FSM Test Translation Through Context

Khaled El-Fakih[1], Alexandre Petrenko[2], and Nina Yevtushenko[3]

[1] American University of Sharjah, UAE
[2] Centre de recherche informatique de Montreal (CRIM), Montreal, Canada
[3] Tomsk State University, Russia
kelfakih@aus.edu, petrenko@crim.ca, yevtushenko@elefot.tsu.ru

Abstract. In this paper, we define a formal approach for translating internal tests derived for a component embedded within a modular system into external tests defined over the external observable alphabets of the system. The system is represented as two communicating complete deterministic finite state machines, an embedded component machine to be tested and a context machine that represents the remaining part of the system. The context is assumed to be fault free and the interactions between the component machines are observable. When an internal test can not be translated in the given context, we demonstrate how another test with the guaranteed fault detection power could be determined (if such a test exists) that can be translated in the given context.

1 Introduction

The problem of testing in context is about testing a component embedded within a modular system that is usually represented as two communicating machines, an embedded component machine and a context machine that models the remaining part of the system and is assumed to be correctly implemented.

A number of test derivation methods have been proposed for testing in context [5,6,7,9,10] when the system components are modeled as Finite State Machines (FSMs). Some of these methods derive test suites with the guaranteed fault coverage directly from the embedded component [7,9,10]. However, such tests are generated in the form of input/output sequences defined over the input/output alphabets of the embedded machine. These tests have then to be translated into external tests defined over the external observable alphabets of the overall system. The problem of translating internal tests into external ones is known as the *fault propagation* or *test translation problem*. Different approaches for solving the translation problem for the case when the internal interactions between the component machines are unobservable are given in [2,7].

In this paper, we formally define and solve the test translation problem for the case when the interactions between the component FSMs are observable. Given an internal test for the embedded component, we present necessary and sufficient conditions for this test to be translated in the given context and show how to translate internal tests into external tests with the same fault detection power (if it is possible). When internal interactions are observable an external test that is a translation of an internal test has the same fault detection power as an internal test, i.e., it detects every faulty implementation of the embedded component that is detectable by the internal test in isolation. If an internal test cannot be translated within the given context, we derive (when possible) another internal test with the same fault detection power that can be translated within the given context. For this purpose, a so-called observable equivalent of the embedded component is derived. The notion of the observable equivalent is close to the notion of the embedded equivalent in [10]. However, in that work, the observable equivalent is derived under the assumption that the internal channels are not observable; in fact, in this paper, the observable equivalent refines a so-called conforming part of the embedded component [10] by restricting it to internal alphabets. Any internal test case derived from the observable equivalent can be translated in the given context.

The paper is organized as follows. Section 2 contains definitions of IOTS, FSM, and other preliminaries. Section 3 includes a formal definition and a method for test translation with simple application examples. Section 4 presents a method for deriving, when possible, internal test suites with the guaranteed fault coverage that can be translated in the given context. Section 5 concludes the paper.

2 Preliminaries

2.1 Input Output Transition Systems and Finite State Machines

We assume in this paper that components of a modular system are FSMs, however, we find it more convenient to compose state machines by encoding them into IOTSs.

An *Input/Output Transition System* is a quintuple $A = \langle S, I, O, \lambda_A, s_0\rangle$, where S is a finite nonempty set of states with the initial state s_0, $I \cup O$ is an alphabet of input and output actions such that $I \cap O = \varnothing$, and $\lambda_A \subseteq S \times (I \cup O \cup \{\tau\}) \times S$ is a transition relation, where $\tau \notin I \cup O$ is the internal action. We say that there is a *transition* from a state s to a state s' labeled with an action $v \in (I \cup O \cup \{\tau\})$ if and only if the triple (s, v, s') is in the transition relation λ_A.

