Fault Propagation by Equation Solving

Khaled El-Fakih[1] and Nina Yevtushenko[2]

[1] American University of Sharjah, PO Box 26666, Sharjah, UAE
kelfakih@aus.ac.ae
[2] Tomsk State University, 36 Lenin str., Tomsk, 634050, Russia
yevtushenko@elefot.tsu.ru

Abstract. In this paper we use equation solving for translating internal tests derived for a component embedded within a composite system into external tests defined over the external alphabets of the system. The composite system is represented as two communicating finite state machines (FSMs), an embedded component FSM, and a context FSM that models the remaining part of the system and which is assumed to be correctly implemented. Application example is given to demonstrate the steps of the method. The method can be adapted for test derivation for a system of two or more communicating FSMs.

1 Introduction

Several methods have been developed for testing a component embedded within a composite system [12]. Usually the composite system is represented as two communicating machines, an embedded component machine, and a context machine that models the remaining part of the system and that is assumed to be correctly implemented.

A number of test derivation methods have been proposed for testing in context when the system components are modeled as Finite State Machines (FSMs). Some of these methods [4, 15] return test suites that satisfy appropriate test purposes. However, these test suites are not complete, i.e. they do not detect all possible faulty implementations of the embedded component. Other methods [for example, 14] return complete but redundant test suites since they consider fault domains that include infeasible implementations that do not correspond to any possible implementation of the embedded FSM. Accordingly, in order to alleviate the problem of infeasible machines, tests can be derived directly from the embedded component machine as proposed in [18, 20]. In this case, a test suite is derived based on the largest set of permissible behaviors of the embedded component FSM that is a largest solution to an appropriate FSM equation. Usually, a largest solution is a nondeterministic FSM, and a test suite is derived w.r.t. the reduction relation. Hence the methods presented in [11, 19, 22] can be used for deriving corresponding test suites. However, tests generated by all of the above methods are given in the form of input/output sequences defined over the input/output alphabets of the embedded machine, i.e. over internal alphabets. These tests are then translated, using adhoc methods, into external tests defined over the external observable input alphabets of the system. The problem of translating internal tests into external ones is called the *fault propagation problem* and is known to have exponential complexity

D. de Frutos-Escrig and M. Núñez (Eds.): FORTE 2004, LNCS 3235, pp. 185-198, 2004.

In this paper we present an equation solving based approach for solving the fault propagation problem. The *equation solving* problem is to describe a behavior of a component of a system knowing the specifications of the other components and the specification of the whole system. In 1980, a first paper [2] (see also [16]) gives a solution to the problem for the case where the system behavior is described in terms of labeled transition systems (LTS). This work was later extended to the cases where the behavior of the components is described in CCS or CSP [17], by FSM [21, 26] or input/output automata [6, 13, 23]. Moreover, the applications of the equation solving problem were first considered in the context of the design of communication protocols [16]. Later it was recognized that equation solving this method could also be useful for the design of protocol converters in communication gateways [10, 13, 24], and for the selection of test cases for testing a module in a context [20]. Another application area of equation solving is the design of controllers for discrete event systems [1, 27].

We solve the fault propagation problem using equation solving as follows: Given the specifications of the context and embedded components, first, we derive the largest set of permissible behaviors of the embedded component FSM as the largest solution to an appropriate FSM equation. The FSM equation is solved using the automata based equation solving method presented in [3]. Then, we derive, using the method proposed in [19], from the largest FSM solution, internal tests for the embedded component FSM. These tests are derived w.r.t. the reduction relation since the largest solution is generally non-deterministic. The internal tests are then represented by an appropriate automaton. This automaton is used with the automaton that represents the context to solve an appropriate automata equation. External tests are then derived from the solution to the latter equation.

This paper is organized as follows. Section 2 includes necessary FSM and automata definitions and an overview of testing in context. Section 3 includes our method for translating internal tests by equation solving with a related application example. Section 4 concludes the paper.

2 Preliminaries

2.1 Finite State Machine

A *finite state machine*, often simply called a *machine*, is a quintuple $A=\langle S, I, O, T_A, s_0\rangle$, where S is a finite nonempty set of states with the initial state s_0, I and O are input and output alphabets, and $T_A \subseteq S \times I \times O \times S$ is a transition relation. We say that there is a transition from a state $s \in S$ to a state $s' \in S$ labeled with an I/O pair i/o, if and only if the 4-tuple (s, i, o, s') is in the transition relation T_A. FSM A is *observable* if for each triple $(i,s,o) \in I \times S \times O$ there exists at most one state $n \in S$ such that $(i,s,n,o) \in T_A$. An FSM A is called *deterministic*, if for each state $s \in S$ and each input $i \in I$ there exist at most one pair of output o and state s', such that $(s, i, o, s') \in T_A$. If A is not deterministic, then it is called *non-deterministic*. FSM $B = (S_B, I, O_B, T_B, s_0)$, where $S_B \subseteq S_A$ and $O_B \subseteq O_A$, is a *submachine* of FSM $A=\langle S_A, I, O_A, T_A, s_0\rangle$, if $\forall(s', i) \in S_B \times I$ $(T_B \subseteq T_A)$.

