Diagnosis of Single Transition Faults in Communicating 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 or transfer) may be present in one of the transitions of a deterministic system represented by a set 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 faults. The algorithm guarantees the correct diagnosis of any single or double faults (output and/or transfer) in at most one of the transitions of a deterministic system which is represented by a set 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 [4, 3, 16, 18, 14, 2, 8]. 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 [6, 19]. Diagnostic is a well documented subject in other areas such as Artificial Intelligence (AI), 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 [11, 15], 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. 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 [11].

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 a 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 in the selection of some additional new tests called **"distinguishing tests"** [5]. 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 problem is a very complicated task, specially for diagnosing complicated systems such as 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 [1]). 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

0-8186-3770-6/93 $3.00 © 1993 IEEE

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 [11]. 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 (e.g., stuck at 0/1 fault models) [12]. 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 in transitions, is used for diagnosing implementations modeled by FSMs [2, 19, 6].

In this paper, we propose a diagnostic algorithm for deterministic systems represented by CFSMs[9, 10]. In such a model, transitions are considered as components in the above described general model, while states have the function of connecting these components. We solve the diagnostic problem for the single transition faults (output and/or transfer) hypothesis, which means that the implementation under test is allowed to have up to two (output and/or transfer) faults in at most one transition. The algorithm in this paper is a generalization to the algorithms presented in [6, 7]. It extends the class of systems to be diagnosed from systems represented by single FSMs to systems represented by N-CFSMs, where $N \geq 2$. It also extends the assumed fault model from a single (an output or a transfer but not both) fault to single transition (an output and/or a transfer) faults. The new proposed algorithm will have the ability of localizing the faults once an error is detected by one or several test cases, which may be generated by one of the existing test selection methods [13].

The fact that in some cases it is possible to transform a set of CFSMs into an equivalent single machine with an exponential algorithm, is not a good reason to stop us from trying to solve the diagnostic question for systems of CFSMs. The equivalent machine is, in general, too big and is less convenient to handle. To avoid the high transformation cost and the state explosion problem in the resulting machine, we propose to solve the diagnostic problem directly for the CFSMs model. Compared with the case of single FSM diagnostics, more work needs to be done to diagnose CFSMs. This becomes evident in Section 3, where a set of transitions suspected of having output faults has to be identified. Such a set was not needed in the case of single FSMs.

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 deterministic system 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 ($i = 1, 2,...N$) in a system of N CFSMs can be represented by a quintuple (S_i, I_i, O_i, NextStaFunc$_i$, OutFunc$_i$) where :

N: Number of FSMs in the system

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

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

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

NextStaFunc$_i$: Next-state function, $S_i \times I_i$ --> S_i

OutFunc$_i$: Output function, $S_i \times I_i$ --> Y_i.

For the rest of the paper, we assume that each machine M_i in the distributed system has a separate external port, P_i, through which input and output symbols are communicated between the machine and the external world. In addition, each machine M_i has N-1 internal input queues: $q_{i<1}, q_{i<1}, ..., q_{i<i-1}, q_{i<i+1}, ..., q_{i<N}$, where $q_{i<j}$ represents the internal input queue for M_i receiving its symbols from the machine M_j.

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. In this paper, we assume that each machine of the CFSMs has an input alphabet composed of two distinct subsets of inputs. The first subset **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.

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 [17]. 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 the different 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 a second machine. If the later transition is also an internal-output one, a third transition will be executed in a third 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 another 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 for external-output transitions in the other machines. 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 of one the other machines in the system instead of the external port), while the output of the second one must be observable in the external port of the receiving machine.

We assume that the input alphabet I_i, of a machine M_i ($i = 1, 2, ..., N$) in a system of CFSMs, is formed by two subsets (i.e., $\mathbf{I_i = IEO_i \cup IIO_i}$, where $\mathbf{IEO_i \cap IIO_i = \emptyset}$). The first subset $\mathbf{IEO_i}$ represents the input symbols, for external-output transitions of M_i, which might be applied from the corresponding external port P_i. $\mathbf{IEO_i}$ includes a subset "$\mathbf{IEOq_{i<1} \cup IEOq_{i<2} \cup ... \cup IEOq_{i<i-1} \cup IEOq_{i<i+1} \cup ... \cup IEOq_{i<N}}$" containing input symbols, for some external-output transitions, which might be received from the different queues of M_i: $q_{i<1}, q_{i<1}, ..., q_{i<i-1}, q_{i<i+1}, ..., q_{i<N}$ *instead of the external port* P_i. *The second subset* $\mathbf{IIO_i}$ represents input symbols for the internal-output transitions of M_i. It is formed by different subsets ("$\mathbf{IIO_i = IIO_{i>1} \cup IIO_{i>2} \cup ... \cup IIO_{i>i-1} \cup IIO_{i>i+1} \cup ... \cup IIO_{i>N}}$", where $\mathbf{IIO_{i>x} \cap IIO_{i>y}} = \emptyset$, $i \neq x,y$ and $x \neq y$). Each subset $\mathbf{IIO_{i>j}}$, $j \neq i$, contains inputs for internal output transitions of M_i which communicate their outputs to the machine M_j. These input symbols are only applied from the external port P_i of M_i.

