Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/130212

Fault Diagnosis of a Protocol Implementation Specified in a System of Communicating Finite-State Machines

Qi-Ping Yang[1] *and Tae-Hyong Kim*[2,*]

[1] Shanghai Research Center for Wireless Communications, China
[2] Kumoh National Institute of Technology, Republic of Korea

Received: 2 Jan. 2019, Revised: 10 Feb. 2019, Accepted: 20 Feb. 2019
Published online: 1 Mar. 2019

Abstract: Since testing is an important but expensive process in the protocol development, formal specification models like Finite-State Machines (FSMs) are used to automate testing process. This paper focuses on specifications in a system of communicating FSMs (SCFSM) for protocols with multiple components, and explores fault diagnosis of protocol implementations based on such specifications. The target implementation is first examined with test cases generated by our test generation method. For the implementation which turns out to have a fault, we propose a heuristic algorithm to isolate and locate the fault in the specification. We also presents the assumptions and conditions required for exact fault localization of an SCFSM. Additionally, an adaptive method is proposed to minimize the fault candidates with the given test cases. The proposed method is evaluated with case studies and complexity analysis, and also software tool development in Java.

Keywords: protocol testing, fault diagnosis, formal methods, communicating finite state machines

1 Introduction

As requirements of network services are getting more various and extensive, protocol specifications have been much more complicated than before. For automated development of protocol products without faults, formal methods have been widely used with well-formed modeling techniques such as Finite-State Machines (FSMs) or the Specification Description Language (SDL) [1]. Testing and verification are necessary tools to generate the next-stage output in formal protocol development. Conformance testing, especially, which verifies if a protocol implementation conforms to its reference specification is required to increase the possibility of successful inter-operation with other existing network products.

In order to optimize testing work which usually costs enormously, there have been a lot of studies, theoretical and practical, how to derive appropriate test cases automatically and efficiently. Since the amount of test cases is related to both test cost and fault coverage of test, which are usually in trade-off relations, automatic generation of test cases with minimal cost but with maximal coverage is an ultimate goal in protocol testing. In test-case generation based on FSMs, output faults and transfer faults are usual types of faults to be detected. The Wp method is a well-known method to generate test cases with full fault coverage under that fault model for FSMs [2].

An interesting yet more complex issue associated with testing is fault diagnosis which includes fault detection and localization. In general, fault diagnosis can be classified into two categories: signal processing-based and model-based. In the signal processing-based fault diagnosis [3], some mathematical or statistical operations are performed on the measurements, or some neural network is trained using measurements to extract the information about the fault. The model-based fault diagnosis tries to predict faults according to the system behavior with well-established formal models [4], [5].

Most of current communication protocols are actually complex concurrent systems which have multiple communicating components inside. Such a protocol is more naturally modeled as a network of Communicating Finite-State Machines (CFSMs), which is usually called a system of CFSMs (SCFSM). A component machine in an SCFSM is an FSM with an input queue which communicates with other components in the system. A

* Corresponding author e-mail: taehyong@kumoh.ac.kr

well-formed system of CFSMs, which ensures generally desirable properties, such as absence of livelocks, deadlocks, and non-executable transitions, needs to be verified through formal methods. A simple technique to explore the behavior of an SCFSM, called reachability analysis, has been used for systematic generation of the global state space of a protocol [6], [7], [8].

In our previous study [9], we investigated how to derive test cases from a specification model in an SCFSM. For test-case generation, an SCFSM is usually transformed into the observably equivalent FSM, which we call the Observable Product Machine (OPM). We analyzed the characteristics of an SCFSM and checked if test-cases generated for its OPM can be applied to SCFSM testing with the same fault coverage. We showed that some faults in component FSMs might be undetected by such test cases and thus proposed a generation method of minimal test cases with full fault coverage using individual reliable resets of component FSMs.

We now focus on fault diagnosis techniques for SCFSM testing in this paper. We try to detect and locate faults of the target protocol implementation with its specification model in an SCFSM. Actually, fault diagnosis of a protocol with concurrent components have been not studied enough yet due to its complexity. We put some appropriate assumptions on the SCFSM model in order to realize feasible fault analysis. A heuristic method is proposed for fault isolation and localization with test cases generated for full fault coverage. We also provide an adaptive method to reduce the number of fault candidates with the given test cases.

This paper proceeds as follows. We first introduce some preliminary work required for SCFSM testing in section 2. Section 3 explains the motivation and problem definition of our fault diagnosis, and present the proposed fault diagnostic method in detail. The proposed method is illustrated with two examples and evaluated by complexity analysis in section 4. Section 5 concludes the paper with some discussions.

2 Preliminary Work

2.1 An SCFSM and its OPM

Definition 1.*A Finite-State Machine (FSM) is a 6-tuple* $M = (\mathscr{S}, \mathscr{I}, \mathscr{O}, \delta, \lambda, s_1)$*, where* $\mathscr{S}$ *is a finite and nonempty set of states,* $\mathscr{I}$ *is a finite and nonempty set of input symbols,* $\mathscr{O}$ *is a finite and nonempty set of output symbols,* $\delta : \mathscr{S} \times \mathscr{I} \to \mathscr{S}$ *is the state transfer function,* $\lambda : \mathscr{S} \times \mathscr{I} \to \mathscr{O}$ *is the output function, and* $s_1 \in \mathscr{S}$ *is the initial state of the machine.*

In an FSM, if the machine is given an input $a \in \mathscr{I}$ when in state $s_i \in \mathscr{S}$, it will move to state s_j and produce output b, where $s_j = \delta(s_i, a)$ and $b = \lambda(s_i, a)$. We write this as $s_i\,[M, a/b\rangle\, s_j$, or if M is understood, simply $s_i\,[a/b\rangle\, s_j$. We extend δ and λ to finite-input sequences by recursively letting, for $s_i \in \mathscr{S}$, $a \in \mathscr{I}$, and $\alpha \in \mathscr{I}^*$, $\delta(s_i, a\alpha) = \delta(\delta(s_i, a), \alpha)$ and $\lambda(s_i, a\alpha) = \lambda(s_i, a)\,\lambda(\delta(s_i, a), \alpha)$, with base cases $\delta(s_i, \varepsilon) = s_i$ and $\lambda(s_i, \varepsilon) = \varepsilon$, where ε is the empty sequence. We use $s_i\,[M, \alpha\rangle\, s_j$ or $s_i\,[\alpha\rangle\, s_j$ to indicate that FSM M moves from state s_i to s_j when an input sequence α is given. An FSM is *completely specified* and *deterministic* if and only if δ and λ are complete functions implying that, for each $s_i \in \mathscr{S}$ and $a \in \mathscr{I}$, there exists exactly one s_j and one b such that $s_i\,[a/b\rangle\, s_j$. Two states s_i and s_j of M are *distinguishable* if there is an input sequence $\alpha \in \mathscr{I}^*$ such that $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$ and otherwise, s_i and s_j are said to be *equivalent*. We use $s_i \not\sim s_j$ and $s_i \sim s_j$ to denote two states are distinguishable or equivalent, respectively. Two FSMs M and M' are *equivalent* if and only if for every state in M there is an equivalent state in M', and vice versa. If M and M' are equivalent, we denote as $M \sim M'$ and otherwise $M \not\sim M'$. Finally, an FSM is *initially connected* if every state can be reached from its initial sate, *strongly connected* if every state can be reached from every state, and *minimal* if it does not contain two equivalent states. An FSM possesses a *reliable reset* if there is an input symbol $r \in \mathscr{I}$ such that, for any state $s_i \in \mathscr{S}$, $\delta(s_i, r) = s_1$ and it is known to have been implemented correctly. Obviously, if M is initially connected and has a reliable reset then M is strongly connected. In this paper, we assume that each FSM we consider is minimal since there are algorithms that convert an FSM into an equivalent minimal FSM [10]. We also assume that FSMs under test are deterministic, completely specified , initially connected and has a reliable reset.

The architecture of a communication protocol usually consists of several components connected via queues. This organization can be modeled as a system of Communicating Finite-State Machines (SCFSM), where each component is a Communicating Finite-State Machine (CFSM). A CFSM can communicate with other CFSM(s) by producing an output that is placed in the input queue of others.