As usual, the transition relation T_A of the FSM A can be extended to sequences over the alphabet I. The extended relation is also denoted by T_A and is a subset of $S \times I^* \times O^* \times S$. By definition, for each state $s \in S$ of the FSM A the tuple $(s, \varepsilon, \varepsilon, s')$ is in the relation T_A. Given a tuple $(s,\alpha,\beta,s')\in T_A$, $\alpha\in I^*$, $\beta\in O^*$, and an input $i\in I$ and an output $o\in O$, the tuple $(s, \alpha i, \beta o, s'') \in T_A$, if and only if $(s', i, o, s'') \in T_A$. Given state s, an I/O sequence $i_1o_1...i_ko_k$, $i_1...i_k \in I^*$, $o_1...o_k \in I^*$, such that $(s, i_1...i_k, o_1...o_k, s') \in T_A$ is called a *trace* of A at state s. The set of all traces at state s is denoted $Tr_A(s)$. We denote Tr_A the set of traces at the initial state s_0, i.e. the set of traces of the FSM A, for short. As usual, to represent the set of traces of an FSM we use the notion of a finite automaton.

A *finite state automaton*, often called an *automaton* throughout the paper, is a quintuple $P=\langle S, V, \delta_P, s_0, F_P\rangle$, where S is a finite nonempty set of states with the initial state s_0 and a subset F_P of *final* (or *accepting*) states, V is an alphabet of actions, and $\delta_P\subseteq S\times V\times S$ is a transition relation. We say that there is a transition from a state s to a state s' labeled with an action v, if and only if the triple (s, v, s') is in the transition relation δ_P. The automaton P is called *deterministic,* if for each state $s\in S$ and any action $v\in V$ there exists at most one state s', such that $(s, v, s')\in \delta_P$. If P is not deterministic, then it is called *nondeterministic*. As usual, the transition relation δ_P of the automaton P is extended to sequences over the alphabet V. These sequences are usually called *traces* of the automaton P. Given a state s of the automaton P, the set of traces $L_P(s) = \{\alpha\in V^* | \exists s'\in F_P\ ((s, \alpha, s')\in \delta_P)\}$ is called the language *generated at the state s*. The language, generated by the automaton P at the initial state, is called the *language generated by the automaton P* and is denoted by L_P, for short.

Given an FSM $A=\langle S, I, O, T_A, s_0\rangle$, we derive the automaton $Aut(A)=\langle S\cup(S\times I), I\cup O, \delta_P, s_0, F_P{=}S\rangle$ [26] with the language that coincides with the set of all traces of the FSM. Each transition (s_i, i, o, s_j) in T_A is represented by the two consecutive transitions $(s_i, i, (s_i,i))$ and $((s_i,i), o, s_j)$ in P. That is the automaton is obtained from the original FSM by replacing each edge labeled by i/o with an edge labeled by i, followed by a new non-accepting state, followed by an edge labeled by o. All original states are accepting. If the FSM A is observable then the automaton $Aut(A)$ is known to be deterministic.

Let $A = \langle S, I, O, T_A , s_0\rangle$ and $B=\langle Q, I, O, T_B, q_0\rangle$ be two FSMs, state q of FSM B is said to be a *reduction* of state s of FSM $A = \langle S, I, O, T_A , s_0\rangle$ (written $q \le s$), if $Tr_B(q) \subseteq Tr_A(s)$. States q and s is said to be *equivalent* (written $q \cong s$) if $q \le s$ and $s \le q$; otherwise, states q and s are not equivalent. Moreover, B is a *reduction* of FSM A, if $Tr_B \subseteq Tr_A$. If $Tr_B = Tr_A$ then FSMs A and B are *equivalent*, written as $A \cong B$. For complete deterministic FSMs the reduction and the equivalence relations coincide.

A non-deterministic automaton can be converted into a deterministic automaton with the same language [9]. For this reason, we consider only observable FSMs. If an FSM is non-observable then it can be transformed into an equivalent observable FSM by determinizing the corresponding automaton. Given a deterministic automaton $P=\langle R, I\cup O, \delta_P, r_0, F_P\rangle$ with the set of traces that is a subset of $(IO)^*$, P can be converted into an observable FSM $FSM(P)$ over input alphabet I and O if for each trace $\alpha io \in Tr_P$ the prefix α also is a trace of the automaton P. States of the $FSM(P)$ are the initial state and all accepting states of the automaton P. Let $P=\langle R, V, \delta_P, r_0, F_P\rangle$ and $R=\langle Q, W, \delta_R, q_0, F_R\rangle$ be two automata. We further describe some operations over finite automata that will be used throughout the paper.

Intersection. If alphabets V and W intersect then the *intersection* $P\cap R$ of automata P and R is the largest connected sub-machine of the automaton $\langle S \times Q, V\cap W, \delta,$

$(s_0, q_0), F_P \times F_R\rangle$. Given an action $a \in V \cap W$ and a state (r, q), there is a transition at the state (r, q) labeled with a, if and only if there are transitions at states s and q labeled with a, i.e. $\delta = \{((r, q), a, (r', q')) \mid (r, a, r') \in \delta_P \wedge (q, a, q') \in \delta_R\}$. The set of traces of the automaton $P \cap R$ accepts the intersection of the sets Tr_P and Tr_R. If V and W are disjoint then intersection of P and R is not defined, since the alphabet of an automaton cannot be empty.

Restriction. Given a sequence α over alphabet V and an alphabet U, the *U-restriction* of α is obtained by deleting from α all symbols that are not in U. If there are no symbols from U in α then the U-restriction of α is equal to the empty sequence ε. Given an automaton P and an alphabet U, the *U-restriction* of P is the deterministic automaton $P_{\downarrow U}$ that is equivalent to the automaton $\langle S, U, \delta, s_0, F_P\rangle$, where $\delta = \{(r, u, r') \mid \exists\, \alpha \in V^* \ (\exists\, (r, \alpha, r') \in \delta_P \ \& \ (\alpha_{\downarrow U} = u))\}$. The set of traces of the automaton $P_{\downarrow U}$ is the set of U-restrictions of all traces of the automaton P, i.e. is the set $\{\alpha \in U^* \mid \exists \beta \in L(P)\ (\alpha = \beta_{\downarrow U})\}$.

