On FSM-based Fault Diagnosis

Zoltán Pap[1], Gyula Csopaki[1] and Sarolta Dibuz[2]

[1] Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics**
Magyar tudósok körútja 2, H-1117, Budapest, HUNGARY
{pap, csopaki}@tmit.bme.hu,
[2] Ericsson, Armborstvägen 14. P.O.B. 1505, 125 25 Älvsjö, Stockholm
sarolta.dibuz@ericsson.com

Abstract. We study the problem of fault diagnosis, i.e., localization of difference(s) between an implementation and a specification in systems modelled by finite state machines. We show that even considering only a single fault in a finite state machine there are some situations when the exact diagnosis of the fault cannot be assured. We give an algorithm for fault diagnosis. If it is possible the procedure exactly locates a single fault, and in case exact localization is unfeasible it provides the set of all potential differences between the implementation and the specification.

Keywords: finite state machine, fault diagnosis, fault localization, output fault, transfer fault

1 Introduction

Conformance testing provides the means to check whether a system behaves according to its specification. Given an implementation, which is a black box – i.e., we can only observe its input/output behavior – and the specification of the system, we test if the implementation conforms to the specification. In case the specification is given as a finite state machine we want to determine whether there are difference(s) between the behavior function of the specification and the implementation machines.

Fault diagnosis – in contrast – addresses the more complex problem of locating the difference(s) between the protocol specification and an implementation if they are found to be different. A solution to this problem has various applications [1]. One of the most important being the correction of a protocol implementation so that it conforms to its specification.

Much research has been done concerning fault diagnosis for different formalisms [2] [3], and using different restrictions on the cardinality of faults . All papers on fault diagnosis in FSMs are considering a fault model with two types of changes between the implementation and the specification: output faults and

** This research is supported by Inter-University Centre for Telecommunications and Informatics (ETIK)

transition faults. A number of papers are using the assumption, that the implementation contains only one – transition or output – fault. There are heuristic procedures presented for diagnosis of single faults in FSMs (finite state machines) [4], and in CFSMs (communicating finite state machines) [5] [6]. An exact fault localization procedure is reported by D. Lee and K. Sabnani capable of locating a single fault in a finite state machine [7]. Limiting the number of differences between the specification and the implementation to a single fault, all of these papers claim to guarantee the precise localization of the difference.

Other contributions consider the case of multiple faults. Procedures for diagnosing multiple faults in FSMs and CFSMs were also reported [8] [9]. These algorithms are not always able to locate the multiple faults of the implementation [9]. The multiple fault diagnosis method for FSMs only guarantees the correct diagnosis of certain configurations of faults in an implementation, which are characterized by a certain type of independence of the different faults [8].

In this paper we concentrate on the case of a single transition or output fault in an FSM. We show that reduced implementation machines with different single faults may have the same observable behavior, and consequently – contrary to the statements found in the literature ([4] [7]) – it is in general not possible to guarantee the precise localization of a single fault in a finite state machine.

We determine a set of sufficient conditions for the guaranteed exact localization of a single output or transfer fault. Based on the analytical results we give an algorithm, a modified version of Lee's procedure [7], for the fault diagnosis problem. If it is possible, the method exactly locates the difference between the implementation and the specification, and in case exact localization is unfeasible it provides the set of all potential single faults.

The rest of the paper is organized as follows. Section 2 provides the definitions of basic terms and notations used in the paper. In Section 3 we show that in some cases fault diagnosis fails to exactly locate a single fault in a finite state machine. In Section 4 we investigate the conditions for guaranteed the exact localization of a single fault in an FSM. In Section 5 we give an algorithm for fault diagnosis, and finally summarize our work in Section 6.

2 Preliminaries

A finite state machine can be used to model a software system. Many specification languages, such as SDL [10] and ESTELLE [11], are extensions of the FSM formalism. Specifications in such languages may be converted into FSMs from which tests can be generated [12]. Finite state systems produce outputs on their state transitions after receiving inputs. A finite state machine A is a 4-tuple (I, O, S, h) where

- I is the finite set of input symbols,
- O is the finite set of output symbols,
- S is the finite set of states,
- h: $D \rightarrow 2^{O \times S} \setminus \varnothing$ is a behavior function where $D \subseteq S \times I$ is the specification domain and $2^{O \times S}$ is the set of all subsets of the set $S \times O$.

In case the specification domain $D = S \times I$, the behavior function is defined for all state-input combinations and the FSM A is said to be completely specified (or completely defined) what we assume for the rest of the paper.

If for each pair $(s, i) \in D$ it holds that $|h(s, i)| = 1$ then FSM A is said to be deterministic. In case of a deterministic FSM instead of behavior function h we use two functions, the transition function δ: $S \times I \rightarrow S$ and the output function λ: $S \times I \rightarrow O$.

For the rest of the paper, we will focus on completely specified and deterministic machines.

FSM A is said to be strongly connected, if for each pair of states (s_j, s_l), there exists an input sequence which takes A from s_j to s_l.

An FSM can be represented by a state transition diagram, a directed graph whose vertices correspond to the states of the machine and whose edges correspond to the state transitions. Each edge is labeled with the input and output associated with the transition. Supposing that the machine is currently in state s_3 and upon input c the machine moves to state s_2 and outputs 1. This transition can be written in a form $s_3 \xrightarrow{c/1} s_2$.

