Multiple Fault Diagnostics for Finite State Machines

Abderrazak Ghedamsi, Gregor v. Bochmann and Rachida Dssouli

Université de Montréal, DIRO, C.P. 6128, Succ. A, Montréal, Canada, H3C 3J7.

Abstract

In this paper, we propose a generalized diagnostic algorithm for the case where more than one fault (output and/or transfer) may be present in the transitions of a system represented by a deterministic finite state machine (FSM). If existing faults are detected, this algorithm permits the generation of a minimal set of diagnoses, each of which is formed by a set of transitions (with specific types of faults) suspected of being faulty. The occurrence in an implementation, of all the faults of a given diagnosis, allows the explanation of all observed implementation outputs. The algorithm guarantees the correct diagnosis of certain configurations of faults (output and/or transfer) in an implementation, which are characterized by a certain type of independence of the different faults. We also propose an approach for selecting additional test cases, which allows the reduction of the number of possible diagnoses. A simple example is used to demonstrate the different steps of the algorithm.

1. Introduction

Systematic test sequence generation for communication protocols in conformance testing has been an active research area during the last decade. Methods were developed to produce optimized test sequences for detecting faults in an implementation under test (IUT). Most of these methods are based on deterministic finite state machine (FSM) models [4, 7, 10, 9, 11, 13, 14]. They are intended to determine whether a given protocol implementation satisfies all properties required by the protocol specification. However, the application of these methods gives only limited information about the locations of detected faults. In general, in the communication protocol area, very little work has been done for the diagnostic and the fault localization problems [6, 16]. However, diagnostics is a well documented subject in other areas, such as artificial intelligence, in complex mechanical systems and medicine. Therefore, most of the concepts and terms used in this paper are imported from those domains.

In model-based diagnostics [8, 12], we assume the availability of the real system (e.g., implementation) which can be observed, and its model (e.g., specification) from which predictions can be made about its behavior. It is necessary to know how the system or the machine under test is supposed to work in order to be able to know why it is not working correctly.

Often the specification of a model-based system is described in a structured manner. Therefore, a system is seen as a set of components connected to each other in a specific way. The **structure** (organization) of a system can be defined as a relationship (e.g., physical connection, procedure call,...) between the different components of the system. A **component** is seen as one of many smaller sub-systems in the larger system. The behavior of the larger system is, therefore, described in terms of its component behaviors. A possible way of describing a component behavior, is through the use of input symbols, which can be applied to the component, and outputs, which might be generated by the same component. **Observations** of inputs and outputs show how the system to be diagnosed is behaving, while **expectations**, derived from its model, tell us how it is supposed to behave. The differences between expectations and observations, which are called **"symptoms"**, hint the existence of one or several differences between the model and its system. In order to explain the observed symptoms, a diagnostic process should be initiated. It consists mainly of performing the following two tasks: the generation of candidates of faults and the discrimination between candidates [8].

Task 1: Generation of candidates: This process uses the identified symptoms and the model to deduce some diagnostic candidates. Each **diagnostic candidate** is defined to be the minimal difference, between the model and its system, capable of explaining all symptoms. It indicates the failure of one or several components in the system.

Task 2: Discrimination between candidates: Once the step of candidate generation terminates, we often end up with a huge number of diagnostic candidates. To reduce their number, two main techniques are used. The first one consists of the selection of some additional new tests called **"distinguishing tests"**. The second technique consists of introducing new observation points in the implementation under investigation and executing the same tests again.

In [6], we introduced a single fault (output or transfer) diagnostic algorithm for systems represented by FSMs.

0743-166X/93 $03.00 © 1993 IEEE

With respect to the above described general model, transitions in an FSM are considered as components, while states have the function of connecting these components. In this paper, we generalize the algorithm of [6] to the case where system implementations are allowed to have multiple faults; several transitions might have output and/or transfer faults. If the occurred faults are detected by one or several test cases, which may have been generated by one of the existing test selection methods, the new algorithm will have the ability to generate a minimal set of diagnoses, each of which is formed by a set of transitions (with specific types of faults) suspected of being faulty. We also propose approach for selecting additional test cases, which allows the reduction of the number of possible diagnoses.

The remainder of the paper is organized as follows. In Section 2, the deterministic finite state machine (FSM) model and a corresponding fault model are introduced. A brief description of some test selection methods and a discussion on their fault localization power are also presented. Section 3 presents an approach for the multiple fault diagnostics of system implementations represented by FSMs. In Section 4, an application example explaining the steps of the proposed diagnostic approach is provided. Section 5 presents an approach for the selection of additional test cases in order to reduce the number of possible diagnoses. An estimation of the complexity of the diagnostic approach is given in Section 6. Finally, Section 7 contains a concluding discussion and points for future research.

2. Finite state machines

A deterministic finite state machine (FSM) M can be represented by a quintuple (S, I, Y, T, O) where :

S is the set of states of M. It includes an initial state s_0,

I is the set of input symbols,

Y is the set of output symbols. It includes the null output (e),

T is the next-state function, S x I --> S,

O is the output function, S x I --> Y.

The notation **s-a/b->s'** is used to represent a **transition**. For each state in the machine, a **reset transition** is used to take the machine to its initial state. It takes the symbol **r** as input and generates the symbol **e** as output.

Finally and in order to deal with null outputs (e.g., e), we assume that the output e is observed during a test by the application of an input and the non-observation of any output during a predetermined lapse of time. After deducing that a null output has occurred, the next input is allowed to be applied.

A graphic representation of a deterministic FSM example, in the form of a **state transition diagram**, is given in Figure 1.

2.1 The FSM fault model

The FSM fault model [2] is based on faults made on labeled transitions. Some of these faults, which are essential for the diagnostic approach discussed in Section 3, are defined as follows:

Definition 1: Output 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.

An implementation has a **single output fault** if one and only one of its transitions has an output fault.

Definition 2: Transfer fault: We say that a transition has a transfer fault if, for the corresponding state and received input, the implementation enters a different state than specified by the Next-state function.