For IOTS $A = <S, I, O, \lambda_A, s_0>$, we use $init(s)$ to denote the set of actions enabled in state $s \in S$, i.e., $init(s) = \{a \in (I \cup O \cup \{\tau\}) \mid \exists s' \in S\ ((s, a, s') \in \lambda_A)\}$. We use $in(s) \subseteq init(s)$ ($out(s) \subseteq init(s)$) to denote input (output) actions in state s. State $s \in S$ is called *stable* or *quiescent* if no output or internal actions are enabled in s: $init(s) \cap (O \cup \{\tau\}) = \varnothing$. Otherwise, s is called *unstable*.

State $s \in S$ with no action enabled, i.e., $init(s) = \varnothing$, is called a *deadlock* state. IOTS A *deadlocks* if there is a deadlock state reachable from the initial state. An IOTS is *deterministic* if it contains no internal action and the transition relation is a function, i.e., $(s, a, s_1), (s, a, s_2) \in \lambda_A$ for $a \in I \cup O$ implies $s_1 = s_2$.

As usual, the transition relation λ_A of the IOTS A is extended to sequences over the alphabet V. These sequences are usually called *traces* of the IOTS A. Given a state s of the IOTS A, the set of traces $Tr(s) = \{\alpha \in V^* \mid \exists s_i \in S\ ((s, \alpha, s_i) \in \lambda_A)\}$ is called the *language generated at the state s*. The language generated by the IOTS A at the initial state is called the *behavior of* or *language generated by* the *IOTS A*, denoted by $Tr(A)$. As usual, given a language L over the alphabet V, the *prefix closure* $\langle L\rangle$ contains each prefix of each sequence of L. The language is *prefix closed* if the language and its prefix closure coincide. By definition, the language of an IOTS is prefix closed.

Given a trace α over alphabet V, the *U-restriction* of α, written $\alpha_{\downarrow U}$, is obtained by deleting from α all symbols that belong to the set $V \backslash U$. Correspondingly, the *U-restriction* of a set T of traces over alphabet V, written $T_{\downarrow U}$, is the set of all sequences $\alpha_{\downarrow U}$, $\alpha \in T$. Given an IOTS $A = \langle S, I, O, \lambda_A, s_0 \rangle$ and $U \subseteq I \cup O$, the *U-restriction* $A_{\downarrow U}$ of A is obtained by replacing actions in $(I \cup O) \backslash U$ with the internal action τ and by determinizing [3] the resulting IOTS.

Let $A = \langle S, I, O, \lambda_A, s_0 \rangle$ and $B = \langle Q, I, O, \lambda_B, q_0 \rangle$ be two IOTSs, state s of IOTS A and state q of IOTS B are (trace) *equivalent*, if $Tr(s) = Tr(q)$. IOTSs A and B are (trace) *equivalent* if $Tr(A) = Tr(B)$.

A *finite state machine* (FSM) is a 7-tuple $M = \langle S, I, O, D_M, \Delta_M, \Lambda_M, s_0 \rangle$, where S is a finite nonempty set of states with the initial state s_0, I and O are input and output alphabets, $D_M \subseteq S \times I$ is the specification domain and Δ_M: $D_M \rightarrow S$ and Λ_M: $D_M \rightarrow O$ are the next state and the output functions. FSM M is called *complete* if $D_M = S \times I$; in this case, D_M can be omitted, i.e., a complete FSM is a 6-tuple $M = \langle S, I, O, \Delta_M, \Lambda_M, s_0 \rangle$. If M is not complete then it is *partial*. In usual way, the next state and output functions are extended to input sequences. Given state s and input sequence $i_1 \ldots i_k$, $\Delta_M(s, i_1 \ldots i_k) = s'$ while $\Lambda_M(s, i_1 \ldots i_k) = o_1 \ldots o_k$ if and only if there exist states $s_1', \ldots, s_{k+1}'$ such that $s_1' = s$, $s_{k+1}' = s'$, and $\Delta_M(s_j', i_j) = s_{j+1}'$ while $\Lambda_M(s_j', i_j) = o_j$ for each $j = 1, \ldots, k$. In this case, the sequence $i_1 o_1 \ldots i_k o_k$ is called an *I/O sequence* at state s. The set of all I/O sequences at the initial state of M is the *language* of FSM M. Each FSM can be represented as a deterministic IOTS with the same language by unfolding each transition [10].