We extend the transition function δ and output function λ from input symbols to finite input sequences (strings) I^* as follows: For a state s_1, an input sequence $x = i_1, ..., i_k$ takes the machine successively to states $s_{j+1} = \delta(s_j, i_j), j = 1, ..., k$ with the final state $\delta(s_1, x) = s_{k+1}$, and produces an output sequence $\lambda(s_1, x) = o_1, ..., o_k$, where $o_j = \lambda(s_j, i_j), j = 1, ..., k$. The input/output sequence $i_1 o_1 i_2 o_2 ... i_k o_k$ is then called a trace of M. Note that since the FSMs in our model are deterministic all their traces are deterministic, because there are no transitions with different next states and/or outputs for the same state-input combination.

Finite state machines may contain redundant states. State minimization is a transformation into an equivalent state machine to remove redundant states.

Two states are equivalent written $s_j \cong s_l$ if and only if for every input sequence the machine will produce the same output sequence regardless of whether s_j or s_l is the starting state. In other words, for all input sequences $x \in I^*$, $\lambda(s_j, x) = \lambda(s_l, x)$. (Note that their succeeding states for a particular input sequence are also pairwise equivalent).

Two states s_j and s_l are distinguishable (inequivalent) if there exists a finite input sequence x which when applied to FSM M causes different output sequences starting in either state. In other words $\exists x \in I^*$, $\lambda(s_j, x) \neq \lambda(s_l, x)$. Such an input sequence is called a separating sequence of the two inequivalent states. If the shortest such sequence is of length k then (s_j, s_l) are k-distinguishable. A FSM M is reduced (minimized), if no two states are equivalent, that is, each pair of states (s_j, s_l) are distinguishable.

Machine equivalence is an equivalence relation on all FSMs with the same input and output sets.

Completely specified deterministic FSMs $M_1 = (I, O, S, \delta, \lambda)$ and $M_2 = (I, O, S', \delta', \lambda')$ are equivalent written $M_1 \cong M_2$ if their sets of traces coincide.

From an other point of view two machines M_1 and M_2 are equivalent if and only if for every state in M_1 there is at least one corresponding equivalent state in M_2, and vice versa.

A homomorphism from M_1 to M_2 is a mapping ϕ from S to S' such that for every state $s \in S$ and for every input symbol $i \in I$, it holds that $\delta'(\phi(s), i) = \phi(\delta(s, i))$ and $\lambda'(\phi(s), i) = \lambda(s, i)$ [13]. If ϕ is a bijection, then it is called an isomorphism. In this case M_1 and M_2 must have the same number of states, and they are identical except for a renaming of states. Two machines are called isomorphic if there is an isomorphism from one to the other. Two isomorphic FSMs are equivalent, but the converse is not true in general.

In each equivalence class there is a reduced machine with the minimal number of states. In an equivalence class, any two reduced machines have the same number of states, furthermore, there is a one-to-one correspondence between equivalent states, which gives an isomorphism between the two machines. That is, the reduced machine in an equivalence class is unique up to isomorphism.

Note that there is a number of equivalence relations of states of machines. They are, however, all the same for completely specified and deterministic machines, and they are only different in case of more general machines like nondeterministic machines.

We say that machine M has a reset capability if there is an initial state $s_0 \in S$ and an input symbol $r \in I$ that takes the machine from any state back to s_0. That is, $\delta(s_j, r) = s_0$ for all states $s_j \in S$. The reset is reliable if it is guaranteed to work properly in any implementation machine M', i.e., $\delta'(s'_j, r) = s'_0$ for all states $s'_j \in S'$, otherwise it is unreliable. Note that reset r is also an input symbol. Thus, if M has reset then M is considered to be strongly connected if all the other states can be reached from the initial state s_0.

According to the previous works on fault diagnosis, we are considering a fault model with two types of faults: the output and the transition fault. We say that a transition has an output fault if, for the corresponding state and received input, the implementation provides an output different from the one specified by the output function. We say that a transition has a transition fault if, for the corresponding state and received input, the implementation enters a different state than specified by the transition function. An implementation has a single output (transition) fault if one and only one of its transitions has an output (transition) fault.

3 Failure of Exact Fault Diagnosis in FSMs

We show that even in the most 'simple' case it is not always possible to solve the fault localization problem. That is, even when considering the strictest assumptions – a single fault in a finite state machine (investigated by Ghedamsi et al. [4] and Lee at al. [7]) – there are some situations where the exact localization of the fault cannot be assured.

For the rest of the paper we will consider a specification finite state machine $Spec = (I, O, S, \delta, \lambda)$. We denote the number of states, inputs, and outputs by

$n = |S|$, $p = |I|$, and $q = |O|$, respectively. We also consider implementation machines $Impl_a = (I, O, S', \delta', \lambda')$, $Impl_b = (I, O, S'', \delta'', \lambda'')$ and so on with the same input and output sets, and the same number of equally labeled states. We use the term "same states" written $s'_j = s''_j$ for states that are labeled alike in different machines. Of course, these states are not necessary equivalent written $s'_j \cong s''_j$.