An implementation has a **single transfer fault** if one and only one of its transitions has a transfer fault.

Definition 3: Additional (missing) transition fault: An implementation has an additional (missing) transition, if for a pair of present state and input, one more (one less) transition (with respect to the specification) is defined.

An implementation has **multiple faults** if and only if some of its transitions have one or several faults defined above.

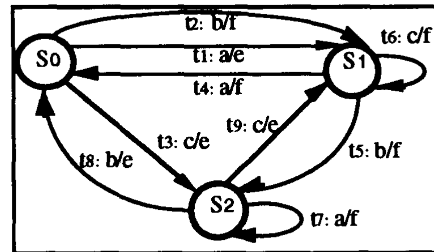

Figure 1: A state transition diagram of an FSM

For our diagnostic approach presented in the following section, we assume the following fault model: **the implementation under test (IUT) may have output faults and/or transfer faults in its transitions.**

This fault model covers single (output or transfer) faults and multiple (output and/or transfer) faults which might occur in the transitions of the machine. In addition, certain cases of missing transition faults may also be explained by a combination of transfer and output faults as explained in the following. A missing transition leads to an incompletely specified implementations. Different implementation assumptions may apply in this case, such as the following:

(a) Blocking: The input is blocked in the input queue, as defined for Estelle [3]. This case can not be modeled in general by a faulty transition.

(b) The input is dropped, as defined for SDL [1]. This case can be modeled by a multiple fault, where the faulty

transition has the empty output and leads back to the same state.

(c) Some error indication: In this case, the fault is detected by the error indication. This case can be modeled by a multiple fault, where the faulty output has the error output and leads back to the same state.

2.2 Test selection methods for FSMs

Many test selection methods have been developed for FSMs [7, 4, 10, 13, 9, 14]. The most important ones are the following:

T-method: The T-method [10] generates a test suite consisting of a single test sequence known as a **"transition tour"**. For a given FSM, a transition tour is an input sequence which takes the FSM from its initial state, traverses every transition at least once, and returns to its initial state. It has the power of detecting all output faults (in the absence of transfer faults), but there is no guarantee of detecting any transfer fault.

DS-method: In the DS-method [7], **a distinguishing sequence (DS)** is used for state identification. An input sequence is said to be a distinguishing sequence for a FSM, if the output sequence produced by the FSM is different for each different starting state. The DS-method uses a two-phase approach. The tests of the first phase check that each state defined by the specification also exists in the implementation. The tests of the second phase check all transitions defined by the specification and not tested during the first phase, for correct output and transfer in the implementation. Under the assumption that the number of states of the implementation is not larger than that of the specification, the DS-method guarantees the detection of all output and transfer faults. Its only disadvantage is that it is not always applicable because not all FSMs possess a DS.

UIO-method: The UIO-method [13] uses a set of **unique input/output (UIO)** sequences for state identification. An UIO sequence for a state s is an I/O behavior not exhibited by any other state in the FSM. The UIO-method generates test sequences which check whether each transition has the correct Next-state and the correct output. To check for the correctness of the Next-state reached by the machine after the execution of the transition under consideration, the corresponding UIO sequence is applied. In general test sequences generated by the UIO-method are shorter than those produced by the DS-method.

Vuong claimed that the UIO-method may leave certain transfer faults undetected [15]. He also proposed a modified version, called UIOv-method, which contains a procedure for verifying the uniqueness of the UIO sequences, thus detecting faults which were otherwise undetectable due to non-unique UIO sequences. The test sequences generated by the UIOv-method guarantee the detection of all output and transfer faults.

W-method: The W-method [4] involves the selection of two sets of input sequences: The **W-set** and the **P-set**. The latter represents a **transition cover set** of the specification. The former represents a **characterization set** of the specification. The set W consists of input sequences that can distinguish between the behaviors of every pair of states in the specification.

The W-method provides a set of test sequences consisting of the concatenation of the sets P and W (i.e. P.W). Each test sequence starts with the initial state, after the application of the reset operation. In this case, to identify a reached state I_k to which a transition t_k transfers, all the sequences contained in the W-set are applied to the implementation, separately. In general, test suites generated by the W-method are longer than those produced by other test selection methods.

Provided that the number of states in the implementation remains within a certain bound, the W-method has the full power of detecting all output and transfer faults.

Wp-method: The Wp-method [5] is a modified version of the W-method. The only difference between the two methods is that instead of using the complete set W to check each reached state S_i, only a subset of this set is used in its second phase. This subset W_i is called an identification set for state S_i. If the reached state is the intended one, the result obtained from the application of W_i will confirm its correctness. On the other hand, if the implementation reaches a faulty state, the result obtained from the application of W_i will be different and hence, indicate the detection of a fault. In such a case, the analysis of the obtained result will in general not have the full power of identifying the reached state. While the Wp-method has the same fault detection power as the W-method, its main advantage is the length reduction of the generated test suite.

2.3 Diagnostic power of test selection methods

Following the discussion of Section 2.2 on the different test selection methods and their fault detection power, the question comes to mind: What is the diagnostic and fault isolation power of these methods?

While a test suite generated by the W-method provides enough information to diagnose a single fault, it cannot localize the faults in an IUT, in general, as shown in Figure 2. This figure shows two faulty implementations, which generate the same output sequences in response to the test sequence, TS, generated by the W method (see Example 1). This failure can be explained by the fact that W is no longer a characterization set for I2 and state s2 is no longer reachable in I1. We see that test sequences generated by existing test selection methods do not, in general, guarantee the localization of multiple faults. It can be expected that less exhaustive test selection methods, such as the UIO, Wp or transition tour methods have even less power of fault location. Therefore, in order to localize implementation faults, additional diagnostic tests are needed. It is important to note that a test sequence with a better fault coverage (i.e. a W test suite rather than a T test

suite) might need less additional diagnostic tests for the process of discrimination between candidate diagnoses.

