ending state s''. Therefore, for each configuration $s\mathbf{v}$ of *EM* and parameterized input $x\boldsymbol{\rho}$ such that the predicate of the transition t is true for the pair $\boldsymbol{\rho}\mathbf{v}$, i.e. $P(\boldsymbol{\rho},\mathbf{v})$=true, and there is a transition $(s\mathbf{v},x\boldsymbol{\rho},y\boldsymbol{\omega},s'\mathbf{v}')$ in the FSM corresponding to *EM*, the *FF-transfer* contains the transition $(s\mathbf{v},x\boldsymbol{\rho},y\boldsymbol{\omega},s''\mathbf{v}')$ for each state s'' of the *EM*.

The set of pairs $(s\mathbf{v},x\boldsymbol{\rho})$ such that the predicate P_t of the transition t from the state s under input x is true for the pair $\boldsymbol{\rho}\mathbf{v}$ is denoted $Dom(P_t)$ and called "domain of P_t". Since an EFSM is consistent, the set of domains $Dom(P_t)$ over all transitions of the *EM* is a partition of the set of all pairs "state-input" of the corresponding FSM.

As an example, consider transition t_2 of EM_1 in Fig. 1. Under the input a, the predicate of t_2 is true for the configurations $(s_0,0)$ and $(s_0,1)$, thus the pairs $)s_0 0,a($ and $(s_0 1,a)$ form a $Dom(P_2)$ of the partition. For configuration $(s_0,0)$, there is the transition $(s_0 0,a,1,s_0 1)$ to the configuration $(s_0,1)$. Therefore, we add transition $(s_0 0,a,1,s_1 1)$ to *FF-Transfer*. Moreover, from configuration $(s_0,1)$, there is transition $(s_0 1,a,2,s_0 2)$ to the configuration $(s_0,2)$. Therefore, we add transition $(s_0 1,a,2,s_1 2)$ to *FF-transfer*. Due to our restrictions, from configurations $(s_0,0)$ and $(s_0,1)$ and under the input a, in any feasible submachine there can only be transitions to either $(s_0,1)$ and $(s_0,2)$ or to $(s_1,1)$ and $(s_1,2)$. We consider each transition of the EM_1 in Fig. 1 and obtain the *FF-Transfer* shown in Fig. 3.

Proposition 1. A submachine of *FF-transfer* is *feasible* only if for each two transitions of the same predicate domain the state parts of the next configurations coincide.

	$(s_0;0)$	$(s_0;1)$	$(s_0;2)$	$(s_1;0)$	$(s_1;2)$	$(s_1;1)$	**Domains**
A	$(s_0;1)/1$ $(s_1;1)/1$	$(s_0;2)/2$ $(s_1;2)/2$	$(s_1;0)/0$ $(s_0;0)/0$	$(s_1;2)/2$ $(s_0;2)/2$	$(s_0;1)/1$ $(s_1;1)/1$	$(s_0;1)/1$ $(s_1;1)/1$	 $\{(s_0 0a),$ $(s_0 1a)\}$ $\{(s_1 0a)\}$ $\{(s_0 2a)\}$ $\{(s_1 2a),$ $(s_1 1a)\}$
B	$(s_0;0)/0$ $(s_1;0)/0$	$(s_0;1)/1$ $(s_1;1)/1$	$(s_0;2)/2$ $(s_1;2)/2$	$(s_1;0)/0$ $(s_0;0)/0$	$(s_0;0)/0$ $(s_1;0)/0$	$(s_1;1)/1$ $(s_0;1)/1$	 $\{(s_0 0b),$ $(s_0 1b),$ $(s_0 2b)\}$ $\{(s_1 0b),$ $(s_1 1b)\}$ $\{(s_1 2b)\}$

Fig. 3. The *FF-Transfer* that corresponds to transfer faults of EM_1 of Fig. 1

Transition $t = (s,x,P,op,y,up,s')$ of a specification EFSM *EM* has a *predicate fault*, if the predicate of the transition of an implementation EFSM is different from P, i.e. an implementation EFSM has the transition (s,x,Q,op,y,up,s') instead of t. Since an implementation EFSM *EM* is consistent, the subset of $D_{inp\text{-}x} \times D_V$ that evaluates the faulty predicate Q to true has to be a subset of that of the reference predicate, i.e. $Dom(Q) \subseteq Dom(P)$. As usual, in order to keep an implementation EFSM complete, we use the completeness assumption. For any item of the set $D_{inp\text{-}x} \times D_V$ where P is true while Q is false, the implementation EFSM has a loop at the state s with the identity update function up_{id} and the *Null* output (written as '-') that has no parameters. In other words, the implementation EFSM remains at the current state s and does not execute the specified action *up*; therefore, the context vector is not changed. Moreover, the implementation EFSM produces the *Null* output, i.e it does not produce any output.