Obviously, without any assumptions conformance testing and fault diagnosis are impossible problems; for any test sequence we can easily construct a machine M_2, which is not equivalent to M_1 but produces the same outputs as M_1 for the given test sequence. There is a number of natural assumptions that are usually made in the literature in order for the test to be at all possible [1]:

- The specification FSM *Spec* is deterministic, completely specified, strongly connected and reduced.
- Implementation machines do not change during the experiment, and have the same input I and output O alphabet as *Spec*.

Furthermore we concentrate on systems with reliable reset capability, and we assume that there is only one difference – an output or a transition fault – between an implementation and the specification machine.

All previous works on the diagnosis of a single fault in a FSM ([4] [7]) used the same assumptions, and they claim to provide methods to precisely locate the single fault.

We show that – contrary to the statements found in the literature – it is in general not possible to guarantee the precise localization of a single fault in a finite state machine, not even considering the assumptions above: Take specification machine *Spec* and two implementation machines: $Impl_a$ differing from *Spec* by a single fault $Fault_a$, and $Impl_b$ differing from the specification by a single fault $Fault_b$. $Fault_a$ and $Fault_b$ are different faults. Evidently, neither $Impl_a$ nor $Impl_b$ can be equivalent to *Spec*, since *Spec* is deterministic, completely specified, strongly connected and reduced. Interestingly, however, $Impl_a$ and $Impl_b$ might be equivalent to each other even though the faults they contain differ. In this case it is impossible to decide between the faults, i.e., it is impossible to exactly locate the fault. The next simple example demonstrates the situation.

Example 1. Take specification machine *Spec* shown on Figure 1. The set of input symbols is $I = \{a, r\}$, where r is the reset input, the set of output symbols is $O = \{1, 2\}$ and the set of states is $S = \{s_0, s_1, s_2\}$ where s_0 is the initial state. Specification machine *Spec* is deterministic, completely specified, strongly connected and reduced. Note that the (reliable) reset transitions are omitted on the figure for the sake of perspicuity. Let us consider two implementation machines $Impl_a$ on Figure 2(a) and $Impl_b$ on Figure 2(b) with the same input and output alphabet as *Spec*.

The difference between $Impl_a$ and *Spec* is a a single transition fault at state s_0, $Fault_a : \delta'(s'_0, a) = s'_0$ instead of s'_1. In case of $Impl_b$ the difference is a single output fault at s_2, $Fault_b : \lambda''(s''_2, a) = 1$ instead of 2.

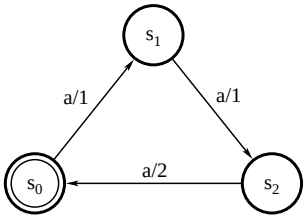


Fig. 1. Specification machine *Spec*.

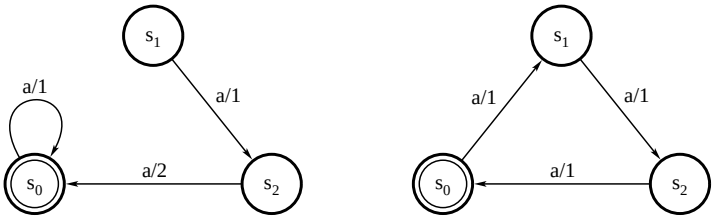


Fig. 2. Faulty implementation machines: (a) $Impl_a$ contains a single transition fault at state s_0, (b) $Impl_b$ contains an output fault at s_2.

The two implementation machines are equivalent, as they both produce the same trace for every input string. Thus, it is impossible to distinguish between them and therefore between the two faults. Or, to formulate more precisely, is it is impossible to distinguish among any faulty implementation machines belonging to the same equivalence class, and therefore among the faults that they contain.

4 Conditions for Guaranteed Fault Diagnosis

We determine a set of sufficient conditions for the guaranteed exact localization of a single output or transfer fault. That is, we analyze when two (or more) implementation machines, each differing from the specification by a single dissimilar fault, cannot be equivalent. Note that we still consider the assumptions made in the previous section, therefore the specification FSM *Spec* is deterministic, completely specified, strongly connected and reduced with reliable reset.

First we show that it is always possible to distinguish two different output faults if the specification machine has reliable reset capability.

Lemma 1. *Suppose that the specification machine under consideration is deterministic, completely specified, strongly connected and reduced with reliable reset capability. Two implementation machines, each differing from the specification by a single and dissimilar output fault, cannot be equivalent, thus any two output faults can be distinguished.*

Proof. Let us consider two implementation machines $Impl_a$ and $Impl_b$, both differing from the specification *Spec* by a single dissimilar output fault. We reset the machines and start to explore the state-space of the two implementations and the specification in parallel. Clearly, until we reach a faulty transition in one of the machines, for any input string x, the traversed states – and the output sequences – are the same in the two implementations and the specification: $\delta'(s'_0, x) = \delta''(s''_0, x) = \delta(s_0, x)$, and $\lambda'(s'_0, x) = \lambda''(s''_0, x) = \lambda(s_0, x)$. When we traverse a faulty transition in one of the implementations (let's say $Impl_a$) with an input string y, we find an inconsistency between $Impl_a$ and *Spec*: $\lambda'(s'_0, y) \neq \lambda(s_0, y)$. However, since $Fault_a \neq Fault_b$ the output of $Impl_b$ at the given transition also cannot be equivalent to the output of $Impl_a$. Therefore, $\lambda'(s'_0, y) \neq \lambda''(s''_0, y)$, i.e., $Impl_a$ and $Impl_b$ are inequivalent, and input sequence y can distinguish them.