Example 1: Given the specification of Figure 1, a possible characterization set for the W-method is:

$W = \{a, b\}$.

Using the above W and applying the W-method, we generate the following test suite:

TS = {aa, ab, bca, bcb, baa, bbb, cab, cca, ccb, cba, bab, bba, caa, cbb}

outputs of S = {ef, ef, fff, fff, ffe, ffe, efe, eef, eef, eee, fff, fff, eff, eef}

The application of TS, to the faulty implementations I1 and I2 shown in Figures 2a and 2b, respectively, generates in both cases the same sequences of outputs listed below:

outputs of I1 and I2 = {ef, ef, fff, fff, ffe, fff, eef, eee, eef, eff, fff, fff, eef, eff}.

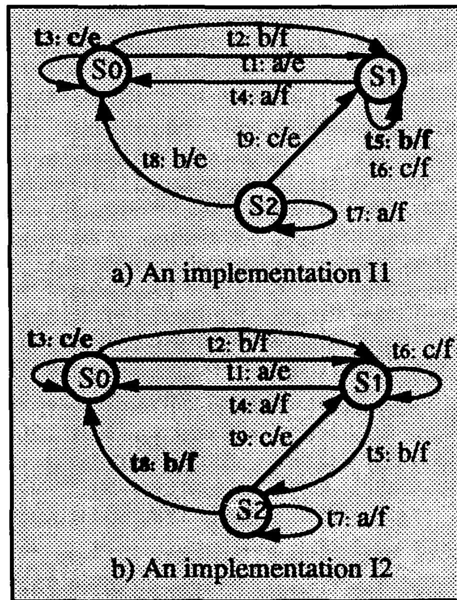

Figure 2: Two faulty implementations non-distinguishable by W

3. The diagnostic approach

3.1 Preliminary definitions

We assume in the following that a specification S is given, as well as an implementation M with output and/or transfer faults, as described in Section 2.1.

A **test suite**, TS, is defined as a set of **test cases**, where each test case is a sequence of input symbols. We write TS = { tc_1; ...; tc_p}, where each tc_i is a test case. If a test case tc_i consists of m_i inputs $i_{i,1}, i_{i,2}, \ldots, i_{i,mi}$, the corresponding sequence of expected outputs is written as $o_i = o_{i,1}, o_{i,2}, \ldots, o_{i,mi}$, where output $o_{i,j}$ is expected after input $i_{i,j}$. $t_{i,j}$ represents the j-th transition executed in test case tc_i according to the specification S.

Any difference, between an **expected output o** as defined by the specification, and the corresponding **observed output ô**, represents a **symptom**.

A minimal set of faults, which has the capability of explaining all observed outputs, is called a **diagnosis**. The corresponding set of transitions, where these faults occur, is called a **diagnostic candidate**.

Definition: A fault f of an implementation M in a transition t of S is said to be **directly reached by a test case tc**, if the execution of tc, as defined by the specification S, leads to the transition t, there is no transfer fault in M on the path that leads from the initial state to t, and the subsequent path of the test case contains a symptom.

Example: If the machine in Figure 3 represents the specification, then all faults (f1, f2, f3, f4, f5, f6, f7) in the implementation of Figure 4, are directly reachable. On the contrary, the fault f3 in the implementation of Figure 5, is not directly reachable.

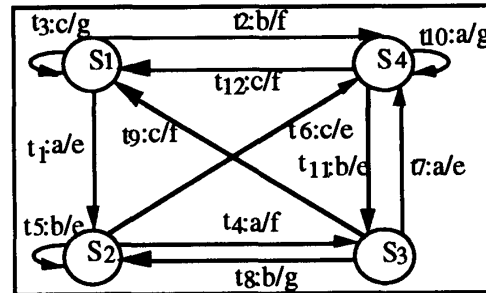

Figure 3: The specification S

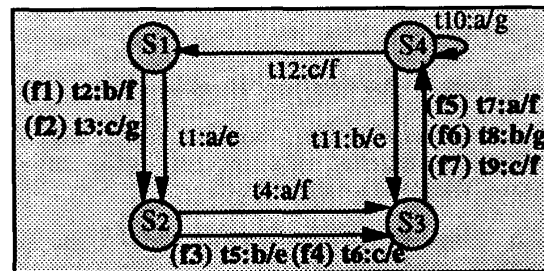

Figure 4: An implementation I1

3.2 The diagnostic algorithm

The diagnostic algorithm described in the following is based on a certain assumption about the faults contained in the implementation under test (IUT) and the test suite TS used to detect the presence of faults. As explained below,

the algorithm ensures correct and complete diagnosis if the following assumption is satisfied.

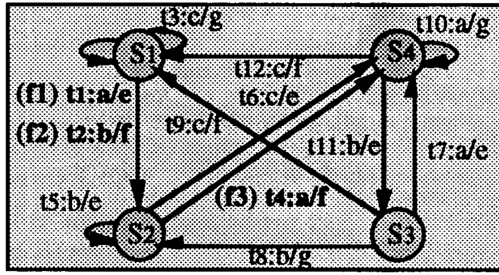

Figure 5: An implementation I2

Assumption: For each fault in the implementation, there is a test case in the applied test suite which reaches that fault directly.

Note: This implies that there is at least one symptom, generated by the application of the test suite, which corresponds to each fault of the implementation. In the case of an output fault, the symptom occurs immediately (therefore the corresponding transition has been called "symptom transition" [6]). In the case of a transfer fault, the symptom occurs after the transition containing the fault, and may be due to the wrong transfer of the fault in question or to other faults which may be present in the implementation. In any case, we say that the fault has been "detected".

3.2.1 Step 1: (Generation of expected and observed outputs)

Application of the test suite, TS, to the specification and the IUT. For each test case tc_i, the expected output sequence is written as $o_i = o_{i,1}, o_{i,2}, ..., o_{i,mi}$, where output $o_{i,j}$ is expected after input $i_{i,j}$, while the observed output sequence is written as: $\hat{o}_i = \hat{o}_{i,1}, \hat{o}_{i,2}, ..., \hat{o}_{i,mi}$.