Definition 2.*A system of Communicating Finite-State Machines (SCFSM) is a set of L machines,* $\mathscr{M} = \{M_k : 1 \leq k \leq L\}$*, where each machine* M_k *is a CFSM that is represented as an FSM by* $(\mathscr{S}_k, \mathscr{I}_k, \mathscr{O}_k, \delta_k, \lambda_k, s_{k1})$ *with an input queue* β_k*.*

Transitions of each CFSM M_k are called *local transitions*. Note that a CFSM M_k has one input queue β_k. Through β_k, M_k can receive inputs from another component $M_j (1 \leq j \neq k \leq L)$ as well as the environment with first come, first served. An input symbol of a CFSM M_k shall be called *external input* if the input can be received from the environment. An output symbol of a CFSM M_k is an *external output* if it is observed by the environment. Otherwise, it is called as *internal output* that cannot be observed by the environment. Let $\mathscr{I} = \bigcup_{1 \leq k \leq L} \mathscr{I}_k$ and $\mathscr{O} = \bigcup_{1 \leq k \leq L} \mathscr{O}_k$. Then $\mathscr{I}_{ext} \subseteq \mathscr{I}$

and $\mathscr{O}_{ext} = \mathscr{O} \setminus \mathscr{I}$ are the sets of external input symbols and external output symbols of $\mathscr{M}$, respectively. We assume the SCFSM $\mathscr{M}$ holds the pairwise disjoint input property, $\forall 1 \leq k,h \leq L, k \neq h \Longrightarrow \mathscr{I}_k \cap \mathscr{I}_h = \emptyset$, and thus the input a is contained in the input symbols of only one CFSM and thus $\mathscr{M}$ is deterministic. If an input $a \in \mathscr{I}_k$ is placed in the input queue β_k of $\mathscr{M}$, this leads to a local transition in M_k , say $s_{k_i}[M_k, a/b\rangle s_{k_j}$. Since one CFSM can interact with another, it is possible for output b to trigger a local transition in other CFSM. If there is a CFSM M_h such that $b \in \mathscr{I}_h$, then the symbol b is placed in the input queue β_h, causing in turn a local transition in M_h, and so on; otherwise b becomes an external output to the environment. Thus, only symbols in $\mathscr{O}_{ext}$ can be sent to the environment while $\mathscr{O}_{int} = \mathscr{O} \cap \mathscr{I}$ are internal symbols which can not be observed by the environment.

The *global state* of the SCFSM is defined by the states of each component CFSM and the contents of the queues. A global state is *stable* if all the queues are empty and otherwise it is *unstable*. We assume a stable initial state and slow environment, i.e., the SCFSM has enough time to consume any symbol in any queue before the environment provides the next input. It is then easy to show that exactly one input queue contains one symbol in any unstable state. For simplicity, we can use a single global queue β of size one instead of separate queue for each component CFSM. Thus, the global state of an SCFSM can be written as σ if it is stable, or $\sigma.a$ if it is unstable, where $\sigma = (s_{1_k}, \cdots, s_{L_k})$ is the state combination of all components of the SCFSM and $a \in \mathscr{O}_{int}$ is the symbol in the input queue β. We also assume that *livelock freeness*, that is, an SCFSM does not allow a cycle of transitions over unstable states. Under the slow environment and livelock freeness assumptions, an SCFSM can be converted to an OPM whose behavior is equivalent to that of the original SCFSM.

Definition 3.*The Observable Product Machine (OPM) of an SCFSM $\mathscr{M}$ is a tuple $\mathscr{P}(\mathscr{M}) = (\Sigma_s^r, \mathscr{I}_{ext}, \mathscr{O}_{ext}, \delta_p, \lambda_p, \sigma_1)$, where Σ_s^r is a finite and nonempty set of reachable and stable global states of $\mathscr{M}$, $\mathscr{I}_{ext}$ is a finite and nonempty set of external input symbols of $\mathscr{M}$, $\mathscr{O}_{ext}$ is a finite and nonempty set of external output symbols of $\mathscr{M}$, $\delta_p : \Sigma_s^r \times \mathscr{I}_{ext} \to \Sigma_s^r$ is the state transfer function, $\lambda_p : \Sigma_s^r \times \mathscr{I}_{ext} \to \mathscr{O}_{ext}$ is the output function, and $\sigma_1 \in \Sigma_s^r$ is the initial global state $(s_{11}, s_{21}, \cdots, s_{L1})$.*

Let $\Sigma = \prod_{1 \leq k \leq L} \mathscr{S}_k$ be the stable state space of $\mathscr{M}$, then the set of reachable and stable global states of an OPM is $\Sigma_s^r = \{\sigma : \exists \alpha \in \mathscr{I}_{ext}^*, \delta_p(\sigma_1, \alpha) = \sigma\} \subseteq \Sigma$. Note that $\Sigma_s^r \subseteq \Sigma$ because some combinations of local states might not be reachable and stable global states if all incoming local transitions to each local state of those combinations would have internal outputs only. A transition of an OPM, upon an external input a at a stable global state σ_m, proceeds according to a sequence of local transitions $\sigma_m.a\,[a/a_i\rangle\,\sigma_i.a_i\,[a_i/a_j\rangle \cdots \sigma_h.a_h\,[a_h/b\rangle\,\sigma_n$, where σ_n is the next stable global state. Such a transition is called *stable global transition*, which is denoted by $\sigma_m\,[\![a/b]\!\rangle\,\sigma_n$. $\delta_p(\sigma_m, a) = \sigma_n$ and $\lambda_p(\sigma_m, a) = b$ represent the state transfer function and output function of this global transition respectively. All transitions of an OPM are *stable global transitions*. When executing a stable global transition, one or more local transitions are traversed. A prefix of a stable global transition is defined as a *transit global transition* and can be denoted by $\sigma_m\,[\![(t_1, \cdots, t_i)a/b]\!\rangle\,\sigma_n.b$, where a is an external input, b is an internal output generated by the last traversed local transition t_i, and $t_1, \cdots, t_i$ are the traversed local transitions.

2.2 Test-case generation for an SCFSM

While an SCFSM can be converted to an OPM that is considered as a deterministic, minimal, completely specified, and strongly-connected single FSM, FSM testing techniques might be used in SCFSM testing as it is. We assume that $\mathscr{M}_{\mathscr{S}} = \{M_k : 1 \leq k \leq L\}$ and $\mathscr{M}_{\mathscr{I}} = \{M'_k : 1 \leq k \leq L\}$ are a specification SCFSM and an implementation SCFSM, respectively, where there is a one-to-one mapping between M_k and M'_k. The corresponding OPMs are $\mathscr{P}(\mathscr{M}_{\mathscr{S}})$ and $\mathscr{P}(\mathscr{M}_{\mathscr{I}})$. $\Im_k$ is the set of all implementation FSMs with the same input alphabet as M_k. $\Im_k$ is called a *fault domain* for M_k. $\Im = \bigcup_{L \geq k \geq 1} \Im_k$ is the fault domain for $\mathscr{M}_{\mathscr{S}}$. In general the traditional fault model <specification, conformance relation, fault domain> is used [11] .

The strongly connectedness and livelock freeness of the OPM of an SCFSM can be guaranteed if we assume there is a global reliable reset and no livelock. However, we found in the previous study that the minimality of the OPM cannot be guaranteed even if all component CFSMs are deterministic, minimal, completely specified, strongly connected and free of livelock. Therefore, for applying test generation techniques for the OPM, it is normally required to minimize the OPM first if it is not minimal.

We also showed that test generation methods for FSMs with full-fault coverage, such as the W method and Wp method, may not guarantee the full fault coverage in testing an SCFSM with its OPM because the stable state space of a faulty implementation might be larger or smaller than that of the correct implementation [9]. If a faulty implementation has an extra state that is not equivalent to an exiting state of the specification machine, that faulty state might be undetected when tested by test cases generated for the OPM of the specification SCFSM.

In order to handle this fault masking problem, we presented a Wp-based test generation method considering state addition faults by converting the target OPM to the full OPM [9]. The generated test suite is complete w.r.t the fault model $< \mathscr{P}(\mathscr{M}_{\mathscr{S}}), \sim, \mathscr{P}(\Im) >$ if the test suite detects all SCFSM implementations whose OPMs are not equivalent to that of the specification SCFSM. In order to reduce the length of test sequences multiplied for checking state addition faults, we also used individual reliable resets of component FSMs.

3 Fault diagnosis for an SCFSM

3.1 Motivation

A test suite consists of multiple test sequences checking if there are differences between an implementation and its reference specification by observing input/output (I/O) behaviors of the implementation. The differences between the observed I/O pairs of the implementation and the expected I/O pairs of the specification, which are called *symptoms*, imply the existence of faults in the implementation [12]. In addition to detecting symptoms, identifying observed symptoms in the specification is an interesting but very complex problem. In order to solve this problem, a diagnostic process called fault diagnosis is required to pinpoint the root cause of those observed differences. The fault diagnosis is very useful in the development of a system since it can make easy the job of correcting an implementation, so that it can conform to its specification [13].

Actually, fault diagnosis has not been studied a lot due to its complexity. A. Ghedamsi *et al.* [12] and D. Lee *et al.* [13] proposed the process of fault diagnosis with FSMs. Both of their work are locating the differences between an implementation and its specification under the assumption that there is only a single fault in the implementation. Ghedamsi *et al.* generated a set of transitions called *conflict set*. These transitions whose failure could explain the faults are further examined to find the faulty transitions by deriving additional test cases with *limited characterization set* [12]. D. Lee *et al.* proposed a fast algorithm which reduced the number of diagnostic candidates by deriving additional test cases called *distinguishing tests* [14]. Ghedamsi *et al.* [15] extended their procedure for diagnosing multiple faults with FSMs under the assumption that each fault is reachable through non-faulty transitions.

A. Ghedamsi et al. proposed some heuristic methods to model the process of fault diagnosis with a system represented by CFSMs [16], [17]. However, they limited the interaction between the component machines by assuming that if a component machine receives an input symbol from another machine, it must execute the corresponding local transition and emit an external output symbol that can be observed by the environment directly. Moreover, it is not always possible to guarantee the precise localization of a single fault because two faulty implementations with a different single fault might cause the same observable behavior. Furthermore, the cost of reducing fault candidates was not considered in their methods.

