Locating a Faulty Machine in a System of Communicating Finite State Machines*

Khaled El-Fakih and Gregor v. Bochmann

School of Information Technology and Engineering, University of Ottawa, Canada
{kelfakih, bochmann} @site.uottawa.ca

Abstract

In this paper, we propose a diagnostic method for the case where the system specification (implementation) is given in the form of two communicating finite state machines (CFSMs). The method decides if it is possible to locate the faulty machine in the system, once a fault has been detected in its corresponding implementation. If this is possible, it also locates the faulty machine. Two simple examples are used to demonstrate the different steps of the method.

1. Introduction

Testing is an important step in the development cycle of both software and hardware systems. In the software domain, where a system is represented by an FSM model, a lot of research work has been directed for such tests (for surveys of test derivation see [1] and [2]).

We consider a system architecture consisting of two FSM machines called components, as shown as "System Under Test" in Figure 1. The system contains a machine, called "Context machine" which communicates with the environment and the other machine, called "Embedded machine". The interactions between the two components (the embedded machine and the context) is assumed to be hidden or unobservable.

In this paper, we present an important complementary step to testing the given system once a fault has been detected in its implementation. The method consists of a new diagnostic method that decides if it is possible to locate the faulty component in the given system. If this is possible, the faulty machine is found.

The method starts by identifying the difference between how the system should behave (expectations), and how it is actually behaving (observations). The difference between these expectations and observations are called symptoms. In order to explain these symptoms, diagnostic candidates for both components are generated. A diagnostic candidate is defined to be the minimal difference between the specification of a given machine and its corresponding implementation, capable of explaining all symptoms. If there are no candidates generated for one of the components then the other component is declared faulty. Otherwise, the method continues to locate the faulty machine.

This paper is organized as follows. In section 2, a system of two CFSMs is presented. In Section 3, a fault model for the given systems is defined. In Section 4, the diagnostic method is described in details followed by two simple examples that demonstrate its different steps in Section 5. Section 6 concludes the paper.

2. A System of Two CFSMs

Complex systems are often specified as a collection of communicating FSMs. A system of two communicating FSMSs, called embedded machine (M2) and context machine (M1), is shown in the upper part of Figure 1. The alphabet X and Y represent the externally observable input/output actions of the system, while the U and Z alphabets represent the internal (hidden) input/output interactions between the two components. As in [3], we assume that the sets of actions X, Y, U, and Z are pair-wise disjoint.

* This work was funded by Communications and Information Technology Ontario

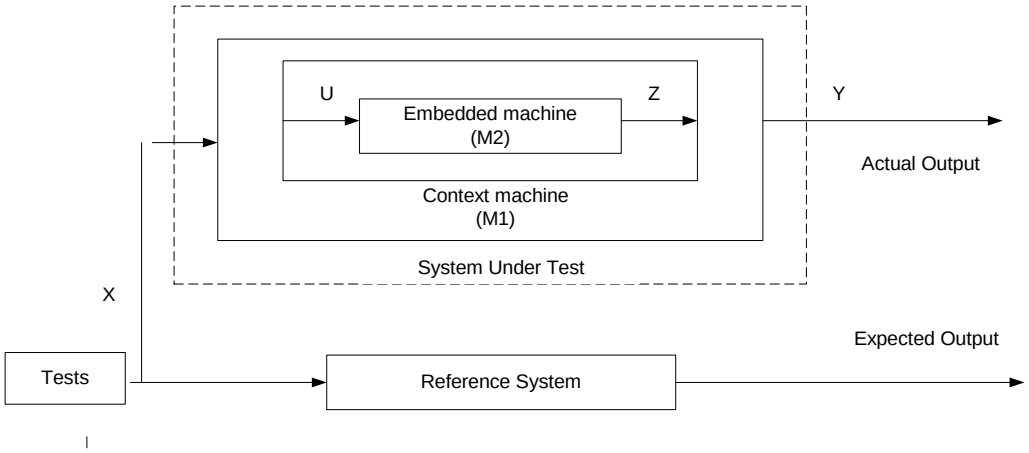

Figure1. A system of two CFSMs and a their Tester

For the rest of the paper, we assume that the two (deterministic) FSMs , M1 and M2, of the given system communicate asynchronously via bounded input queues where the internal actions of U and Z are stored.

