

Diagnosing Distributed Systems Modeled by Communicating Finite State Machines

A. Ghedamsi, G. v. Bochmann and R. Dssouli

Université de Montréal

Département d'Informatique et de Recherche Opérationnelle

C.P.6128, Succ. "A", Montréal, Canada, H3C 3J7

Abstract

We propose a diagnostic algorithm for the case where a distributed system specification (implementation) is given in the form of communicating finite state machines (CFSMs). Such an algorithm localizes the faulty transition in the distributed system once the fault has been detected. It generates, if necessary, additional diagnostic test cases which depend on the observed symptoms and which permit the location of the detected fault. The algorithm guarantees the correct diagnosis of any single (output or transfer) fault in a system of communicating FSMs. A simple example is used to demonstrate the functioning of the different steps of the proposed diagnostic algorithm.

1. Introduction

Testing is an important step in the development cycle of any system (i.e. software, communication protocol or hardware). A lot of research work has been directed towards such tests [Fuji 91, More 90, Davi 88, Sabn 88, Ural 87, Nait 81, Chow 78, Gone 70]. At the same time, in the software domain where a system may be represented by an FSM model, very little work has been done for diagnostic and fault localization problems [Ghed 92, Ghed 92a, Koka 90, Kore 88]. Diagnostic is a well documented subject in other areas such as Artificial Intelligence (AI), complex mechanical systems and medicine [Scho 76]. Therefore, most of the concepts and terms used in this paper are imported from those domains.

In model-based diagnostics [Klee 87, Reit 87], we assume the availability of the real system (i.e. implementation) which can be observed, and its model (i.e. 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. In order to diagnose this kind of systems, models and their corresponding systems are assumed to have the same components and the same structure. **Observations** of inputs and outputs show how the system 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 and the discrimination between candidates [Klee 87].

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**" [Gene 84]. The second technique consists of introducing new observation points in the implementation under investigation and executing the same tests again.

We recall that in general, the diagnostic process is a very complicated task, specially for diagnosing complicated systems such as the distributed systems. This complexity makes the achievement of the candidate generation and discrimination tasks harder. In order to solve this problem, the use of fault models is necessary (see for instance [Boch 91]). Given the hierarchical system description, corresponding fault models may be established using the different levels of abstraction. Some of these fault models give all possible failures of each component in the system. They help to ease the diagnostic procedure, specially by reducing the number of the different cases which have to be considered, and hence, in reducing the number of diagnoses to be generated. It is important to note that different fault models may be used during both tasks of the diagnostic process. In the simplest case and for high level abstractions, the following fault model, based on the system decomposition into components and connections, may apply during the candidate generation phase. Each component may either be faulty or operating correctly [Klee 87]. On the other hand, and for lower level abstractions (i.e. gates or transitions levels), different

uses of precise and more concrete fault models, are recorded in different areas such as the diagnostics of hardware circuits (i.e, stuck at 0/1 fault models) [Stru 89, Klee 89]. These fault models may be used during the phase of discrimination between candidates. In the software area and more precisely for FSMs, another simple fault model, based on transfer and output faults of state transitions, can be used for diagnosing software systems modelled by FSMs [Chow 78, Vuon 90, Ghed 92]. The same fault model is also used for the diagnostic approach presented in this paper.

The remainder of the paper is organized as follows. In Section 2, the model of communicating finite state machines and a corresponding fault model are introduced. Section 3 includes all the details of an approach for the diagnostic of distributed systems implementations represented by the CFSMs model . In Section 4, an application example explaining the steps of the proposed diagnostic algorithm is provided. Section 5, finally, contains a concluding discussion and points for future research.

2. Communicating Finite State Machines

2.1 Principles of the CFSMs model

A system of communicating finite state machines with distributed ports consists of a finite number of deterministic finite state machines which communicate with each other through input queues in addition to their communication with the environment through their respective external ports.

Definition 1: A deterministic FSM M_i ($i = 1, 2, \dots, N$) in a system of CFSMs can be represented by a quintuple $(S_i, I_i, Y_i, T_i, O_i)$ where :

N : Number of FSMs in the system,

S_i : Set of states of M_i . It includes an initial state s_{i0} ,

I_i : Set of input symbols. It includes the reset input (r),

Y_i : Set of output symbols. It includes the null output (-),

T_i : Next-state function, $S_i \times I_i \rightarrow S_i$,

O_i : Output function, $S_i \times I_i \rightarrow Y_i$.

In this paper, we describe the diagnostic problem and a proposed solution for $N = 2$. We assume that each machine in the distributed system has a separate external port through which input and output symbols are communicated between the machine and the external world. In addition, the

two machines are connected to each other by two one way internal queues: one input queue for each machine in the system.

For each deterministic FSM in a system of CFSMs, we distinguish two types of transitions. The first type is called "**external-output transitions**" or simply "**transitions**". It is the kind of transitions which deliver their outputs to a corresponding external port. The second type is called "**internal-output transitions**". They are those transitions which communicate their outputs to another machine, instead of communicating them to the external port of the corresponding machine. A possible way to distinguish the two types of transitions is by using two distinct sets of inputs for their execution. The first set **IEO**, called "**inputs for external outputs**", contains input symbols which can be applied to only external-output transitions. The second set **IIO**, called "**inputs for internal outputs**", contains inputs which can be applied only to internal-output transitions. In the following sections, we make use of this type of input sets for each machine in the system.