We say that an IOTS has an *FSM behavior* if the IOTS is deterministic, has non-empty input and output sets, inputs are enabled only at stable states, and stable and unstable states alternate, i.e., for every stable state s and input $a \in in(s)$, $(s, a, s') \in \lambda_A$ implies that s' is an unstable state and for every unstable state s and output $a \in out(s)$, $(s, a, s') \in \lambda_A$ implies that s' is a stable state while the initial state is stable. If all input actions are enabled at every stable state, we say that the IOTS has a *behavior of a complete FSM*. If each input is followed by a single output, i.e., $|out(s)| = 1$ for each unstable state s, we say that the IOTS has a *behavior of a deterministic FSM*.

2.2 Parallel Composition of IOTSs

To compose complete FSMs we consider their IOTS counterparts. The joint behavior of k deterministic IOTSs $A_j = <S_j, I_j, O_j, \lambda_j, s_{j0}>$, $j = 1, \ldots, k$, is described by the parallel composition of IOTSs. The *parallel composition* $\|A_j$ (written also as $A_1 \| A_2 \ldots \| A_k$) is the IOTS $<R, \bigcup I_j \backslash \bigcup O_j, \bigcup O_j, \lambda, s_{10} \ldots s_{k0}>$, where the set of states $R \subseteq \times S_j$ and the transition relation λ are the smallest sets obtained by applying the following inference rules:

- $s_{10} \ldots s_{k0} \in R$;
- given $(s_1 \ldots s_k) \in R$, $(s_1' \ldots s_k') \in \times S_j$ and $a \in \bigcup I_j \cup \bigcup O_j$, $(s_1 \ldots s_k, a, s_1' \ldots s_k') \in \lambda$, if for each $j \in \{1, \ldots, k\}$ it holds that

if $a \in I_j \cup O_j$ then $(s_j, a, s_j') \in \lambda_j$ and if $a \notin I_j \cup O_j$ then $s_j' = s_j$.

Sometimes we need to hide some actions that are not observable in the resulting composition. This is achieved using the U-restriction defined above. In particular,

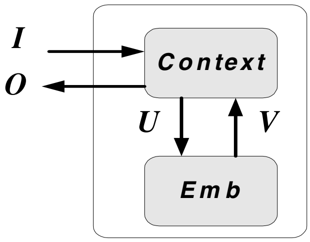

Fig. 1. Parallel Composition of the IOTSs *Context* and *Emb*

given a subset $I \subseteq \bigcup I_j \backslash \bigcup O_j$ and a subset O of the set $\bigcup O_j$, $I \cup O \neq \varnothing$, $(\|A_j)_{\downarrow I \cup O}$ is obtained by restricting the IOTS $\|A_j$ to the alphabet $I \cup O$.

In this paper, we consider a system of two complete deterministic FSMs, each of which is represented as an IOTS. The system consists of the context IOTS *Context* = $\langle S, I \cup V, O \cup U, \lambda_{Con}, s_0\rangle$ and the embedded IOTS *Emb* = $\langle Q, U, V, \lambda_{Emb}, q_0\rangle$, as shown in Figure 1. 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 IOTSs. As usual, for the sake of simplicity, we assume that the sets I, O, V, U are pair-wise disjoint. We also assume that the composition works in a slow environment, i.e., an external input can be applied to the composition after the latter has produced an external output to a previous external input. A behavior of such an environment can be represented by the IOTS $MAX = \langle\{p_0, p_1\}, I, O, \lambda_{Max}, p_0\rangle$, $\forall\ i \in I$ $(p_0, i, p_1) \in \lambda_{Max}$ and $\forall\ o \in O$ $(p_1, o, p_0) \in \lambda_{Max}$.