3.2.2 Step 2: (Generation of symptoms)

Compare each observed output sequence with its corresponding expected output sequence and identify all observed symptoms for all test cases.

3.2.3 Step 3: (Construction of the set of tentative candidate sets)

We construct in the following a **set of tentative candidate sets**, called STC. This set has the following properties: Each tentative candidate set TC is a set of fault candidates, and a **fault candidate** is a transition name annotated by o or *, which corresponds to a output fault candidate or a transfer fault candidate, respectively. For example, the candidate t* is a transfer fault candidate, and represents the assumption that the transition t has a transfer fault. A tentative candidate $\{t_1^*, t_2^*, t_1^o\}$ represents the assumption that the transition t_1 has an output and a transfer fault, that t_2 has a transfer fault, and that all other transitions have (definitely) no faults.

Definition: The **actual set of faulty transitions** of the implementation is the set of transitions of the implementation that contain faults, annotated by o and/or *, as in the case of the tentative candidate sets.

The following construction of STC ensures that the constucted STC will contain the actual set of faulty transitions. For each given test case tc_i, we construct a set of fault hypothesis SFH_i as follows. Each fault hypothesis is a pair <FC, CC>, where FC is a set of fault candidates and CC is a set of correct candidates, that is, fault candidates that are assumed not to be present.

(a) If tc_i has no symptom, SFH_i contains a single element of the form $\{ <\{\},\{t_{i,1}^o\} >\}$, which indicates that the first transition, $t_{i,1}$, executed by tc_i, does not have an output fault.

(b) If there are m symptoms in tc_i, we consider the different symptoms in order and include in SFH_i fault hypotheses which correspond to the assumption that the symptom considered corresponds to a fault that is directly reached by the test case tc_i. The following situations may occur for the j-th symptom. Because of the Assumption, the j-th symptom is only considered under the hypothesis that all earlier symptoms j' < j correspond to output faults. We have the following two hypotheses:

(1) The j-th symptom corresponds to an output fault (t_{i,k_j}^o), and there is no transfer fault on the execution path ($t_{i,k(j-1)}, t_{i,k(j-1)+1}, ..., t_{i,k_j-1}$) that leads to this symptom transition. In this case, the next symptom will also be considered (if it exists) since it corresponds to a directly reached fault. If there is no next symptom, then the fault hypothesis <X, Y> is added to SFH_i, where $X = \{t_{i,k_1}^o, t_{i,k_2}^o, ..., t_{i,k_j}^o\}$, $Y = \{t_{i,1}^o, t_{i,1}^*, t_{i,2}^o, t_{i,2}^*, ..., t_{i,k_j}^*\} - X$, and t_{i,k_j} is the k_j-th transition where the j-th symptom has been observed.

(2) The j-th symptom corresponds to a transfer fault which is located in one of the transitions ($t_{i,k(j-1)}, t_{i,k(j-1)+1}, ..., t_{i,k_j-1}$) from the symptom transition (output fault in $t_{i,k(j-1)}^o$) of the previous symptom (or from the initial state, respectively) to the symptom in question in the transition t_{i,k_j}. In this case, the next symptom will not be considered, since it corresponds to a fault which is not directly reached by this test case, although it may be directly reached through another test case. For each transition $t_{i,n}$, $n = k(j-1), k(j-1)+1, ..., k_j-1$, in the transition sub-sequence starting at $t_{i,k(j-1)}$ and ending at the transition t_{i,k_j-1}, the fault hypotheses <X, Y> will be

added to SFH_i, where $X = \{t_{i,k_1}{}^o, t_{i,k_2}{}^o, ..., t_{i,k(j-1)}{}^o, t_{i,n}{}^*\}$, $Y = \{t_{i,1}{}^o, t_{i,1}{}^*, t_{i,2}{}^o, t_{i,2}{}^*, ..., t_{i,k_j}{}^o\} - X$.

The following example shows the construction of a set of fault hypotheses for a test case tc_i:

If for the test case tc_i, we have the following:

Spec. Transitions:
$tr_i = t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}, t_{i,5}, t_{i,6}, t_{i,7}$
Spec. outputs:
$o_i = o_{i,1}, o_{i,2}, o_{i,3}, o_{i,4}, o_{i,5}, o_{i,6}, o_{i,7}$
Observed Outputs
$o_i = o_{i,1}, o_{i,2}, \hat{o}_{i,3}, \hat{o}_{i,4}, \hat{o}_{i,5}, \hat{o}_{i,6}, o_{i,7}$
where $o_{i,j} \neq \hat{o}_{i,j}$ $j = 3, 5, 6$

then the following set of fault hypotheses is constructed:

$SFH_i = \{\langle\{t_{i,1}{}^*\}, \{\}\rangle,$
/* first symptom caused by a transfer fault in $t_{i,1}$ */
$\langle \{t_{i,2}{}^*\}, \{\{t_{i,1}{}^o, t_{i,1}{}^*\}\rangle,$
/* first symptom caused by a transfer fault in $t_{i,2}$; $t_{i,1}$ not faulty */
$\langle\{t_{i,3}{}^o, t_{i,3}{}^*\}, \{t_{i,1}{}^o, t_{i,1}{}^*, t_{i,2}{}^o, t_{i,2}{}^*\}\rangle,$
/* first symptom caused by an output fault in $t_{i,3}$; second symptom caused by a transfer fault in $t_{i,3}$, no faults in $t_{i,1}$ and $t_{i,2}$ */
$\langle\{t_{i,3}{}^o, t_{i,4}{}^*\}, \{t_{i,1}{}^o, t_{i,1}{}^*, t_{i,2}{}^o, t_{i,2}{}^*, t_{i,3}{}^*\}\rangle,$
$\langle \{t_{i,3}{}^o, t_{i,5}{}^o, t_{i,5}{}^*\}, \{t_{i,1}{}^o, t_{i,1}{}^*, t_{i,2}{}^o, t_{i,2}{}^*, t_{i,3}{}^*, t_{i,4}{}^o, t_{i,4}{}^*\}\rangle,$
$\langle \{t_{i,3}{}^o, t_{i,5}{}^o, t_{i,6}{}^o\}, \{t_{i,1}{}^o, t_{i,1}{}^*, t_{i,2}{}^o, t_{i,2}{}^*, t_{i,3}{}^*, t_{i,4}{}^o, t_{i,4}{}^*, t_{i,5}{}^*\rangle\}$