Each time an input symbol is applied to a machine in the system, we assume that enough time is given to observe its effect, which will be an output interaction in one of the existing external ports. Hence, the application of the next external input should be preceded by the observation of the output implied by the previous input. Therefore, only one message will be circulating in the whole system at any time. Such an assumption, which we call "**the synchronization assumption**", guarantees the deterministic behavior of the global system. Related issues to the synchronization problem are discussed in more details in [Sari 84]. With such an assumption, only one global sequence of output symbols is expected for a given global sequence of input symbols (i.e., a mixture of input symbols belonging to both machines). A possible way of implementing such a feature, is by providing some coordinating procedures between the different external ports of the system.

From the above described model, it is obvious that the execution of an internal-output transition in one machine implies the execution of another transition in the other machine. If the later transition is also an internal-output one, a third transition will be executed in the starting machine, before any output is observed in any of the ports. This process will continue until an external-output transition is invoked. In such a case, the output of that last transition will be observed in the external port of the machine to which that transition belongs. Because of the complexity of the diagnostic process, we restrict ourselves to the study of systems where the execution of an internal-output transition in one machine will only imply the execution of an external-output transition in the other machine. In other words, the set of output symbols of internal-output

transitions in one machine should be a subset in the set of inputs of external-output transitions in the other machine. Hence, for a pair (Internal-output transition, external-output transition) of transitions provoked by a single internal input, the output of the first transition is hidden (since it is communicated to an internal queue instead of an external port), while the output of the second one must be observable.

We assume that the input alphabet I_i , of a machine M_i ($i = 1, 2$) in a system of CFSMs, is formed by two subsets (i.e, $I_i = IEO_i \cup IIO_i$, where $IEO_i \cap IIO_i = \emptyset$). The first subset IEO_i represents the input symbols, for external-output transitions of M_i , which might be applied from the corresponding external port P_i . IEO_i includes a subset " IEO_{q_i} " containing input symbols, for some external-output transitions, which might be received from the queue q_i instead of the external port P_i . The second subset IIO_i represents input symbols, for the internal-output transitions of M_i , which are also applied from the external port P_i . Similarly, the set of output symbols O_i of a machine M_i can be seen as the union of two subsets (i.e, $O_i = OEO_i \cup OIO_{q_j}$). The first subset OEO_i is formed by the output symbols generated by external-output transitions of M_i and addressed to M_i external port P_i . The second subset OIO_{q_j} is formed by output symbols generated by internal-output transitions of M_i and addressed to the input queue q_j of machine M_j . It is important to note that the input subset IEO_{q_i} of machine M_i is equal to the output subset OIO_{q_i} of machine M_j . From the implementation point of view, the input symbols of the subset IEO_{q_i} (if received from the queue q_i) and the output symbols of the subset OIO_{q_j} of a machine M_i are hidden and can not be observed by an external observer.

A graphic representation of an CFSMs example, in the form of **state transition diagram**, is given in Figure 1 where a system of two communicating machines with two distributed ports is presented. Each machine has an external port for both external input and output interactions and one input queue which receives messages sent by the other machine. Internal transitions are shown in bold lines and external-output transitions are shown in simple lines.

For the example in Figure 1, we have the following sets of inputs and outputs for both machines:

$$IEO_1 = \{b, e\}; IIO_1 = \{a\}; IEO_{q_1} = \{b, e\}; I_1 = IEO_1 \cup IIO_1 \quad \implies I_1 = \{a, b, e\}$$

$$IEO_2 = \{b, f\}; IIO_2 = \{c\}; IEO_{q_2} = \{f\}; I_2 = IEO_2 \cup IIO_2 \quad \implies I_2 = \{b, c, f\}$$

$$OEO_1 = \{c, f\}; OIO_{q_2} = \{f\}; O_1 = OEO_1 \cup OIO_{q_2} \quad \implies O_1 = \{c, f\}$$

$$OEO_2 = \{a, e\}; \quad OIO_{q_1} = \{b, e\}; \quad O_2 = OEO_2 \cup OIO_{q_1} \quad \implies \quad O_2 = \{a, b, e\}$$