This paper presents a heuristic method by concentrating on removing the limitation on communications among component machines of fault localization of implementations with a single fault. The proposed method can precisely locate the difference between the specification and implementation if the root cause of observed symptoms is unique; otherwise, it provides a set of all potential single faults. It also provides an adaptive selection of test cases and a fast procedure to discriminate diagnostic candidates for reducing the cost of fault diagnosis.

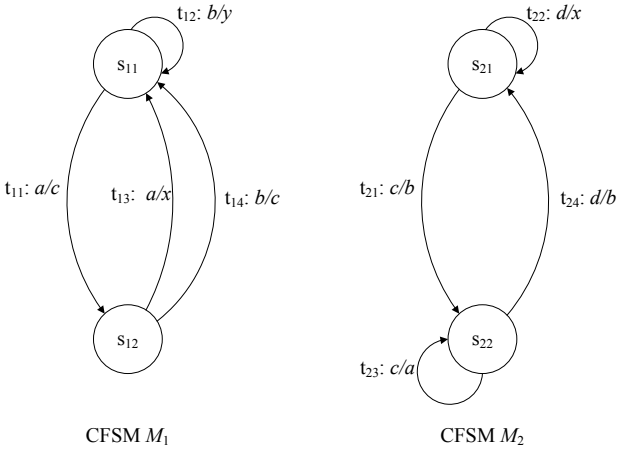

Fig. 1: An example SCFSM $\mathscr{M}_1$

3.2 Problem statement

It is important to note that application of an input symbol in a test case might trigger execution of one or more local transitions, depending on whether that input produces an external or an internal output. Although we assume that a single output fault or transfer fault exists in only one component machine, generation of diagnostic fault candidates still needs to consider all possible differences for traversed local transitions. In fault diagnosis of a single FSM, if an implementation has only one output fault, it must occur in the transition traversed by the last input in the observed symptom. However, it is not true for an SCFSM; the output fault can occur in any local transitions constructing the last global transition of the symptom.

Let us consider an SCFSM $\mathscr{M}_1$ shown in Figure 1, where $\mathscr{M}_1 = \{M_1, M_2\}$ with $\mathscr{I}_{ext} = \{a,b,c,d\}$ and $\mathscr{O}_{ext} = \{x,y\}$. Assume that the target implementation of M_1 has an output fault, where t_{11} produces x instead of c in response to input a. A test case $TC = rccb$ is applied to the faulty implementation and the expected and observed output sequences are shown in Table 1. Here, for analyzing the problem, we list each global transition and its corresponding traversed local transitions separately.

Based on the observed outputs of Table 1, if we assume there is a single output fault, the diagnostic candidate should include the following candidate: t_{11} produces x instead of c in response to the input a in CFSM M_1. By observing, the local transition t_{11} is not traversed by the final global transition.

Table 1: Expected and observed output sequences of $\mathscr{M}_1$

Test Case	$TC = rccb$		
Inputs	c	c	b
Local transitions	t_{21}, t_{12}	t_{23}, t_{11}, t_{13}	t_{12}
Expected outputs	y	x	y
Observed outputs	y	x	x

Another issue to be considered is that, for a faulty implementation, it might not be possible to guarantee the precise localization of a single fault. Z. Pap *et al.* [18] and K. El-fakih *et al.* [19] handled similar problems for FSMs and a system of two communicating machines called embedded machine and context machine respectively. However, fault diagnosis can be guaranteed only if a faulty implementation of CFSMs is assumed to be a minimized machine, and the system modeled in the work of K. El-fakih et al. is not suitable for general FSMs. We extend this issue to the SCFSM with arbitrary number of component machines. We first illustrate the problem with two examples and present a fault diagnostic method which is a modified version of that of K. El-fakih *et al.*. The proposed method exactly locates the difference between an implementation and its specification, or provide the set of all potential single faults.

The SCFSM $\mathscr{M}_1$ shows a situation that we cannot distinguish different faults for one local transition. If the local transition t_{12} is falsely implemented to produce x or c instead of y, we get a symptom easily by a test case rb. However, we cannot distinguish the diagnostic candidates between the above two changes since only one external output x can be observed in the faulty implementation.

Table 2: Two faulty implementations of $\mathscr{M}_1$

Faulty machine	Original transition	Faulty transition
$\mathscr{M}_1'$	t_{13}: $s_{12}\,[a/x\rangle\, s_{11}$	t_{13}': $s_{12}\,[a/y\rangle\, s_{11}$
$\mathscr{M}_1''$	t_{23}: $s_{22}\,[c/a\rangle\, s_{22}$	t_{23}'': $s_{22}\,[c/b\rangle\, s_{22}$

Let us consider the case when we cannot distinguish between faults in different machines. Suppose there are two faulty implementations $\mathscr{M}_1'$ and $\mathscr{M}_1''$ formed by injecting different single faults to $\mathscr{M}_1$ as listed in Table 2. Table 3 shows the expected and observed output sequences when a test case $TC = rca$ is fed into the faulty implementation $\mathscr{M}_1'$ of $\mathscr{M}_1$.

Table 3: Observation of the faulty implementation $\mathscr{M}_1'$

Test Case	TC
Inputs	rca
Local transitions	$t_{21}, t_{12}, t_{11}, t_{23}, t_{13}$
Expected outputs in $\mathscr{M}_1$	yx
Observed outputs in $\mathscr{M}_1'$	yy

Based on Table 3, a symptom is identified and the further fault diagnosis should be executed to locate the faulty transition. Unfortunately, we cannot be sure that these observed outputs have been from the faulty transitions t_{13}' of M_1 in $\mathscr{M}_1'$ or from t_{23}'' of M_2 in $\mathscr{M}_1''$. The above two examples show that some faulty implementations with different singles faults can be observably equivalent.

Note also that the work of [16] and [17] did not consider the cost of fault diagnosis. Fault diagnosis of an SCFSM requires generation of its OPM. If we remove the limitation of interactions, when discriminating diagnostic candidates by additional tests with distinguishing tests [13] or when identifying diagnostic candidates by conformance testing the mutant implementations generated by injecting those diagnostic candidates. Since the generation cost of OPMs and test sequence based on the OPMs are quite high, we need to reduce the number of diagnostic candidates as much as possible in order to reduce generation of OPMs.

3.3 The proposed fault diagnosis method

Now we present a diagnostic approach for a system modeled by an SCFSM. The proposed method starts by detecting a symptom when applying a given test suite to the implementation. In order to explain the symptom, a conflict set of local transitions which are supposed to participate in generation of the symptom is derived. For each transition t_k in the conflict set, its corresponding diagnostic candidates may include three kinds of diagnosis: $Outputs(t_k)$, $EndStates(t_k)$, and $OutStates(t_k)$ which are formed by analyzing all possible single output faults, all possible single transfer faults, and both types of faults respectively. Usually, as the total number of diagnostic candidates is very large, an adaptive procedure of test case selection for discriminating diagnostic candidates is adopted. In this procedure, each selected test case updates the current conflict set and diagnostic candidates until at most one diagnostic candidate is left or other termination conditions are satisfied. A fast procedure using so-called cross verification [13] is used in case that the remaining number of diagnostic candidates is more than one after the previous procedure. Finally, the fault diagnosis is sure to produce one of the following results: *exact localization with one single fault*, *a set of possible single faults but no exact localization*, or *multiple faults*.

Algorithm 1. The proposed fault diagnosis algorithm

Step 1: Detecting a single symptom

Given a specification SCFSM $\mathscr{M}_{\mathscr{S}} = \{M_k = (\mathscr{S}_k, \mathscr{I}_k, \mathscr{O}_k, \delta_k, \lambda_k, s_{k1}) : 1 \le k \le L\}$ and its implementation $\mathscr{M}_{\mathscr{I}} = \{M_k' = (\mathscr{S}_k', \mathscr{I}_k', \mathscr{O}_k', \delta_k', \lambda_k', s_{k1}') : 1 \le k \le L\}$, a test suite $TS = \{tc_1, ..., tc_p\}$ can be derived using the test selection methods proposed in our previous work[9]

based on the OPM $\mathscr{P}(\mathscr{M}_{\mathscr{S}}) = (\Sigma_s^r, \mathscr{I}_{ext}, \mathscr{O}_{ext}, \delta_p, \lambda_p, \sigma_1)$. Each test case $tc_k = \langle a_{k,1}, a_{k,2}, \cdots, a_{k,m_k} \rangle \in TS$ generates its expected output sequences $\langle o_{k,1}, o_{k,2}, \cdots, o_{k,m_k} \rangle$ and observed output sequences $\langle o'_{k,1}, o'_{k,2}, \cdots, o'_{k,m_k} \rangle$ by applying tc_k to both $\mathscr{M}_{\mathscr{S}}$ and $\mathscr{M}_{\mathscr{I}}$, where o_{k,m_k} is the expected output and o'_{k,m_k} is the observed output after input a_{k,m_k}, respectively. If differences between $o_k = \langle o_{k,1}, o_{k,2}, \cdots, o_{k,m_k} \rangle$ and $o'_k = \langle o'_{k,1}, o'_{k,2}, \cdots, o'_{k,m_k} \rangle$ exist, $\mathscr{M}_{\mathscr{I}}$ has at least one faulty transition. The first difference between o_k and o'_k denoted by $(o_{k,l} \neq o'_{k,l})$, where $1 \leq l \leq m_k$, is a symptom.