A deterministic FSM M_i (i = 1, 2) in such a system of 2 CFSMs can be represented by a quintuple (S_i, I_i, O_i, $NextStaFunc_i$, $OutFunc_i$) where :

S_i is the set of states of M_i. It includes the initial state s_{i0},
I_i is the set of input symbols. It includes the reset input (r),
O_i is the set of output symbols. It includes the null output (-),
The next-state function is $NextStaFunc_i$: $S_i \times I_i \rightarrow S_i$,
The output function is $OutFunc_i : S_i \times I_i \rightarrow Y_i$.

We also assume that the system at hand has never more than one message in transit, i.e. a next external input is submitted to the system only after it has produced an external output y to the previous input. Under these assumptions, the joint behavior of M1 and M2 can be described by means of a composed machine, called Reference System, RS=M1 ∘ M2. RS describes the joint behavior of M1 and M2 in terms of external inputs x and external outputs y. Consider for example the two machine M1 and M2 as shown in Figure 2, and their corresponding Reference System RS = M1 ∘ M2 as shown in Figure 3. The set of external inputs is $X=\{x_1, x_2, x_3\}$, the set of external outputs is $Y = \{y_1, y_2, y_3\}$, the set of internal inputs is $U =\{u_1, u_2, u_3\}$, and the set of internal outputs is $Z = \{ z_1, z_2, z_3 \}$.

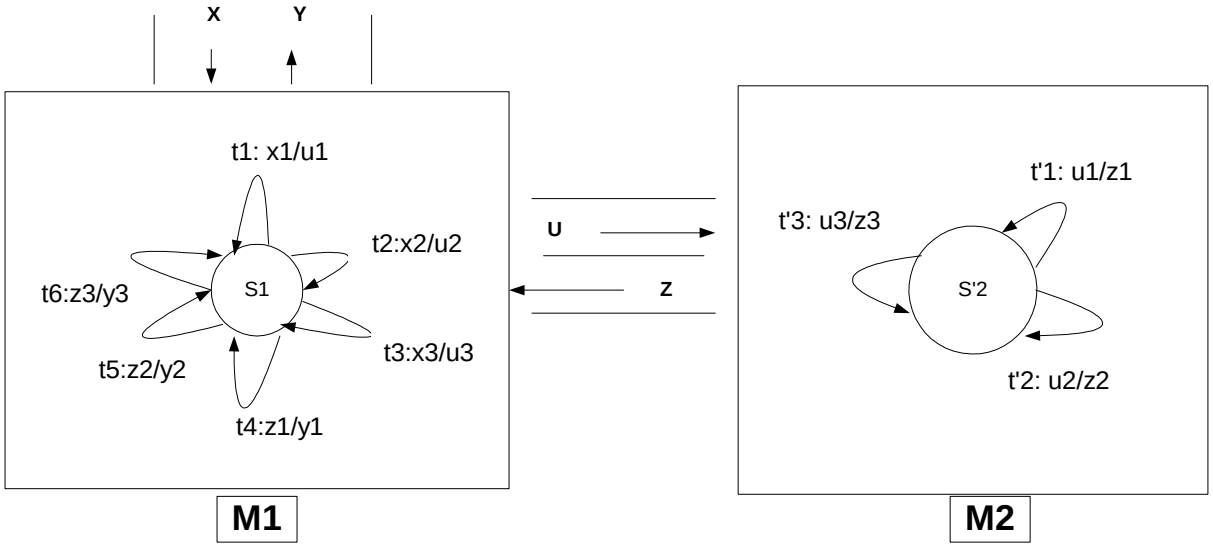

Figure 2. A system of two CFSMs, M1 and M2.

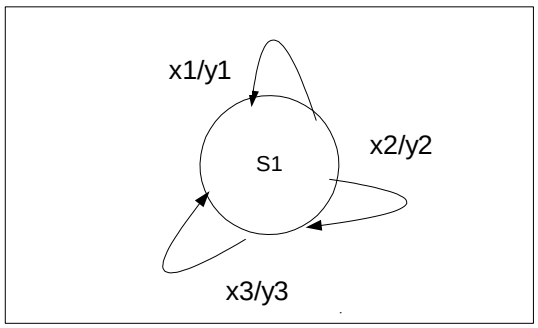

Figure 3. Reference System of the M1 and M2 of

The tester, lower part of Figure 1, implements a given test by executing external input sequences (test cases) simultaneously against both the system under test (SUT) consisting of the implementation of M1 and M2, and the reference system in order to generate the observed and expected outputs.