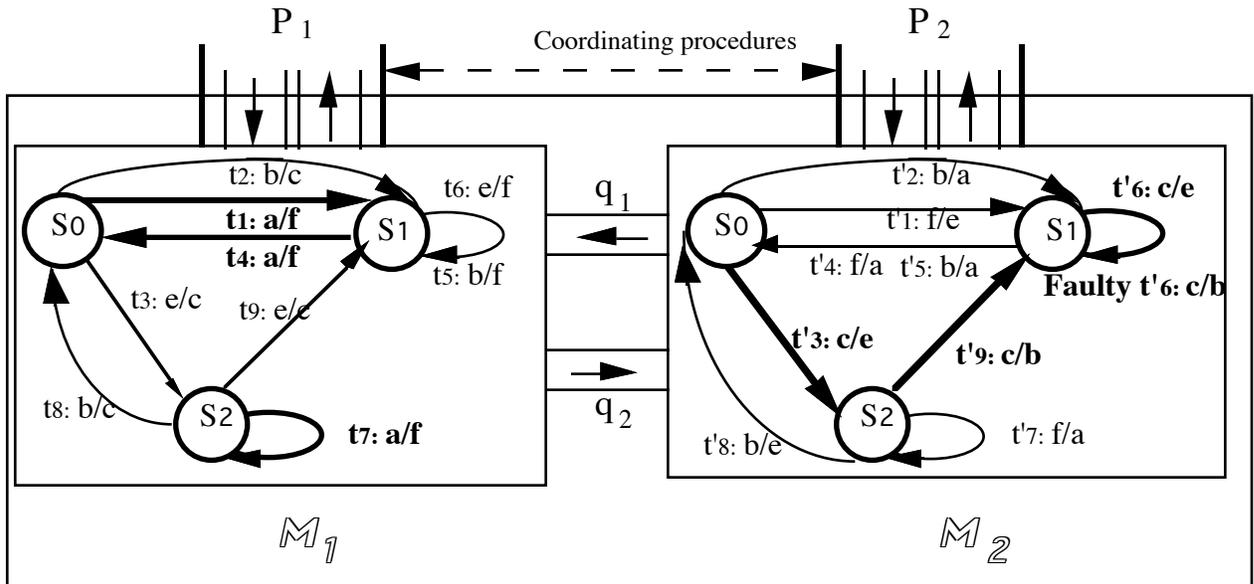


Figure 1: A state transition diagram of a two CFSMs

2.2 The CFSMs fault model

The CFSMs fault model is based on errors and faults made on labeled transitions of the machines. Some of these faults, which are essential for the CFSM-based diagnostic approach discussed in Section 3, are defined as follows:

Definition 2: 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 3: 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.

In the CFSMs model defined in the above subsection, an output can be seen as the composition of two portions: the message type and the address to which that message is destined (i.e., the environment queue or another machine queue). Therefore, output faults can occur in either components. For our diagnostic approach presented in the following section, we assume the following fault model: the implementation under test (IUT) may have a single output fault, where the fault can occur only in the message type component and not in the address component, or a single transfer fault in one of its machines.

3. The diagnostic approach

In [Ghed 92], we proposed a single fault diagnostic algorithm for systems represented by FSMs. In this section, we generalize such an algorithm to the case where distributed system implementations and their models are represented by CFSMs. In such a context, transitions (which may be faulty) in each machine in the distributed system can be seen as the components of the general structured system described earlier.

The following algorithm consists of diagnosing (with respect to its specification CFSMs) an IUT CFSMs for possible faulty transitions. Its main purpose is to identify the faulty transition and to determine the type of its fault (i.e. output or transfer). This work is mainly executed within Step 5 and Step 6 of the following algorithm. In particular, Step 5 might end up with different diagnostic candidates. In such a case, additional diagnostic tests should be selected in Step 6, in order to be able to isolate the faulty transition and more precisely to know to which state (in case of a transfer fault) that transition has transferred.

ALGORITHM:

Step 1: Generation of expected outputs

We assume that a test suite "TS" is given. The test suite consists of a number of test cases which are sequences of input symbols. We write $TS = \{ tc_1; \dots; tc_p \}$, where each tc_i is a test case.

If a test case tc_i consists of m_i inputs: $iP_{i,1}, iP_{i,2}, \dots, iP_{i,m_i}$, the corresponding sequence of expected outputs is written as: $o_i = o_{i,1}^g, o_{i,2}^g, \dots, o_{i,m_i}^g$, where p and g are ports (i.e. $p, g = 1, 2$: external ports in the system) through which interactions get applied or observed and $o_{i,j}^g$ is

expected after input $iP_{i,j}$. In other words, the input symbols in the test cases and their corresponding outputs can be applied and observed in different external ports. It is important to note that the application of an input symbol in a test case might imply the execution of one or two transitions in both machines, depending on whether that input is external or internal.

Step 2: Execution of test cases

Application of the test suite to the IUT. For each test case tc_i , a corresponding output sequence is observed in the ports of the IUT. It is written as: $\hat{o}_i = \hat{o}_{i,1}^g, \hat{o}_{i,2}^g, \dots, \hat{o}_{i,m_i}^g$

Definition 4: The transition $T_{i,j}$ of the specification machine M_k where the symptom ($o_{i,j}^g \neq \hat{o}_{i,j}^g$) has been observed, is called a **symptom transition**. If we have the same symptom transition for all symptoms, that transition is called the **unique symptom transition (ust)**. The observed output generated by the ust, is called the **unique symptom output (uso)**.

Step 3: Generation of symptoms

Compare observed outputs with expected ones and identify all symptoms. Any difference ($o_{i,j}^g \neq \hat{o}_{i,j}^g$) represents a **symptom**. The faulty output corresponding to a symptom is called a **symptom output**.

Step 4: Generation of conflict sets

Algorithm: For each symptom ($o_{i,j}^g \neq \hat{o}_{i,j}^g$) and for each machine M_l in the system, determine its corresponding conflict set. A **conflict set** for a given symptom is defined to be the set of transitions which are supposed to participate (through their execution) in the generation of the symptom output; therefore, at least one of these transitions must be faulty. The conflict set for a machine M_l is formed by all transitions executed in the M_l specification when the corresponding test case is applied. No transitions, executed after the observation of the symptom in a test case, will be included in the conflict set.

Note: In order to continue the diagnostic process, different approaches might be used depending on whether the single or the multiple fault hypothesis is made. In the following, we make the assumption that **the IUT has a single fault, either output or transfer**.

Step 5: Generation of diagnostic candidates and their diagnoses

Diagnostic candidates are transitions which are suspected to be faulty. Therefore, each one of them should belong to each of the last step generated conflict sets. It also has to be consistent with all observations in all initially given test cases.