Step 2: Generating a conflict set

With each symptom for $\mathscr{M}_{\mathscr{S}}$ and $\mathscr{M}_{\mathscr{I}}$, a conflict set is generated which is a set of local transitions supposed to be traversed for generation of the symptom. Since $\mathscr{M}_{\mathscr{I}}$ has only one single faulty transition, one of these local transitions must be faulty and the transitions after the symptom in the test case can be ignored. Suppose, for a symptom $(o_{k,l} \neq o'_{k,l})$, the sequence of expected global transitions in $\mathscr{P}(\mathscr{M}_{\mathscr{S}})$ is $\langle \tau_{k,1}, ..., \tau_{k,l} \rangle$, where each $\tau_{k,l}$ consists of one or more local transitions. Therefore, the conflict set is the union of all local transitions constructing $\langle \tau_{k,1}, ..., \tau_{k,l} \rangle$, represented as $CS_{tr} = \bigcup_{1 \leq i \leq l} LOCAL(\tau_{k,i}) = \{t_{k,1}, t_{k,2}, ..., t_{k,n}\}$, where $t_{k,i}$ is a local transition for $1 \leq i \leq n \leq \sum_{1 \leq k \leq L}(|S_k||I_k|)$, and $LOCAL(\tau)$ is the set of local transitions constructing a global transition τ. Note that the maximum size of the conflict set is the number of total local transitions, that is $\sum_{1 \leq k \leq L}(|S_k||I_k|)$ when a global transition traverses all local transitions.

Step 3: Generating diagnostic candidates

Given $CS_{tr} = \{t_{k,1}, t_{k,2}, ..., t_{k,n}\}$, we need to determine fault candidates of each local transition that might lead to the observed symptom $(o_{k,l} \neq o'_{k,l})$. A number of diagnostic candidates for each local transition $t_{k,i} \in CS_{tr}$ that are suspected to be faulty can be formed by injecting a possible output fault, transfer fault or both to $t_{k,i}$ with all remaining local transitions of $\mathscr{M}_{\mathscr{S}}$ unchanged; this leads to a mutant specification SCFSM $\tilde{\mathscr{M}}_{\mathscr{S}}$. If the expected external output sequences of a test case tc_k for $\mathscr{M}_{\mathscr{S}}$ and the observed output sequences of tc_k for $\mathscr{M}_{\mathscr{I}}$ are equal, the output fault, transfer fault or both faults under consideration will be included in the corresponding diagnostic candidate set, $Outputs(t_{k,i})$, $EndStates(t_{k,i})$, or $OutStates(t_{k,i})$, respectively. If $Outputs(t_{k,i})$, $EndStates(t_{k,i})$, and $OutStates(t_{k,i})$ are all empty for some $t_{k,i}$, the conflict set CS_{tr} is updated by removing this local transition $t_{k,i}$.

Step 4: Minimizing the size of diagnostic candidates

For convenience of presentation, we let DC_{tr} denote the set of total diagnostic candidates, and $DC_{tr}(t_{k,i})$ denote the set of total diagnostic candidates for a single local transition $t_{k,i}$ such that $DC_{tr}(t_{k,i}) = Outputs(t_{k,i}) \cup EndStates(t_{k,i}) \cup OutStates(t_{k,i})$. Given a test suite $TS = \{tc_1, ..., tc_p\}$, if the first symptom is observed by applying the test case tc_k in the step 1, we reduce the size of diagnostic candidates by using the adaptive procedures shown in Figure 2 and Figure 3 for the following two sub test suites: $TS_1 = \{tc_1, ..., tc_{k-1}\}$ and $TS_2 = \{tc_{k+1}, ..., tc_p\}$, respectively. The former has been already applied to the faulty implementation and the latter has not been tested yet.

input : $\mathscr{M}_{\mathscr{S}}$, $\mathscr{M}_{\mathscr{I}}$, TS_1, CS_{tr}, DC_{tr}
output: Updated CS_{tr}, Updated DC_{tr}

1 **foreach** $tc_i \in TS_1$ *and* $LOCAL(tc_i) \cap CS_{tr} \neq \emptyset$ **do**
2 **if** $|DC_{tr}| \leq 1$ **then** e;
3 xit;
4 $o' \leftarrow$ `ObservedOutputSequence` $(\mathscr{M}_{\mathscr{I}}, tc_i)$;
5 **foreach** $t_{k,j} \in LOCAL(tc_i) \cap CS_{tr}$ **do**
6 **foreach** *diagnostic candidate* $dc_l \in Outputs(t_{k,j})$ **do**
7 $\tilde{\mathscr{M}}_{\mathscr{S}} \leftarrow$ `GenerateMutant` $(\mathscr{M}_{\mathscr{S}})$;
8 $\tilde{o} \leftarrow$ `ExpectedOutputSequence` $(\tilde{\mathscr{M}}_{\mathscr{S}}, tc_i)$;
9 **if** $\tilde{o} \neq o'$ **then**
10 update DC_{tr} by removing dc_l from $Outputs(t_{k,j})$;
11 **end**
12 **end**
13 **foreach** *diagnostic candidate* $dc_l \in Endstates(t_{k,j})$ **do**
14 $\tilde{\mathscr{M}}_{\mathscr{S}} \leftarrow$ `GenerateMutant` $(\mathscr{M}_{\mathscr{S}})$;
15 $\tilde{o} \leftarrow$ `ExpectedOutputSequence` $(\tilde{\mathscr{M}}_{\mathscr{S}}, tc_i)$;
16 **if** $\tilde{o} \neq o'$ **then**
17 update DC_{tr} by removing dc_l from $Endstates(t_{k,j})$;
18 **end**
19 **end**
20 **foreach** *diagnostic candidate* $dc_l \in OutStates(t_{k,j})$ **do**
21 $\tilde{\mathscr{M}}_{\mathscr{S}} \leftarrow$ `GenerateMutant` $(\mathscr{M}_{\mathscr{S}})$;
22 $\tilde{o} \leftarrow$ `ExpectedOutputSequence` $(\tilde{\mathscr{M}}_{\mathscr{S}}, tc_i)$;
23 **if** $\tilde{o} \neq o'$ **then**
24 update DC_{tr} by removing dc_l from $OutStates(t_{k,j})$;
25 **end**
26 **end**
27 **if** $DC_{tr}(t_{k,j}) = \emptyset$ **then**
28 update CS_{tr} by removing $t_{k,j}$;
29 **end**
30 **end**
31 **end**

Fig. 2: Adaptive minimizing procedure based on TS_1

In the procedure presented in Figure 2, we need only to consider $tc_i \in TS_1$ if tc_i traverses some local transitions in the conflict set, that is $LOCAL(tc_i) \cap CS_{tr} \neq \emptyset$ where $LOCAL(tc_i)$ represents the set of local transitions traversed by executing tc_i. This is because we assume only one local transition has faults. We now check diagnostic candidates for each local transition that exists in the intersection of $LOCAL(tc_i)$ and CS_{tr}. First, the function *GenerateMutant* in the Figure 2 generates a mutant $\tilde{\mathscr{M}}_{\mathscr{S}}$ that is formed by injecting a possible diagnostic candidate in a local transition of $\mathscr{M}_{\mathscr{S}}$. Then we compare the expected output sequence of $\tilde{\mathscr{M}}_{\mathscr{S}}$ and the observed output sequence of $\mathscr{M}_{\mathscr{I}}$ (the observed output sequence has been already observed in Step 1). If a symptom is found, we remove the corresponding diagnostic candidate since $\mathscr{M}_{\mathscr{I}}$ can never be equivalent to the mutant $\tilde{\mathscr{M}}_{\mathscr{S}}$. If $DC_{tr}(t_{k,j})$ turns to be empty for a certain local transition $t_{k,j} \in CS_{tr}$, we remove $t_{k,j}$ from CS_{tr} since we have considered all possible fault candidates for that local transition.

For each $tc_i \in TS_2$, we apply the input sequence to $\mathscr{M}_{\mathscr{I}}$ and compare the observed output sequence with the expected output sequence of $\mathscr{M}_{\mathscr{S}}$ if tc_i traverses some local transitions in the conflict set. If a symptom is found, we generate a new conflict set with tc_i by the process in step 2, and update the original conflict set by making an intersection with this new conflict set. At the same time, we can reduce the diagnostic candidates if some local transitions are removed from the conflict set. The above process is represented from lines 3 to 11 in the procedure shown in Figure 3. Lines 12 to 37 in Figure 3 do the same checking as lines 4 to 29 in Figure 2 for reducing the number of diagnostic candidates.

Note that we terminate both procedures if at most one diagnostic candidate remains in the procedures shown in Figure 2 and Figure 3. However, if the test suite for conformance testing is too long, we can relax the terminate condition, such as if the number of total diagnostic candidates is less than some reasonable constant or if the diagnostic candidates are not reduced after a few test cases that are examined in the procedure.