Note that the statement made in Lemma 1 only holds if the specification machine has reliable reset capability. We demonstrate a counter-example of Lemma 1 in case the specification machine does not have reliable reset capability.

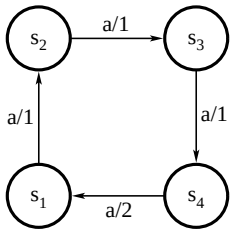
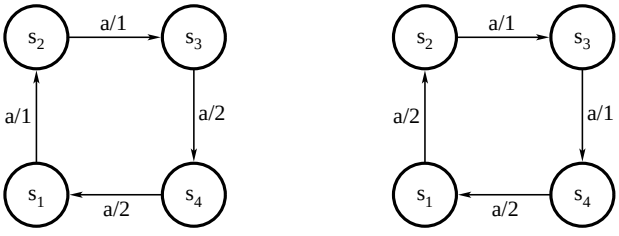


Fig. 3. Specification machine *Spec* without reliable reset capability.



(a) $Impl_a$ (b) $Impl_b$

Fig. 4. Faulty implementation machines: (a) $Impl_a$ contains a single output fault at state s_3, (b) $Impl_b$ contains an output fault at s_1.

Example 2. Take specification machine *Spec* shown on Figure 3

The set of input symbols is $I = \{a\}$, the set of output symbols is $O = \{1, 2\}$ the set of states is $S = \{s_1, s_2, s_3, s_4\}$.

Specification machine *Spec* is deterministic, completely specified, strongly connected and reduced. Take two implementation machines $Impl_a$ on Figure 4(a) and $Impl_b$ on Figure 4(b) with the same input and output alphabet as *Spec*.

The difference between $Impl_a$ and *Spec* is a a single output fault at state s_3, $Fault_a : \lambda'(s'_3, a) = 2$ instead of 1. In case of $Impl_b$ the difference is a single output fault at s_1, $Fault_b : \lambda''(s''_1, a) = 2$ instead of 1. The two implementation machines are clearly equivalent.

Next we show that it is always possible to distinguish a single output and a single transition fault if the *faulty machines* are reduced.

Lemma 2. *Suppose that the specification machine under consideration is deterministic, completely specified, strongly connected and reduced with reliable reset capability. Two implementation machines, one differing from the specification by a single output fault, the other by a single transition fault cannot be equivalent if the implementation machines are reduced.*

Proof. Let us consider two implementations $Impl_a$ and $Impl_b$. One of the implementations (let's say $Impl_a$) contains a transition, the other ($Impl_b$) an output fault. The two implementation machines are reduced therefore they must be isomorphic to be equivalent. That is, there has to be a one-to-one mapping ϕ from S' to S'' such that for every state s' in S' and for every input symbol i in I, $\delta''(\phi(s'), i) = \phi(\delta'(s', i))$ and $\lambda''(\phi(s'), i) = \lambda'(s', i)$ should hold.

Let's say the output fault in $Impl_b$ is at s''_x. Since the output of one of its transitions has changed s''_x has to map to an other state (say s'_y) of $Impl_a$ where $\lambda''(s''_x, i) = \lambda'(s'_y, i),\ \forall i \in I$. However, this mapping cannot be one-to-one, as there is no output fault in $Impl_a$, and therefore $Impl_a$ has one less states with the same output characteristic as s''_x. Thus, there is clearly no one-to-one mapping ϕ fulfilling $\lambda''(\phi(s'), i) = \lambda'(s', i),\ \forall s' \in S',\ \forall i \in I$.

Finally we show that it is always possible to distinguish two different single transition faults if the *faulty machines* are reduced.

Lemma 3. *Suppose that the specification machine under consideration is deterministic, completely specified, strongly connected and reduced with reliable reset capability. Two implementation machines, each differing from the specification by a single and dissimilar transition fault, cannot be equivalent if the implementation machines are reduced.*

Proof. Let us assume the following situation:
$Fault_a$ in $Impl_a$: $(s_c \xrightarrow{i_l/o_f} s_d) \Rightarrow (s'_c \xrightarrow{i_l/o_f} s'_e)$
$Fault_b$ in $Impl_b$: $(s_u \xrightarrow{i_m/o_g} s_v) \Rightarrow (s''_u \xrightarrow{i_m/o_g} s''_w)$
First, take the special case when the two faults are applied to the same transition in the two implementations, i.e., $c = u$ *and* $l = m$. In this case the outputs are also the same ($f = g$) but the next states are not ($e \neq w$) because $Fault_a$ and $Fault_b$ are dissimilar. In this case there is only one difference between $Impl_a$

and $Impl_b$, and therefore, if the implementation machines are reduced ($s'_e \ncong s'_w$) we can certainly find an input sequence distinguishing them for example using Chow's method [14]: Let y be a separating sequence distinguishing states s'_e and s'_w. We apply an input sequence (say x) corresponding to the path of the tree from the initial state to s'_c, input i_l and then apply y. This input sequence $x \cdot i_l \cdot y$ certainly distinguishes $Impl_a$ and $Impl_b$, thus the implementation machines cannot be equivalent.