Step 5A: Generation of initial tentative candidate sets

Algorithm: For each machine M_i in the system, the initial tentative candidate set "**ITCⁱ**" will be formed by the **intersection** of all conflict sets of M_i . Each element T_k in ITC^i represents a tentative candidate transition (with an output or transfer fault) which may explain all symptoms.

Step 5B: The FTC, the ending state and the outputs sets

Algorithm: For each generated initial tentative candidate set ITC^i , if there is a unique symptom transition "**ustⁱ**", it will be contained in the ITC^i (i.e, see definition 3). In that case, we split the ITC^i into the set "**ustsetⁱ**", which will initially contain the ust^i , and the final tentative candidates set for transitions with transfer faults "**FTCtrⁱ**", which will contain the rest of the transitions in ITC^i . Otherwise, the full ITC^i set forms the $FTCtr^i$ set. A third set, called the final tentative candidate set for internal-output transitions with output faults "**FTCcoⁱ**", will be formed by the internal-output transitions included in ITC^i . Each set will be processed separately, as explained in the following paragraphs.

It is clear from the above steps that at most one of the $ustset^i$'s will not be empty. If a ust^i exists, the $ustset^i$ will be processed as follows. All test cases in the initially given test suite "TS" are scanned for transitions that are equal to the ust^i . If for all found transitions their corresponding observed output is equal to the uso^i and for the remaining transitions in the corresponding test cases all observed and expected outputs are equal, which means the ust^i explains all observations, then the ust^i is considered a diagnostic candidate for an output fault.

```
Procedure ust-processing (ustseti)
Forall tcm ∈ TS DO
    Forall iPm,n ∈ tcm DO    {if in iPm,n, (p = i) or iPm,n is an internal input, Tim,n is assigned
                             the corresponding transition of Mi, otherwise, it is null}
        IF (Tim,n = usti) THEN                                {usti is the only element of ustseti}
            IF (δgm,n <> usoi) THEN
                ustseti = ∅; exit                            {the usti is not a diagnostic candidate}
            ELSE IF ogm,n+1 <> δgm,n+1 THEN {l=1, 2, ..., im where n+im is the
                                                             length of the test case tcm}
                ustseti = ∅; exit
        ENDForall
    ENDForall
```

For each transition T_k in the $FTCco^i$'s ($i = 1, 2$), we compute the set of all faulty outputs called "**outputs_k**", T_k might generate. For each transition, we consider all outputs in OC_{qj} (i.e. the set

of output alphabet the internal-output transitions in the machine M_i might generate), with the exception of the expected output of T_k , one at a time. For each output o under consideration, o will be included in $outputs_k$, if under the assumption that o is the output of T_k , the expected and observed outputs are equal for all succeeding transitions in all test cases.

```

Procedure findoutputs (FTCCoi);
  Forall  $T_k$  in FTCCoi Do                                { $T_k$  is the k-th internal-output transition in FTCCoi }
    outputsk := ∅;                                       {outputsk is the set of all faulty outputs  $T_k$  might generate }
    Forall output  $o \in OC_{qj}$  and  $o \neq Output(T_k)$  Do
      flag := true
      Forall  $tc_m \in TS$  Do
        Forall  $iP_{m,n} \in tc_m$  Do {if in  $iP_{m,n}$ , ( $p = i$ ) or  $iP_{m,n}$  is an internal input,
           $T_{m,n}^i$  is assigned the corresponding transition of  $M_i$ , otherwise, it is null}
          IF ( $T_{m,n}^i = T_k$ ) THEN
            Output'( $T_{m,n}^i$ ) =  $o$ ;
            Apply the test case  $tc_m$  to the modified specification
            IF (newly expected outputs  $\neq$  observed outputs) THEN
              flag := false; exit
            ENDForall
          ENDForall
        IF (flag = true) THEN
          outputsk := outputsk ∪ { $o$ }
        ENDForall
      ENDForall
    ENDForall
  ENDForall

```

For each transition T_k in the FTCTrⁱ's ($i = 1,2$), we compute the set of all faulty transfer states called "**EndStates_k**", to which T_k might transfer. For each transition, we consider all states in the machine, with the exception of the expected NextState of T_k , one at a time. For each state s under consideration, s will be included in EndStates_k, if under the assumption that s is the NextState of T_k , the expected and observed outputs are equal for all succeeding transitions in all test cases.

```

Procedure findendingstates (FTCTri);
  Forall  $T_k$  in FTCTri Do                                { $T_k$  is the k-th transition in FTCTri }
    EndStatesk := ∅                                       {EndStatesk is the set of all states to which  $T_k$  might transfer }
    Forall state  $s \in S$  and  $s \neq NextState(T_k)$  Do
      flag := true
      Forall  $tc_m \in TS$  Do
        Forall  $iP_{m,n} \in tc_m$  Do {if in  $iP_{m,n}$ , ( $p = i$ ) or  $iP_{m,n}$  is an internal input,
           $T_{m,n}^i$  is assigned the corresponding transition of  $M_i$ , otherwise, it is null}
          IF ( $T_{m,n}^i = T_k$ ) THEN
            NextState'( $T_{m,n}^i$ ) =  $s$ ;
            Apply the test case  $tc_m$  to the modified specification
            IF (newly expected outputs  $\neq$  observed outputs) THEN

```

```

                                flag := false; exit
                                ENDForall
                                ENDForall
                                IF (flag = true) THEN
                                    EndStatesk := EndStatesk ∪ {s}
                                ENDForall
                                ENDForall

