Test Result Analysis and Diagnostics for Finite State Machines

A. Ghedamsi and G. v. Bochmann

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

Abstract

Systematic test sequence generation for communication protocols in conformance testing has been an active research area during the last decade. Methods were developed to produce optimized test sequences for detecting faults in an implementation under test (IUT). However, the application of these methods gives only limited information about the locations of detected faults. Therefore, we propose a complementary step which localizes the faulty transition in a deterministic finite state machine (FSM) once the fault has been detected. A diagnostic algorithm will generate, if necessary, additional diagnostic test cases which depend on the observed symptom and which permit the location of the detected fault. The algorithm guarantees the diagnostic of any single (output or transfer) fault in an FSM. An application example, explaining the functioning of the algorithm, is provided in the paper.

1. Introduction

A protocol implementation has to be checked as to whether it conforms to its protocol specification or not. This activity is called protocol conformance testing. A lot of research work has been directed towards such tests [16]. In particular, a protocol specification can be viewed as consisting of two portions: a control portion and a data portion, which may be addressed separately in the context of conformance testing. The testing of the data portion, which is concerned with checking the parameters of input and output primitives and local variables, has been investigated by a number of researchers using some forms of static data flow analysis [15, 21, 23].

In this paper, we are concerned with the control aspects. We assume that this aspect is modeled by finite state machines. Various test selection methods have been described for this aspect [2, 8, 11, 12, 14, 19]. These methods are intended to determine whether a given protocol implementation satisfies all properties required by the protocol specification. The purpose of a test selection method is to come up with a set of test cases, usually called "test suite", which has the following (conflicting) properties:

(a) The test suite should be relatively short, that is, the number of test cases should be small, and each test case should be fast and easily executable in relation with the implementation under test.

(b) The test suite should cover, as much as possible, all faults which any implementation may contain.

Existing test selection methods differ in the kind of compromise which is reached between these two conflicting objectives, and in the amount of formalism which is used to define them.

The remainder of the paper is organized as follows. In Section 2, we introduce the diagnostic problem, in general, and discuss the main steps required by any diagnostic process. In Section 3, the test selection methods are studied in respect to their capability of solving diagnostic and fault localization problems, which have not been addressed by the above references. Test suites, generated by one of these methods, are used by our diagnostic algorithm which is presented and discussed in Section 4. An application example is introduced in Section 5. Finally in Section 6, concluding remarks and future research points are included.

2. The diagnostic problem

In distributed systems and the communication protocol area, very little work has been done for diagnostic and fault localization problems [22]. At the same time, diagnostic is a well documented subject in other areas such as Artificial Intelligence (AI) [3, 5], complex mechanical systems and medicine [17]. Therefore, most of the concepts and terms used in this paper are imported from those domains.

2.1 Methodology for test result analysis and diagnostics

In general, diagnostics can be classified into two classes. The first class is called "Experimental diagnostic". It is mainly used in medicine [17] and similar domains. Experimental diagnostic is not covered in our paper. Our main interest is related to the second class of diagnostic called "Diagnostic based on a model" [9]. The main idea of this type of diagnostic is that it is necessary to know how the system or the machine under test is supposed to work correctly in order to be able to know why it is not working correctly. Different reasoning systems based on models were developed for this approach. The most important ones

0-8186-2865-0 /92 $3.00

are the following: HT (Hardware Troubleshooting) [3], DART (Diagnosis Assistance Reference Tool) [5], and GDE (General Diagnostic Engine) [9, 20]. A brief survey on these systems can be found in [7].

Often the specification of a model-based system is structured in a hierarchical manner. The system is seen as a set of components connected to each other in a specific way. The **structure** (organization) of a system can be defined as a relationship (i.e, physical connection, procedure call,...) between the different components of the system. A **component** is seen as one of many smaller sub-systems in a larger system. The behavior of the larger system is, therefore, described in terms of its components behaviors. Each component can itself described at a more detailed level, possibly in terms of sub-components, and so on. Depending on the need, we might have several different descriptions of the same component, one in terms of the composition of sub-components, and another which describes the behavior of the component in terms of its interactions with the environment.