Expansion. Given an alphabet U, the *U-expansion* of P is the automaton $P_{\uparrow U} = \langle R, V \cup U, \delta, r_0, F_P\rangle$, where $\delta = \delta_P \cup \{(r, u, r) \mid r \in R \ \& \ u \in U \setminus V\}$. The automaton $P_{\uparrow U}$ is obtained from P by adding at each state a loop transition labeled with each action of the alphabet $U \setminus V$. If U is a subset of V then the automaton $P_{\uparrow U}$ coincides with the automaton P. Automaton $P_{\uparrow U}$ has the set of traces $\{\alpha \in (V \cup U)^* \mid \exists\, \beta \in Tr_P\ (\alpha_{\downarrow V} = \beta)\}$.

2.2 Parallel Composition of FSMs

We consider a system of two Communicating FSMs of the context FSM *Context*=$\langle S, I \cup V, O \cup U, T_A, s_0\rangle$ and of the embedded FSM *Emb*=$\langle T, U, V, T_B, t_0\rangle$, as shown in Fig. 1. above. The alphabets I and O represent the *external* inputs and outputs of the system, while the alphabets V and U represent the *internal* interactions between the two machines. As usual, for the sake of simplicity, we assume that the sets I, O, V, U are pair-wise disjoint. The system produces an output in response to each input. We assume that the system at hand has at most one message in transit, i.e. the next external input is submitted to the system only after it produces an external output to the previous input. Under these assumptions, the collective behavior of the two communicating FSMs *Context* and *Emb* can be described by an FSM as follows [26]:

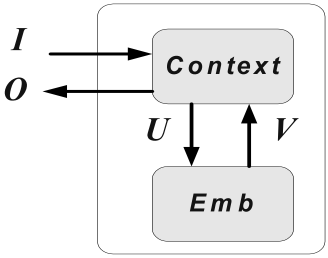


Fig. 1. Parallel Composition of two FSMs

First, we transform the two FSMs *Context* and *Emb* into the corresponding automata *Aut*(*Context*) and *Aut*(*Emb*). Then, we derive the automaton *Aut*(*Context*) $\Diamond$ *Aut*(*Emb*) = $(Aut(Context) \cap Aut(Emb)_{\uparrow I \cup O})_{\downarrow I \cup O}$. Then, we intersect *Aut*(*A*) $\Diamond$ *Aut*(*B*) with the automaton of the chaos FSM defined over the alphabet $I \cup O$. The obtained automaton is shown to have an FSM language over the alphabets *I* and *O* [26]. The FSM corresponding to the obtained automaton is called the *parallel composition* of FSMs *Context* and *Emb*, and is written as *Context* $\Diamond$ *Emb*. In this paper, the context and the embedded FSMs are assumed to be complete and deterministic.

As an example, consider the two FSMs shown in Figures 4.1 and 7, respectively. The set of external inputs is $I = \{x_1, x_2\}$, the set of external outputs is $O = \{o_1, o_2, o_3\}$, the sets of internal interactions are $V=\{v_1, v_2\}$ and $U=\{u_1, u_2\}$. The corresponding composed FSM is shown as the specification FSM *Spec* in Fig 3.1.

2.3 Testing in Context

Testing in context deals with the generation of tests for implementations of the embedded machine *Emb* assuming that the implementation of the context machine is fault free [20, 18]. Moreover, usually it is assumed that the implementation system has been tested w.r.t. livelocks, for example, as proposed in [7], and found to be livelock free; thus, the system under test *Context* $\Diamond$ *Imp*, where *Imp* is a complete deterministic implementation of *Emb*, is assumed to be complete and deterministic. Under these assumptions embedded implementations are tested w.r.t. external equivalence or equivalence in context.

Given complete deterministic FSMs $Context=\langle S, I \cup V, O \cup U, T_A, s_0\rangle$ and embedded FSM $Emb = \langle T, U, V, T_B, t_0\rangle$, let the composed FSM *Context* $\Diamond$ *Emb* be also deterministic and complete. FSM $Imp = \langle Q, U, V, T_B, q_0\rangle$ is said to be *externally equivalent* (or *equivalent in the context*) to the embedded FSM *Emb* if the FSMs $Context \Diamond Emb \cong Context \Diamond Imp$ are equivalent, i.e. $Context \Diamond Emb \cong Context \Diamond Imp$.

A test suite *TS* w.r.t. external equivalence is a set of external input sequences defined over the alphabet *I*. Given a set $\Re$ of possible implementations of the embedded machine *Emb*, called the *fault domain* of *Emb*, a test suite *TS* is said to be *complete* w.r.t. the fault domain $\Re$ if and only if for each FSM *Imp* of $\Re$ such that *Context* $\Diamond$ *Imp* is not equivalent to *Context* $\Diamond$ *Emb*, there exists a test case in *TS* that eliminates *Imp*.