Now take the general case when the two faults are applied to different transitions. Let's reset the machines and start to explore the state-space of the two implementations and the specification in parallel. Until we reach a faulty transition in one of the machines for any input string the traversed states – and the output sequences – are the same in the two implementations and the specification. Let's say we first encounter $Fault_a$ in $Impl_a$ with an input string x, i.e., with x we reach the state s'_c in $Impl_a$, s''_c in $Impl_b$ and s_c in *Spec*. If we input i_l after x, $Impl_a$ will transit to s'_e, *Spec* to s_d and $Impl_b$ to s''_d. Let Y be the set of all separating sequences distinguishing states s_e and s_d. Any input sequence $x \cdot i_l \cdot y_j$ where $y_j \in Y$ will clearly distinguish $Impl_a$ and *Spec*. Any of these input sequences will also distinguish $Impl_a$ and $Impl_b$, except if all $y_j \in Y$ starting from s''_d in $Impl_b$ traverse $Fault_b$ making $\lambda(s'_0, x \cdot i_l \cdot y_j)$ and $\lambda(s''_0, x \cdot i_l \cdot y_j)$ consistent for all $y_j \in Y$. For that, also in *Spec* all separating sequences distinguishing states s_e and s_d starting from s_d traverse transition (s_u, i_m); and if $\delta(s_u, i_m) = s_v$ then s_e and s_d are separable, if $\delta(s_u, i_m) = s_w$ then they are not separable. From that it follows that s''_e and s''_d in $Impl_b$ are not separable. Thus, the implementation machines cannot be minimal.

Theorem 1. *Suppose that the specification machine under consideration is deterministic, completely specified, strongly connected and reduced with reliable reset capability. Two implementation machines, each differing from the specification by a single and dissimilar fault, cannot be equivalent if the implementation machines are reduced.*

Proof. The proof follows from Lemmas 1, 2 and 3.

The theorem shows that if there is only one difference between an implementation and a specification and the implementation is minimal then it is unique, no other fault can induce the same change in behavior. Thus it is possible to identify the given fault.

5 Exact Algorithm for Fault Diagnosis

We give an algorithm – a modification of Lee's method [7] – for the localization of single transfer or output faults in finite state machines. We incorporate the analytical results of Section 4. to quickly verify if the first fault candidate the algorithm identifies is certainly the only possible one. If it is we conclude that the difference between the implementation and the specification can be exactly located, otherwise the algorithm moves on and provides the set of all potential single faults.

Let us consider a specification finite state machine *Spec* and an implementation *Impl* to be diagnosed. The algorithm is made up of two steps:

Step 1 Conformance testing is used to determine if there is difference between the specification and the diagnosed implementation.
Step 2 Localization of the fault.

5.1 Step 1: Detection of the Fault

For Step 1 of the algorithm a checking sequence needs to be constructed.

Definition 1. *Let M be a finite state machine with n states and initial state s_0. A checking sequence for M is an input sequence x that distinguishes M from all other machines with n states. That is, any machine with at most n states not equivalent to M produces a different output than M on input x starting from the initial state.*

There are a number of conformance testing methods developed for finite state machines constructing checking sequences. These include the transition tour [15], the Unique Input Output (UIO) method [16], the Distinguishing Sequence method [17], the "W-method" [14] and the Wp method [18]. In our algorithm we create the checking sequence using the W-method proposed by Chow for machines with reliable reset. It consists of no more than pn^2 test sequences of length less than $2n$ interposed with reset. We apply the checking sequence to the specification and to the diagnosed implementation. If we do not find an inconsistency of the observed outputs then we conclude that the implementation machine is equivalent to the specification, thus there is either no fault in the implementation or there are more than one, and end of the algorithm. If we find a difference we move on to Step 2.

5.2 Step 2: Localization of the Fault

During conformance testing an inconsistency was found between the specification and the diagnosed implementation. Thus, there is at least one of the pn^2 test sequences (say x) detecting the fault, i.e. $\lambda(s_0, x) \neq \lambda'(s'_0, x)$. Let us assume that the earliest inconsistency between $\lambda(s_0, x)$ and $\lambda'(s'_0, x)$ is at the k^{th} output symbol where $1 \leq k \leq 2n$. Let's say that the first k elements (inputs) of x carry the specification machine from s_0 to $s_1, s_2, ..., s_k$, where these $k + 1$ states may or may not be different.

We assume *Impl* has only a single output or transition fault. In case the diagnosed implementation machine contains an output fault, x has to traverse the fault at the k^{th} transition. If *Impl* contains a transition fault, then x has to traverse the fault during the first $k - 1$ transitions.

Note that if there are more than one test sequences detecting the fault, we may use either of them for the localization of the fault (for practical reasons we should choose the shortest sequence). If multiple test cases detect the fault, we might also check if the set of possibly faulty transitions can be narrowed: For

each test sequence detecting the fault we determine the transitions it traverses in the specification machine prior to the first inconsistency. Trivially, in Step 2, we only have to consider the intersection of these traversed transitions.