Therefore, the behavior of *Context* and *Emb* in the slow environment can be described by the parallel composition *MAX* ‖ *Context* ‖ *Emb*. We note that the IOTS *MAX* ‖ *Context* ‖ *Emb* does not have an FSM behavior. The reason is that the input set is empty. The following proposition states how the language of the IOTS *Context* ‖ *Emb* is constrained by a slow environment.

Proposition 1. The language of the IOTS *MAX* ‖ *Context* ‖ *Emb* is a subset of the prefix closure of the language $(I(UV)^*O)^*$.

Proposition 1 states that when an environment is slow, the component machines can execute a sequence of the set $(UV)^*$ before an external output is produced by the context in response to external input $i \in I$ received from the environment. Only after the context has produced an external output to a previous input, a next external input can be applied to the context.

3 Fault Propagation

3.1 Test Definitions

Definition 1. Given a specification IOTS $A = \langle S, I, O, \lambda_A, s_0\rangle$, a *test case* (*test*) is a non-empty sequence over alphabet $I \cup O$. A test αb is said to be *reduced* (w.r.t. the given specification A) if α is the longest prefix of αb that is a trace of the specification.

Given an IOTS specification A, the set of all possible implementations of A that are IOTSs over the alphabet $I \cup O$, is called the *fault domain* of A, denoted by $\mathfrak{I}(A)$. When A is clear from the context, we use the notation $\mathfrak{I}$ instead of $\mathfrak{I}(A)$. The fault domain includes both, conforming and nonconforming implementations, where the trace equivalence of IOTSs is the conformance relation. Thus, a fault to be detected by a test occurs when an implementation IOTS has a trace that is not a valid trace of the specification IOTS. To be more specific, such invalid trace has always an output as its last symbol. This is true for any IOTS that encodes a complete FSM, as well as for an IOTS that describes the composition of such IOTSs. It is not difficult to demonstrate that for this class of IOTSs either only all input actions are enabled or only output actions are enabled in each state, i.e., either $init(s) = I$ or $init(s) \subseteq O$ for all $s \in S$. Thus, traces of specification and implementation (deterministic) IOTSs may only differ on outputs and not on inputs.

Definition 2. Given the specification IOTS A, an implementation IOTS $B \in \mathfrak{I}$ that is not trace equivalent to A, and a test α, we say that *α detects B* if there exists a prefix of α that is a trace of the implementation IOTS B and not of A.

Given the specification IOTS A, the set $\mathfrak{I}$ of implementation IOTSs over the alphabet $I \cup O$, and a test α, $\mathfrak{I}_\alpha \subseteq \mathfrak{I}$ denotes the subset of implementations that are detected by α. The set $\mathfrak{I}_\alpha$ can be empty, it is the case when, for example, α is a trace of the specification.

Definition 3. A *test suite* is a finite set of tests. An implementation IOTS $B \in \mathfrak{I}$ that is not trace equivalent to A is said to be *detected* by a test suite if the test suite has a test that detects B.

If a test suite $TS = \{\alpha_1, ..., \alpha_k\}$ and $\mathfrak{I}_{TS} \subseteq \mathfrak{I}$ denotes the subset of implementations that are detected by TS then $\mathfrak{I}_{TS} = \mathfrak{I}_{\alpha_1} \cup ... \cup \mathfrak{I}_{\alpha_k}$.

Given a test α over the alphabet $I \cup O$, we derive a tree IOTS $IOTS_\alpha = <T, I, O, \lambda, \varepsilon>$, where T is the prefix closure of α with the empty sequence ε as the initial state. Given a proper prefix β of α and symbol $a \in I \cup O$, $(\beta, a, \beta a) \in \lambda$ if βa is a prefix of α. By definition, state α is a deadlock state.