```

Step 5C: Identification of diagnostic candidates and generation of diagnoses

Algorithm: In this step we remove all correct (i.e. transitions with empty ending state sets or empty outputs sets) transitions from the final tentative candidate sets. All transitions in the resulting "D Ctr^i " sets (if not empty) are diagnostic candidates with transfer faults. For each transition T_k in the D Ctr^i 's ($i = 1, 2$) and for each state s_{ik} in the EndStates_k, a diagnose, stating that T_k might transfer to state s_{ik} , is generated. Similarly, all transitions in the resulting "D Cco^i " sets (if not empty) are diagnostic candidates with output faults. For each transition T_k in the D Cco^i 's ($i = 1, 2$) and for each output o_{ik} in the outputs_k, a diagnose, stating that T_k might have a faulty output o_{ik} , is generated. An extra diagnose, stating that the ust^i might have an output fault, is also generated, if the $ustset^i$ is not empty.

Step 6: Additional diagnostic tests

Depending on the results of the previous steps, the following different possibilities might be present.

Case 1: One of the $ustset^i$'s contains the ust^i transition, the D Ctr^i 's and the D Cco^i 's ($i = 1, 2$) are empty. In such a case, the ust^i is the faulty transition with the output fault uso^i and no further diagnostic tests are required.

Case 2: The $ustset^i$'s are empty and all of the D Ctr^i 's and the D Cco^i 's ($i = 1, 2$) are empty, except one of them which is a singleton with a corresponding singleton ending state set or a corresponding singleton outputs set. If the D Ctr^i is not empty, its only transition has a transfer fault to the state in the ending state set, otherwise, the only transition of the D Cco^i has a faulty output included in the corresponding outputs set. No further tests are required.

Case 3: The $ustset^i$'s are empty and one or more of the other sets has more than one element. Therefore, any element of the D Ctr^i 's or the D Cco^i 's ($i = 1, 2$) might be the faulty transition. In such a case, we should process the elements of these sets in order to derive further tests with the purpose of identifying the faulty transition and localizing the exact fault.

Algorithm for Case 3:

Step (a): For each transition T_k in the DCtrⁱs, additional test cases have to be selected and executed, in order to be able to know exactly to which state it transfers. These test cases should have the ability of distinguishing between the different states contained in the corresponding ending state set "EndStates_k" and possibly the correct ending state of the transition. Therefore, a "**limited characterization set**" W_k has to be computed for the states in EndStates_k and the correct state. It is different from the characterization set defined in [Chow 78], since it concerns only a subset of states rather than the whole set of states in the machine. It is formed by sequences of inputs such that if applied to the machine in one of the states in EndStates_k, the produced outputs will be different from the outputs obtained if the same input sequences were applied to the machine in any other state of EndStates_k or the correct state. Each additional test case is a concatenation of an input sequence, called transfer sequence, required to take the machine from its initial state to the starting state of T_k , the input for T_k and a sequence of inputs from the W_k .

Step (b): For the internal-output transitions in the DCcoⁱs, a similar approach to Step (a) is used. Therefore, each additional test case for a transition T_k in a DCcoⁱ is a concatenation of an input sequence, called transfer sequence, required to take the machine M_i from its initial state to the starting state of T_k , the input for T_k and a sequence of inputs from "**the distinguishing set**" U_k . The characteristic of the sequences in U_k is that once incorporated in the additional test cases, they will have the ability of distinguishing between the different possible outputs which might be generated by T_k and communicated to the machine M_j . In other words, if M_j in a state s receives an input symbol x (i.e. the output of T_k) from M_i , it will execute a precise corresponding transition t and will reach a state s' , then, a sequence from U_k will be applied to M_j in state s' . If a faulty input symbol x' (instead of x) is received by M_j in state s , a different transition t' will be executed and possibly a different output will be generated and a different state will be reached. Therefore, the different sequences of U_k will identify such an anomaly. Consequently, if the application of these additional tests generates the expected outputs, the transition T_k is declared correct and is removed from the DCcoⁱ. When a faulty transition is found, the analysis of observed outputs will identify the faulty output of that transition and the search is stopped.

In order to avoid any ambiguities, the transfer sequence, the limited characterization set and the distinguishing set should be chosen in such a manner that they do not involve any candidate transition in any of the DCtrⁱs or the DCcoⁱs ($i = 1, 2$) sets. Figure 2 illustrates the progressive construction of the additional test cases needed to distinguish the faulty transition from the rest of

the diagnostic candidates of DCtr's. A similar picture will illustrate the progressive construction of additional tests for DCco's.