In structured model-based diagnostics, 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 [9, 13]. Both systems and their corresponding models are assumed to have the same components and the same structure. **Observations** of inputs and outputs show how the system is behaving, while **expectations** 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. The process of comparing observations and expectations is called **"test result analysis"**. 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 [9].

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. A good candidate generator should be complete, non redundant and optimal. It is complete if it generates all candidates which explain all identified symptoms. It is non redundant when it does not generate the same candidate more than once. Finally, it is optimal if it generates only minimal candidates and no super-sets of them.

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

2.2 Fault models

We recall that in general, the diagnostic process is a very complicated task, specially for diagnosing complicated 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 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 [9]. 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, stack at 0/1 fault models) [20]. These fault models may be used during the phase of discrimination between candidates. In the protocol area and more precisely for FSMs, another simple fault model, based on transfer and output faults of state transitions, can be used for diagnosing protocol entities modelled by FSMs [2, 7, 22].

2.3 A general diagnostic method

In the following method, we only consider the diagnostic of structured systems which are assumed to be composition of different components. These components will be tested for their correctness during the diagnostic process. We assume that the behavior of these systems is described in terms of inputs and outputs. We also assume that the application of a sequence of inputs to the system will imply the involvement of a specific set of components in the system and the generation of a specific sequence of outputs.

The diagnostic method

Given a structured system to be diagnosed and its model, the purpose of the following method is to present the main steps required by any diagnostic process. Therefore, we do not include any specific techniques at this stage. A detailed and a more specific algorithm for diagnosing systems modelled by deterministic FSMs is discussed in Section 4.

Step 1: Generation of expected outputs

We assume that a test suite "TS" is given; it may have been obtained by one of the existing test selection methods. 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 $i_{i,1}i_{i,2}....i_{i,mi}$, the corresponding sequence of expected outputs is written as $o_i = o_{i,1}o_{i,2}....o_{i,mi}$, where $o_{i,j}$ is expected after input $i_{i,j}$.

Step 2: Generation of observed outputs

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

Step 3: Generation of symptoms

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

Step 4: Generation of conflict sets

For each symptom $(o_{i,j} \neq \hat{o}_{i,j})$, determine its corresponding conflict set. A **conflict set** for a given symptom is defined to be the set of components which are supposed to be involved in the generation

of the symptom output; therefore, at least one of these components must be faulty.

Step 5: Generation of diagnostic candidates

Diagnostic candidates are components which are suspected to be faulty. Therefore, each one of them should have a non empty intersection with each conflict set. It also has to be consistent with all observations. A diagnostic candidate with more than one component corresponds to multiple faults in the IUT.

Step 6: Additional diagnostic tests

In this step, additional diagnostic tests or different points of observation might be needed in order to reduce the number of diagnostic candidates, if possible to a single diagnosis. In such a case, specific techniques might be applied in order to achieve such a goal (for more details, see for instance Section 4).

3. The finite state machine model

A deterministic FSM M can be represented by a quintuple (S, I, Y, T, O) where :
S : Set of states of M. It includes an initial state s_0
I : Set of input symbols
Y : Set of output symbols. It includes the null output (-),
T : Next-state function, S x I --> S,
O : Output function, S x I --> Y.

A graphic representation of a deterministic FSM, called **"State transition diagram"**, is given in Figure 1.

3.1 Principles of an FSM fault model

The FSM fault model is based on errors and faults made on labeled transitions or states of the machine. These definitions are also essential for the FSM-based test selection methods discussed in the following sections.

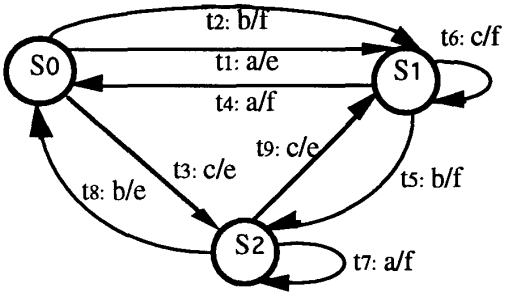

Figure 1: A state transition diagram of an FSM

Definition 1: Output fault: We say that a transition has an output fault if, for the corresponding state and received input, the implementation provides an output different from the one specified by the output function.