In the algorithm we consider two cases. First we presume that the fault in $Impl$ is an output fault and verify if it's a potential candidate. If the verification succeeds, then we try to confirm whether it is the only potential candidate. If it is the only one, then we located the fault and end of algorithm. Otherwise we move on and presume that the fault could be a transition fault, and similarly analyze each possibilities. If at the end we don't find any potential candidates we conclude that there are more than one fault in $Impl$.

Output Fault We assume that the fault in $Impl$ is an output fault, i.e., $\lambda(s_{k-1}, x_k) \neq \lambda'(s'_{k-1}, x_k)$ where x_k is the k^{th} input of x. For the verification we modify *Spec* according to the supposed fault: we change the output symbol at state s_{k-1} upon input x_k to the faulty output symbol $\lambda'(s'_{k-1}, x_k)$. We denote the modified specification C_1. We conduct a checking experiment (conformance testing) on $Impl$ with respect to C_1.

C_1, however, is not necessarily minimal. To use Chow's method for checking sequence generation, we first have to minimize machine C_1 and get $C_{1\ reduced}$. Let m be the number of states of reduced machine $C_{1\ reduced}$. If $m < n$, that is, if the reduced conjectured machine has less states than the specification, then according to Chow we have to use a Z set instead of a W set for test sequence generation. A Z set can be created by extending the W set the following way [14]:

$$Z: \ W \ U \ I \cdot W \ U \ \ldots \ U \ I^{n-m} \cdot W$$

Where "U" is the union operator, "$\cdot$" is the string concatenation operator and I is the input alphabet. The checking sequence is then created by the concatenation of the sets of sequences P and Z.

If we find that $Impl$ conforms to C_1, we conclude that C_1 is a potential candidate. Then we try to confirm if it is the only possible candidate. For that we simply have to check if the reduced machine $C_{1\ reduced}$ has equivalent number of states to the specification. If it has, we conclude that C_1 is certainly the only potential candidate, and therefore we exactly located the fault in $Impl$, end of algorithm.

If $Impl$ conforms to C_1, but the reduced machine has less states than the specification, we conclude that C_1 is a potential candidate, store it in the set of potential candidate machines PC, and proceed to the following step.

If $Impl$ does not conform to C_1 we proceed to the following step.

Transition Fault A transition fault can occur in one of the first $k-1$ transitions, i.e., $\delta(s_j, x_{j+1}) \neq \delta'(s'_j, x_{j+1})$ where $j = 0, ..., (k-2)$. We assume, that the fault occurs in the j^{th} transition and verify each assumption in turn. On input x_{j+1} at state s_j the implementation machine is supposed to transit to s_{j+1}. But instead $Impl$ transits to s_r, where s_r can be any of the $n-1$ states except the right state s_{j+1}. We verify each possibilities in turn.

In each turn we modify *Spec* according to the supposed fault: we create candidate machine C_{l+1} where l is the cardinality of the set PC, by changing the next-state symbol of the given transition to the supposed wrong state s_r. We minimize the candidate machine and conduct a checking experiment on $Impl$ with respect to $C_{l+1\ reduced}$.

If we find that $Impl$ conforms to $C_{l+1\ reduced}$, we conclude that C_{l+1} is a potential candidate. If PC is not empty ($l \geq 1$), we store C_{l+1} in PC, conclude that the exact localization of the fault is not possible and move on to the next turn.

If $l = 0$ then we try to confirm if it is the only possible candidate. We simply check if the reduced machine $C_{l+1\ reduced}$ has equivalent number of states to the specification. If it has, we conclude that C_{l+1} is certainly the only candidate, and therefore we exactly located the fault in $Impl$, end of algorithm.

If $l = 0$ and $Impl$ conforms to C_{l+1}, but the reduced machine has less states than the specification, we conclude that C_{l+1} is a potential candidate, store it in set PC, and proceed to the next turn.

If $Impl$ does not conform to C_{l+1} we move on to the next turn.

For each assumed transition there are $n-1$ possible next states. Thus, there are no more than $2n^2$ turns. At the end of the last turn there are three possibilities:

- If $l = 0$, we conclude that there are more than one faults in $Impl$, end of algorithm.
- If $l = 1$, there is only one potential candidate, therefore we exactly located the fault in $Impl$, end of algorithm.
- If $l > 1$, we conclude that the exact localization of the fault is not possible, and PC is the set of all potential candidates i.e., we determined the set of all potential single faults, end of algorithm.

Example 3. We use an example to demonstrate the algorithm given above. Take the specification machine *Spec* shown on Figure 5(a) The set of input symbols is $I = \{a, b, r\}$, where r is the reset input, the set of output symbols is $O = \{1, 2\}$ and the set of states is $S = \{s_0, s_1, s_2, s_3\}$ where s_0 is the initial state. Reset transitions are again omitted on the figure. Implementation machine on Figure 5(b) contains a single transition fault at state s_2. This transition fault is to be located using the algorithm.