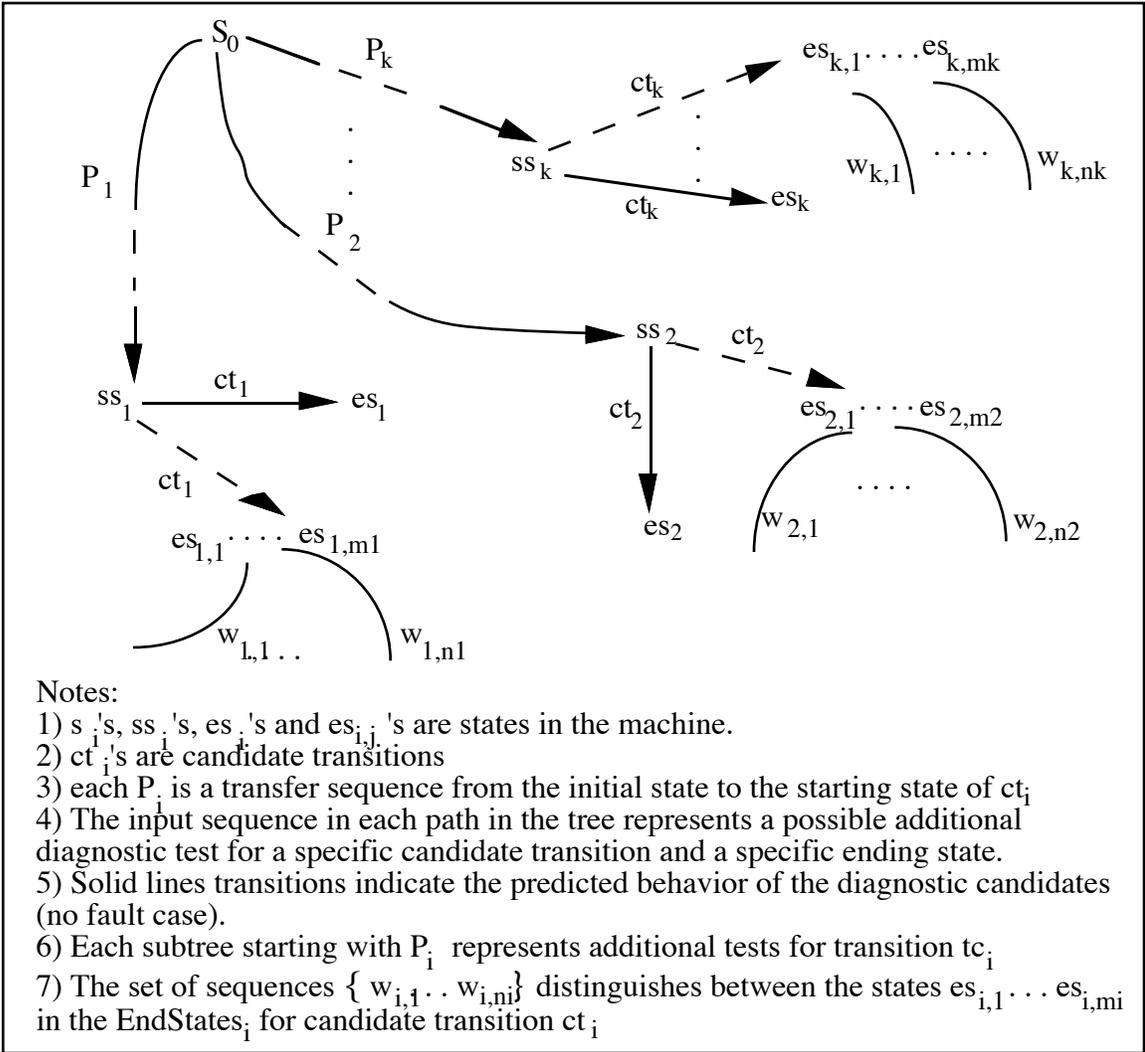


Figure 2 : Construction of additional diagnostic tests

The construction of the additional tests is progressive because if the fault is located, the rest of these additional tests need not be generated, since we work under the single fault hypothesis. If some of the generated tests are already included in the initially given test suite, this will be taken into consideration for the analysis of the obtained outputs, but they need not be applied again to the IUT. If the application of these additional tests generates the expected outputs, the transition is declared correct and is removed from the corresponding diagnostic candidate set. When a faulty transition is found, the analysis of observed outputs identify the wrong transfer or the wrong output of the transition and the search is stopped.

Case 4: One of the $ustset^i$ s contains the ust^i transition and one or more of the other sets has more than one element. In such a case, we first check the ust^i transition by generating for it an additional test case which will imply the execution of a sequence of transitions not included in any of the sets $DCtr^i$ s and $DCco^i$ s ($i = 1, 2$). Such an additional test case should terminate by the input of ust^i . If its application generates the expected output, then the ust^i is declared correct and the search for the faulty transition in the other sets has to be done as in Case 3, otherwise, ust^i is the transition with an output fault and the search is stopped.

4. An application example

Suppose that the following initial test suite for the two CFSMs specification shown in Figure 1, is given:

$$TS = \{R, b^2, c^2, a^1, c^2, b^1; R, a^1, f^2, c^2, b^1; R, b^1, c^2, a^1, b^2, e^1, f^2\}$$

Step 1 and 2: The application of TS to the specification and the implementation (i.e. equal to the specification with the exception of transition t'_6 has an output fault) of Figure 1, yields the expected and observed output sequences, as shown in Table 1.

tc. #	tc2	tc1
Input	a1, c2, b1	a1, f2, c2, b1, b1, c2, a1, b2, e
Spec. transitions	tr, t'2, t'3, t'4, t'5	t1t'1, t'4, t'3t6, t'2, t'3t6, t4t'7,
Expected output	a2, c1, a2, c1, f1	e2, a2, f1, f1, c1, f1, a2,
Observed output	a2, c1, a2, c1,	-, e2, a2, f1, f1, c1, f1, a2,

Table 1: Test cases and their outputs

In Table 1, a reset transition tr is assumed to be available for both the specification and the implementation. It resets both machines in the system to their initial states. We use the symbol "R" to denote the input for such a transition and the symbol "-" to denote its output.

Step 3: A difference between observed and expected outputs is detected for test cases tc_1 .