Given a test suite TS consisting of test cases $\alpha_1, ..., \alpha_k$ over the alphabet $I \cup O$, a tree IOTS $IOTS_{TS}$ is determined by first deriving the union of the IOTSs $IOTS_{\alpha_1}$, $IOTS_{\alpha_2}$, ..., $IOTS_{\alpha_k}$ [HoUI79] and then determinizing the obtained IOTS.

Definition 4. Given the specification IOTS A and a set $\mathfrak{I}' \subseteq \mathfrak{I}$ of implementation IOTSs over alphabets $I \cup O$, let TS be a test suite. The test suite TS is *exhaustive* in $\mathfrak{I}'$, if the test suite detects each $B \in \mathfrak{I}'$ that is not trace equivalent to A.

Given a test case $\alpha\gamma$ that is not reduced and $B \in \mathfrak{I}_{\alpha\gamma}$, in order to detect B we can use the shortest prefix α of $\alpha\gamma$ that is not a trace of the specification A. In other words, in order to detect all possible faulty implementations of the fault domain $\mathfrak{I}_{\alpha\gamma}$ it is sufficient to use the reduced test α, i.e., the following statement holds.

Proposition 2. Given the specification IOTS A, let α and $\alpha\gamma$ be test cases such that α is not a trace of the specification A. The set of implementation IOTSs that are detected by $\alpha\gamma$ coincides with the set of those implementations that are detected by α, i.e., $\mathfrak{I}_\alpha = \mathfrak{I}_{\alpha\gamma}$.

Given a test suite exhaustive in $\Im'$, we can reduce the length of this test suite by deleting every test that is a trace of the specification and replacing each remaining non-reduced test with its shortest prefix that is not a trace of the specification. According to Proposition 2, the resulting test suite is also exhaustive in $\Im'$.

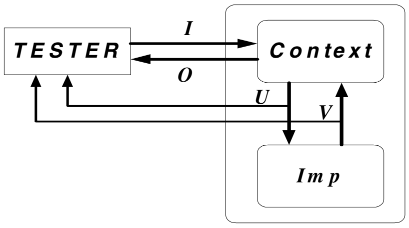

Fig. 2. Test architecture

3.2 Test Architecture

We consider the composition of IOTSs *Context* and *Emb* with the IOTS *TESTER*, and assume that during testing all actions can be observed (Figure 2). In this case, the closed system is the parallel composition *TESTER* || *Context* || *Emb* with the output set $I \cup O \cup U \cup V$.

As usual, we assume that the *Context* component is fault free and only an implementation of the embedded component may be faulty. Moreover, we assume that each possibly faulty implementation is a complete deterministic FSM with a restricted number of states represented as an IOTS and denote $\Im(Emb)$ the fault domain of *Emb*, i.e., $\Im(Emb)$ is the set of IOTSs that represent all possible *Emb* implementations. Thus, a fault domain of the system *MAX* || *Context* || *Emb* is $\Im(Con\text{-}Emb) = \{MAX \,||\, Context \,||\, Imp : Imp \in \Im(Emb)\}$. Given $Imp \in \Im(Emb)$, *Imp* is said to be a *conforming* (in the given context) implementation of *Emb* if IOTSs *MAX* || *Context* || *Imp* and *MAX* || *Context* || *Emb* are trace equivalent. Otherwise, *Imp* is a *nonconforming* implementation. Not every implementation of the embedded component that is not trace equivalent to *Emb* and thus, can be detected in isolation, is a nonconforming implementation in context [10]. As an example, consider the specification *Emb* and the faulty implementation Imp_1 shown in Figures 3a and 3b, respectively. The context IOTS is shown in Figure 4. The composition *MAX* || *Context* || Imp_1 is trace equivalent to the *MAX* || *Context* || *Emb*. Therefore, the fact that the implementation Imp_1 is not trace equivalent to *Emb* cannot be established within the given context.