Thus, if the predicate fault occurs for the transition $t = (s,x,P,op,y,up,s')$ of a specification EFSM *EM* where the predicate P is changed to predicate Q, then a corresponding implementation EFSM has transitions (s,x,Q,op,y,up,s') and $(s,x,R,\text{-},up_{id},s)$ where the predicate R is true for each item of the set $D_{inp\text{-}x} \times D_V$ where P is true while Q is false,. Therefore, the FSM corresponding to the faulty implementation has transitions $(s\mathbf{v},x\boldsymbol{\rho},\text{-},s\mathbf{v})$ for each pair $(s\mathbf{v},x\boldsymbol{\rho}) \in Dom(R)=Dom(P)\backslash Dom(Q)$.

Thus, when deriving *FF-Predicate*, for each transition $(s\mathbf{v},x\boldsymbol{\rho},y\boldsymbol{\omega},s'\mathbf{v}')$ of the FSM obtained from the transition $t = (s,x,P,op,y,up,s')$ of the specification EFSM we add the transition $(s\mathbf{v},x\boldsymbol{\rho},\text{-},s\mathbf{v})$ for each $(s\mathbf{v},x\boldsymbol{\rho}) \in Dom(P)$.

As an example, consider transition t_2 of EM_1 in Fig. 1. Under the input a, the predicate of t_2 is true for the configurations $(s_0,0)$ and $(s_0,1)$. For configuration $(s_0,0)$, there is transition $(s_0 0,a,1,s_0 1)$. Therefore, we add transition $(s_0 0,a,\text{-},s_0 0)$ to *FF-Predicate*. For configuration $(s_0,1)$, there is transition $(s_0 1,a,2,s_0 2)$. Therefore, we add transition $(s_0 1,a,\text{-},s_0 1)$. We consider each transition of EM_1 in Fig. 1 and obtain *FF-Predicate* shown in Fig. 4.

A transition of an implementation EFSM *EM* has an *assignment fault* if it has an action other than that specified by the transition. Consider a transition $t = (s,x,P,op,y,up,s')$ of the specification EFSM *EM*. If t has an assignment fault, then

after the execution of t, the context vector can be any vector of the set D_V except that specified by t. Thus, the corresponding FSM has a transition ($s\mathbf{v}$,$x\boldsymbol{\rho}$,$y\boldsymbol{\omega}'$,$s'\mathbf{v}'$) for each pair ($s\mathbf{v}$, $x\boldsymbol{\rho}$) $\in$ $Dom(P)$ and each $\mathbf{v}' \in D_V$ and $\boldsymbol{\omega}' \in D_{out\text{-}y}$[1] .

	$(s_0;0)$	$(s_0;1)$	$(s_0;2)$	$(s_1;0)$	$(s_1;2)$	$(s_1;1)$
a	$(s_0;1)/1$ $(s_0;0)/-$	$(s_0;2)/2$ $(s_0;1)/-$	$(s_1;0)/0$ $(s_0;2)/-$	$(s_1;2)/2$ $(s_1;0)/-$	$(s_0;1)/1$ $(s_1;2)/-$	$(s_0;1)/1$ $(s_1;1)/-$
b	$(s_0;0)/0$	$(s_0;1)/1$	$(s_0;2)/2$	$(s_1;0)/0$ $(s_1;0)/-$	$(s_0;0)/0$ $(s_1;2)/-$	$(s_1;1)/1$ $(s_1;1)/-$

Fig. 4. The *FF-Predicate* that corresponds to the predicate faults of EM_1 of Fig. 1

Thus, when deriving the *FF-Assignment*, for each transition ($s\mathbf{v}$,$x\boldsymbol{\rho}$,$y\boldsymbol{\omega}'$,$s'\mathbf{v}'$) of the FSM corresponding to the specification EFSM we include into *FF-Assignment* a transition ($s\mathbf{v}$,$x\boldsymbol{\rho}$,$y\boldsymbol{\omega}'$,$s\mathbf{v}'$) for each possible context vector $\mathbf{v}'$ and each possible output vector $\boldsymbol{\omega}'$. Similar to the *FF-transfer*, the set of domains $Dom(P_t)$ over all transitions of the *EM* is a partition of the set of all pairs "state-input" of the corresponding FSM.