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

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

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

3.2 Test selection methods for FSMs

Many test selection methods have been developed for FSMs. The most important ones are the following:

(a) The transition tour method called "T-method" [12] detects any set of output faults in the absence of transfer faults.

(b) The Distinguishing Sequence method called "DS-method" [8] and the UIOv-method [23] detect any set of output and transfer faults, assuming that the number of states of the implementation is the same as for the specification.

(c) The W [2] and the Wp [4] methods detect in addition transfer faults with additional states, if the number of additional states is limited. However, the test suite length (and cost) increases in general exponentially when the number of expected additional states is increases.

Various test suite optimization techniques have been described based on UIO sequences [18] which try to reduce the cost of the test suite by keeping its fault coverage invariant. The invariance of the fault coverage is not shown in all these cases. However, the fault diagnostic process might become more complicated. This is might be caused by the possible loss of information about some parts of the

machine once its corresponding small test cases, included in larger ones, get eliminated by the optimization process.

3.3 Fault diagnosis for FSMs

In this section, we first give a brief overview of a recently proposed approach for diagnosing FSMs, and then a discussion on the diagnostic power of the above test selection methods.

3.3.1 A general diagnostic method for FSMs: Vuong presented a diagnostic method [22], which is based on the test sequence generation "Constraint Satisfaction Problem (CSP)" approach [10]. In such a method, it is assumed that an observed input/output sequence for the implementation to be diagnosed, is given. That sequence can be used as the initial sequence for the CSP method. Different FSMs might result from the resolution of the CSP. Each solution represents a possible FSM model for the given IUT. If none of these FSMs is equivalent to the FSM specification, the observed sequence indicates that the given IUT failed the test and that some of its transitions are faulty. To distinguish the real implementation from the set of solutions, additional tests are required.

The described method has some major disadvantages concerning the high complexity of resolving the CSP problem which is known in general, to be NP-complete. The other problem is the growing space of diagnostic candidates which might be generated by the CSP approach. The number of solutions could grow exponentially with the number of states in the given FSM and hence the number of additional tests will also grow fast.

3.3.2 Diagnostic power of test selection methods: Following the discussion of Section 3.2 on the different test selection methods and their fault detection power, the question comes to mind: what is the diagnostic and fault isolation power of these methods?

Concerning the W- and DS- methods in the case of single faults, both methods have the full power of diagnosing and localizing the fault. If there is a transfer fault, the test sequences are able to tell to which state the faulty transition has transferred. For the case of the UIO- and the Wp- methods, however, the generated test sequences do not guarantee the localization of transfer faults because no conclusion about the reached state can be made. Therefore, a diagnostic process is in general needed if test suites are generated by the UIO-, Wp-, or T- methods. In such a case, distinguishing diagnostic tests may be required in order to reduce the number of the generated diagnostic candidates. We note that a test sequence with better fault coverage (i.e. an UIO test suite rather than a T test suite) might need less additional diagnostic tests for the diagnostic process.

4. Diagnostics for sequential machines

In this section, we adapt the general diagnostic method (in Section 2.3) to the specific case where implementations and their models are represented by deterministic FSMs. In such a context, transitions (which may be faulty) of an FSM to be diagnosed, can be seen as the components of the general structured system described in Section 2.1.

The adapted version of the diagnostic method consists of diagnosing (with respect to its specification FSM) an IUT FSM for possible faulty transitions. We assume the following fault model: " the IUT may have an output or transfer fault in at most one of its transitions". Its purpose is the identification of the faulty transition and the type of its fault (i.e. output or transfer). This work is mainly executed within the Step 5 and the Step 6 of the algorithm. In particular, Step 5 might end up with different diagnostic candidates. In such a case and in Step 6, additional diagnostic tests should be selected in order to be able to isolate the faulty transition and more precisely to know to which state (in case of transfer fault) that transition has transferred.