Similarly, the set of output symbols O_i of the machine M_i *can be seen as the union of two subsets (i.e.,* $\mathbf{O_i = OEO_i \cup OIO_i}$). The first subset OEO_i is formed by the output symbols generated by external-output transitions of M_i and addressed to M_i's external port, P_i. The second subset, ("$\mathbf{OIO_i = OIO_{i>1} \cup OIO_{i>2} \cup ... \cup OIO_{i>i-1} \cup OIO_{i>i+1} \cup ... \cup OIO_{i>N}}$", is formed by output symbols generated by internal-output transitions of M_i and addressed to the input queues: $q_{1<i}$, $q_{2<i}, ..., q_{i-1<i}, q_{i+1<i}, ..., q_{N<i}$ of machines $M_1, M_2, ..., M_{i-1}, M_{i+1}, ..., M_N$, respectively. It is important to note that the input subset $IEOq_{i<j}$ of machine M_i is equal to the output subset $OIO_{j>i}$ *of machine* M_j. From the implementation point of view, the input symbols of the subset $IEOq_{i<j}$ of the machine M_i (if received from the queue $q_{i<j}$) and the output symbols of the subset $OIO_{j>i}$ of a machine M_j 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 three communicating machines with three distributed ports is presented. Each machine M_i in the system has an external port for both external input and output interactions and an input queue for each machine M_j ($j \neq i$) which receives messages sent by M_j. In each machine M_i of the system, we show external-output transitions in simple lines, one group of internal transitions (with outputs designated to one machine) in continued bold lines, while the other group (with outputs designated to the other machine) are shown in dashed bold lines.

For the example in Figure 1, we have the following *finite sets of inputs and outputs for the three machines in* the system:

$IEO_1 = \{a, b\}$; $IEO_{q1<2} = \{a, b\}$; $IEO_{q1<3} = \{a, b\}$;
$IIO_{1>2} = \{c, d\}$; $IIO_{1>3} = \{e, f\}$ ==>$IO_1 = \{c, d, e, f\}$
$I_1 = IEO_1 \cup IIO_1$ ==> $I_1 = \{a, b, c, d, e, f\}$
$OEO_1 = \{c', d'\}$;
$OIO_{1>2} = \{c', d'\}$; $OIO_{1>3} = \{c', d'\}$ ==> $OIO_1 = \{c', d'\}$
$O_1 = OEO_1 \cup OIO_1$ ==> $O_1 = \{c', d'\}$
$IEO_2 = \{c', d', o, p\}$; $IEO_{q2<1} = \{c', d'\}$; $IEO_{q2<3} = \{o, p\}$;
$IIO_{2>1} = \{q, r\}$; $IIO_{2>3} = \{s, t\}$ ==>$IIO_2 = \{q, r, s, t\}$
$I_2 = IEO_2 \cup IIO_2$ ==> $I_2 = \{c', d', o, p, q, r, s, t\}$
$OEO_2 = \{a, b\}$;
$OIO_{2>1} = \{a, b\}$; $OIO_{2>3} = \{u, v\}$ ==>$OIO_2 = \{a, b, u, v\}$
$O_2 = OEO_2 \cup OIO_2$ ==> $O_2 = \{a, b, u, v\}$
$IEO_3 = \{c', d', u, v\}$; $IEO_{q3<1} = \{c', d'\}$; $IEO_{q3<2} = \{u, v\}$
$IIO_{3>1} = \{w, x\}$; $IIO_{3>2} = \{y, z\}$ ==>$IIO_3 = \{w, x, y, z\}$
$I_3 = IEO_3 \cup IIO_3$ ==> $I_3 = \{c', d', u, v, w, x, y, z\}$
$OEO_3 = \{a, b\}$;