The set STC of tentative candidate sets is formed by all possible unions of fault alternatives taken from all test cases. More formally,

$$STC = \{TC \mid TC = \bigcup_{i=1,2,...Ls} FC_i \quad \wedge \quad TC \cap \left(\bigcup_{i=1,2,...Ls} CC_i \right) = \varnothing,$$

where $\langle FC_i, CC_i\rangle \in SFH_i$ and Ls is the number of test cases$\}$.

It is possible to **reduce** the set *STC*, by removing some of its elements, if some additional knowledge is available, such as :

1) The maximum number of transitions which might be faulty in the IUT, or

2) The maximum number of faults (transfer and output) which might be present in the IUT,

3) The maximum number of output faults and the maximum number of transfer faults which might be present in the IUT.

As an example, if we know that the maximum number of faults in the IUT does not exceed N faults, then all elements in *STC*, having a cardinality which is strictly greater than N, can be removed. As the special case of single fault diagnostics, we may keep in *STC* only elements with cardinality one. In this case the algorithm described here reduces to the one described in [6].

3.2.4 Step 4: (Generation of PossFaults sets, diagnostic candidates and diagnoses)

A **diagnostic candidate** is a tentative candidate and an assignment of faults (specific output and/or transfer to a specific state) to all its transitions which succeed to explain all observations. Note that a given tentative candidate may lead to several diagnostic candidates. All diagnostic candidates can be obtained by checking for each possible assignment of faults of all diagnostic candidates whether it explains all observations. The checking process can be done through the application of all test cases in *TS*, to the mutant machine, which corresponds to the diagnostic candidate in question. If the outputs obtained from the mutant are identical to the outputs observed from the IUT the diagnostic candidate is confirmed. To compute all possible faults for each tentative candidate, and hence, all corresponding diagnostic candidates, we proceed as follows:

Suppose that the tentative candidate, "**$Cand_i$**", in *STC* is under consideration and has n_1 transitions suspected of having transfer faults, and n_2 transitions suspected of having output faults. We change in the specification machine the ending states of all $Cand_i$'s n_2 transitions suspected of having transfer faults. We also assign the remaining transitions in $Cand_i$ the corresponding symptom outputs. All remaining specification transitions are left unchanged. We apply the test cases in TS on the resulting machine (mutant). If the resulting outputs are equal to those of the IUT, then the specific set of faults, introduced in the elements of $Cand_i$, will be saved as an element in a set called: "**PossFaults[$Cand_i$]**".

The above process is repeated until all combinations of faults (the different assignments of ending states to $Cand_i$'s n_2 elements suspected of having transfer faults, in addition to the new outputs assigned to the remaining elements of $Cand_i$) for $Cand_i$'s elements are considered. If all combinations of faults for $Cand_i$'s elements fail to produce the same outputs as those obtained from the IUT, then $Cand_i$'s PossFaults set will be kept empty and the tentative candidate $Cand_i$ will not be considered as a diagnostic candidate.

We remove all tentative candidates with empty PossFaults sets from the tentative candidate set, *STC*. Each element in the corresponding PossFaults sets

represents a diagnostic candidate, simply called **"diagnosis"**. It consists of the minimal set of faults (output and/or transfer), which might be present in the given implementation and which have the ability of explaining all observed outputs.

4. An application example

Suppose that the following initial test suite, obtained through the Wp method, is given for the FSM specification shown in Figure 1:

TS = {raa; rab; rbca; rbcb; rbaa; rbbb; rcab; rcca; rccb; rcba}

Step 1: The application of this TS to the specification of Figure 1 and the implementation of Figure 2a, yields the expected and observed output sequences shown in Table 1. A reset transition tr is assumed to be available for both the specification and the implementation. We use the symbol "r" to denote the input for such a transition and the symbol "-" to denote its output.

Step 2: Differences between observed and expected outputs are encountered in test cases tc_6, tc_7, tc_8, and tc_{10}. Therefore, the following symptoms are generated:

$Symp_{6,1} = (o_{6,3} \neq \hat{o}_{6,3})$,
$Symp_{7,1} = (o_{7,2} \neq \hat{o}_{7,2})$,
$Symp_{72} = (o_{7,3} \neq \hat{o}_{7,3})$,
$Symp_{8,1} = (o_{8,3} \neq \hat{o}_{8,3})$,
$Symp_{10,1} = (o_{10,2} \neq \hat{o}_{10,2})$,
$Symp_{10,2} = (o_{10,3} \neq \hat{o}_{10,3})$.

Step 3: Corresponding to the above symptoms, we construct the following sets of fault hypothesis:

$SFH_1 = \{ <\{\},\{t_1^o\}> \}$
$SFH_2 = \{ <\{\},\{t_1^o\}> \}$
$SFH_3 = \{ <\{\},\{t_2^o\}> \}$
$SFH_4 = \{ <\{\},\{t_2^o\}> \}$
$SFH_5 = \{ <\{\},\{t_2^o\}> \}$
$SFH_6 = \{ <\{t_2^*\},\{\}>, <\{t_5^*\}, \{t_2\}>, <\{t_8^o\}, \{t_2, t_5\}> \}$
$SFH_7 = \{ <\{t_3^*\},\{\}>, <\{t_7^o, t_7^*\}, \{t_3\}>, <\{t_7^o, t_8^o\}, \{t_3, t_7^*\}> \}$
$SFH_8 = \{ <\{t_3^*\},\{\}>, <\{t_9^*\}, \{t_3\}>, <\{t_4^o\}, \{t_3, t_9\}> \}$
$SFH_9 = \{ <\{\},\{t_3^o\}> \}$
$SFH_{10} = \{ <\{t_3^*\},\{\}>, <\{t_8^o, t_8^*\}, \{t_3\}>, <\{t_8^o, t_1^o\}, \{t_3, t_8^*\}> \}$