Several fault models have been proposed for testing an embedded FSM w.r.t. external equivalence [18]. For example, one can explicitly enumerate all possible implementations of the embedded component if the number of these implementations is not huge. When the fault domain is huge, one can use for test derivation the methods that generate tests without the explicit enumeration of the fault domain machines. For instance, the W-method [5] and its modifications, namely the Wp, UIOv, and the HIS methods, can be used if an upper bound on the number of states of an implementation system is known. In this case, tests are derived without taking into account the fact that the context is assumed to be fault free. Thus, the considered fault domain includes all possible implementations of the embedded and context machines. Therefore, in this case, the derived tests are known to be redundant and an optimization procedure such as that proposed in [25] is needed to reduce redundant tests. As an alternative approach, one can consider as a fault domain for the embedded machine, the set of all submachines of an appropriate nondeterministic FSM. This non-deterministic FSM is combined with the context machine and a test suite is then derived from the obtained Mutation Machine (MM) [28]. However, a mutation machine is known to have infeasible machines that do not correspond to any possible implementation system. The number of these machines can still be large even if we decrease their number by using several mutation machines as done in [8, 7]. In order to avoid fault domains with infeasible machines, one can use as a fault domain for the embedded machine the largest solution M to the equation $Context \Diamond X \cong Context \Diamond Emb$. A complete deterministic implementation FSM Imp is not externally equivalent to the specification embedded machine Emb if and only if Imp is not a reduction of M [20]. Therefore, we can derive a complete test suite from a largest solution M w.r.t. the fault domain $\Re$ and the reduction relation. However, in this case, the sequences of an obtained test suite are defined over the internal alphabets U and V and thus, have to be translated to tests defined over the external input alphabets (i.e. external tests). In [20] some adhoc recommendations for such translation have been proposed. In the following sections, we propose a rigorous equation solving based approach for translating internal tests to external ones.

3 Translating Internal Tests by Equation Solving

In this section we use equation solving for translating internal tests of an embedded machine into external ones defined over the external input alphabets of the system. First, in subsection 3.1, we present a method for solving an FSM equation [3, 21], assuming that the internal interactions between the system components are unobservable, then, in subsection 3.2, we propose a method for translating internal tests by solving an appropriate automata equation.

3.1 Solving an FSM Equation

We consider the equation $Context \Diamond X \cong Spec$, where $Spec = Context \Diamond Emb$. We recall that this equation has a largest solution M that contains all possible implementations that are externally equivalent to the embedded component Emb.

An FSM B over the alphabets U and V is called a *solution* to the equation $Context \Diamond X \cong Spec$ if $Context \Diamond B \cong Spec$. A complete solution M is called *largest* if it includes as its reductions all complete solutions to the equation $Context \Diamond X \cong Spec$, i.e. each solution to the equation is a reduction of the largest solution. In order to derive M, we use the methods proposed in [3, 21].

We replace the FSMs $Context$ and $Spec$ with the corresponding automata $Aut(Context)$ and $Aut(Spec)$ and solve the automata equation $Aut(Context) \Diamond X \cong Aut(Spec)$. Since we are interested in an FSM solution, we derive the largest automaton with the set of traces that is a subset of $(UV)^*$. Thus, we derive as a largest solution the largest reduction of the automaton $Aut(Chaos\text{-}UV)$, where $Chaos\text{-}UV = \langle R, U, V, T_{Ch}, r_0\rangle$ is the chaos FSM over the alphabets U and V.

Similar to [3] we first derive the automaton $\Lambda(Aut(Context), Aut(Chaos\text{-}UV), Aut(Spec)) = Aut(Context) \cap Aut(Chaos\text{-}UV)_{\uparrow I \cup O} \cap Aut(Spec)_{\uparrow U \cup V}$. A state (s,r,q) of the automaton is called *forbidden* if the external restriction of the language generated at state (s,r,q) is not equal to the language generated at state q of the specification

Spec. We restrict the automaton to the alphabets U and V of the embedded FSM, replace each subset that has a forbidden state with the designated state '*FAIL*', and then convert the obtained automaton into an FSM defined over the alphabets U and V. Each undefined transition in the obtained FSM is specified as a transition to the DNC (don't care or chaos) state, and the '*FAIL*' state and its incoming and outgoing transitions are deleted. The DNC state accepts all input/output sequences of the set $(UV)^*$. The largest complete submachine of the obtained FSM (if it exists) is the largest complete solution M to the FSM equation $Context \Diamond X \cong Spec$. In our case, M always exists since the equation has a solution, in particular the embedded component FSM *Emb* is a solution to the equation. In the following subsection we illustrate the above steps through an application example.

3.2 Translating Internal Tests

Given the largest complete solution M to the equation $Context \Diamond X \cong Spec$, consider a complete test suite TS derived from M w.r.t. the fault domain $\mathfrak{R}$ and the reduction relation. The sequences of TS are defined over the internal alphabet U. A test suite TS is said to be *complete* if for each implementation FSM $Imp \in \mathfrak{R}$ that is not a reduction of M, there exists a test case $\alpha \in TS$ s.t. the set of output responses of the FSM *Imp* to α is not a subset of the set of output responses of M to α. Since the FSM *Imp* is deterministic, the latter means that if the implementation FSM *Imp* is not a reduction of M, then there exists an input sequence $\alpha \in TS$ s.t. the trace of FSM *Imp* with the U-restriction α does not intersect the set of traces of M with the U-restriction α. Based on the complete test suite TS, we derive the set $\widehat{TS}$ of input/output sequences of the set $(UV)^*$ such that the set $\widehat{TS}$ intersects the set of traces of each possible implementation $Imp \in \mathfrak{R}$ that is not a reduction of M. For each non-empty prefix $u_1v_1 \ldots u_kv_k$ of each trace of the set $\{\beta : \beta \in \text{Traces of } M \;\&\; \beta_{\downarrow U} \in TS\}$ we include into the set $\widehat{TS}$ the set of sequences $\{u_1v_1 \ldots u_kv : u_1v_1 \ldots u_kv \notin \text{Traces of } M\}$ (if it exists).

Proposition 1. Given a largest solution M to the equation $Context \Diamond X \cong Spec$. An implementation FSM $Imp \in \mathfrak{R}$ is not a reduction of M if and only if the set of traces of *Imp* intersects the set $\widehat{TS}$.