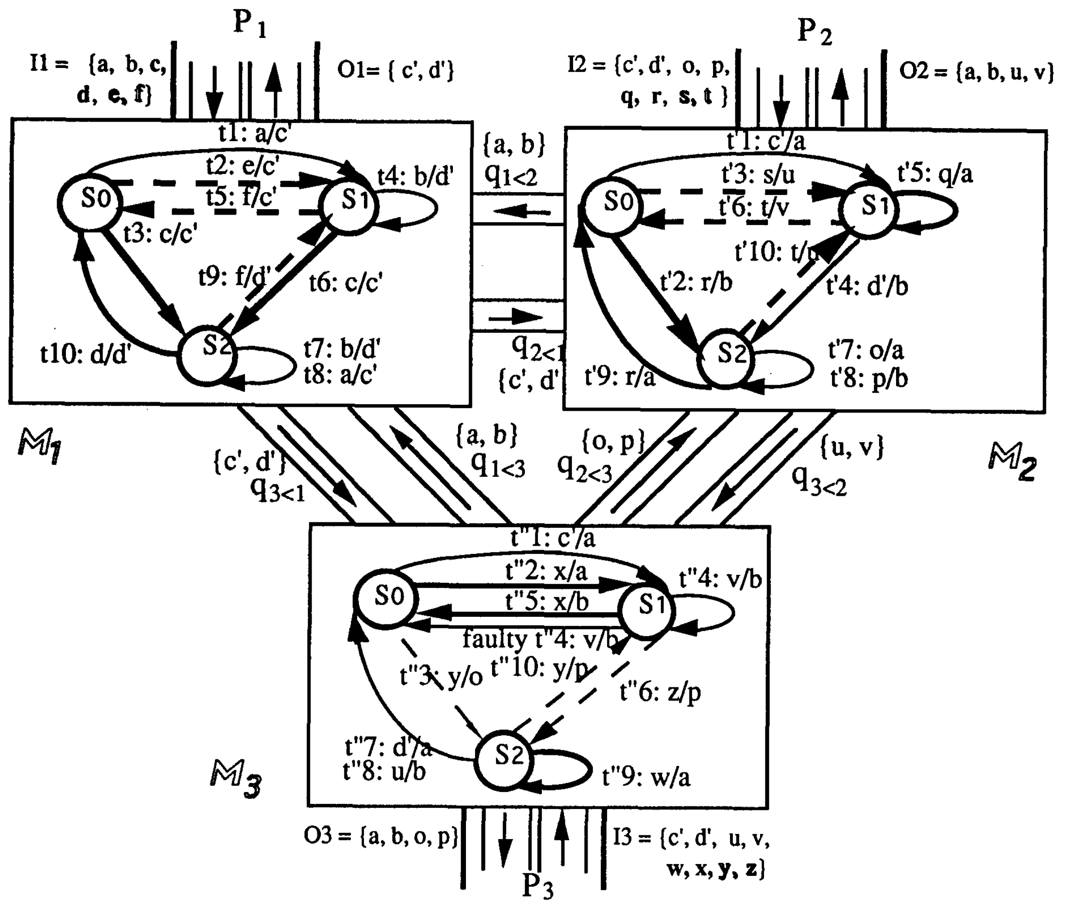


Figure 1: A state transition diagram for a three Communicating Finite State Machines

$OIO_{3>1} = \{a, b\}; OIO_{3>2} = \{o, p\} \Longrightarrow OIO_3 = \{a, b, o, p\}$

$O_3 = OEO_3 \cup OIO_3 \Longrightarrow O_3 = \{a, b, o, p\}$

2.2 The CFSMs fault model

The CFSMs fault model is based on 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 is considered to be composed of two components: the message type and the address to which that message is destined (e.g., the environment queue or another machine queue). Output faults may occur in either component. For our diagnostic approach presented in the following section, we assume the following fault model: the implementation under test (IUT) may have **an output fault**, where the fault can occur only in the message type component and not in the address component, **and/or a transfer fault in at most one transition in one of its machines.**

3. The diagnostic approach

In this section, we present a diagnostic algorithm for deterministic systems represented by CFSMs. Such an 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 faults (i.e., output and/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; ...; tc_p}, where each tc_i is a test case.

If a test case tc_i consists of m_i inputs: $i^p_{i,1}, i^p_{i,2}, \dots, i^p_{i,mi}$, the corresponding sequence of expected outputs is written as: $o_i = o^g_{i,1}, o^g_{i,2}, \dots, o^g_{i,mi}$, where p and g are ports (i.e. p, g ∈ {1, 2, ..., N}: external ports in the system) through which interactions get applied or observed and $o^g_{i,j}$ is expected after input $i^p_{i,j}$. 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 for an external or an internal output.