Using the above sets of fault hypothesis, we construct the following set of tentative candidate sets:

STC = $\{\{t_2^*, t_3^*\}, \{t_2^*, t_7^o, t_7^*, t_9^*, t_8^o, t_8^*\}, \{t_2^*, t_7^o, t_7^*, t_4^o, t_8^o, t_8^*\}, \{t_2^*, t_7^o, t_9^*, t_8^o, t_8^*\}, \{t_2^*, t_7^o, t_4^o, t_8^o, t_8^*\}, \{t_3^*, t_5^*\}, \{t_5^*, t_7^o, t_7^*, t_9^*, t_8^o, t_8^*\}, \{t_5^*, t_7^o, t_7^*, t_4^o, t_8^o, t_8^*\}, \{t_5^*, t_7^o, t_9^*, t_8^o, t_8^*\}, \{t_5^*, t_7^o, t_4^o, t_8^o, t_8^*\}, \{t_3^*, t_8^o\}, \{t_7^o, t_7^*, t_9^*, t_8^o, t_8^*\}, \{t_7^o, t_7^*, t_4^o, t_8^o, t_8^*\}, \{t_7^o, t_8^o, t_8^*, t_9^*\}, \{t_7^o, t_4^o, t_8^o, t_8^*\}\}$

Step 4: For each element in STC, all possible assignments of faults, having the ability of explaining all observed outputs, lead to the following possibilities:

PossFaults[$\{t_2^*, t_3^*\}$] = {}
PossFaults[$\{t_2^*, t_7^o, t_7^*, t_9^*, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_2^*, t_7^o, t_7^*, t_4^o, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_2^*, t_7^o, t_9^*, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_2^*, t_7^o, t_4^o, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_3^*, t_5^*\}$] = {{t_3->s_0, t_5->s_0},
{t_3->s_0, t_5->s_1}}
PossFaults[$\{t_5^*, t_7^o, t_7^*, t_9^*, t_8^o, t_8^*\}$] = {
{t_5->s_0, t_7-e->, t_7->s_0, t_8-f->, t_8->s_1, t_9->s_0}
{t_5->s_0, t_7-e->, t_8-f->, t_7->s_0, t_8->s_1, t_9->s_2}
{t_5->s_0, t_7-e->, t_8-f->, t_7->s_1, t_8->s_1, t_9->s_0}
{t_5->s_0, t_7-e->, t_8-f->, t_7->s_1, t_8->s_1, t_9->s_2}
{t_5->s_1, t_7-e->, t_8-f->, t_7->s_0, t_8->s_1, t_9->s_0}
{t_5->s_1, t_7-e->, t_8-f->, t_7->s_0, t_8->s_1, t_9->s_2}
{t_5->s_1, t_7-e->, t_8-f->, t_7->s_1, t_8->s_1, t_9->s_0}
{t_5->s_1, t_7-e->, t_8-f->, t_7->s_1, t_8->s_1, t_9->s_2}}
PossFaults[$\{t_5^*, t_7^o, t_7^*, t_4^o, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_5^*, t_7^o, t_9^*, t_8^o, t_8^*\}$] = {
{t_5->s_0, t_7-e->, t_8-f->, t_8->s_1, t_9->s_0}
{t_5->s_0, t_7-e->, t_8-f->, t_8->s_1, t_9->s_2}
{t_5->s_1, t_7-e->, t_8-f->, t_8->s_1, t_9->s_0}
{t_5->s_1, t_7-e->, t_8-f->, t_8->s_1, t_9->s_2}}
PossFaults[$\{t_5^*, t_7^o, t_4^o, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_3^*, t_8^o\}$] = {{t_3-->s_0, t_8-f->}}
PossFaults[$\{t_7^o, t_7^*, t_9^*, t_8^o, t_8^*\}$] = {
{t_7-e->, t_7->s_0, t_8-f->, t_8->s_1, t_9->s_0}
{t_7-e->, t_7->s_0, t_8-f->, t_8->s_1, t_9->s_2}
{t_7-e->, t_7->s_1, t_8-f->, t_8->s_1, t_9->s_0}
{t_7-e->, t_7->s_1, t_8-f->, t_8->s_1, t_9->s_2}}
PossFaults[$\{t_7^o, t_7^*, t_4^o, t_8^o, t_8^*\}$] = {}
PossFaults[$\{t_7^o, t_8^o, t_8^*, t_9^*\}$] = {
{t_7-e->, t_8-f->, t_8->s_1, t_9->s_0},
{t_7-e->, t_8-f->, t_8->s_1, t_9->s_2}}
PossFaults[$\{t_7^o, t_4^o, t_8^o, t_8^*\}$] = {}

Note: t_i-->s_j means that t_i might have transferred to s_j, while t_k-x-> means that t_k might have the output fault of x.