For each diagnostic candidate in the candidate sets DC_{tr}, a mutant specification SCFSM is generated by assigning the output fault, transfer fault or both faults to the conjectured local transition. Assuming the number of total diagnostic candidates denoted by $|DC_{tr}|$ is n, the set of mutant SCFSMs is $\{\tilde{\mathscr{M}}_1, \cdots, \tilde{\mathscr{M}}_n\}$ and the set of OPMs is $\{\mathscr{P}(\tilde{\mathscr{M}}_1), \cdots, \mathscr{P}(\tilde{\mathscr{M}}_n)\}$, where each OPM is assumed to be minimized.

Now we examine a pair of OPMs, $(\mathscr{P}(\tilde{\mathscr{M}}_j), \mathscr{P}(\tilde{\mathscr{M}}_{j+1}))$, $j = 1, \cdots, n-1$. If they are equivalent, we group the corresponding fault candidates by using $\mathscr{P}(\tilde{\mathscr{M}}_j, \tilde{\mathscr{M}}_{j+1})$ to represent the equivalent OPM and consider the next pair $(\mathscr{P}(\tilde{\mathscr{M}}_j, \tilde{\mathscr{M}}_{j+1}), \mathscr{P}(\tilde{\mathscr{M}}_{j+2}))$. Otherwise, we can easily derive a test sequence that distinguishes the two machines [20], and we apply this sequence to both machines and also to $\mathscr{M}_{\mathscr{I}}$. If the

input : $\mathscr{M}_{\mathscr{S}}, \mathscr{M}_{\mathscr{I}}, TS_2, CS_{tr}, DC_{tr}$
output: Updated CS_{tr}, Updated DC_{tr}

1 **foreach** $tc_i \in TS_2$ *and* $LOCAL(tc_i) \cap CS_{tr} \neq \emptyset$ **do**
2 **if** $|DC_{tr}| \leq 1$ **then** e;
3 xit;
4 $o \leftarrow$ `ExpectedOutputSequence` $(\mathscr{M}_{\mathscr{S}}, tc_i)$;
5 $o' \leftarrow$ `ObservedOutputSequence` $(\mathscr{M}_{\mathscr{I}}, tc_i)$;
6 **if** $o \neq o'$ **then**
7 $CS'_{tr} \leftarrow$ `GenerateConflictSet` (tc_i);
8 **foreach** $t_{k,j} \in CS_{tr}$ *and* $t_{k,j} \notin CS'_{tr}$ **do**
9 update DC_{tr} by removing $DC_{tr}(t_{k,j})$;
10 **end**
11 $CS_{tr} \leftarrow CS'_{tr} \cap CS_{tr}$;
12 **end**
13 **foreach** $t_{k,j} \in LOCAL(tc_i) \cap CS_{tr}$ **do**
14 **foreach** *diagnostic candidate* $dc_l \in Outputs(t_{k,j})$ **do**
15 $\tilde{\mathscr{M}}_{\mathscr{S}} \leftarrow$ `GenerateMutant` $(\mathscr{M}_{\mathscr{S}})$;
16 $\tilde{o} \leftarrow$ `ExpectedOutputSequence` $(\tilde{\mathscr{M}}_{\mathscr{S}}, tc_i)$;
17 **if** $\tilde{o} \neq o'$ **then**
18 update DC_{tr} by removing dc_l from $Outputs(t_{k,j})$;
19 **end**
20 **end**
21 **foreach** *diagnostic candidate* $dc_l \in Endstates(t_{k,j})$ **do**
22 $\tilde{\mathscr{M}}_{\mathscr{S}} \leftarrow$ `GenerateMutant` $(\mathscr{M}_{\mathscr{S}})$;
23 $\tilde{o} \leftarrow$ `ExpectedOutputSequence` $(\tilde{\mathscr{M}}_{\mathscr{S}}, tc_i)$;
24 **if** $\tilde{o} \neq o'$ **then**
25 update DC_{tr} by removing dc_l from $Endstates(t_{k,j})$;
26 **end**
27 **end**
28 **foreach** *diagnostic candidate* $dc_l \in OutStates(t_{k,j})$ **do**
29 $\tilde{\mathscr{M}}_{\mathscr{S}} \leftarrow$ `GenerateMutant` $(\mathscr{M}_{\mathscr{S}})$;
30 $\tilde{o} \leftarrow$ `ExpectedOutputSequence` $(\tilde{\mathscr{M}}_{\mathscr{S}}, tc_i)$;
31 **if** $\tilde{o} \neq o'$ **then**
32 update DC_{tr} by removing dc_l from $OutStates(t_{k,j})$;
33 **end**
34 **end**
35 **if** $DC_{tr}(t_{k,j}) = \emptyset$ **then**
36 update CS_{tr} by removing $t_{k,j}$;
37 **end**
38 **end**
39 **end**

Fig. 3: Adaptive minimizing procedure based on TS_2

observed output sequence of one machine equals to that of $\mathscr{M}_{\mathscr{J}}$, we keep this machine for further checking and discard the other since that machine can never be equivalent to $\mathscr{M}_{\mathscr{J}}$. If neither of them has the same observed output sequence as that of $\mathscr{M}_{\mathscr{J}}$, we discard both of them. In each examination, we may remove one or two machines. Therefore, we can reduce the number of diagnostic candidates until either no or one machine, grouped or not, is left. If the set is empty, $\mathscr{M}_{\mathscr{J}}$ has multiple faults and does not match our fault model, so the diagnostic process is finished. If one ungrouped machine is left, then a single diagnostic candidate is obtained. If the remaining OPMs are grouped, the single fault can not be located uniquely.

Step 5: Confirming the fault diagnosis While one machine is left in the set of OPMs, sometimes we need to confirm the diagnosis by the final confirmation: do execute conformance testing on $\mathscr{M}_{\mathscr{J}}$ by assuming that remaining machine as the specification. If the remaining machine is not grouped and is equivalent to $\mathscr{M}_{\mathscr{J}}$, we can locate the fault in the implementation exactly. If the machine is grouped and is equivalent to $\mathscr{M}_{\mathscr{J}}$, we have a set of possible faults in the implementation so we can not identify the fault in the implementation uniquely. Since we have considered all possible fault candidates with a single fault assumption, if the machine is not equivalent to $\mathscr{M}_{\mathscr{J}}$, we should conclude that the implementation has multiple faults.

4 Evaluation

4.1 Case studies

First, a case study is given to evaluate the proposed diagnostic method. Consider the SCFSM specification $\mathscr{M}_1$ shown in the Figure 1, where $\mathscr{I}_{ext} = \{a,b,c,d\}$ and $\mathscr{O}_{ext} = \{x,y\}$. Let σ_1 represent global state (s_{11},s_{21}) and σ_2 represent global state (s_{11},s_{22}). A test suite TS with 11 test cases is listed in the Table 4 derived from the OPM of Figure 4 by our test case generation method [9].

Faulty implementation 1 Given a faulty implementation $\mathscr{M}_1'$ which is equal to $\mathscr{M}_1$ except a faulty transition, t_{11}' : $s_{11}\,[M_1,a/x\rangle\, s_{12}$, converted from t_{11}: $s_{11}\,[M_1,a/c\rangle\, s_{12}$. We apply the above test suite TS to $\mathscr{M}_1'$ until a symptom $(o_{tc_{6,4}}, o'_{tc_{6,4}})$ is observed at the 4^{th} input of the 6^{th} test case tc_6 where, according to the OPM, $\sigma_2\,[d/y\rangle\,\sigma_1$ have produced. The expected and observed outputs of traversed local transitions in the test cases that have been executed for detecting the symptom (shown in bold) are listed in Table 5.

Corresponding to the above symptom and the traversed local transitions, the following candidate set CS_{tr} which includes local transitions that are suspected to be faulty is determined as: $CS_{tr} = \{t_{11},t_{12},t_{13},t_{14},t_{21},t_{23},t_{24}\}$.

A single local transition of the conflict set with one possible output fault, transfer fault or both faults is replaced in the specification to form a mutant, one at a time. If the behavior of this mutant for test case tc_6 cannot explain the observed symptom, that fault is not a diagnostic candidate. For example, if a mutant that converts the output of t_{13} in M_1 to y instead of x is the faulty implementation, it must produce the output sequence xxx for tc_6. However, its expected output sequence for tc_6 is yyy. Then, the output fault converting to y of t_{13} is not a diagnostic candidate. The whole diagnostic candidates that correspond to the above conflict set are shown in Table 6. Since the diagnostic candidates of t_{13} and t_{14} are empty, we update the conflict set to $CS_{tr} = \{t_{11},t_{12},t_{21},t_{23},t_{24}\}$ by removing these two local transitions.