According to the above proposition, if an implementation FSM $Imp \in \mathfrak{R}$ has a trace of the set $\widehat{TS}$ then *Imp* is not externally equivalent to the embedded machine *Emb*. The traces of the set $\widehat{TS}$ can be represented as traces of an automaton $Aut\widehat{TS}$, where each trace of $\widehat{TS}$ leads to the designated *Trap* state of $Aut\widehat{TS}$. The state *Trap* is the only accepting state of the automaton and the language generated at the state *Trap* is the set $(UV)^*$. We obtain a test generator that generates all sequences over $(IO)^*$ such that for each $Imp \in \mathfrak{R}$ that is not externally equivalent to *Emb*, the generator induces in *Imp* at least one trace of the set $\widehat{TS}$ in order to detect that *Imp* is not a reduction (i.e. *Imp* is a nonconforming implementation of *Emb*) of M,. Due to the test architecture shown in Figure 2, the generator is obtained by solving the equation $Aut(Context) \Diamond X \cong Aut\widehat{TS}$.

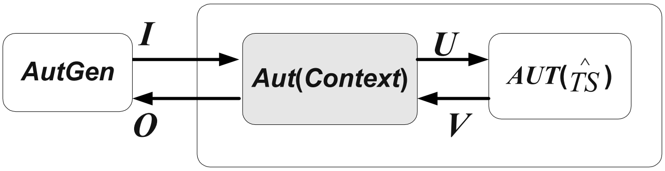
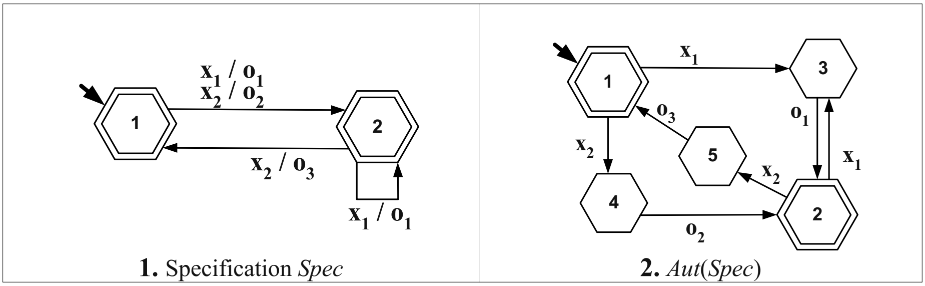


Fig. 2. Test Architecture

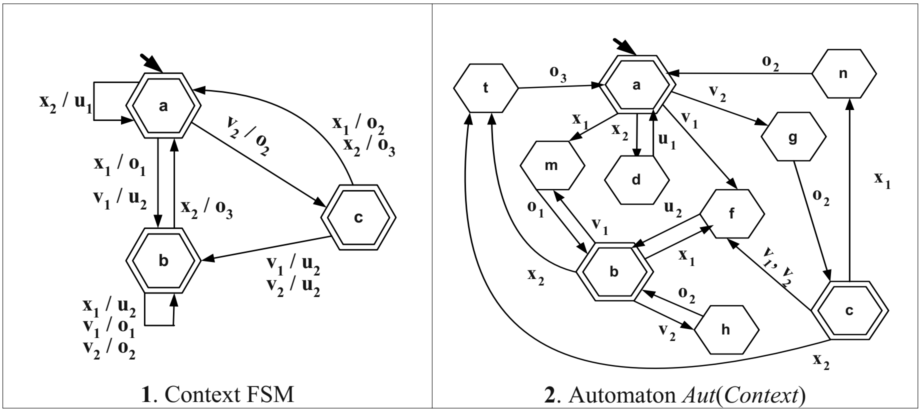
As an application example, consider the specification FSM *Spec*, shown in Fig 3.1, defined over the inputs $I = \{x_1, x_2\}$ and outputs $O = \{o_1, o_2, o_3\}$. The corresponding automaton *Aut*(*Spec*) is shown in Fig 3.2. Moreover, consider the context FSM *Context*, shown in Figures 4.1, defined over the external inputs $I = \{x_1, x_2\}$, external outputs $O = \{o_1, o_2, o_3\}$, internal inputs $V=\{v_1, v_2\}$ and internal outputs $U=\{u_1, u_2\}$. The automaton *Aut*(*Context*) that corresponds to *Context* is shown in Fig 4.2. Accepting states of both automata are shown by double lines. We are required to solve the FSM equation $Context \Diamond X \cong Spec$ and obtain its largest solution *M*. This solution is defined over FSMs, thus it is a submachine of automaton *Chaos-UV* shown in Fig. 5. In order to solve the equation, we combine the automata *Aut*(*Spec*) and *Aut*(*Context*) with *Aut*(*Chaos-UV*) and obtain the automaton Λ(*Aut*(*Context*), *Aut*(*Spec*), *Aut*(*Chaos-UV*)) shown in Fig. 6. Each state (*s*, *j*, *Q*) where the external restriction of the set of traces at the state does not coincide with the set of traces at state *j* of the specification, i.e. states h3A, m4A, n3A, c2A, and g4A are declared as the designated state '*FAIL*'. We restrict the automaton onto the alphabet $\{u_1, u_2, v_1, v_2\}$ of the solution. Each subset of states of the restricted automaton that includes the '*FAIL*' state is designated as the '*FAIL*' state. We add the DNC state for the transitions (f4A, u_1) and (f3A, u_1) since there are no transitions from these states under the input u_1, delete the '*FAIL*' state with its incoming transitions and obtain the largest FSM solution *M* shown in Fig 7. Then, we derive from *M* the complete internal test suite $TS = \{u_1\, u_2\, u_2,\ u_2\, u_2\}$ w.r.t. all FSMs with at most two states using the HIS method [19]. The set of all traces of the FSM *M* with the *U*-restriction in the set *TS* is $\{u_1/v_1\ u_2/v_2\ u_2/v_1,\ u_2/v_1\ u_2/v_1\}$. According to Proposition 1, if an implementation of the embedded component has one of the sequences of the set $\widehat{TS} = \{u_1/v_2;\ u_1/v_1\ u_2/v_1;\ u_1/v_1\ u_2/v_2\ u_2/v_2;\ u_2/v_2;\ u_2/v_1\ u_2/v_2\}$ then this implementation is not externally equivalent to the specification of the embedded component FSM. We represent the sequences of $\widehat{TS}$ as the finite automaton $Aut\widehat{TS}$ shown in Fig 8 and solve the equation $Aut(Context) \Diamond X \cong Aut\widehat{TS}$.