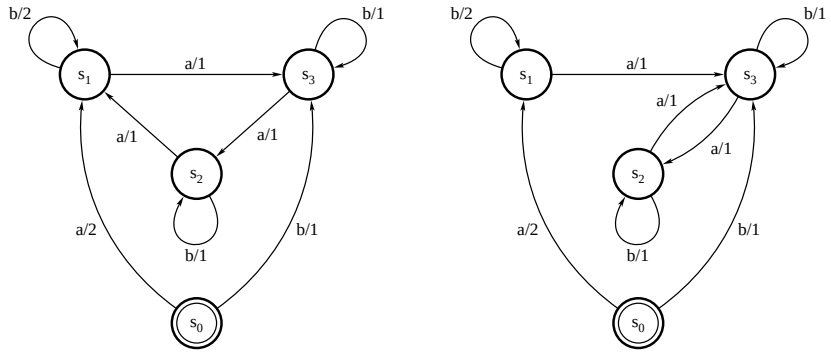
For the detection of the fault (step 1 of the algorithm) we need to construct a checking sequence. Since our emphasis is not on checking experiments, we omit the details. A P-set of *Spec* can be constructed based on a testing tree.

$$P: \ \{r, ra, raa, rab, rb, rba, rbb, rbaa, rbab\}$$

The characterizing set (W-set) of *Spec* is:

$$W: \ \{ab, b\}$$

By concatenating P and the characterizing set we get a basic test set of the checking sequence, interposed with reset. Obviously, if a prefix of a sequence can



(a) Specification machine *Spec* (b) Implementation machine *Impl*

Fig. 5. Faulty implementation machine *Impl* contains a single transition fault at state s_2.

detect a fault then the whole sequence also can. Thus, we can remove all the sequences that are prefix of other sequences. As a result we get the following test set:

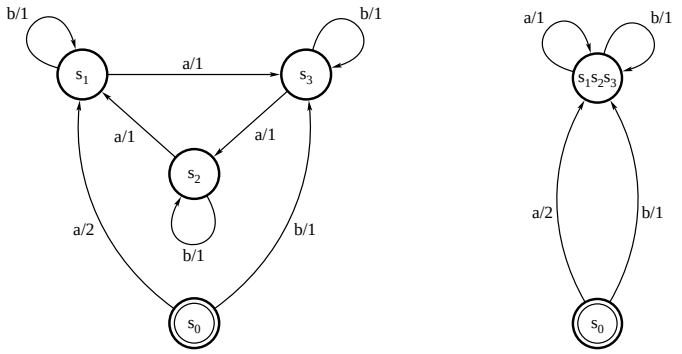
$$\{raaab, raab, rabab, rabb, rbaaab, rbaab^*, rbabab^*, rbabb, rbbab, rbbb\}$$

We execute the test set on *Impl*. The test sequences marked with * detect the fault. We use the shortest sequence – *rbaab* – for the rest of the algorithm. Note that we can not narrow the set of possibly faulty transitions, because in *Spec* sequence *rbabab* traverses all transitions that *rbaab* does.

If applied to *Spec* *rbaab* produces the output sequence 1112, and if applied to *Impl* we get 1111. That is, the fourth outputs are different ($k = 4$). First we presume that the fault in *Impl* is an output fault (occurring at the fourth transition). Since the sequence *rbaab* carries *Spec* from s_0 to s_3, s_2, s_1, s_1, we change the output at state s_1 input b from 2 to 1. We get the machine C_1 on Figure 6(a) We reduce C_1 and get the machine $C_{1\ reduced}$ on Figure 6(b) We conduct a checking experiment (conformance testing) on *Impl* with respect to $C_{1\ reduced}$. We find that the two machines are not equivalent (for example *rab* finds the difference), therefore, we move on and presume that the fault is a transition fault occurring in one of the first three $(k-1)$ transitions.

We first conjecture that the first transition goes to a different state than specified. We have three possibilities: at state s_0, on input b the machine goes to s_0, s_1 or s_2 instead of s_3. We build the according conjectured machines and verify them in turn. Omitting the details, we find that none of the machines conform to *Impl* (*rba*, *rbb* and *rbab* rule out the possibilities respectively).

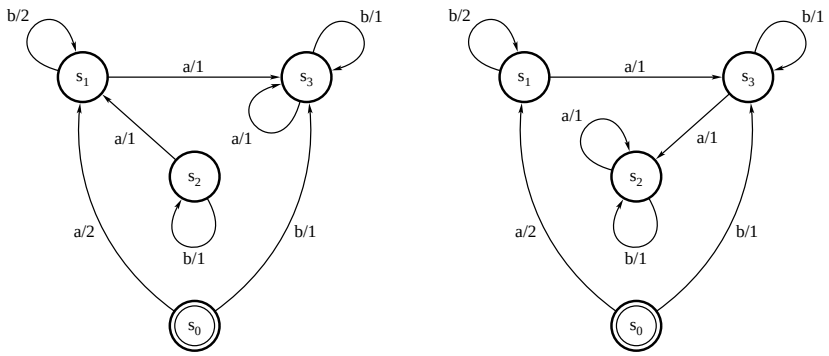
We move on and conjecture that the fault is at the second transition: at state s_1, on input a the machine goes to s_0, s_1 or s_3 instead of s_2. After building the machines and conducting the checking experiments we rule out the first two



(a) C_1 (b) $C_{1\ reduced}$

Fig. 6. Conjectured machine C_1 with an output fault at state s_1 (a), and C_1 after minimization (b).