Therefore, the symptom is:

"Symp₁ = (o¹_{1,6} ≠ ô¹_{1,6})" with the symptom transition t₅.

Step 4: Corresponding to the above symptom, we generate the following two conflict sets for both machines:

$$\text{Conf}^1_1 = \{t_3, t_7, t_9, t_5\}, \text{Conf}^2_1 = \{t'_2, t'_6, t'_4, t'_3\}$$

Step 5A: Since there is only one conflict set for each machine, no intersection is needed. The two initial sets of tentative candidates for both machines are the following:

$$\text{ITC}^1 = \{t_3, t_7, t_9, t_5\}, \text{ITC}^2 = \{t'_2, t'_6, t'_4, t'_3\}$$

Step 5B: For each ITCⁱ (i = 1, 2), we generate its corresponding FTCtrⁱ, FTCcoⁱ and the ustsetⁱ sets:

$$\begin{aligned} \text{FTCtr}^1 &= \{t_3, t_7, t_9\}, & \text{ustset}^1 &= \{t_5\}, & \text{FTCco}^1 &= \{t_7\} \\ \text{FTCtr}^2 &= \{t'_2, t'_6, t'_4, t'_3\}, & \text{ustset}^2 &= \{\}, & \text{FTCco}^2 &= \{t'_6, t'_3\} \end{aligned}$$

The processing of the above sets and the computation of the outputs and the ending state sets for the transitions in FTCtrⁱs and FTCcoⁱs (i = 1, 2) leads to:

$$\begin{aligned} \text{ustset}^1 &= \{\}, \\ \text{EndStates}[t_3] &= \{\}, & \text{EndStates}[t_7] &= \{s_0\}, & \text{EndStates}[t_9] &= \{s_0, s_2\} \\ \text{outputs}[t_7] &= \{\} \end{aligned}$$

$$\begin{aligned} \text{ustset}^2 &= \{\}, \\ \text{EndStates}[t'_2] &= \{s_0, s_2\}, & \text{EndStates}[t'_6] &= \{s_2\}, \\ \text{EndStates}[t'_4] &= \{s_2\}, & \text{EndStates}[t'_3] &= \{\} \\ \text{outputs}[t'_6] &= \{b\}, & \text{outputs}[t'_3] &= \{b\} \end{aligned}$$

Step 5C: The transitions with empty ending state sets or empty outputs sets are correct, therefore they are removed from their final tentative candidate sets. The resulting diagnostic candidates sets are the following:

$$\begin{aligned} \text{DCtr}^1 &= \{t_7, t_9\}, \text{ustset}^1 = \{\}, \text{DCco}^1 = \{\} \\ \text{DCtr}^2 &= \{t'_2, t'_6, t'_4\}, \text{ustset}^2 = \{\}, \text{DCco}^2 = \{t'_6, t'_3\} \end{aligned}$$

With the use of the ending state sets and the outputs sets generated in Step 5B, the following diagnoses are deduced:

Diag1: t_7 might transfer to state s_0 instead of state s_2 .

Diag2: t_9 might transfer to state s_0 instead of state s_1 .

Diag3: t_9 might transfer to state s_2 instead of state s_1 .

Diag4: t'_2 might transfer to state s_0 instead of state s_1 .

Diag5: t'_2 might transfer to state s_2 instead of state s_1 .

Diag6: t'_6 might transfer to state s_2 instead of state s_1 .

Diag7: t'_4 might transfer to state s_2 instead of state s_0 .

Diag8: t'_6 might have an output fault of b instead of e .

Diag9: t'_3 might have an output fault of b instead of e .

Step 6: In order to reduce the number of these diagnoses, additional diagnostic tests have to be selected. Since output faults are in general easier to be tested and require less tests, we start with Diag8 . As indicated in the proposed algorithm, other diagnostic candidates have to be avoided from the path of transitions executed by the additional test case. A possible transfer sequence which will take the machine to the starting state s_1 of t'_6 is " R, f^2 ". After the execution of t'_6 in the machine M_2 and depending on the generated output, the machine M_1 in state s_0 will execute t_3 or t_2 and consequently will transfer to states s_2 or s_1 , respectively. Since t_3 and t_2 generate the same output, another input to be applied to machine M_1 is needed to distinguish the states s_2 and s_1 . A possible input symbol is " b ". Therefore, after the application of the additional test case " R, f^2, c^2, b^1 ", we observe " R, e^2, c^1, f^1 " as output. Such a result confirms that t'_6 is faulty and generates the output " b " instead of " e " as specified. Since it is assumed that there is at most one fault in IUT, the fault is localized and the remaining diagnoses are discarded.

5. Concluding discussion

In [Ghed 92], we proposed a diagnostic algorithm for systems represented by a single FSM. Such an algorithm localizes the faulty transition in the system once the fault has been detected. It generates, if necessary, additional diagnostic test cases which depend on the observed symptoms and which permit the location of the detected fault. The algorithm guarantees the correct diagnosis of any single (output or transfer) fault in a system represented by a single FSM. In this paper, we generalize the diagnostic approach to the case where distributed system specifications

and implementations are represented by CFSMs. The main differences, in comparison with [Ghed 92] introduced in this paper, are the following:

a) Two groups of conflict sets for both machines in the distributed system are generated, instead of only one in [Ghed 92]. They resulted from the split of the sequences of transitions corresponding to the test cases where symptoms have been observed. If a symptom is observed in a test case which does not include internal-output transitions or, it includes internal-output transitions of only one machine M_i and the symptom transition belongs to the same machine, conflict sets are generated only for the machine to which the symptom transition belongs. In such a case, important processing time will be saved in the remaining steps by declaring the other machine correct. In all other cases, the identification of the faulty transition will precede the identification of the faulty machine.

b) A new set of special diagnostic candidates might be created for each machine in the system. It is formed by those internal-output transitions which are candidates for possible output faults. Such a set and the algorithm for its computation are not needed when systems to be diagnosed are represented by a single FSM. Consequently, special techniques were added to Step 6 of the proposed algorithm, in order to generate additional test cases for those candidates suspected for having output faults.