1. Specification *Spec*

2. *Aut*(*Spec*)

Fig. 3. Specification



1. Context FSM

2. Automaton *Aut*(*Context*)

Fig. 4. Context FSM and Automaton *Aut*(*Context*)

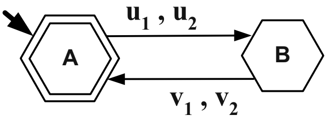


Fig. 5. Automaton *Aut*(*Chaos-UV*)

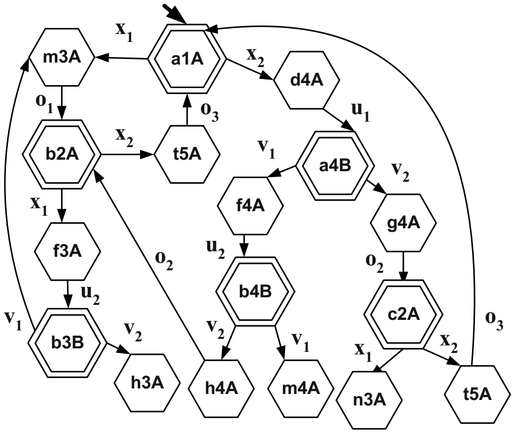
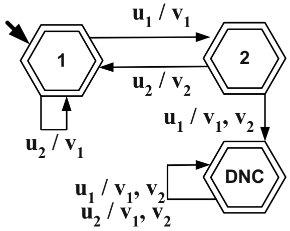


Fig. 6. Automaton Λ(*Aut*(*Context*), *Aut*(*Spec*), *Aut*(*Chaos-UV*))



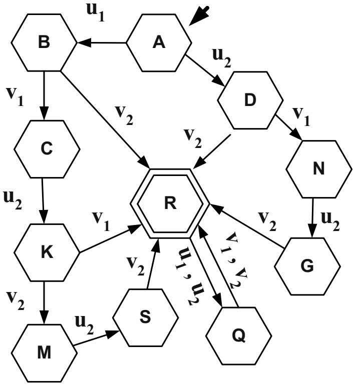


Fig. 7. Largest FSM Solution

Fig. 8. Automaton *Aut* $\widehat{TS}$

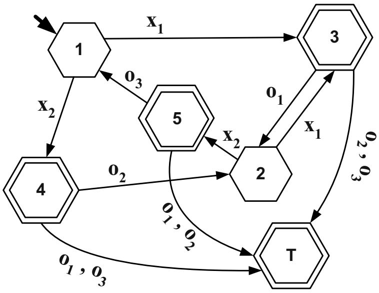


Fig. 9. *Aut*($\overline{Spec}$)

When no access to the internal interactions is available, some faults of the embedded component become latent. To illustrate latent faults consider a faulty implementation *Imp* of the embedded component that has the trace u_1v_2 instead of u_1v_1. The trace u_1v_2 in the faulty implementation can be induced by the external input x_2. However, when the internal outputs are unobservable, the composed system *Context* ◊ *Imp* has the expected output response o_2 to the input x_2. In order to detect that *Imp* has the wrong trace u_1v_2, we have to apply after x_2 the input x_1. In this case the composed system will reply with the unexpected output o_2. Therefore, since the internal channels are unobservable, it is insufficient to have a generator that induces at least one forbidden trace of each non-conforming implementation of the embedded machine. The consequences of the fault have to be externally observable, i.e. have to be propagated to the external environment. In other words, a test generator has to be a reduction of the complement of the specification machine.

Thus, when solving the equation *Aut*(*Context*) ◊ $X \cong Aut\widehat{TS}$ we look for a solution that is a reduction of the complement of the specification machine $\overline{Spec}$. This is done since an internal fault is detected if and only if an unexpected output is produced to some external test case. The following statement holds.

Proposition 2. Given a solution *Gen* to the automaton equation *Aut*(*Context*) ◊ $X \cong Aut\widehat{TS}$, let *AutGen* be a reduction of the automaton *Aut*($\overline{Spec}$) and have a finite number of traces. The *I*-restriction of the traces of the automaton *AutGen* is a complete (external) test suite w.r.t. the fault domain $\mathfrak{R}$ and the external equivalence relation.