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: $\hat{o}_i = \hat{o}^g_{i,1}, \hat{o}^g_{i,2}, \dots, \hat{o}^g_{i,mi}$.

Definition 4: The transition $T_{i,j}$ of the specification machine M_k where the first symptom ($o^g_{i,j} \neq \hat{o}^g_{i,j}$) in test case tc_i has been observed, is called a **symptom transition.** If we have the same symptom transition for all test cases first 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 ($o^g_{i,j} \neq \hat{o}^g_{i,j}$) represents a **symptom.** The faulty output corresponding to a symptom is called a **symptom output.**

Note: In order to continue the diagnostic process, different approaches might be used depending on the assumed fault model. In the following, we make the assumption that **the IUT might have at most one faulty transition with an output and/or a transfer fault.**

Step 4: Generation of conflict sets

Algorithm: For each test case tc_i with symptoms and for each machine M_l in the system, determine its corresponding conflict set. A **conflict set** for a given test case is defined to be the set of transitions which are supposed to participate (through their execution) in the generation of the symptom outputs in tc_i. 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 first symptom in tc_i, will be included in the corresponding conflict set of M_l. More formally, we suppose that the following two output sequences (the first is expected and the second is observed) correspond to the test case, "tc_i = R, $i^p_{i,1}$, $i^p_{i,2}$, $i^p_{i,3}$, ..., $i^p_{i,m}$, $i^p_{i,m+1}, \dots, i^p_{i,n}$", with one or more symptoms:

$o_i = o^g_{i,1}, o^g_{i,2}, o^g_{i,3}, \dots, o^g_{i,m}, o^g_{i,m+1}, \dots, o^g_{i,n}$

$\hat{o}_i = o^g_{i,1}, o^g_{i,2}, o^g_{i,3}, \dots, o^g_{i,m}, \hat{o}^g_{i,m+1}, \dots, \hat{o}^g_{i,n}$

where ($o^g_{i,m+1} <> \hat{o}^g_{i,m+1}$) is a symptom. The conflict set for the machine M_l is formed by the projection on the transitions of M_l which belong to the specification sub-sequence of transitions which corresponds to the input sub-sequence "$i^p_{i,1}$, $i^p_{i,2}$, $i^p_{i,3}$, ..., $i^p_{i,m}$, $i^p_{i,m+1}$".

Note: The **flag** is set to true if If ($o^g_{i,m+2}, \dots, o^g_{i,n} < > \hat{o}^g_{i,m+2}, \dots, \hat{o}^g_{i,n}$).

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 conflict sets generated in the last step. 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 "**ITC^i**" will be formed by the **intersection** of all its conflict sets. Each element T_k in ITC^i represents a tentative candidate transition (with an output and/or a transfer fault) which may explain all symptoms.

Step 5B: The *FTC*, the *EndStates*, the *outputs*, and the *statout* sets

Algorithm: For each generated initial tentative candidate set ITC^i, if there is a unique symptom transition "**ust^i**", it will be contained in the ITC^i (i.e, see definition 4). In that case, we split the ITC^i into the set "**ustset^i**", which will initially contain the ust^i, and the set of final tentative candidates for transitions with transfer faults "**FTCtr^i**", 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 possibly only output faults or both output and transfer faults, "**FTCco^i**", will be formed by the internal-output transitions included in ITC^i. Each group of sets 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. Depending on the value of flag established in Step 4, two cases are distinguished:

Case 1 (flag = true): The ust^i transition should be shecked for having both (output and transfer) faults.

Case 2 (flag = false): The ust^i transition should be shecked only for having an output fault.

In the following procedure, called *ustprocessing*, we consider the above both cases. If flag is true a set called *statout* will be computed for all couples (state, uso^i) of faults in ust^i which explain all observed outputs. If not a different set called *outputs* will be computed for uso^i and will contain the faulty output uso^i if the assumption that ust^i has the output uso^i explains all observed outputs.

```
Procedure ustprocessing (ustset^i, flag)
  If ( (ustset^i ≠ ∅) and flag) Then
  {statout(ust^i) is the set of all couples, (state, uso^i), of faults ust^i might have}
      statout(ust^i) := ∅
      processtate&out(ust^i, i, uso^i, statout)
  Else If (ustset^i ≠ ∅) Then
      outputs(ust^i) := ∅
      calouts (ust^i, uso^i, outputs)
```