For example, the last possibility {t_7-e->, t_8-f->, t_8->s_1, t_9->s_2} corresponds to the following diagnosis: The IUT has t_7 with an output fault of e, t_8 with an output fault of f and a transfer fault to s_1, and t_9 with a transfer fault to s_2.

tc. #	tc1	tc2	tc3	tc4	tc5
Inputs	r, a, a	r, a, b	r, b, c, a	r, b, c, b	r, b, a, a
Specified transitions	tr,t1,t4	tr,t1,t5	tr,t2,t6,t4	tr, t2,t6,t5	tr,t2,t4,t1
Expected outputs	-, e, f	-, e, f	-, f, f,f	-, f, f,f	-,f, f,e
Observed outputs	-, e, f	-, e, f	-, f, f,f	-, f, f,f	-,f, f,e

tc. #	tc6	tc7	tc8	tc9	tc10
Inputs	r, b, b, b	r, c, a, b	r, c, c, a	r, c, c,b	r, c, b, a
Specified transitions	tr,t2,t5,t8	tr,t3,t7,t8	tr,t3,t9,t4	tr, t3,t9,t5	tr,t3,t8,t1
Expected outputs	-, f, f,e	-, e,f, e	-, e, e,f	-, e, e,f	-,e,e,e
Observed outputs	-, f, f,f	-, e,e,f	-, e, e,e	-, e, e,f	-,e, f,f

Table 1: Test cases and their outputs

5. Additional tests for reducing the number of diagnoses

Recall that the main purpose of testing and diagnostics is the localization of implementation faults and their correction. Therefore, if the diagnostic process ends up with multiple diagnoses, additional tests are needed to help reducing the number of diagnoses, if possible, to a single diagnosis. In other words, additional tests should be selected and applied to the implementation until a set of faults in one of the diagnoses is confirmed and consequently, all remaining diagnoses could be removed.

To distinguish between different diagnoses, we use an approach which can be based on a test method described by Gill [Gill 62]. This method determines a test sequence which allows the distinction between any two given finite state machines. In our context, each diagnosis corresponds to a particular (faulty) implementation determined by the specification and the faults predicted by the diagnosis. Gill's test method can be used to distinguish between any pair of diagnoses (mutants) by a single test sequence derived by his method.

Given a set of n diagnoses for a given implementation, Gill's method may be applied to distinguish between any two selected diagnoses, say Di and Dj. The application of the derived test sequence to the implementation will lead to one of the following situations:

(1) The observed output is equal to the one expected for Di.

(2) The observed output is equal to the one expected for Dj.

(3) The observed output is different from both of the outputs expected for Di and Dj.

In cases (1) or (2), we know that Dj or Di, respectively, is a wrong diagnosis. In case (3), we know that both, Di and Dj are wrong diagnosis. We have therefore reduced the number of possible diagnoses and may continue until only one diagnosis remains.

Gill's algorithm for the selection of a test sequence to distinguish between two given implementations can be described as follows:

Given two machines M_1 and M_2, generate a pruned tree breath first (nodes are labelled by pairs of states from M_1 and M_2, respectively)

Create the root node n_0 of a tree T, $n_0 = [s_0^1, s_0^2]$, where s_0^1 and s_0^2 are the initial states of M_1 and M_2, respectively.

for each non-closed node, $n_k = [s_i^1, s_j^2]$, in the current level of the tree **do**

for each input symbol i **do**

if $(O_1(s_i^1, i) = O_2(s_j^2, i))$ **then**
Create the new node, n_l = [nextState$_1(s_i^1, i)$, nextstate$_2(s_j^2, i)$], in the next level of the tree
Create a new branch labeled $i/(O_1(s_i^1, i))$ between nodes n_k and n_l

if $(n_l = n_m)$, where n_m is an existing node in the tree **then**
Close n_l by marking it with an x (it will not be considered for further expansion).
else
Form a test case sequence, tc, by the input part in the labels of the path, which starts in the root node and ends in node n_k
Concatenate the input i to the end of the test case tc.
stop.

The number of different nodes in the constructed tree is bounded by $O(n^2)$, where n is the number of states in the specification machine. Since each node in the constructed tree is considered at most once (due to node closing) by all possible inputs, the number of leafs in such a tree is bounded by $O(I.n^2)$, where I is the number of input symbols accepted by the specification machine. Hence, the overall complexity of the selection of the additional tests to distinguish between the N diagnoses, is bounded by $O(N.I.n^2)$. Knowing that the number of diagnoses is bounded by $O((Lc^S).n^F)$, where Lc is the number of inputs in the longest test case, S is the estimated maximum number of test cases with symptoms, and F is the maximum number of transfer faults, the overall complexity is $O(I.Lc^S.n^{F+2})$.

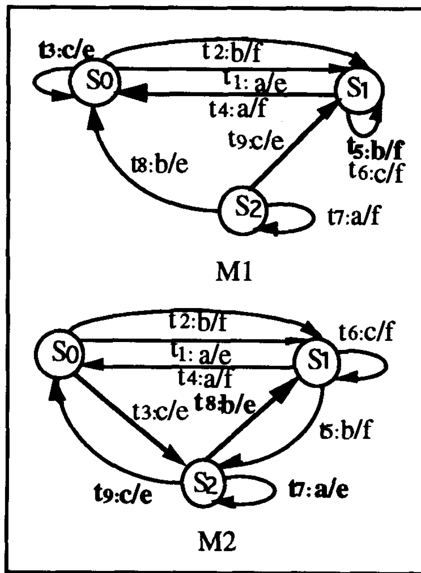

Figure 7: Machines corresponding to diagnoses Diag1 and Diag2

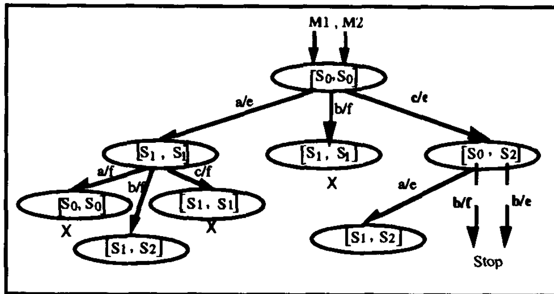

Figure 8: The test-tree for M_1 and M_2

If Step 4 of the diagnostic approach produces N diagnoses, at most $(N-1)$ additional diagnostic tests will be needed, in order to reduce the set of diagnoses to a single diagnosis. Since the initial states of the mutant machines are known, Corollary 4.1 in [Gill 62] guarantees that the length of each of the additional tests will be at most $(2n -1)$, where n is the number of states in these machines.