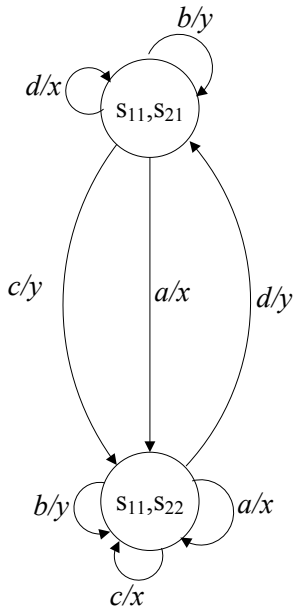

Fig. 4: The OPM of SCFSM $\mathscr{M}_1$

We now consider reducing the number of diagnostic candidates. For the tested test cases $TS_1 = \{tc_1,\cdots,tc_5\}$, if the intersection of traversed local transitions for each test case and the current conflict set is empty, we skip checking that. Otherwise, we check each test case and update the conflict set and diagnostic candidates. Table 7 gives the details of test cases in TS_1. Column "Status" indicates whether the test case requires checking or not. Column "New CS_{tr}" and "New DC_{tr}" gives the updated conflict set and diagnostic candidates after examination of the test case, respectively. For tc_1, the intersection is $\{t_{12},t_{21}\}$ and the corresponding diagnostic candidates of these two transitions are $Outputs(t_{12}) = \{x,c\}$, $OutStates(t_{12}) = \{(x,s_{12}),\ (c,s_{12})\}$, and $OutStates(t_{21}) = \{(x,s_{21}),\ (a,s_{21})\}$. If we convert the output of t_{12} to x instead of y in machine M_1, the expected output sequence of tc_1 for this mutant is xx, which is different with the observed output yy in Table 5, so this diagnostic candidate is eliminated. We can eliminate all diagnostic candidates

Table 4: Test sequences generated for SCFSM $\mathscr{M}_1$

Test Cases	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8	tc_9	tc_{10}	tc_{11}
Inputs	*rbc*	*rcb*	*rcd*	*rdc*	*raab*	*raad*	*rabb*	*rabd*	*racb*	*racd*	*radc*

Table 5: Output sequences of $\mathscr{M}_1$ and $\mathscr{M}_1'$

Test Cases	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6
Inputs	*rbc*	*rcb*	*rcd*	*rdc*	*raab*	*raad*
Local transitions	t_{12} t_{21},t_{12}	t_{21},t_{12} t_{12}	t_{21},t_{12} t_{24},t_{12}	t_{22} t_{21},t_{12}	$t_{11},t_{21},t_{14},t_{23}$, t_{11},t_{23},t_{13} t_{11},t_{23},t_{13} t_{12}	$t_{11},t_{21},t_{14},t_{23}$, t_{11},t_{23},t_{13} t_{11},t_{23},t_{13} t_{24},t_{12}
Expected	*yy*	*yy*	*yy*	*xy*	*xxy*	*xxy*
Observed	*yy*	*yy*	*yy*	*xy*	*xxy*	***xx*x**

Table 6: Initial diagnostic candidates of $\mathscr{M}_1'$

DC_{tr}	$Outputs$	$EndStates$	$OutStates$
t_{11}	$\{x\}$	$\{\}$	$\{(x,s_{11})\}$
t_{12}	$\{x,c\}$	$\{\}$	$\{(x,s_{12}),(c,s_{12})\}$
t_{13}	$\{\}$	$\{\}$	$\{\}$
t_{14}	$\{\}$	$\{\}$	$\{\}$
t_{21}	$\{\}$	$\{\}$	$\{(x,s_{21}),(a,s_{21})\}$
t_{23}	$\{\}$	$\{\}$	$\{(x,s_{21})\}$
t_{24}	$\{x,a\}$	$\{\}$	$\{(x,s_{22}),(a,s_{22})\}$

related to t_{12} and t_{21} in this way and update the conflict set and diagnostic candidates as shown in Table 7. Neither tc_2 nor tc_4 needs to be examined since the test cases do not contain any local transition related to the conflict set. tc_3 and tc_5 are examined in the same way as tc_1 except that the set of considered transitions are $\{t_{24}\}$ and $\{t_{11},t_{23}\}$, respectively.

Furthermore, we can discriminate the diagnostic candidates by examining the remaining untested test cases $TS_2 = \{tc_7,\cdots,tc_{11}\}$. For tc_7, a symptom is found between the specification and the implementation and the conflict set is updated by interacting with the new conflict set generated with the new observed symptom. Then, each diagnostic candidate is examined for discrimination as the process of TS_1. Eventually, the procedure terminates after t_8 is examined since only one diagnostic candidate is left. The details of examining each test case in TS_2 is shown in table 8.

For the final confirmation, the output of local transition t_{11} is converted to x instead of c, which leads to a mutant SCFSM. We do conformance testing on the implementation with this mutant specification. As the mutant SCFSM is equivalent to the implementation, we have identified that the implementation has a single faulty transition at t_{11}; upon input a, t_{11} outputs an external output x, instead of internal output c.

Faulty implementation 2 Consider another faulty implementation $\mathscr{M}_1''$ where t_{23} is converted to $s_{22}\,[M_2,c/b\rangle\, s_{22}$. We apply the test suite TS in Table 4 to $\mathscr{M}_1''$ until a symptom $(o_{tc_{5,2}},o'_{tc_{5,2}})$ is observed at the 2^{nd} input of the 5^{th} test case tc_5 where, according to the OPM, $\sigma_2\,[d/y\rangle\,\sigma_1$ have produced. The expected and observed outputs with traversed local transitions in the test cases that have been executed for detecting the symptom (shown in bold) are listed in Table 9.

Corresponding to the above symptom and the traversed local transitions, the following candidate set CS_{tr} is determined as: $CS_{tr} = \{t_{11},t_{13},t_{14},t_{21},t_{23}\}$. The whole diagnostic candidates that correspond to the above conflict set are shown in Table 10.

For the tested test cases $TS_1 = \{tc_1,\cdots,tc_4\}$, Table 11 gives the details of discriminating diagnostic candidates by examining each test case in TS_1. Since test cases tc_1, tc_2, and tc_3 include only one local transition t_{21} in the conflict set, the diagnostic candidate $EndStates(t_{21}) = \{s_{21}\}$ is examined. Upon input c, the machine M_2 moves from state s_{21} to s_{21}. As the output sequence of test case tc_3 is yx, different from the observed output sequence of the implementation, this candidate is removed. t_{21} is also removed from the conflict set since the other candidate sets related to t_{21} are all empty. We need not check the test case tc_4 because the intersection between its traversed local transitions and the current conflict set is empty.

We now discriminate the diagnostic candidates by examining the remaining untested test cases $TS_2 = \{tc_6,\cdots,tc_{11}\}$. The details of examining each test case in TS_2 is shown in table 12. Actually, we can observe that the conflict set and diagnostic candidates are reduced once only after examining test case tc_6. For avoiding unnecessary check of the remaining test cases, we can use different termination conditions such as the minimum threshold number of remaining diagnostic candidates. In this example, we can set this threshold to 3 and then only tc_6 is only to be tested.

The remaining three diagnostic candidates form three mutants and the corresponding OPMs are shown in Figure 5. Let $\mathscr{P}(\tilde{\mathscr{M}}_1)$, $\mathscr{P}(\tilde{\mathscr{M}}_2)$, and $\mathscr{P}(\tilde{\mathscr{M}}_3)$ be the OPMs with $Outputs(t_{13}) = \{y\}$, $Outputs(t_{23}) = \{b\}$, and

Table 7: Conflict set and diagnostic candidates updated by analyzing TS_1

Test Cases	Status	New CS_{tr}	New DC_{tr}
tc_1	Y	$\{t_{11},t_{23},t_{24}\}$	$Outputs(t_{11})=\{x\}$ $Outputs(t_{24})=\{x,a\}$ $OutStates(t_{11})=\{(x,s_{11})\}$ $OutStates(t_{23})=\{(x,s_{21})\}$ $OutStates(t_{24})=\{(x,s_{22}),(a,s_{22})\}$
tc_2	N	-	-
tc_3	Y	$\{t_{11},t_{23}\}$	$Outputs(t_{11})=\{x\}$ $OutStates(t_{11})=\{(x,s_{11})\}$ $OutStates(t_{23})=\{(x,s_{21})\}$
tc_4	N	-	-
tc_5	Y	$\{t_{11},t_{23}\}$	$Outputs(t_{11})=\{x\}$ $OutStates(t_{11})=\{(x,s_{11})\}$ $OutStates(t_{23})=\{(x,s_{21})\}$

Table 8: Conflict set and diagnostic candidates updated by analyzing TS_2

Test Cases	Local Transitions	Status	New CS_{tr}	New DC_{tr}
tc_7 $(rabb)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23}	Y	$\{t_{11},t_{23}\}$	$Outputs(t_{11})=\{x\}$ $OutStates(t_{11})=\{(x,s_{11})\}$ $OutStates(t_{23})=\{(x,s_{21})\}$
tc_8 $(rabd)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	Y	$\{t_{11}\}$	$Outputs(t_{11})=\{x\}$
tc_9 $(racb)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23}	N	-	-
tc_{10} $(racd)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	N	-	-
tc_{11} $(radc)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	N	-	-

Table 9: Output sequences of $\mathscr{M}_1$ and $\mathscr{M}_1''$

Test Cases	tc_1	tc_2	tc_3	tc_4	tc_5
Inputs	rbc	rcb	rcd	rdc	$raab$
Local transitions	t_{12} t_{21},t_{12}	t_{21},t_{12} t_{12}	t_{21},t_{12} t_{24},t_{12}	t_{22} t_{21},t_{12}	$t_{11},t_{21},t_{14},t_{23}$, t_{11},t_{23},t_{13} t_{11},t_{23},t_{13} t_{12}
Expected	yy	yy	yy	xy	xxy
Observed	yy	yy	yy	xy	$\mathbf{y}yy$