In our application example we are interested in a largest solution to the equation *Aut*(*Context*) ◊ $X \cong Aut\widehat{TS}$ that is a reduction of *Aut*($\overline{Spec}$) of Fig 9. In order to obtain this solution, we derive the automaton Λ(*Aut*(*Context*), *Aut*($\overline{Spec}$), $Aut\widehat{TS}$) shown in Fig. 10. The language of the *I*-restriction of the obtained largest solution is $\mathrm{x}_1(\mathrm{x}_2\mathrm{x}_1)^*\mathrm{x}_1\mathrm{x}_1$, $(\mathrm{x}_1(\mathrm{x}_2\mathrm{x}_1)^*\mathrm{x}_1(\mathrm{x}_2\mathrm{x}_1)^*$, $\mathrm{x}_2\mathrm{x}_2(\mathrm{x}_1\mathrm{x}_2)^*\mathrm{x}_1$, $\mathrm{x}_2\mathrm{x}_1$, $\mathrm{x}_2\mathrm{x}_2\mathrm{x}_2^*(\mathrm{x}_1\mathrm{x}_2)^*$, and correspondingly a reduction of this largest solution that has the sequences $TS = \{\mathrm{x}_1\mathrm{x}_1\mathrm{x}_1, \mathrm{x}_2\mathrm{x}_2\mathrm{x}_1, \mathrm{x}_2\mathrm{x}_1, \mathrm{x}_2\mathrm{x}_2\mathrm{x}_2\}$ (these sequences are the labels of all simple paths from the initial state of the reduction to a final state that includes the trap state *R*) is also a solution to the equation *Aut*(*Context*) ◊ $X \cong Aut\widehat{TS}$ .The external test suite *TS* is a complete test suite for the embedded component w.r.t. the external equivalence relation.

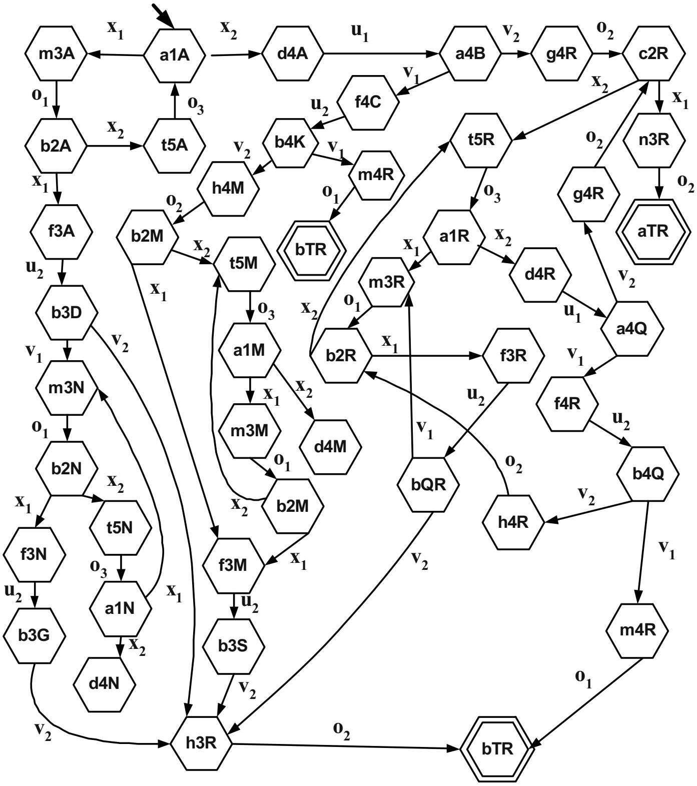


Fig. 10. Automaton $\Lambda(Aut(Context), Aut(\overline{Spec}), Aut\,\widehat{TS})$

A Summary of the Fault Propagation Approach:

Input: A deterministic $Context = \langle S, I \cup V, O \cup U, T_A, s_0 \rangle$, a deterministic embedded component $Emb = \langle T, U, V, T_B, t_0 \rangle$, and a deterministic specification $Spec = \langle Q, I, O, T_{Spec}, q_0 \rangle = Context \Diamond Emb$, and the fault domain $\mathfrak{R}$ of the embedded component *Emb*.

Output: A complete external test suite defined over the external alphabet *I* for testing the embedded component *Emb*. The test suite is complete w.r.t. the fault model $<Context \Diamond Emb, \cong, Context \Diamond \mathfrak{R}>$.

Step 1: Derive a largest FSM solution *M* to the equation $Context \Diamond X = Spec$.

Step 2: Derive a complete test suite *TS* w.r.t. the fault model $<M, \leq, \mathfrak{R}>$. Then, derive from *TS* the set of sequences $\widehat{TS}$, over the alphabet $(UV)^*$, that intersects the set of traces of each possible implementation *Imp* in $\mathfrak{R}$ that is not a reduction of *M*. Represent the sequences of $\widehat{TS}$ by the automaton $Aut\,\widehat{TS}$.

Step 3: Derive a largest solution *AutGen* to the automata equation $Aut(Context) \Diamond X = Aut\widehat{TS}$ that is a reduction of the automaton $Aut(\overline{Spec})$, where $\overline{Spec}$ is the complement of *Spec.*

Step 4: Derive an external test suite from the automaton *AutGen* by projecting *AutGen* on the external alphabet *I* and by considering in the obtained automaton all simple paths from the initial state to each final state that includes the trap state *R*. The labels of these simple paths form a complete (external) test suite for the embedded component *Emb* w.r.t. the external equivalence relation.

4 Conclusion

In this paper we presented an equation solving based approach for translating internal tests derived for a component embedded within a composite system into external tests defined over the external alphabets of the system. The system is represented as two communicating finite state machines, an embedded component machine and a context machine that represents the remaining part of the system. The context is assumed to be fault free. The method can be adapted for generating tests for a system of two or more communicating finite state machines. This is part of our current research work.

References

[1] G. Barrett and S. Lafortune, “Bisimulation, the supervisory control problem, and strong model matching for finite ftate machines”, Discrete Event Dynamic Systems: Theory and Application, 8(4), 377-429, 1998.