Example: Suppose that, we are given the specification S of Figure 1, the implementation I of Figure 6, and the following two diagnoses among the several diagnoses produced in Step 4 of the diagnostic approach:

> **Diag1:** The IUT might have t_3 transferred to s_0 and t_5 transferred to s_1
>
> **Diag2:** The IUT might have t_7 generated e as output, t_8 transferred to s_1 and generated f as output, and t_9 transferred to s_0

We generate the tree of Figure 8 for the machines M_1 and M_2 (shown in Figure 7) corresponding to $Diag_1$ and $Diag_2$, respectively. The corresponding constructed additional test is rcb. The application of this test case to the implementation I of Figure 2a generates the output -ef. Such a result eliminates the diagnose $Diag_2$ from the list.

6. Complexity approximation of the diagnostic approach

The proposed diagnostic approach has four sequential steps. Therefore, the overall complexity of this approach is equal to the complexity of the most complex step. A close look at the different steps leads to the conclusion that Step 4 has the highest complexity since during the process of diagnoses generation all n states in the machine have to be checked as to whether it might be the ending state for a transition suspected of having a transfer fault. Since in this step, we make use of tentative candidates determined during Step 3, we also need to study the complexity of Step 3.

To estimate the upper bound of the number of tentative candidate sets q we have to study the algorithm in Step 3, which uses Ls test cases. The maximum number of elements in a set of fault hypothesis for a given test case is Lc. Only the test cases with symptoms contribute to the construction of the set of tentative candidate sets. If the number of test cases with symptoms is S, then the maximum number of tentative candidate sets q, which is the number of all combination of elements in the S sets of fault hypothesis, is bounded by Lc^S.

In Step 4, each element in the set of tentative candidates contains several transitions some of which are suspected of having transfer faults and the rest is suspected of having output faults. If F is the upper bound of the number of transfer faults (this number can never be larger than Ls, the number of test cases, because of the Assumption) in any of

the Lc^S tentative candidates determined during Step 3, the complexity of Step 4, in terms of possible number of diagnoses, will be $O((Lc^S).n^F)$. If F and S are reasonably small, say 3 or 4, the complexity of the algorithm will remain manageable. Even when F and S become large, the complexity of our algorithm remains comparable with the complexity of other existing methods; for example, Ko's method generates up to $n^{I.n}$, diagnoses, where n and I are the number of states and the number of inputs, respectively.

7. Conclusion

In this paper, we generalized the diagnostic approach proposed in [6] to the case where system implementations, represented by FSMs, are allowed to have multiple faults. Such an approach is mainly motivated by the fact that even strong test selection methods (e.g., the W-method) do not have in general full fault localization power. If existing faults are detected, this algorithm permits the generation of a minimal set of diagnoses, each of which is formed by a set of transitions (with specific types of faults) suspected of being faulty. The occurrence in an implementation, of all the faults of a given diagnosis, allows the explanation of all outputs observed during the test of the implementation. We also proposed two approaches for selecting additional test cases, which allow the reduction of the number of possible diagnoses.

The proposed diagnostic approach provides the guarantee of correct diagnosis only in those situations where each fault of the implementation is directly reachable by a test case in the given test suite. An interesting research project would be the extension of our work to the diagnostics of machines not respecting this assumption. Another challenging question, which needs to be solved, is to extend the diagnostic approach to systems modelled by extended FSMs.

Acknowledgments: The authors would like to thank A. Petrenko for discussions on the test methods described in this paper. This work was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols.

References

[1] F. Belina et al., "The CCITT specification and description language SDL", Computer Networks and ISDN Systems, Vol. 16, pp. 311-341, 1989.

[2] G.v. Bochmann et al., "Fault models in testing", Invited paper in 4-th IWPTS, Leidschendam, Holland, Oct. 1991.

[3] S. Budkowski et al., "An introduction to Estelle: a specification language for distributed systems", Computer Networks and ISDN Systems, Vol. 14, No. 1, pp. 3-23, 1987.

[4] T.S. Chow, "Testing Design Modelled by Finite-State Machines", IEEE Trans. S.E. 4, 3, 1978.

[5] S. Fujiwara et al., "Test selection based on finite state models", IEEE Trans. on S.E., Vol. 17, No. 6, June 1991, pp. 591-603.

[6] A. Ghedamsi and G.v. Bochmann, "Test result analysis and diagnostics for finite state machines", Proceedings of the 12-th IWPTS, Yokohama, Japan, June 9-12, 1992.

[7] G. Goenenc, "A method for the design of fault detection experiments", IEEE Trans. Computer, Vol. C-19, pp. 551-558, June 1970.

[8] J. de Kleer et al., "Diagnosing multiple faults", Artificial Intelligence 32(1), 1987, pp. 97-130.

[9] R.E. Miller and G.M. Lundy, "Testing protocol implementations based on a formal specification", 3rd IWPTS, McLean, Virginia, Oct. 30 - Nov. 1, 1990.

[10] S. Naito et al., "Fault Detection for Sequential Machines by Transition-Tours", Proc. of FTCS (Fault Tolerant Computing Systems), pp.238-243, 1981.

[11] A. F. Petrenko, "Checking experiments with protocol machines", Proc. of the 4-th IWPTS, Leidschendam, Holland, 15 - 17 Oct. 1991, pp. III-21-III.31.

[12] R. Reiter, "A theory of diagnosis from first principles", Artificial Intelligence 32(1), 1987, pp. 57-96.

[13] K.K. Sabnani et al., "A protocol Testing Procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, 1988.

[14] D. P. Sidhu and T.K. Leung, "Formals Methods for Protocols Testing: A Detailed Study", IEEE Trans. on S.E., vol. 15. No. 4 , April 1989.

[15] S. T. Vuong, W. W. L. Chan and M. R. Ito, "The UIOv-Method for protocol test sequence generation", in the 2-nd IWPTS, Berlin, Germany, Oct. 3-6, 1989.

[16] S.T. Vuong et al., "A novel approach to protocol test sequence generation", IEEE Global telecomm. conference and exhibition, San Diego, California, Dec. 2-5, 1990, vol. 3, 904.1.1 - 904.1.5.