Definition: The transition $T_{i,j}$ of the specification where the symptom $(o_{i,j} \neq \hat{o}_{i,j})$ 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)**.

The diagnostic algorithm

Given a FSM specification and the FSM implementation (to be diagnosed), the following algorithm develops the general steps of diagnostic method of Section 2.3. It also includes some techniques for the computations needed in each step.

Step 1 to 3: (generation of expected outputs, generation of observed outputs and generation of symptoms): These steps are as described in Section 2.3.

Step 4: Generation of conflict sets

For each symptom, a corresponding conflict set is formed which consists of all transitions executed in the FSM specification when the corresponding test case is applied. No transitions, executed after the observation of the symptom under consideration, will be included in the conflict set.

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

Step 5A: Generation of the initial tentative candidate set

The initial tentative candidate set **"ITC"** will be formed by the intersection of all conflict sets. Each element T_k in ITC represents a tentative candidate transition (with an output or transfer fault) which may explain all symptoms.

Step 5B: The FTC set and the ending state set

If there is a unique symptom transition "**ust**", it will be contained in the ITC. In that case, we split the ITC set into the set "**ustset**" which will initially contain the ust and the final tentative candidates set "**FTC**" which will contain the rest of transitions in ITC. Otherwise, the full ITC set forms the FTC set. If applicable, we first process the ust as explained below. A separate processing will be done for each transition in the FTC set.

The ust is processed as follows. All test cases in the initially given test suite "TS" are scanned for transitions that are equal to the ust. If for all found transitions their corresponding observed outputs is equal to the uso, which means the ust explains all observations, then the ust is considered a diagnostic candidate.

```
Procedure ust-processing (ust)
  Forall tc_m ∈ TS DO
    Forall I_m,n ∈ tc_m DO
      IF (T_m,n = ust) THEN
        IF (ô_m,n <> uso) THEN
            ustset = ∅; exit        {the ust is not a
                                     diagnostic candidate}
        ELSE IF o_m,n+l <> ô_m,n+l THEN
                {l =1, 2, ..., i_m where n +i_m is the
                        length of the test case tc_m}
             ustset = ∅; exit
    ENDForall
  ENDForall
```

For each transition T_k in FTC, we compute the set of all states called "**$EndStates_k$**", to which the transition 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 (FTC);
  Forall T_k in FTC Do {T_k is the k-th transition in FTC}
    EndStates_k := ∅        {EndStates_k is the set of all
        states to which the transition T_k might transfer}
    Forall state s ∈ S and s ≠ NextState(T_k) Do
      flag := true
      Forall tc_m ∈ TS Do
        Forall I_m,n ∈ tc_m Do
          IF (T_m,n = T_k ) THEN
            IF (O(s, I_m,n+l) <> ô_m,n+l) THEN
                {l =1, 2, ..., i_m where n +i_m is the
                        length of the test case tc_m}
              flag := false; exit
        ENDForall
      ENDForall
      IF (flag = true) THEN
        EndStates_k := EndStates_k ∪ {s}
    ENDForall
  ENDForall
```

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

In this step we remove all correct (i.e. transitions with empty ending state sets) transitions from the final tentative candidate set FTC. All transitions in the resulting "**DCtr**" set (if not empty) are diagnostic candidates with transfer faults. For each transition T_k in the DCtr and for each state s_{ik} in the $EndStates_k$, a diagnostic, stating that T_k might transfer to state s_{ik}, is generated. An extra diagnostic, stating that the ust might have an output fault, is also generated, if the ustset is not empty.

Step 6: Additional diagnostic tests

Depending on the result of the previous step, different possibilities might be present.

Case 1: The ustset contains the ust transition and DCtr is empty. In such a case, the ust is the faulty transition with an output fault and no further diagnostic tests are required.

Case 2: The ustset is empty and the DCtr is a singleton with a corresponding singleton ending state set. In such a case, the transition in DCtr has a transfer fault to the state in the ending state set. No further tests are required.

Case 3: The ustset is empty and the DCtr is a singleton with a corresponding ending state set with more than one element, or the DCtr has more than one element. Therefore, each element in DCtr might be the faulty transition with a transfer fault. In such a case, we should process the elements of DCtr to derive further tests with the purpose of identifying the faulty transition and the state to which it transfers.

We propose the following approach and algorithm for the derivation of additional diagnostic tests. For each transition in the DCtr, the following step is executed:

Given the transition under consideration T_k in the DCtr, additional test cases have to be selected and executed, in order to be able to know exactly to which state T_k 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, an "**limited characterization set**" W_k [2] has to be computed for the states in $EndStates_k$ and the correct state. 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 was applied to 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.

In order to avoid any ambiguities, the transfer sequence and the limited characterization set should be chosen in such a manner that they do not involve any transition in DCtr.

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 DCtr. When a faulty transition is found, the observed outputs identify the wrong transfer of the transition.

Case 4: The ustset contains the ust transition and DCtr is not empty. In such a case, we first check the ust transition by generating for it additional test cases in a similar way as in Case 3. If the application of these test cases generates the expected output, then ust is declared correct (no output fault) and the search for a faulty transition with a transfer fault has to be done as in Case 3.

The above cases are covered by the following algorithm:

```
IF (ustset = {ust} AND DCtr = ∅)
THEN Print "The ust transition has an output fault"
ELSE IF ustset = ∅ AND DCtr = {T1 } AND
   EndStates1 = {s1}
   THEN Print "T1 is the faulty transition with transfer
                fault to s1"
   ELSE IF ustset = ∅  AND DCtr = {T1 , ..., Td }
      THEN Findtransferfault (DCtr)
      ELSE IF ustset = {ust} AND DCtr = {T1 , ..., Td }
         THEN select test cases for the ust;
         apply these tests to the IUT;
         IF (observed output <> expected output)
            THEN Print "The ust has an output fault and
            all other transitions are correct"
            ELSE Findtransferfault (DCtr)
```

```
Procedure Findtransferfault (DCtr)
   flag := false; k := 1;
   REPEAT                          {Tk is a transition in DCtr}
      select diagnostic tests for Tk ;
      apply these tests to the implementation;
      IF (observed output <> expected output)
      THEN flag := true;
      Print "Tk has a transfer fault. Its ending state is
         deduced from the analysis of the observed outputs.
         All other transitions are correct"
      ELSE Print "Tk is correct"
      k := k + 1
   UNTIL (flag = true)
```

5. An application example

Given the FSM specification of Figure 1, we execute the steps of the proposed diagnostic algorithm as follows:

Step 1: Generation of expected outputs

Suppose that the initial test suite is given as follows:

TS = {rab, rbca, rcab, rcca}

The expected outputs for these tests are: {-ef, -fff, -efe, -eef}

Step 2: Generation of observed outputs

The application of the given TS to the IUT of Figure 2 generates the observed output sequences included in the following table.

tc. #	tc_1	tc_2	tc_3	tc_4
Input	rab	rbca	**rcab**	rcca
Transition	t_1t_5	$t_2t_6t_4$	$t_3t_7t_8$	$t_3t_9t_4$
Exp. output	-ef	-fff	**-efe**	-eef
Obs. output	-ef	-fff	**-eff**	-eef

Table 1: Test cases and their outputs

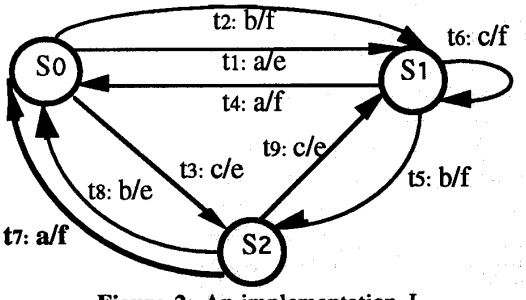

Figure 2: An implementation I

Step 3: Generation of symptoms

A difference between observed and expected outputs is detected during the application of test case tc_3. Therefore, we have a single symptom "$Symp_3 = (o_{3,3} \neq \hat{o}_{3,3})$" with the symptom transition t_8.

Step 4: Generation of conflict sets

The conflict set for the given symptom is "$Conf_3$ = {t_3, t_7, t_8}", namely, the transitions of test case tc_3 that lead to the symptom.

Step 5A: Generation of the initial tentative candidates set

Because there is only one conflict set, the resulting initial tentative candidate set ITC is equal to the conflict set $Conf_3$. Each transition in ITC is a tentative candidate.

Step 5B: The FTC set and the ending state sets

The unique symptom transition t_8 is included in ITC. Therefore, ITC is split into ustset and the final tentative candidate set FTC as follows:

ustset = $\{t_8\}$ FTC := $\{t_3, t_7\}$

The computation of the ending state sets for the transitions in FTC leads to:

EndStates[t_3] = { } EndStates[t_7] = $\{s_0, s_1\}$

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

The transitions with empty ending state sets are correct, therefore they are removed from the final tentative candidate set FTC. The resulting set DCtr is the one containing the diagnostic candidates with transfer faults. We obtain:

ustset = $\{t_8\}$ DCtr = $\{t_7\}$.

With the use of the ustset and the ending state sets generated for transitions in DCtr by Step 5B, the following diagnostics are generated:

D1: t_7 might transfer to state s_0 instead of s_2.

D2: t_7 might transfer to state s_1 instead of s_2.

D3: t_8 might have an output fault of f instead of e.

Step 6: Additional diagnostic tests

We determine the faulty transition by completing the initial test suite with additional diagnostic tests. In order to discriminate between the two diagnostic candidates t_7 and t_8, additional diagnostic tests have to be selected.