We now consider each set $FTCco^i$ (i = 1, 2, ..., N), one at a time. For each transition in $FTCco^i$, we check all outputs in the set $OIO_{i>j}$ (i.e., the set of the output alphabet for the sub-set of internal-output transitions of M_i to which T_k belongs and used to communicate with the machine M_j), with the exception of the expected output of T_k, one at a time. For each output o and transition T_k, of the machine M_i, and depending on the value of the flag, the following procedure, called *inttransproc*, computes the set *statout* or the set *outputs*. If flag is true a set called *statout* will be computed for T_k with all couples (state, o) of faults which explain all observed outputs. If not a different set called *outputs* will be computed for T_k and will contain all output faults o which explain all observed outputs.

```
Procedure inttransproc (FTCco^i, flag);
  Forall1 T_k in FTCco^i Do
  {T_k is the k-th internal-output transition in FTCco^i }
   if flag then
       statout[T_k ]:= ∅;
  {statout[T_k ] is the set of all couples, (state, output), of faults T_k might have}
       Forall2 o ∈ OIO_{i>j} and o ≠ Output(T_k) Do
           processtate&out(T_k , i, o, statout)
       ENDForall2
   else
       outputs[T_k ] := ∅;
  {outputs[T_k ] is the set of all faulty outputs T_k might generate}
       Forall2 o ∈ OIO_{i>j} and o ≠ Output(T_k) Do
          calouts (T_k , o, outputs)
       ENDForall2
  ENDForall1
```

```
Procedure processtate&out(cand, i, o, statout)
Forall1 s ∈ S_i and s ≠ NextState(cand) Do
      flag1 := true
      Forall2 tc_m ∈ TS  Do
{if in i^p_{m,n} we have p = i, or i^p_{m,n} is an internal input, then T^p_{m,n} is assigned the corresponding transition of M_i, otherwise, it is null}
          Forall3 i^p_{m,n} ∈ tc_m Do
              If  (T^p_{m,n} = cand) Then
{let the ending state of T^p_{m,n} in the specification be s}
                  NextState'(T^p_{m,n}) := s;
{let the output of T^p_{m,n} in the specification be o}
                  Output''(TP_{m,n}) := o
      Apply the test case tc_m to the modified specification
      If (newly set of expected output sequences <> set of observed outputs) Then  flag_1 := false; exit
            EndForall3
        EndForall2
        IF (flag_1 = true) Then
        statout(cand) := statout(cand) ∪ {[s,o]}
   EndForall1
```

```
Procedure calouts (cand, o, outputs)
      flag1 := true
      Forall1 tc_m ∈ TS  Do
          Forall2 i^p_{m,n} ∈ tc_m Do
              IF  (T^p_{m,n} = cand) THEN
                  Output'(T^p_{m,n}) = o;
      Apply the test case tc_m to the modified specification
          IF (newly expected outputs <> observed outputs)
              THEN    flag1 := false; exit
            ENDForall2
          ENDForall1
          IF (flag1 = true) THEN
           outputs(cand) := outputs(cand) ∪ {o}
```