[2] G. v. Bochmann and P. M. Merlin, “On the construction of communication protocols”. ICCC (1980) 371-378, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ. (1981).

[3] S. Buffalov, K. El-Fakih, N. Yevtushenko, & G.v. Bochmann, “Progressive solutions to a parallel automata equation”. In Proc. of the IFIP 23rd International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2003), Berlin, Germany, Published as LNCS 2767, pp.367-382, 2003.

[4] Cavalli, , D. Lee, D., C. Rinderknecht, , and F. Zaidi, “Hit-or-Jump: An algorithm for embedded testing with applications to IN services”. Proceedings of Joint Inter. Conf. FORTE/PSTV99, pp: 41-58, 1999.

[5] T. S. Chow, “Test design modeled by finite-state machines,” IEEE Trans. SE, vol. 4, no.3, pp. 178-187, 1978.

[6] J. Drissi and G. v. Bochmann, Submodule Construction for systems of I/O Automata. Technical Report #1133, DIRO, Universite' de Montreal, Canada, 1999.

[7] K. El-Fakih, V. Trenkaev, N. Spitsyna, N. Yevtushenko, “FSM Based Interoperability Testing Methods”, in Proc. of the IFIP 16th International Conference on Testing of Communicating Systems, Oxford, U.K., Published as LNCS 2978, pp. 60-75, 2004.

[8] K. El-Fakih, S. Prokopenko, N. Yevtushenko, and G. v. Bochmann, “Fault diagnosis in extended finite state machines”, in Proc. of the IFIP 15th International Conference on Testing of Communicating Systems, France, published as LCNC 2644, pp. 197-210, 2003.

[9] J. E. Hopcroft, and J. D. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley, N.Y., 1979.

[10] S. G. H. Kelekar, Synthesis of protocols and protocol converters using the submodule construction approach. In. A. Danthine et al, editors, Protocol Specification, Testing, and Verification- PSTV XIII, 1994.

[11] R. Hierons and H. Ural, "Concerning the ordering of adaptive test sequences", In Proc. of the IFIP 23rd International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2003), Berlin, Germany, Published as LNCS 2767, pp.289-302, 2003.

[12] Information technology. "Open systems interaction. Conformance testing methodology and framework". International standard IS-9646, 1991.

[13] R. Kumar, S. Nelvagal, and S. I. Marcus. "A discrete event systems approach for protocol conversion", Discrete Event Dynamical Systems: Theory and Applications, 7(3) 295-315, 1997.

[14] L. P Lima, , and A. R. Cavalli, "A pragmatic approach to generating test sequences for embedded systems". Proceedings of 10th IWTCS, pp: 125-140, 1997.

[15] D. Lee, K. Sabnani, D. M. Kristol, and S. Paul, "Conformance testing of protocols specified as communicating finite state machines - a guided random walk based approach". IEEE Transactions on Communications, 44(5): 631-640, 1996.

[16] P. Merlin and G. v. Bochman. On the construction of submodule specifications and communication protocols, ACM Trans. On Programming Languages and Systems. 5(1) 1-25, 1983.

[17] J. Parrow, Submodule construction as equation solving in CCS, Theoretical Computer Science, 68, 1989.

[18] Petrenko, N. Yevtushenko, G. v. Bochmann. "Fault models for testing in context", FORTE '96.

[19] Petrenko, N. Yevtushenko, and G. v. Bochmann. "Testing deterministic implementations from their nondeterministic specifications". Proceedings of the IFIP 9th International Workshop on Testing of Communicating Systems, Germany, pp. 125-140, 1996.

[20] Petrenko, N. Yevtushenko, G. v. Bochmann, and R. Dssouli, "Testing in context: framework and test derivation", Computer communications, Vol. 19, pp. 1236-1249, 1996.

[21] Petrenko and N. Yevtushenko, Solving asynchronous equations. In S. Bukowski, A. Cavalli, and E. Najm, editors, Formal Description Techniques and Protocol Specification, Testing, and Verification- FORTE XI/PSTVXVIII '98, Chapman-Hall, 231-247, 1998.

[22] Petrenko, N. Yevtushenko, A. Lebedev, and A. Das, "Nondeterministic State Machines in Protocol Conformance Testing," Proc. of the IFIP 6th IWPTS, France, pp. 363-378, 1993.

[23] H. Qin and P. Lewis, Factorisation of finite state machines under strong and observational equivalences, Journal of Formal Aspects of Computing, 3, 284- 307, 1991.

[24] Z. Tao, G. v. Bochmann and R. Dssouli, A formal method for synthesizing optimized protocol converters and its application to mobile data networks. Mobile Networks & Applications, 2(3) 259-69, 1997.

[25] N. Yevtushenko, A. R. Cavalli, and L.P. Lima, "Test minimization for testing in context". Proceedings of the 11th IWTCS, pp: 127-145, 1998.

[26] N. Yevtushenko, T. Villa. R. K. Brayton, A. Petrenko, A. Sangiovanni-Vincentelli. Solution of parallel language equations for logic synthesis. In Proc. of the International Conference on Computer-Aided Design, 103-110, 2001.

[27] W. M. Wonham and P. J. Ramadge, On the supremal controllable sublanguage of a given language. SIAM J. Control. Optimization. 25(3) (1987) 637-659.

[28] K. El-Fakih, N. Yevtushenko, and G. v. Bochmann,"Diagnosing multiple faults in communicating finite state machines", In Proc. of the IFIP 21st International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2001), Cheju Island, Korea), pp. 85-100, 2001.