From an optimization point of view, it is recommended to combine Steps 5 and 6 of the diagnostic algorithm. In other words, it will be more efficient to process one candidate using Steps 5B and 5C, then select its additional tests using Step 6 before the processing of the next candidate. If the application of those tests identifies the faulty transition and localizes its fault, there will be no need to process the rest of the candidates which will, in general, save us a lot of computation time. If not, we repeat the same process until the localization of the fault.

Many important questions are left for future work, such as the diagnostic of distributed systems which are represented by CFSMs and have non-deterministic behaviors. The non-determinism can be caused by the absence of synchronization between the different queues for the different machines of the distributed system. The extension of the CFSMs fault model is also recommended to cover, for example, addressing faults, which are not considered in this paper. Another important question, is the diagnostic of systems having multiple faults, which is known to be a very difficult problem. A possible starting point is to try to solve such a question for at least some special classes of multiple faults.

Acknowledgments: The authors would like to thank G. Luo for discussions on the CFSMs model introduced in this paper. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada, the Ministry of Education of Québec and the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols.

References

- [Boch 91] G.v. Bochmann, R. Dssouli, A. Das, M. Dubuc, A. Ghedamsi and G. Luo, "Fault models in testing", Invited paper in 4-th IWPTS, Leidschendam, Holland, 15 - 17 Oct. 1991.
- [[Chow 78] T.S. Chow, "Testing Design Modelled by Finite-State Machines", IEEE Trans. S.E. 4, 3, 1978.
- [Davi 88] R. Davis, and W. Hamscher, "Model-based reasoning: Troubleshooting", in: Exploring Artificial Intelligence, edited by Shrobe, H. E. and the American Association for Artificial Intelligence, pp. 297-346, Morgan Kaufman, 1988.
- [Fuji 91] S. Fujiwara, G.v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, "Test selection based on finite state models", IEEE Trans. on Software Engineering, Vol. 17, No. 6, June 1991, pp. 591-603.
- [Gene 84] M.R. Genesereth, "The use of design descriptions in automated diagnosis", Artificial Intelligence 24(3), 1984, pp. 411-436.
- [Ghed 92] A. Ghedamsi and G.v. Bochmann, "Test result analysis and diagnostics for finite state machines", accepted at the 12-th international conference on distributed systems, Yokohama, Japan, June 9-12, 1992.
- [Ghed 92a] A. Ghedamsi and G.v. Bochmann, "Diagnostic Tests for Finite State Machines", TR No. 807, Univ de Montréal, Montréal, January 1992.
- [Gone 70] G. Goenenc, "A method for the design of fault detection experiments", IEEE Trans. Computer, Vol. C-19, pp. 551-558, June 1970.
- [Klee 87] J. de Kleer, and B.C. Williams, "Diagnosing multiple faults", Artificial Intelligence 32(1), 1987, pp. 97-130.
- [Klee 89] J. de Kleer, and B. C. Williams, Diagnosing with behavioral models, Proceedings IJCAI, Detroit-Michigan, 1989, pp. 1324-1330.
- [Koka 90] K.C. KO, "Protocol test sequence generation and analysis using AI techniques", Master thesis, Dept of Comp. Sci., UBC, Jul. 1990.
- [Kore 88] B. Korel, "PELAS-Program error-locating assistant system", IEEE Trans. on Software Engineering, Vol. 14, No. 9, September 1988.
- [More 90] L. J. Morell, "A theory of fault based testing", IEEE Trans. on Software Engineering, Vol. 16, No. 8, August 1990.
- [Nait 81] S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition-Tours", Proc. of FTCS (Fault Tolerant Computing Systems), pp.238-243, 1981.
- [Reit 87] R. Reiter, "A theory of diagnosis from first principles", Artificial Intelligence 32(1), 1987, pp. 57-96.
- [Sabn 88] K.K. Sabnani and A.T. Dahbura, "A protocol Testing Procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, 1988.
- [Sari 84] B. Sarikaya, and G.v. Bochmann, Synchronization and specification issues in protocol testing, IEEE Trans. on Communications, Vol. COM-32, No. 4, April 1984, pp. 389-395.
- [Scho 76] E.H. Shortlife, "Computer-based Medical Consultations : MYCIN", Elsevier, New-York, 1976.

- [Stru 89] P. Struss, and O. Dressler, "Physical Negation - Integrating Fault Models into the General Diagnostic Engine", Proceedings IJCAI, Detroit - Michigan, 1989, pp. 1318-1323.
- [Ural 87] H. Ural, "A Test Derivation Method for Protocol Conformance Testing", Proc. of the 7th IFIP Symposium on Protocol Specification, Testing and Verification, Zurich, May 5-8 1987.
- [Vuon 90] S.T. Vuong and K.C. Ko, "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.