3. Fault Model for the System of CFSMs

We assume a fault model which is based on faults made on labeled transitions [4]. Here, we assume that one and only of the two machines of the given system has a single output fault. We say a transition has an output fault if, for the corresponding state and received input, the implementation of the component 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.

4. A Method for Locating the Faulty Machine in the Given System

We assume that a test suite (TS) is given and that at least one erroneous output is detected. We apply the TS to the SUT and the reference system. For each test case tc_i of TS, the expected output sequence is written as $o_i = o_{i,1},o_{i,2},...,o_{i,mi}$, where output $o_{i,j}$ is expected after input $i_{i,j}$, while the observed output sequence is written as: $\hat{o}_i = \hat{o}_{i,1},\hat{o}_{i,2},...,\hat{o}_{i,mi}$. We compare the observed outputs with the expected outputs and identify all symptoms. Any difference ($o_{i,j} \neq \hat{o}_{i,j}$) represents a symptom.

For each symptom ($o_{i,j} \neq \hat{o}_{i,j}$), and for each machine Mi in the system (i=1,2), we do the following:

(i) To determine a corresponding conflict paths. A conflict path for a given symptom is the sequence of transitions that are supposed to be executed by the machine, according to the specification of the two components, for the generation of the symptom output; therefore, at least one of these transitions must be faulty if the given machine contains the fault. For example, the conflict path for machine M_1 is formed by all transitions executed by M_1 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 path.

(ii) To determine the transitions which are suspected to be faulty (called tentative candidate transitions), we form the intersection of the transitions of all conflict paths of (i). For each tentative candidate transition Tk of machine Mi, we form its corresponding tentative diagnostic candidates. These are candidates that may succeed to explain the observable behavior of the given SUT. Each candidate is formed by computing and assigning to Tk a possible output fault and by leaving all remaining transitions of Mi unchanged. This process is repeated for all faulty outputs of Tk (all outputs except the specified output).

(iii) Afterwards, we eliminate from the tentative diagnostic candidates all candidates that do not succeed to explain all observations of the SUT. A particular tentative diagnostic candidate fails to explain all observations, if its expected outputs are not equal to the observed outputs of the SUT for at least one test case of TS. All remaining candidates are considered as diagnostic candidates.

Let NumCandM1 be the number of diagnostic candidates of M1 and $DC_{M1,k}$ (k = 1… NumCandM1) be the diagnostic candidates of M1. Let NumCandM2 be the number of diagnostic candidates of M2 and $DC_{M2,k}$ (k = 1… NumCandM2) be the diagnostic candidates of M2.

If NumCandM1 = 0, then M2 is the erroneous machine, and if NumCandM2 = 0 then M1 is erroneous, else we proceed as follows:

a- From : $DC_{M1,k} \circ M2$ for k =1… NumCandM1
b- From : $DC_{M2,k} \circ M1$ for k =1… NumCandM2

Two FSMs are said to be equivalent if and only if for all possible input sequences, they produce the same output sequences. If any of the machines computed in (a) is equivalent to any machine computed in (b), then the faulty machine (M1 or M2) can not be

identified. This is due to the fact that there is at least one possible fault in M1 and a possible fault in M2 such that the behavior of the composed system for both of these possible faults is the same. However, if none of the machines computed in (a) is equivalent to any machine computed in (b), we can locate the faulty machine as follows:

We generate additional tests for distinguishing between the diagnostic candidates using the test development approach described by Gill [4]. This method determines a test sequence which allows the distinction between any two given finite state machines. In our context, each diagnosis, $DC_{Mi,k}$ ∘ Mj (for i, j = 1,2, j # i, k ≤ NumCandMi) corresponds to a particular (faulty) implementation of Mi determined by its k-th fault predicted by $DC_{Mi,k}$ and the assumed non-faulty implementation of Mj.

Given a set of n diagnoses, Gill's method may be applied to distinguish between any two selected diagnoses, say $D^{(1)}$ and $D^{(2)}$. The application of the derived test sequence to the implementation will lead to one of the following situations:

(1) The observed output is equal to the one expected for $D^{(1)}$.

(2) The observed output is equal to the one expected for $D^{(2)}$.