Table 10: Initial diagnostic candidates of $\mathscr{M}_1''$

DC_{tr}	$Outputs$	$EndStates$	$OutStates$
t_{11}	$\{y\}$	$\{s_{11}\}$	$\{(y,s_{11})\}$
t_{13}	$\{y\}$	$\{\}$	$\{(y,s_{12})\}$
t_{14}	$\{y\}$	$\{\}$	$\{(y,s_{12})\}$
t_{21}	$\{\}$	$\{s_{21}\}$	$\{\}$
t_{23}	$\{b\}$	$\{\}$	$\{(b,s_{21})\}$

$OutStates(t_{13})=\{(y,s_{12})\}$, respectively. Since $\mathscr{P}(\tilde{\mathscr{M}}_1)$ is equivalent to $\mathscr{P}(\tilde{\mathscr{M}}_2)$, we group these two mutants and consider the pair $(\mathscr{P}(\tilde{\mathscr{M}}_1,\tilde{\mathscr{M}}_2),\mathscr{P}(\tilde{\mathscr{M}}_3))$. An input sequence can be easily found that produces different output sequences from the two machines. If that sequence is $radd$, the corresponding output sequence is yyx for $\mathscr{P}(\tilde{\mathscr{M}}_1,\tilde{\mathscr{M}}_2)$, and yyy for $\mathscr{P}(\tilde{\mathscr{M}}_3)$. Since the corresponding output sequence from the faulty implementation, yyx is different from $\mathscr{P}(\tilde{\mathscr{M}}_3)$, the diagnostic candidate $OutStates(t_{13})=\{(y,s_{12})\}$ is discarded. The only remaining candidates are $Outputs(t_{13})=\{y\}$ and $Outputs(t_{23})=\{b\}$ which have the equivalent OPM.

For the final confirmation, conformance testing is done on the implementation with the OPM for the remaining grouped diagnostic candidates. The mutant

Table 11: Conflict set and diagnostic candidates updated by analyzing TS_1

Test Cases	Status	New CS_{tr}	New DC_{tr}
tc_1	Y	$\{t_{11},t_{13},t_{14},t_{21},t_{23}\}$	$Outputs(t_{11})=\{y\}$,$Outputs(t_{13})=\{y\}$ $Outputs(t_{14})=\{y\}$,$EndStates(t_{11})=\{s_{11}\}$ $Outputs(t_{23})=\{b\}$,$EndStates(t_{21})=\{s_{21}\}$ $OutStates(t_{11})=\{(y,s_{11})\}$ $OutStates(t_{13})=\{(y,s_{12})\}$ $OutStates(t_{14})=\{(y,s_{12})\}$ $OutStates(t_{23})=\{(b,s_{21})\}$
tc_2	Y	$\{t_{11},t_{13},t_{14},t_{21},t_{23}\}$	$Outputs(t_{11})=\{y\}$,$Outputs(t_{13})=\{y\}$ $Outputs(t_{14})=\{y\}$,$EndStates(t_{11})=\{s_{11}\}$ $Outputs(t_{23})=\{b\}$,$EndStates(t_{21})=\{s_{21}\}$ $OutStates(t_{11})=\{(y,s_{11})\}$ $OutStates(t_{13})=\{(y,s_{12})\}$ $OutStates(t_{14})=\{(y,s_{12})\}$ $OutStates(t_{23})=\{(b,s_{21})\}$
tc_3	Y	$\{t_{11},t_{13},t_{14},t_{23}\}$	$Outputs(t_{11})=\{y\}$,$Outputs(t_{13})=\{y\}$ $Outputs(t_{14})=\{y\}$,$Outputs(t_{23})=\{b\}$ $EndStates(t_{11})=\{s_{11}\}$ $OutStates(t_{11})=\{(y,s_{11})\}$ $OutStates(t_{13})=\{(y,s_{12})\}$ $OutStates(t_{14})=\{(y,s_{12})\}$ $OutStates(t_{23})=\{(b,s_{21})\}$
tc_4	N	-	-

Table 12: Conflict set and diagnostic candidates updated by analyzing TS_2

Test Cases	Local Transitions	Status	New CS_{tr}	New DC_{tr}
tc_6 $(raad)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	Y	$\{t_{13},t_{23}\}$	$Outputs(t_{13})=\{y\}$ $Outputs(t_{23})=\{b\}$ $OutStates(t_{13})=\{(y,s_{12})\}$
tc_7 $(rabb)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23}	Y	$\{t_{13},t_{23}\}$	$Outputs(t_{13})=\{y\}$ $Outputs(t_{23})=\{b\}$ $OutStates(t_{13})=\{(y,s_{12})\}$
tc_8 $(rabd)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	Y	$\{t_{13},t_{23}\}$	$Outputs(t_{13})=\{y\}$ $Outputs(t_{23})=\{b\}$ $OutStates(t_{13})=\{(y,s_{12})\}$
tc_9 $(racb)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23}	Y	$\{t_{13},t_{23}\}$	$Outputs(t_{13})=\{y\}$ $Outputs(t_{23})=\{b\}$ $OutStates(t_{13})=\{(y,s_{12})\}$
tc_{10} $(racd)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	Y	$\{t_{13},t_{23}\}$	$Outputs(t_{13})=\{y\}$ $Outputs(t_{23})=\{b\}$ $OutStates(t_{13})=\{(y,s_{12})\}$
tc_{11} $(radc)$	$t_{11},t_{12},t_{13},t_{14}$, t_{21},t_{23},t_{24}	Y	$\{t_{13},t_{23}\}$	$Outputs(t_{13})=\{y\}$ $Outputs(t_{23})=\{b\}$ $OutStates(t_{13})=\{(y,s_{12})\}$

SCFSM turns out to be equivalent to the implementation by that testing, so we cannot exactly locate the single fault. Accordingly, the following set of potential single faults is finally obtained: either t_{13} with faulty output y upon input a, or t_{23} with fault output b upon input c.

4.2 Complexity analysis

In order to easily calculate the complexity of each step of the diagnostic method, we assume that m is the number of component machines which have the same number of states n and the same number of output symbols q. We also let p be the number of external input symbols of a given SCFSM and k be the number of internal output symbols of

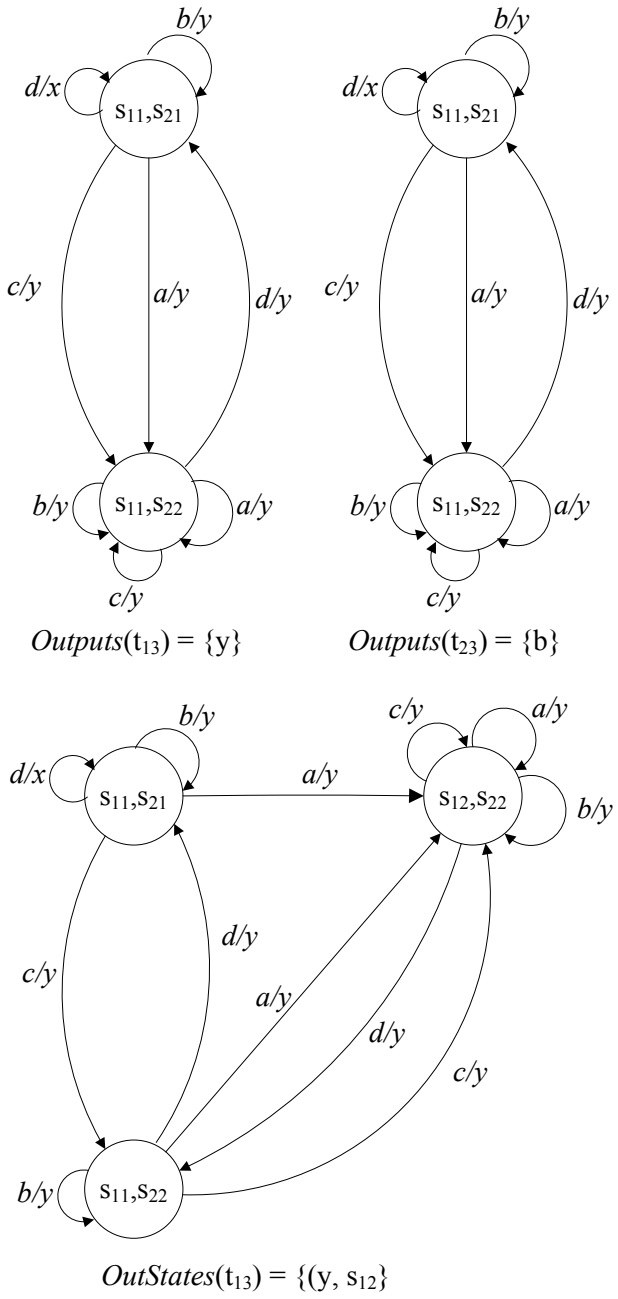

Fig. 5: The OPMs for diagnostic candidates

that SCFSM. We also let L_s be the length of test sequences and L_c be the number of inputs in the longest test case.