For each transition T_k in the $FTCtr^i$s (i = 1, 2, ..., N), we compute the set of all faulty transfer states called "*EndStates*(T_k)", to which T_k might transfer. For each transition, we consider all states in the machine M_i, 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*(T_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 (FTCtr^i);
  Forall1 $T_k$ in $FTCtr^i$ Do
{$T_k$ is the k-th transition in $FTCtr^i$ }
    EndStates($T_k$) := $\varnothing$
{EndStates($T_k$) is the set of all states to which $T_k$ might transfer }
    Forall2 state $s \in S_i$ and $s \neq NextState(T_k)$ Do
      flag1 := true
      Forall3 $tc_m \in TS$ Do
        Forall4 $i^p_{m,n} \in tc_m$ Do
{if in $i^p_{m,n}$ we have p = i, or $i^p_{m,n}$ is an internal input, then $T^p_{m,n}$ is assigned the corresponding transition of $M_i$, otherwise, it is null}
          IF ($T^p_{m,n} = T_k$) THEN
            NextState'($T^p_{m,n}$) = s;
      Apply the test case $tc_m$ to the modified specification
        IF (newly expected outputs <> observed outputs)
          THEN flag1 := false; exit
        ENDForall4
      ENDForall3
      IF (flag1 = true) THEN
        EndStates($T_k$) := EndStates($T_k$) $\cup$ {s}
    ENDForall2
  ENDForall1
```

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

Algorithm: In this step we remove all correct (i.e. transitions with empty *EndStates*, empty *statout* and empty *outputs*) transitions from the final tentative candidate sets. All remaining transitions in an $FTCtr^i$ set form a **"$DCtr^i$"** set (if not empty) of diagnostic candidates with transfer faults in machine M_i. For each transition T_k in the $DCtr^i$s (i = 1, 2, ..., N) and for each state s_{ik} in the set EndStates(T_k), a diagnose, stating that T_k might transfer to state s_{ik}, is generated. If *statout* sets are not empty, we generate corresponding diagnoses which suspect the remaining transitions in **"$DCco^i$"** sets and the $ustset^i$ (if not empty), for having both output and transfer faults. If *outputs* sets are not empty, we generate corresponding diagnoses which suspect the remaining transitions in **"$DCco^i$"** sets and the $ustset^i$ (if not empty) for having only output faults.

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 with a corresponding singleton *outputs* set, the $DCtr^i$s and the $DCco^i$s (i = 1, 2, ..., N) are all 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: One of the $ustset^i$s contains the ust^i transition with a corresponding singleton *statout* set, the $DCtr^i$s and the $DCco^i$s (i = 1, 2, ..., N) are empty. In such a case, the ust^i is the faulty transition with the output fault uso^i and the transfer fault to the state belonging to the only element of *statout*. No further diagnostic tests are required.

Case 3: The $ustset^i$s are empty and all of the $DCtr^i$s and the $DCco^i$s (i = 1, 2, ..., N) are empty, except one of the $DCtr^i$s ($DCco^i$s) which is a singleton with a corresponding singleton *EndStates* set (a corresponding singleton *outputs* set or a singleton *statout* set). In this case, the only transition of $DCtr^i$ ($DCco^i$) has a transfer fault to the state in *EndStates* (a faulty output corresponding the only element in *outputs* or both an output and a transfer faults corresponding to the only element in *statout*). No further tests are required.

Case 4: The $ustset^i$s are empty and one or more of the other sets has more than one element. Therefore, any element of the $DCtr^i$s or the $DCco^i$s (i = 1, 2, ..., N) 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 faults.

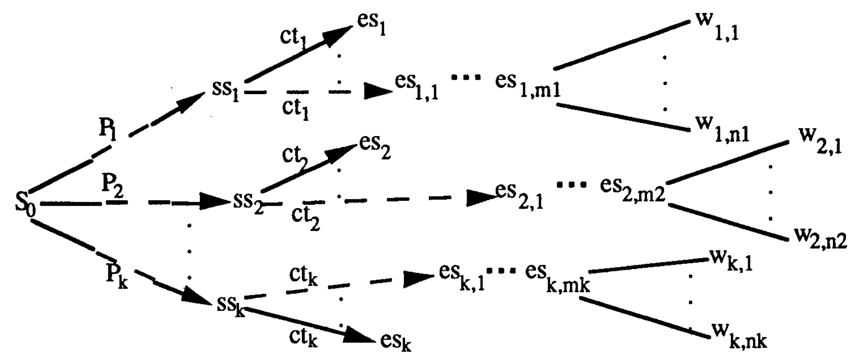
Algorithm for Case 4:

Step (a): For each transition T_k in the $DCtr^i$s, 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*(T_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*(T_k) and the correct state. It is different from the characterization set defined in [2], 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*(T_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*(T_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 $DCco^i$s, a similar approach to Step (a) is used. If the *statout* set is not empty, two groups of additional tests will be needed for each transition T_k in $DCco^i$. The first group concerns the ending state of T_k and can be found using the algorithm of Step (a), while the second group concerns the output of T_k and can be found using the present algorithm. If the ***outputs*** set is not empty, only one group of additional tests will be needed for each transition T_k in $DCco^i$. This group concerns the output of T_k and can also be found using the present algorithm. An additional test for finding the output of a transition T_k in $DCco^i$ 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 confirmed to do not have an output fault. If at the same time T_k is confirmed not having a transfer fault, it can then be removed from the corresponding $DCco^i$. 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 $DCtr^i$s or the $DCco^i$s (i = 1, 2, ..., N) 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 $DCtr^i$s. A similar picture would illustrate the progressive construction of additional tests for $DCco^i$s.