(3) The observed output is different from both of the outputs expected for $D^{(1)}$ and $D^{(2)}$.

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

5. Two Application Examples

In the following two subsections, two examples are given to demonstrate the different steps of the diagnostic method described in Section 4. In these examples, a reset transition tr is assumed to be available for both the specification and the implementation. We use the symbol "r" to denote the input for such a transition and the null symbol "-" to denote its output. A reset input r resets both machines in the system to their initial states.

5.1. A Simple Example

Suppose that the test suite TS = {r-x1, r-x2, r-x3 } is given for the two CFSMs specification shown in Figure 2 .

The application of TS to the specification of Figure 2 and its corresponding implementation of M1 and M2 (which equal to the specification with the exception that t1 of M1 has the output fault u2) yields the expected and observed output sequences depicted in Table 1.

Tc#	tc_1	tc_2	tc_3
Inputs	r, x_1	r, x_2	r, x_3
Specified transitions	t_1, t'_1, t_4	t_2, t'_2, t_5	t_3, t'_3, t_6
Expected Output	y_1	y_2	y_3
Observed Output	y_2	y_2	y_3

Table1. Test cases and their outputs

A difference between observed and expected outputs is detected for test cases tc_1. Therefore, the symptom is: $Symp_1 = (o_{tc1,1} \# \hat{o}_{1,1})$

Corresponding to the above symptom, we determine the following conflict paths for both machines M1 and M2, which are equal to tentative candidate faulty transitions for this particular example:

$Confp^{M1}{}_1 = t_1, t_4$

$Confp^{M2}{}_1 = t'_1$

Corresponding to these tentative candidate transitions, we compute the following tentative diagnostic candidates for M1 and M2:

$Tdiagc^{M1}{}_1$ = M1 where t1 has been changed to x1/u2 instead of x1/u1
$Tdiagc^{M1}{}_2$ = M1 where t1 has been changed to x1/u3 instead of x1/u1
$Tdiagc^{M1}{}_3$ = M1 where t4 has been changed to z1/y2 instead of z1/y1
$Tdiagc^{M1}{}_4$ = M1 where t4 has been changed to z1/y3 instead of z1/y1
$Tdiagc^{M2}{}_1$ = M2 where t'1 has been changed to u1/z2 instead of u1/z1
$Tdiagc^{M2}{}_2$ = M2 where t'1 has been changed to u1/z3 instead of u1/z1

Notice that $Tdiagc^{M1}{}_2$, $Tdiagc^{M1}{}_4$, and $Tdiagc^{M2}{}_2$ do not explain all observable outputs of the SUT, and thus are not considered as diagnostic candidates. For example, if the fault is as specified in $Tdiagc^{M1}{}_2$ (t1: x1/u3), the SUT should produce for the external output y2 for the external input r-x1 of Tc1; however, it produces the external output y2 as shown in Table 1. The remaining tentative diagnostic candidates are considered as diagnostic candidates $Diagc^{M1}{}_1$, $Diagc^{M1}{}_3$, and $Diagc^{M2}{}_1$, respectively. For these candidates, we form the following composed machines $Diagc^{M1}{}_1 \circ M2$, $Diagc^{M1}{}_3 \circ M2$, and $Diagc^{M2}{}_1 \circ M1$. These machines are equivalent, and therefore we can not determine which machine is faulty by testing the composed system in the given architecture.

5.2 A More Complex Example

Suppose that the test suite TS = {r-x1-x1, r-x2-x1-x2-x1} is given for the two CFSMs specification shown in Figure 4 .

The application of TS to the specification of Figure 4 and its corresponding implementation of M1 and M2 (which equal to the specification with the exception that t'_3 of M2 has the output fault z_3) yields the expected and observed output sequences depicted in Table 2.