According to the above test architecture, during the testing process a tester applies actions of the set I to the external input of *Context* and draws a conclusion whether an implementation *Imp* of the embedded component conforms to its specification by observing the outputs over the set $O \cup U \cup V$. Thus, traces of a tester are defined over the alphabet $I \cup O \cup U \cup V$. Since we are interested in the system of communicating IOTSs *Context* and *Imp* that work in a slow environment, the tester has also to be slow, i.e., the tester can apply the next symbol $i \in I$ only after it has obtained, from *Context*, an external output $o \in O$ to the previously applied input of the set I. We call such tester a *slow* tester and according to Proposition 1, a slow tester executes traces in the set $(I(UV)^*O)^*$. Each trace of a tester is called an *external* test (case). In fact, a tester is a tree IOTS derived from an external test suite. As usual, an *external* test suite is a finite set of external tests. Following Definitions 2 and 3, an external test detects each implementation system *MAX* || *Context* || *Imp* of the set $\mathfrak{I}$(*Con-Emb*) if some prefix of the external test is a trace of *MAX* || *Context* || *Imp*, but not a trace of the composition *MAX* || *Context* || *Emb*. In this case, the tester detects a fault that makes *Imp* nonconforming. Otherwise, i.e., when the external test is a trace of *MAX* || *Context* || *Emb*, the implementation *Imp* has no faults that can be detected by this test.

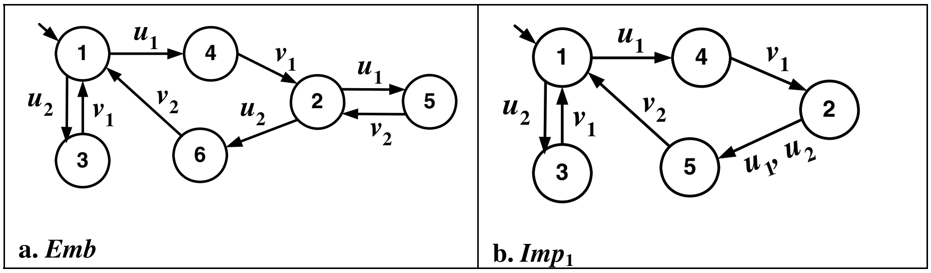

Fig. 3. Specification *Emb* and a faulty implementation Imp_1

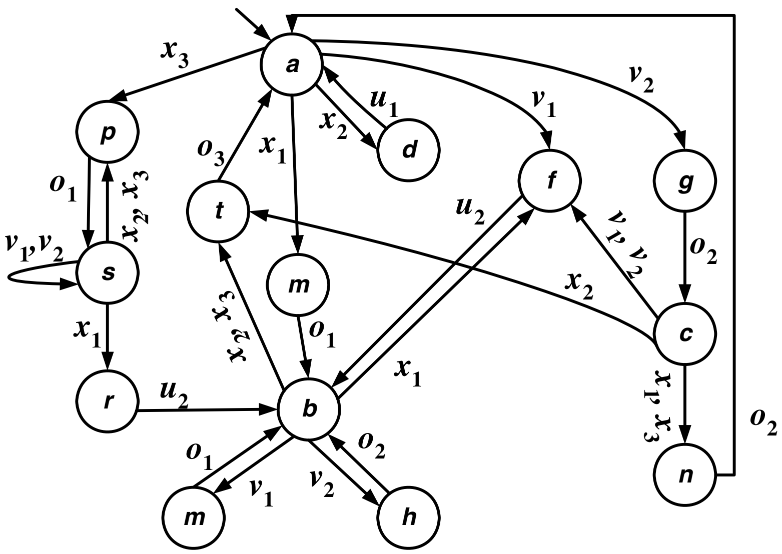

Fig. 4. Context IOTS

3.3 Problem Definition

Given the embedded component *Emb* over input alphabet *U* and output alphabet *V*, an *internal* test (case) is a trace over the alphabet $U \cup V$. Since the IOTS *Emb* has an

FSM behavior, an internal test is a non-empty sequence of the language $(UV)^*$. Correspondingly, an *internal* test suite is a finite set of internal tests.