The construction of the additional tests is progressive because if the faults are located, the rest of these additional tests need not be generated, since we work under the single transition faults 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 the observed outputs identify the wrong transfer and/or the wrong output of the transition and the search is stopped.



Notes:
1) s_i's, ss_i's, es_i's and $es_{i,j}$'s are states in the machine.
2) ct_i's are candidate transitions
3) each P_i is a transfer sequence from the initial state to the starting state of ct_i
4) The input sequence in each path in the tree represents a possible additional diagnostic test for a specific candidate transition and a specific ending state.
5) Solid lines transitions indicate the predicted behavior of the diagnostic candidates (no fault case).
6) Each subtree starting with P_i represents additional tests for transition tc_i
7) The set of sequences $\{ w_{i,1} .. w_{i,ni}\}$ distinguishes between the states $es_{i,1} ... es_{i,mi}$ in the $EndStates_i$ for candidate transition ct_i

Figure 2 : Construction of additional diagnostic tests

tc. #	tc1	tc2
Input	R, a1, c'3, c1, t2, x3,	R, a1, c'2, d'2, c'3, x3, f1
Spec. transitions	**tr, t1, t"1, t6t'1, t'6t"4, t"5t7**	-, t1, t'1, t'4, t"1, t"5t4, t5t"1
Expected output	-, c'1, a3, a2, b3, d'1	-, c'1, a2, b2, a3, d'1, a3
Observed output	-, c'1, a3, a2, b3, **c'1**	-, c'1, a2, b2, a3, d'1, a3

Table 1: Test cases and their outputs

Case 5: One of the ustset^{i}s contains the ust^i transition and one or more of the other sets have more than one element. In such a case, we first check the ust^i transition by generating for it one (or several) additional test case(s) depending on the corresponding *statout* set is empty or not. If *statout* is empty and *outputs* contains one element, then only one additional test is needed. This test should not imply the execution of any transition in the sets DCtr^{i}s and DCco^{i}s (i = 1, 2, ..., N). It 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 4, otherwise, ust^i is the transition with an output fault and the search is stopped. If the set *statout* which corresponds to the ust^i transition, is not empty, then additional tests should first be selected for ust^i using the algorithm in Step (a) of Case 4. If these tests confirm the correctness of the ust^i transition, we select additional tests for the transitions in the other sets using the algorithms in Case 4.

4. An application example

Suppose that we are given the three CFSMs specification (implementation) shown in Figure 1. We execute the diagnostic algorithm presented in Section 3 with the following test suite:

TS = {R, a^1, c'^3, c^1, t^2, x^3; R, a^1, c'^2, d'^2, c'^3, x^3, f^1}

Step 1 and 2: The application of TS to the specification and the implementation (i.e. equals to the specification with the exception of transition t''_4 which has a transfer fault) of Figure 1, yields the expected and observed output sequences, as shown in Table 1.

In Table 1, a reset transition tr is used. It is assumed to be available for both the specification and the implementation. It resets all 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 case tc_1. Therefore, the symptom is: "$Symp_1 = (o^1{}_{1,6} \neq \hat{o}^1{}_{1,6})$" with the symptom transition t_7.

Step 4: Corresponding to the above symptom, we generate a conflict set for each machine in the system:

$Conf^1{}_1 = \{t_1, t_6, t_7\}$, $Conf^2{}_1 = \{t'_1, t'_6\}$, $Conf^3{}_1 = \{t''_1, t''_4, t''_5\}$

Step 5A: Since there is only one conflict set for each machine, no intersection is needed. The three initial sets of tentative candidates for the three machines are the following: $ITC^1 = \{t_1, t_6, t_7\}$, $ITC^2 = \{t'_1, t'_6\}$, $ITC^3 = \{t''_1, t''_4, t''_5\}$

Step 5B: For each ITC^i (i =1, 2), we generate its corresponding FTCtr^i, FTCco^i and the ustset^i sets:

$FTCtr^1 = \{t_1, t_6\}$, $ustset^1 = \{t_7\}$, $FTCco^1 = \{t_6\}$

$FTCtr^2 = \{t'_1\}$, $ustset^2 = \{\}$, $FTCco^2 = \{t'_6\}$

$FTCtr^2 = \{t''_1, t''_4\}$, $ustset^2 = \{\}$, $FTCco^2 = \{t''_5\}$

The processing of the above sets and the computation of the outputs and the ending state sets for the transitions in FTCtr^{i}s and FTCco^{i}s (i =1, 2, 3) leads to:

$ustset^1 = \{t_7\}$,

$EndStates[t_1] = \{\}$, $EndStates[t_6] = \{\}$ $outputs[t_6] = \{\}$

$ustset^2 = \{\}$,

$EndStates[t'_1] = \{\}$, $outputs[t'_6] = \{\}$,

$ustset^3 = \{\}$,

$EndStates[t''_1] = \{\}$, $EndStates[t''_4] = \{s_0\}$ $outputs[t''_5] = \{a\}$,

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

$DCtr^1 = \{\}$, $ustset^1 = \{t_7\}$, $DCco^1 = \{\}$

$DCtr^2 = \{\}$, $ustset^2 = \{\}$, $DCco^2 = \{\}$

$DCtr^3 = \{t''_4\}$, $ustset^3 = \{\}$, $DCco^3 = \{t''_5\}$

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 have the output fault of c' instead of d'.

Diag2: t''_4 might transfer to state s_0 instead of state s_1.

Diag3: t''_5 might have an output fault of a instead of b.

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 Diag1 . 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 M_1 to the starting state s_2 of t_7 is

"R, c^1". We concatenate this sequence with the input of t7. The execution of the resulting additional test "R, c^1, b^1"generates the output sequence "-, a^2, d'^1". This result confirms the correctness of t_7 in M_1 and the search for the faulty transition should be continued.

Let us consider now Diag2. A possible transfer sequence which will take the machine M_3 to the starting state s_1 of t''_4 is "R, c'^3". A possible sequence which will distinguish between states s_0 and s_1 is the input "v^3". After the application of the additional test case "R, c'^3, v^3, v^3", we observe "-, a^3, b^3, ε^3" as output. Such a result confirms that t''_4 is faulty and transfers to state s_0 instead of state s_1 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 this paper, we proposed a generalized diagnostic approach for deterministic systems represented by CFSMs. We showed that even for a single transition faults (output and/or transfer) hypothesis, a lot of work needs to be done for its diagnosis. The main advantage of the diagnostic approach is the need of shorter test suites for localizing detected faults. The optimization factor comes from the fact that only suspicious transitions requires additional tests, rather than every transition in the CFSMs, such as done in existing test selection methods with a strong diagnostic power (i.e., W or DS methods for single deterministic FSMs).

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 ports of 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 diagnostics 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 Quėbec and the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols.

References

[1] G.v. Bochmann, et al., "Fault models in testing", Invited paper in 4-th IWPTS, Leidschendam, Holland, 15 - 17 Oct. 1991.

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

[3] R. Davis, et al., "Model-based reasoning: Troubleshooting", in: Exploring Artificial Intelligence, edited by Shrobe, H. E., pp. 297-346, Morgan Kaufman, 1988.

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

[5] M.R. Genesereth, "The use of design descriptions in automated diagnosis", Artificial Intelligence 24(3), 1984, pp. 411-436.

[6] A. Ghedamsi et al., "Test result analysis and diagnostics for finite state machines", Proc. of the 12-th ICDCS, Yokohama, Japan, June 9-12, 1992, pp. 244-251.

[7] A. Ghedamsi et al., "Diagnostic tests for communicating finite state machines", to appear in the Proc. of the IPCCC-93, Scottsdale, USA, March 24-26, 1993.

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

[9] D. Harel, "Statecharts: a visual formalism for complex systems", Science of Computer Programming, 8, 1987.

[10] C.A.R. Hoare, "Communicating sequential processes", Comm. ACM 21, 1978, pp. 666-677.

[11] J. de Kleer, and B.C. Williams, "Diagnosing multiple faults", Artificial Intelligence 32(1), 1987, pp. 97-130.

[12] J. de Kleer, and B. C. Williams, Diagnosing with behavioral models, Proceedings IJCAI, Detroit-Michigan, !989, pp. 1324-1330.

[13] G. Luo, et al., "Test generation for concurrent programs modeled by communicating nondeterministic finite state machines", TR 823, DIRO, Univ. of Montreal, Montreal, Canada.

[14] S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition-Tours", Proc. of FTCS, pp.238-243, 1981.

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

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

[17] B. Sarikaya, and G.v. Bochmann, Synchronization and specification issues in protocol testing, IEEE Trans. on Comm., Vol. COM-32, No. 4, April 1984, pp. 389-395.

[18] H. Ural, "A Test Derivation Method for Protocol Conformance Testing", Proc. of the 7th IFIP Symposium on PSTV, Zurich, May 5-8 1987.

[19] S.T. Vuong and K.C. Ko, "A novel approach to protocol test sequence generation", IEEE Glocomm, San Diego, California, Dec. 2-5, 1990, vol. 3, 904.1.1 - 904.1.5.