A difference between observed and expected outputs is detected for the test cases tc_1 and tc_2. Therefore, the symptoms are:

$Symp_1 = (o_{tc1,2} \# \hat{o}_{tc_{1,2}})$

$Symp_2 = (o_{tc_{2,4}} \# \hat{o}_{tc_{2,4}})$

Let us start by determining the conflict paths, tentative diagnostic candidates, and the diagnostic candidates of M1. Corresponding to the above symptoms, we determine the following conflict paths for machine M1:

$Confp^{M1}{}_1 = t_1, t_4, t_4$

$Confp^{M1}{}_2 = t_2, t_5, t_1, t_4, t_2, t_3, t_1, t_4$

The tentative candidate transitions that correspond to the above conflict paths are t1, and t4. Corresponding to these transitions, we compute the following tentative diagnostics:

$Tdiagc^{M1}{}_1$ = M1 where t1 has been changed to x1/u2 instead of x1/u1

$Tdiagc^{M1}{}_2$ = M1 where t4 has been changed to z2/y1 instead of z2/y2

$Tdiagc^{M1}{}_3$ = M1 where t4 has been changed to z2/y3 instead of z2/y2

The reader can check that all these tentative candidates do not explain the observable behavior of the given SUT. For example, if the fault is as specified in $Tdiagc^{M1}{}_1$ (t1:x1/u2), the SUT should produce the external output y3 for the external input r-x1 of tc1; however, it produces y2 as shown in Table 2. Therefore, M1 cannot be the faulty machine. The fault must be located in M2. Actually, for this example, the diagnostic candidate of M2 that explains all observed outputs of the given SUT, is where t'3 of M2 has been changed from u1/z2 to u1/z3.

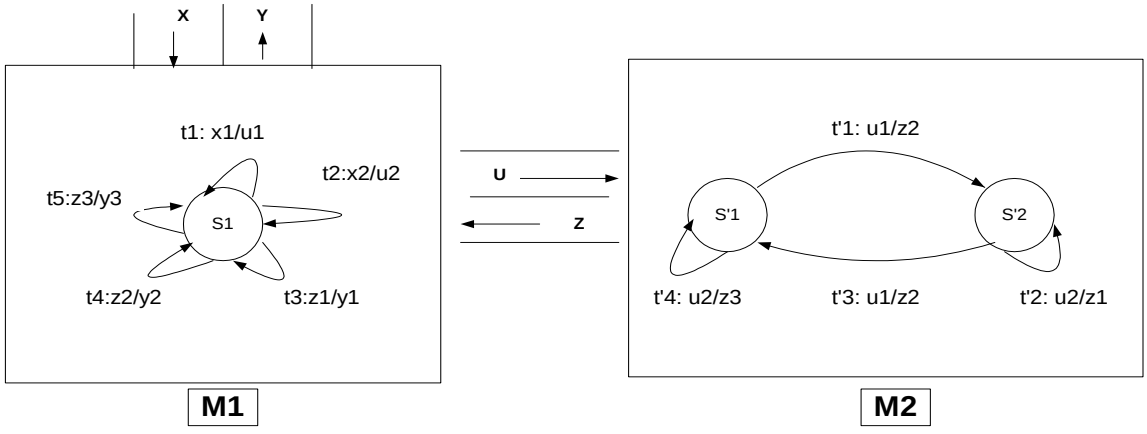

Figure 4. A system of two CFSMs M1 and M2.

tc#	tc1	tc2
Inputs	r, x_1, x_1	r, x_2, x_1, x_2, x_1
Specified transitions	t_1, t'_1, t_4, t_1, t'_3, t_4	t_2, t'_4, t_5, t_1, t'_1, t_4, t_2, t'_2, t_3, t_1, t'_3, t_4
Expected Outputs	y_2 y_2	y_3 y_2 y_1 y_2
Observed Outputs	y_2 **y_3**	y_3 y_2 y_1 **y_3**

Table 2. Test cases and their corresponding outputs

6. Conclusion and future work

In this paper, we proposed a diagnostic method that decides if it is possible to locate the faulty machine in a system of two CFSMs, once a fault has been detected in its implementation. If this is possible, it also locates the faulty machine. Currently, we are enhancing our method to cover an extended fault model. This fault model would also cover transfer faults. 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.

References

[1] D.P. Sidhu and T.K. Leung, "Formal methods for protocols: A detailed study", IEEE Trans. SE-15, 4, 1989.

[2] G. v. Bochmann and A. Petrenko,"Protocol testing: Review of methods and relevance for software testing", Département IRO, Publication départementale #107, Université de Montréal, June 1994.

[3] A. Peternko, N. Yevtushino, G.v. Bochmann, R. Dssouli,(1996) "Testing in context: Framework and test derivation", Computer Communications Journal, Special issue on protocol engineering 19, pp.1236-1249.

[4] A. Gill, Introduction to the Theory of Finite State Machines, McGraw-Hill, New York, 1962