An OPM is obtained by performing reachability analysis. The complexity of building such a machine is the complexity of building its states and transitions. The number of stable global states is bounded by n^m, and the number of global transitions is bounded by pn^m. Each global transition may traverse all unstable global states in the worst case. The number of unstable global states is bounded by kn^m. Therefore the complexity of building an OPM is $O(n^m) + O(pn^m kn^m) = O(pkn^{2m})$.

The complexity of Step 1: The worst case is that we execute all test cases for finding a symptom. The complexity of this step is $O(L_s L_c)$.

The complexity of Step 2: In order to determine the local transitions that are traversed by the test case which generates the symptom, we insert the local transitions traversed in Step 1 while executing this test case against the OPM into a conflict set. This set would include, in the worst case, all the local transitions of the SCFSM. The complexity of this step is the number of all local transitions, that is $O(np)$.

The complexity of Step 3: In order to form the diagnostic candidates, we need to consider all possible single faults of each local transition in the conflict set. The number of possible single faults of a local transition is $nq-1$, so the complexity of generating all diagnostic candidates is $O((nq-1)np) = O(pqn^2)$. To check each diagnostic candidate whether it has the same output as the implementation for the test case, we need to trace the test case on the mutant machine. As the complexity of this check is $O(L_c kn^m)$, the complexity of this step is $O(pqn^2 L_c kn^m) = O(L_c pqkn^{m+2})$.

The complexity of Step 4: First, we consider the complexity of using test case in TS to eliminate diagnostic candidates. For each test case, in the worst case, we need to trace on the mutant machine to compare with the observed outputs for each diagnostic candidate in step 3, so this complexity is $O(L_s L_c pqkn^{m+2})$. Second, we consider the complexity of using OPMs to eliminate diagnostic candidates. The complexity of generating OPMs for all mutant machines is $O(pqn^2 pkn^{2m}) = O(p^2 qkn^{2m+2})$. To cross-check a pair of OPMs takes time $O(pn^m)$ and the total time for all mutants is $O(pn^m pqn^2) = O(p^2 qn^{m+2})$. The complexity of step 4 is thus $O(L_s L_c pqkn^{m+2}) + O(p^2 qkn^{2m+2}) + O(p^2 qn^{m+2}) = O(L_s L_c pqkn^{m+2}) + O(p^2 qkn^{2m+2})$.

The complexity of Step 5: Finally, the confirmation only considers one product machine, so it takes time $O(pn^{3m})$.

Overall Complexity: The overall complexity of the heuristic method is $O(pkn^{2m}) + O(L_s L_c) + O(np) + O(L_c pqkn^{m+2}) + O(L_s L_c pqkn^{m+2}) + O(p^2 qkn^{2m+2}) + O(pn^{3m}) = O(L_s L_c pqkn^{m+2}) + O(p^2 qkn^{2m+2}) + O(pn^{3m})$.

If we use Wp method for generating test sequence TS without extra states, TS is bounded by $O(pn^{3m})$ and TC is bounded by $O(2pn^m)$. In this case, the overall complexity is $O(L_s L_c pqkn^{m+2}) + O(p^2 qkn^{2m+2}) + O(pn^{3m}) = O(pn^{3m} 2pn^m pqkn^{m+2}) + O(p^2 qkn^{2m+2}) + O(pn^{3m}) = O(p^3 qkn^{5m+2})$.

5 Conclusions

This paper studies fault diagnosis for testing a communication protocol which can be modeled in an SCFSM. We propose a heuristic method that attempts to

locate a fault of the target implementation which has been detected by the given test cases under some assumptions. We have shown the condition required to enable fault localization in an SCFSM, and have discussed the possibility of precise locating the fault of an implementation which is assumed to have only one fault. According to the organization of component machines and their communications in the SCFSM, the proposed method can locate the fault exactly, which explains the root cause of faulty behavior in the implementation, or gives a set of all potential faults. We have also evaluated the proposed method with two case studies and complexity analysis. Especially we have developed a software tool in Java which performs the proposed fault diagnosis method to validate its reliability and completeness.

When testing an SCFSM based on its OPM, the main problem is the state explosion problem. Some studies are attempting to test local transitions individually without constructing the OPM. However, those methods usually require strict assumptions or may not guarantee the full fault coverage. Other studies are trying to use a system of Communicating Extended Finite-State Machines (SCEFSM) rather than an SCFSM, which allows specifications of data elements explicitly. The testing techniques proposed in this paper could be also applied to SCEFSM models after some adjustments. We also study how to reduce the state space required for the proposed fault diagnosis in order to guarantee its scalability.

Acknowledgement

This paper was supported by Research Fund, Kumoh National Institute of Technology.

References

[1] ITU, CCITT Specification and Descritpion Language (SDL), ITU-T, Recommendation Z.100 (2007).
[2] S. Fujiwara, G.V. Bochmann, F. Khendek, M. Amalou and A. Ghedamsi, Test Selection Based on Finite State Models, IEEE Trans on Software Eng, Vol.17, pp.591–603 (1991).
[3] C. Furse, P. Smith, C. Lo, Y.C. Chung, P. Pendayala and K. Nagoti, Spread spectrum sensors for critical fault location on live wire networks, Structural Control and Health Monitoring, Vol.12, pp.257–267 (2005).
[4] J. de Kleer and B.C. Williams, Diagnosing multiple faults, Artificial Intelligence, Vol.32, pp.97–130 (1987).
[5] R. Patton and P. Frank, Fault diagnosis in dynamic systems, theory and applications Prentice Hall: Englewood Cliffs (1989).
[6] D. Brand and P. Zafiropulo, On Communicating Finite-State Machines, J. ACM, Vol.30, pp.323–342 (1983).
[7] G.V. Bochmann, Conformance testing methodologies and architectures for OSI protocols, IEEE Computer Society Press: Los Alamitos, CA, USA, Section Finite state description of communication protocols, pp. 66–77 (1995).
[8] K. Özdemir and H. Ural, Protocol validation by simultaneous reachability analysis, Computer Communications, Vol.20, pp.772–788, (1997).
[9] Qi-Ping Yang and Tae-Hyong Kim, Test Generation for a System of Communicating Finite State Machines, International Journal of Engineering and Technology (IJET), Vol. 5, No. 4, pp. 3504–3513 (2013).
[10] D. Lee and M. Yannakakis, Principles and methods of testing finite state machines-a survey, Proceedings of the IEEE, Vol.84, pp.1090–1123 (1996).
[11] A. Petrenko, N. Yevtushenko and G.V. Bochmann, Fault models for testing in context, IFIP TC6/ 6.1 international conference on formal description techniques IX/protocol specification, testing and verification XVI on Formal description techniques IX, Chapman & Hall, Ltd., pp.163–178 (1996).
[12] A. Ghedamsi and G.V. Bochmann Test result analysis and diagnostics for finite state machines, Proceedings of the 12th International Conference on Distributed Computing Systems, pp. 244–251 (1992).
[13] D. Lee, and K. Sabnani, Reverse-engineering of communication protocols, Proceedings of the 1st International Conference on Network Protocols, pp. 208–216 (1993).
[14] M.R. Genesereth, The use of design descriptions in automated diagnosis, Artif. Intell., Vol.24, pp.411–436 (1984).
[15] A. Ghedamsi, G.V. Bochmann and R. Dssouli, Multiple fault diagnosis for finite state machines, Proceedings on the 12th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol.2, pp. 782–791 (1993).
[16] A. Ghedamsi, G.V. Bochmann and R. Dssouli, Diagnostic tests for communicating finite state machines, Annual International Phoenix Conference on Computers and Communications, pp. 254 –260 (1993).
[17] A. Ghedamsi, G.V. Bochmann and R. Dssouli, Diagnosis of single transition faults in communicating finite state machines, Proceedings on the 13th International Conference on Distributed Computing Systems, pp. 157–166 (1993).
[18] Z. Pap, G. Csopaki and S. Dibuz, On FSM-Based Fault Diagnosis, Lecture Notes in Computer Science, Vol. 3502, pp. 370–370 (2005).
[19] K. El-fakih and G.V. Bochmann, Locating a Faulty Machine in a System of Communicating Finite State Machines, Proceedings of the EEEL Workshop on Software Embedded Systems and Testing; pp.75–80 (1999).
[20] E.F. Moore, Gedanken Experiments on Sequential Machines, Automata Studies, Princeton U., pp.129–153 (1956).

Qi-Ping Yang received the PhD degree in Computer Engineering at Kumoh National Institute of Technology (KIT) in the Republic of Korea in 2012. He joined the Shanghai Research Center for Wireless Communications, China in 2013. His research interests are computer networks and protocol engineering, especially formal methods in development of network protocols and performance analysis of wireless networks.

Tae-Hyong Kim is a Professor of Computer Engineering at Kumoh National Institute of Technology (KIT) in the Republic of Korea. He received the PhD degree in Electric and Electronic Engineering at Yonsei University in the same country in 2001. He was a post-doctoral researcher at the University of Ottawa in 2002 and a visiting scholar at the University of California, Riverside in 2008. His main research interests are networking and intelligent processing, especially protocol engineering in formal methods, next-generation wireless networks, the internet of things, and data analysis with machine learning.