As an example, consider transition t_2 of EM_1 in Fig 1. Under the input a, the predicate of t_2 is true for the configurations $(s_0,0)$ and $(s_0,1)$. For configuration $(s_0,0)$, w=$up(0)$= 1 and other possible values of w are 0 and 2. Thus, we add transitions $(s_0 0,a,0,s_0 0)$ and $(s_0 0,a,2,s_0 2)$ to *FF-assignment*. Moreover, for $(s_0,1)$, w =$up(1)$=2 and other possible values of w are 0 and 1. Thus, we add transitions $(s_0 1,a,0,s_0 0)$ and $(s_0 1,a,1,s_0 1)$. Domain $Dom(P_2)$ comprises the pairs $(s_0 0,a)$ and $(s_0 1,a)$. We consider each transition of EM_1 of Fig. 1 and obtain *FF-Assignment* shown as in Fig. 5 below.

	$(s_0;0)$	$(s_0;1)$	$(s_0;2)$	$(s_1;0)$	$(s_1;2)$	$(s_1;1)$
a	$(s_0;1)/1$ $(s_0;0)/0$ $(s_0;2)/2$	$(s_0;2)/2$ $(s_0;1)/1$ $(s_0;0)/0$	$(s_1;0)/0$ $(s_1;1)/1$ $(s_1;2)/2$	$(s_1;2)/2$ $(s_1;0)/0$ $(s_1;1)/1$	$(s_0;1)/1$ $(s_0;0)/0$ $(s_0;2)/2$	$(s_0;1)/1$ $(s_0;0)/0$ $(s_0;2)/2$
b	$(s_0;0)/0$	$(s_0;1)/1$	$(s_0;2)/2$	$(s_1;0)/0$	$(s_0;0)/0$ $(s_0;1)/1$ $(s_0;2)/2$	$(s_1;1)/1$

Fig. 5. The *FF-Assignment* that corresponds to assignment faults of EM_1 of Fig.1

Proposition 2. A submachine of *FF-assignment* is *feasible* only if for each two transitions of the same predicate domain the context parts of the next configurations and the output vectors are updated in the same way.

4 Removing Sub-machines from a Given Fault Function

In this section, we present a new method for reducing the FFs. This is done by removing from each FFs those sub-machines (i.e. possible faulty implementations) whose output responses to the test cases of the given *TS* disagree with those observed by applying these test cases to the SUT.

[1] Here we assume the specification domain of each output parameter is finite.

Algorithm 1. Removing from FSM *A* those sub-machines that do not match the set *V* of deterministic I/O sequences

***Input*:** A non-deterministic FSM $A = (S, I, O, h, s_0)$ representing a given *FF*, and a set *V* of deterministic sequences over alphabet $(IO)^*$.

***Output*:** The smallest sub-FSM $A^r = (S, I, O, h, s_0)$ of *A*, that contains all sub-machines of *A* whose traces include *V*, if such an A^r exists.

Step-1. Given FSM *A* and the set *V* of deterministic sequences over alphabet $(IO)^*$, we derive, using *A* and the sequences of *V*, the set of all families of deterministically compatible paths labelled with the sequences of *V* that satisfy the feasibility requirements of the FF *A*. Each family is represented as a path in a tree *Tree*. We build *Tree* as follows: Starting from the initial state s_0 of *A* (the root node of *Tree*), we select a sequence from *V* and derive all deterministic paths of FSM *A* that agree with the selected sequence and satisfy the feasibility requirements of *A*. We then, select a new sequence from *V*, and instead of deriving all deterministic paths for the new sequence starting again from the root node of *Tree*, as done in [1], we start from the leaf node of each path that is associated with the initial state s_0. Moreover, each new path has to be deterministically compatible with the path it appends. If a family of deterministically compatible paths does not exist for a given set *V*, then there is no sub-machine in *A* that has *V* as a subset of its set of traces. In this case, A^r does not exist; end of Algorithm 1.