At this stage, we know for sure that all transitions, with the exception of t_7 and t_8, are correct. In order to check whether t_8 has an output fault or not, we just have to use a path which takes the machine to the starting state of t_8, then we execute it. For this example, a possible additional test case is "rcb". The execution of this test case generates "-ee" as output. This result confirms that t_8 is not faulty and therefore, t_7 has a transfer fault.

Consequently, another diagnostic test is required in order to distinguish between the two diagnoses of t_7 (D1 and D2). We don't know whether the faulty transition t_7 transfers to state s_0 or to state s_1. A possible transfer sequence which will take the machine to the starting state s_2 of t_7 is "rc". A possible sequence which will distinguish between states s_0 and s_1 is the input "a". If in s_0, the machine produces "e" as output for the input "a", and "f" for the same input if it was in state s_1. Hence, the resulting additional test case is "rcaa" where the first "a" input represents the input of the transition t_7. If after the application of "rcaa" the observed output is "-efe", t_7 transfers to state s_0 as shown in Figure 4, otherwise, it would transfer to state s_1, as assumed by the second diagnosis.

6. Concluding discussion

6.1 The diagnostic algorithm and test selection methods

It is important to note that the proposed algorithm depends closely on its first step, where different test selection methods might be used for generating the initial test suite. In the following, we discuss this dependance in more details.

For a poor test selection method, such as the T method, it is not even guaranteed that a fault is detected. If the fault is not detected, no diagnostics will be generated and Steps 4 to 6 of the algorithm will simply not apply . However, if a symptom is identified, the diagnostic process may generate a larger number of diagnostic candidates and therefore correspondingly, an even larger number of diagnostics, because the initially applied tests were not very thorough. In order to distinguish between these diagnostics, a large number of additional diagnostic tests will be needed. We conclude that the less complete an initial test suite is used, the more complex is the task of diagnosing the location of a detected fault.

For a better initial test suite, such as generated by the UIOv- or Wp- methods, faults are guaranteed to be detected, but they are not necessarily localized. In such a case, the diagnostic algorithm might end up with several diagnostics at the end of Step 5 and additional tests might be generated by Step 6. It is interesting to use the UIOv- or the Wp-methods since their corresponding test suites are shorter than those generated by the W-method. It is important to mention that, even after complementing the test suites generated by the UIOv- or the Wp- methods with the additional diagnostic tests, the resulting test suites will, in general, remain shorter than those generated by the W-method.

In the case of the W- or DS- methods finally, the computation required by the steps of the algorithm will be simplified considerably. The generated test suites can detect and localize any single (output or transfer) fault in the implementation. At the end of Step 5 of the algorithm, only a single diagnostic candidate will remain and hence there will be no need for additional diagnostic tests.

6.2 Complexity of the Algorithm

A detailed study about the complexity of the proposed diagnostic algorithm is given in [6]; in this subsection, we only give the result of this study. The overall complexity of the algorithm is found to be of $O(nL_sL_c^3)$ where n, L_S, L_C are variables representing the number of states in the specification, the number of test cases in the initially given

test suite, and the number of inputs in the longest test case, respectively.