possibilities with sequences *rbaa* and *rbab* respectively. We also find that the third conjectured machine (C_1 on Figure 7(a)) conforms to $Impl$. Since the set of potential candidate machines PC is empty, we try to confirm if it is the only possible candidate. We find that after minimization C_1 has less states than the specification. Thus, we conclude that C_1 is a potential candidate, store it in set PC, and proceed to the next turn. We conjecture that the fault is at the third



(a) C_1 (b) C_2

Fig. 7. Conjectured machines C_1 and C_2

transition: at state s_2, on input a the machine goes to s_0, s_2 or s_3 instead of s_1. The sequence *rbaaa* rules out the first possibility, but the other two conjectured

machines – C_2 (Figure 7(b)), and C_3 (Figure 5(b)) – conform to $Impl$. Since PC is not empty, we know that the exact localization of the fault is not possible and store both machines in PC. As $k = 4$, transition fault may only occur at the first three transitions, therefore, we have reached the end of the algorithm.

As a result we conclude that the exact localization of the fault is not possible. PC is the set of possible faulty machines C_1, C_2 and C_3 including all potential single faults.

6 Conclusion

We study the problem of fault diagnosis. The scope of fault diagnosis is beyond the scope of the fault detection (or conformance testing) problem. While the latter is concerned with determining if there are difference(s) between the behavior of the specification and the implementation machines, the former also tries to identify and locate the difference(s).

We concentrate on the diagnosis of a single transition or output fault in an FSM. Clearly, the problem cannot be exactly solved if there are two or more equivalent implementation machines, each differing from the specification machine by a single dissimilar fault. We show that implementation machines with different single faults may have the same observable behavior and thus in general it is not possible to guarantee the exact localization of a single fault in a finite state machine.

We analyze under what circumstances the exact localization of a single output or transfer fault can be guaranteed. That is, we determine a set of sufficient conditions when two (or more) implementation machines, each differing from the specification by a single dissimilar fault, cannot be equivalent. We incorporate the analytical results into an algorithm for the fault diagnosis problem. In case it is possible, the algorithm exactly locates the difference between the implementation and the specification, and when the exact localization is not possible, it provides the set of all potential single faults.

References

1. Lee, D., Yiannakakis, M.: Principles and methods of testing finite state machines – a survey. Proc. IEEE **43** (1996) 1090–1123
2. El-Fakih, K., Prokopenko, S., Yevtushenko, N., Bochmann, G.V.: Fault diagnosis in extended finite state machines. Testing of Communicating Systems - Proceedings of 15th IFIP International Conference, TestCom 2003 (2003)
3. Ghedamsi, A., Dssouli, R., Bochmann, G.V.: Diagnostic tests for single transition faults in non-deterministic finite state machines. Proceedings of the IFIP TC6/WG6.1 Fifth International Workshop on Protocol Test Systems V (1992)
4. Ghedamsi, A., v. Bochmann, G.: Test result analysis and diagnostics for finite state machines. Proc. 12th Int. Conf. on Distributed Systems (1992)
5. Ghedamsi, A., Bochmann, G., Dssouli, R.: Diagnosis for single transition faults in communicating finite state machines. IEEE International Conference on Distributed Computing Systems (ICDCS'93), Pittsburgh, USA (1993)

6. Ghedamsi, A., Dssouli, R., Bochmann, G.: Diagnosing distributed systems modeled by communicating finite state machines. Revue Reseaux et Informatique Repartie **3** (1993) 343–363
7. Lee, D., Sabnani, K.: Reverse-engineering of communication protocols. Proc. of the IEEE International Conference on Network Protocols, California (1993) 208–216
8. Ghedamsi, A., Bochmann, G., Dssouli, R.: Multiple fault diagnostics for finite state machines. IEEE INFOCOM'93 (1993)
9. El-Fakih, K., Yevtushenko, N., von Bochmann, G.: Diagnosing multiple faults in communicating finite state machines. Formal Techniques for Networked and Distributed Systems, FORTE 2001, IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems (2001) 85–100
10. ITU-T: Recommendation z.100: Specification and description language (2000)
11. TC97/SC21, I.: Estelle – a formal description technique based on an extended state transition model. international standard 9074 (1988)
12. Luo, G., Das, A., Bochmann, G.V.: Generating tests for control portion of sdl specifications. Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI (1993)
13. Moore, E.F.: Gedanken-experiments on sequential machines. In: Automata Studies. Princeton University Press, Princeton, N.J. (1956) 129–153
14. Chow, T.: Testing software design modelled by finite-state machines. IEEE Trans. Software Eng. **4** (1978)
15. Naito, S., Tsunoyama, M.: Fault detection for sequential machines by transition tours. Proc. of FTCS (Fault Tolerant Computing Systems (1981) 238–243
16. Sabnani, K., Dahbura, A.: A protocol testing procedure. Computer Networks and ISDN Systems **15** (1988) 285–297
17. Gonenc, G.: A method for the design of fault detection experiments. IEEE Trans. Computer **C-19** (1970) 551–558
18. Fujiwara, S., Bochmann, G.V., Khendec, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state model. IEEE Trans. Softrw. Eng. **17** (1991) 591–603