Step-2. For each deterministic path $Path_j$ of the *Tree,* we derive the corresponding FSM B_j by copying the transitions of this path. Moreover, for each pair (s,i) such that the path has no transition from state *s* under input *i*, we copy into B_j all corresponding transitions from the original FF *A*. Then, we obtain the FSM A^r as the union of all FSMs B_j over all deterministic paths of *Tree*.

Step-3: Remove from the obtained machine A^r all states that are not reachable from the initial state of A^r.

As an application example, consider the *FF-Transfer* shown in Fig. 3, and the set *V*={*a*/1*a*/2*a*/0*a*/2*a*/2, *b*/0*a*/1} of I/O sequences of a given SUT. At Step-1, using the observed behavior of the SUT and the set *V*, we derive a Tree of all deterministic compatible paths that satisfy the *FF-Transfer* feasibility constraints.

For example, according to *FF-Transfer*, if t_2 has a transfer fault, then from the root node $(s_0,0)$ and under input *a*, the machine moves to state $(s_1,1)$ and produces the output 1. However, we do not consider the paths from the ending node $(s_1,1)$ since the observed output of the SUT after applying the second input symbol *a* is 2, while according to *FF-Transfer*, any submachine at state $(s_1,1)$ produces 1 to the input *a*. For the same reason, after obtaining the output 1202 to the input sequence *aaaa*, we do not consider the paths from the ending node of the transition $(s_0,2)$-*a*/0->$(s_0,0)$. Finally, we observe that according to *FF-Transfer*, a sub-machine of the FF that produces 1202 to the input sequence *aaaa* can be either at state $(s_1,2)$ or $(s_0,2)$. At each of these states only the output 1 can be produced for the next input *a*, while due to the observed behavior the SUT produces the output 2. Thus, we eliminate *FF-Transfer* since there is no sub-machine in *FF-Transfer* that has *V* as a subset of its traces, i.e. the SUT has no transfer faults. We apply Algorithm 1 to *FF-Predicate* and assure the SUT has no predicate faults. It is therefore expected to have assignment fault(s).

Then, we apply Algorithm 1 to *FF-Assignment* of Figure 5 and we obtain the machine shown in Fig. 6 below. Now we locate the faulty transition $(s_1 2,a,2,s_0 2)$ that

	$(s_0;0)$	$(s_0;1)$	$(s_0;2)$	$(s_1;0)$	$(s_1;2)$
a	$(s_0;1)/1$	$(s_0;2)/2$	$(s_1;0)/0$	$(s_1;2)/2$	$(s_1;2)/2$
b	$(s_0;0)/0$	$(s_0;1)/1$	$(s_0;2)/2$	$(s_1;0)/0$	$(s_0;0)/0$ $(s_0;1)/1$ $(s_0;2)/2$

Fig. 6. Machine obtained after applying Algorithm 1 to *FF-Assignment* of Fig. 5

corresponds to the assignment fault of transition t_6 of the reference EFSM. The context variable is updated to w:=2 instead of w:=1.

5 Other Steps of the Diagnostic Method

In this section we derive an identifier of a given SUT based on an adaptive experiment with the SUT. We refer to an implementation identifier as an *adaptive diagnostic test suite* and we propose a method for the derivation of an identifier of a given SUT. We recall that the identifier can be derived up to the set of equivalent implementations that are equivalent to the SUT. In this case, we know that the fault corresponds to one of remaining equivalent implementations.

According to [3], if there exists a distinguishing set, say *DIS*, for two given FFs, say *FF-predicate* and *FF-assignment*, then after applying the sequences of *DIS* to the SUT and observing corresponding output responses, we can always determine if the SUT is a submachine of *FF-predicate* or *FF-assignment*, or if it is not a submachine of any of them. Therefore, if the machines *FF-predicate* and *FF-transfer* and *FF-assignment* are pair-wise distinguishable we can always identify the type of the fault.

However, we do not need to derive a complete distinguishing set based on the intersection of two FFs. Instead, we state simpler sufficient conditions for the derivation of test cases that allow us to reduce two given FFs. These conditions are also applicable when the two FFs have no distinguishing set.

Theorem 1. Given two FFs FF_1 and FF_2, let state *st* be a deterministically reachable state of the intersection $FF_1 \cap FF_2$ through an input sequence β, such that the sets of outputs of FF_1 and FF_2 at states *s* and *t* under some input *a* do not coincide. Then after observing the output response of the SUT to the last input symbol *a* of $r.\beta.a$ we can delete from FF_1 and FF_2 the outgoing transitions of states *s* and *t* with inputs *a* and outputs that do not match the observed output of the SUT. If for *s* or *t* all outgoing transitions with an appropriate input label *a* are removed we can delete FF_1 or FF_2.