It is important to note that the two variables L_S and L_C are interrelated, if one of them is reduced the other one will be increased. Hence, it is possible to reduce the execution time of the proposed algorithm by reducing as much as possible L_C, since in general, L_S will only register minor increases. For example, suppose we use the transition tour method to select an initial test suite consisting of a single test case of length 50 (i.e. $L_S = 1$, $L_C = 50$). It may be possible to reduce dramatically the execution time of the diagnostic algorithm by just selecting several small test cases corresponding to sub-tours of the whole sequence, for instance 15 sub-tours not longer than 5 (i.e. $L_S = 15$, $L_C = 5$). From the practical point of view and as indicated in [14], for most of the known protocols $L_C \leq 5$.

6.3 Presence of multiple faults

As explained above, the proposed algorithm is designed for diagnosing single faults. In general, the initial test suites which might be used by the algorithm, will not necessarily guarantee the detection of all faults and hence the algorithm will not ensure their diagnosis. In the presence of multiple faults, and if detected, the diagnostic algorithm needs to be modified in order to accommodate the change in the fault model assumption. In particular in Step 5, the generation of the diagnostic candidates will be more complicated, since they might be formed by more than one transition. As a consequence, Step 6 should also be modified. It is not obvious how to select additional tests which will have the ability of distinguishing between sets of transitions and their corresponding diagnostics.

Acknowledgments: The authors would like to thank F. Khendek for discussions on test methods discussed 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

[1] 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.

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

[3] 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.

[4] 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.

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

[6] A. Ghedamsi and G.v. Bochmann, "Diagnostic Tests for Finite State Machines", TR No. 807, Univ de Montréal, Montréal, January 1992.

[7] A. Ghedamsi, "Test selection and diagnostic methods", TR, Univ de Montréal, Montréal, February 1991.

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

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

[10] A.K. Mackworth, "Consistency in networks of relations", Artificial Intelligence 8(1), 1977, pp. 99-118.

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

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

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

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

[15] B. Sarikaya, "An Estelle based test generation tool", Univ. de Montreal, DIRO, Montreal.

[16] B. Sarikaya, "Conformance Testing: Architecture and Test Sequences", Computer Networks and ISDN Systems 17, pp. 111-126, 1989.

[17] E.H. Shortlife, "Computer-based Medical Consultations : MYCIN", Elsevier, New-York, 1976.

[18] Y.N. Shen, F. Lambardi and A.T. Dahbura, "Protocol conformance testing using multiples UIO sequences" in E. Brinksma, G. Scollo, C.A. Vissers (eds), PTSV IX, Amsterdam 1989.

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

[20] P. Struss, and O. Dressler, "Physical Negation - Integrating Fault Models into the General Diagnostic Engine", Proceedings IJCAI, Detroit - Michigan, 1989, pp. 1318-1323.

[21] 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.

[22] 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. no. 3, 904.1.1 - 904.1.5.

[23] S. T. Vuong, W. W. L. Chan and M. R. Ito, "The UIOv-Method for protocol test sequence generation", in the 2-nd International Workshop on Protocol Test Systems, Berlin, Germany, Oct. 3-6, 1989.