When the implementation is tested through the context, the internal inputs of the embedded component are not directly controllable; except for the context of FIFO queues [4,11]. For other types of contexts, internal tests have to be translated to external tests.

Given an internal test *InTest*, let $\Im_{InTest}(Emb) \subseteq \Im(Emb)$ denote the set of possible faulty implementations of the embedded component *Emb* that can be detected by *InTest* when testing the IOTS *Emb* in isolation. Naturally, it makes sense to consider internal tests which detect at least one nonconforming implementation, i.e., tests which belong to the set $(UV)^*\backslash Tr(Emb)$. We first introduce the notion that relates fault detection capability of internal and external tests.

Definition 5. Given $InTest \in (UV)^*\backslash Tr(Emb)$, an external test *ExtTest* has the *same fault detection power* as *InTest* if *ExtTest* detects each implementation system $MAX \parallel Context \parallel Imp$, where $Imp \in \Im_{InTest}(Emb)$. Similarly, given an internal test suite $InTS \subseteq (UV)^*\backslash Tr(Emb)$, an external test suite *ExtTS* has the *same fault detection power* as *InTS*, if *ExtTS* detects each implementation system $MAX \parallel Context \parallel Imp$, where $Imp \in \Im_{InTS}(Emb)$.

The problem of translating *InTest* is to determine an external test (if it exists) with the same fault detection power, i.e., to determine an external test case that detects each IOTS $MAX \parallel Context \parallel Imp$, where $Imp \in \Im_{InTest}(Emb)$. The problem is called the *test translation* or the *fault propagation* problem [2,7].

In the rest of the paper, given an internal test suite, we propose a method of translating (when possible) it into an external one with the same fault detection power. Moreover, in Section 4, we propose methods for deriving internal test suites with the guaranteed fault coverage that can be translated within the given context.

3.4 Translation of an Internal Test Case

Given an internal test case $InTest \in (UV)^*\backslash Tr(Emb)$, let $Imp \in \Im_{InTest}(Emb)$ be an implementation that is detected by *InTest*, i.e., *Imp* has a trace that is not a trace of the embedded component *Emb*. Therefore, a tester, that induces *InTest* at the channels *U* and *V* in the composition $TESTER \parallel Context \parallel Imp$, will detect that *Imp* is a nonconforming implementation. In other words, if the IOTS $(TESTER \parallel Context \parallel Imp)_{\downarrow U \cup V}$ has a trace *InTest*, then a tester detects the nonconforming implementation *Imp*, and, thus, we have the following definition that relates internal and external tests.

Definition 6. Given $InTest \in (UV)^*\backslash Tr(Emb)$, an external test $\in \langle (I(UV)^*O)^* \rangle$ is a *translation* of *InTest*, denoted *Transl*(*InTest*), if the IOTS $(IOTS_{Transl(InTest)} \parallel Context \parallel IOTS_{InTest})_{\downarrow U \cup V}$ is trace equivalent to the IOTS $IOTS_{InTest}$. Correspondingly, given an internal test suite $InTS \subseteq (UV)^*\backslash Tr(Emb)$, an external test suite $\subseteq \langle (I(UV)^*O)^* \rangle$ is a *translation* of *InTS*, denoted *Transl*(*InTS*), if the $(IOTS_{Transl(InTS)} \parallel Context \parallel IOTS_{InTS})_{\downarrow U \cup V}$ is trace equivalent to the IOTS $IOTS_{InTS}$.

The following statement is implied immediately.

Proposition 3. Given $InTS \subseteq (UV)^*\backslash Tr(Emb)$, an external test suite *Transl*(*InTS*) detects each implementation system $MAX \parallel Context \parallel Imp$, $Imp \in \Im_{InTS}(Emb)$, i.e., *Transl*(*InTS*) has the same fault detection power as *InTS*.