Hereafter, we assume that the type of a faulty transition is already identified, i.e. only one fault function FF remains. Now, to locate a faulty sub-machine (implementation) of the remaining FF, we distinguish between the different deterministic submachines of FF. This can be done by deriving input sequences as described in the following theorem. These sequences then are applied to the SUT in order to reduce the FF according to the observed behavior. Afterwards, if only a single implementation remains, then the fault is uniquely located, else if a number of equivalent implementations remain, then we can locate the fault up to the set of equivalent implementations (equivalent faults).

Theorem 2. Given an *FF*, let state *s* of *FF* be deterministically reachable through an input sequence β. If there exists an input *a* such that *FF* has at least two different output responses to *a* then after observing the response of the SUT to the last input

symbol a of $r.\beta.a$ we can delete the outgoing transitions of s with input labels a and outputs different than the observed one. Otherwise, if there exist some outgoing transitions of s labeled with the same input/output a/o such that in FF the destination states of these transitions are separable by the sequence γ, then after observing the output responses of the SUT to the sequence γ of $r.\beta.a.\gamma$ we can delete the outgoing transitions of s that are labeled with a/o and have destination states with outputs to γ different than the observed ones.

As an application example, we consider the *FF-assignment* of Figure 6 obtained after applying Algorithm 1. The state $(s_1,2)$ is deterministically reachable through the I/O sequence $a/1a/2a/0a/2$. Moreover, there are different output responses to the input b at state $(s_1,2)$. Therefore, the second fault can be identified after applying the input sequence $aaaa.b$ to the SUT. If the output response of the SUT to the last input symbol of this sequence is 2, then the second faulty transition is $(s_1 2,b,2,s_0 2)$. This transition corresponds to the assignment fault of t_7, where the context variable is updated as w:=2 instead of w:=0.

We note that if the FFs cannot be further reduced based on the above two theorems, then similar to [1], we suggest to divide these FFs into several FFs by fixing some of its transitions as deterministic transitions. In the worst case, we may need to explicitly enumerate all deterministic sub-machines of a given FF. However, our preliminary experiments show that the FFs obtained after applying Algorithm 1 are almost deterministic.

6 Conclusions

In this paper we have proposed an original method for the fault diagnosis in Extended Finite State Machines (EFSMs). The method assumes that an implementation EFSM is complete, consistent and it can have either predicate, transfer, or assignment faults. However, our ability to locate the fault is known to essentially depend on the observability (i.e. outputs produced) of the implementation EFSM. In software implementations, it is easy to increase the observability by reading some internal variables (eg. some context and state variables). Currently we are investigating the problem of determining the minimum number of variables that we need to observe in order to locate the faulty transitions. Moreover, we are adapting the method for inconsistent implementation EFSMs and we are experimenting with the method to assess its applicability to realistic application examples.

References

1. El-Fakih, K., Yevtushenko, N., Bochmann, G.: Diagnosing Multiple Faults in Communicating Finite State Machines. In Proc. of the IFIP 21st FORTE, Korea (2001) 85-100
2. Ghedamsi, A. Bochmann. G.: Test Result Analysis and Diagnostics for Finite State Machines. Proc. of the 12-th ICDS, Yokohama Japan (1992)
3. Koufareva, I.: Using Non-Deterministic FSMs for Test Suite Derivation. Ph.D. Thesis, Tomsk State University, Russia (2000)

4. Lee, D., Sabnani, K.: Reverse Engineering of Communication Protocols. Proc. of ICNP, October (1993) 208-216
5. Petrenko, A., Boroday, S., Groz, R.: Confirming Configurations in EFSM'. Proc. of the IFIP joint conference on FORTE XII and PSTV XIX, China (1999)
6. Petrenko, A., Yevtushenko, N.: Test Suite Generation for a FSM with a Given Type of Implementation Errors. Proc. of the 12th Int. Workshop Protocol Specification, Testing and Verification, (1992)
7. Starke, P.: Abstract Automata. North-Holland/American Elsevier (1972)
8. Yevtushenko, N., Koufareva, I.: Relations Between Non-Deterministic FSMs. Tomsk, Spectrum (2001)