Due to Definition 6, in order to determine a translation of a given internal test *InTest* we have to establish conditions under which the composition (*TESTER* || *Context* || *Imp*)$_{\downarrow U \cup V}$ has the trace *InTest.* According to the definition of the parallel composition, the following statement holds for traces of the composition (*TESTER* || *Context* || *Imp*)$_{\downarrow U \cup V}$.

Proposition 4. Given an internal test case *InTest*, the composition (*TESTER* || *Context* || *Imp*)$_{\downarrow U \cup V}$ has a trace *InTest* if and only if the IOTS *Imp* and the composition (*TESTER* || *Context*)$_{\downarrow U \cup V}$ have the trace *InTest.* Correspondingly, given an internal test suite *InTS*, the set of traces of the composition (*TESTER* || *Context* || *Imp*)$_{\downarrow U \cup V}$ contains *InTS* if and only if the set of traces of the IOTS *Imp* and of the composition (*TESTER* || *Context*)$_{\downarrow U \cup V}$ contains the set *InTS.*

Here we note that not each internal case can be translated, as the context may render it impossible. According to Proposition 4, the following sufficient and necessary conditions can be established for the translation of an internal test in the given context.

Let $IOTS^{Aug}_{InTest}$ denote the IOTS obtained from $IOTS_{InTest}$ by adding self-loops labeled with all $i \in I$ (input) and $o \in O$ (output) at every non-deadlock state.

Theorem 1. Given a context *Context* and internal test *InTest*, the test *InTest* can be translated in the context if and only if the IOTS (*MAX* || *Context* || $IOTS_{InTest}$)$_{\downarrow U \cup V}$ is trace equivalent to $IOTS_{InTest}$. Moreover, if the test *InTest* is reduced and can be translated in the context then the set of traces with the ($U \cup V$)-restriction *InTest* that take the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ into a deadlock state coincides with the set of all reduced translations of the test *InTest.*

In fact, the first statement of the theorem is a direct corollary to Proposition 4. In the second statement of the theorem, we use $IOTS^{Aug}_{InTest}$ instead of $IOTS_{InTest}$ in the composition, to force a tester to stop after the first unexpected output *v* of *InTest* is produced by an implementation. Thus, each trace with the ($U \cup V$)-restriction *InTest* that takes the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ into a deadlock state is a reduced external test if *InTest* is reduced. As the set of traces the IOTS *MAX* || *Context* has each trace that can occur in the composition of the system *Context* || *Imp*, *Imp* $\in$ $\Im$(*Emb*), with a slow tester, the set of all traces with the ($U \cup V$)-restriction *InTest* that take the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ into a deadlock state coincides with the set of all reduced translations of the test *InTest.*

According to Theorem 1, each trace with the ($U \cup V$)-restriction *InTest* that takes the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ into a deadlock state has the same fault detection power as the internal test *InTest* and is a translation of *InTest.*

In general, the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ has many traces that lead to a deadlock state and have the ($U \cup V$)-restriction *InTest.* Each such trace can be selected as a translation of *InTest.* However, we are interested in a shortest translation, i.e., in the translation that is a reduced external test. According to Theorem 1, in order to determine a shortest translation of *InTest* it is sufficient to find a shortest trace that takes *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ to a deadlock state and has the ($U\cup V$)-restriction *InTest*. Therefore, the problem of finding a shortest translation *Transl*(*InTest*) of *InTest* can be solved by determining a shortest trace that takes the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTest}$ to a deadlock state and has *InTest* as its ($U\cup V$)-restriction.

In our working example, consider the internal test u_2v_2. By direct inspection (Figure 5), one can assure that the trace $x_1o_1x_1u_2v_2$ is a translation of u_2v_2. For the internal test u_2v_2, we have two shortest translations $x_1o_1x_1u_2v_2$ and $x_3o_1x_1u_2v_2$ (Figure 5).

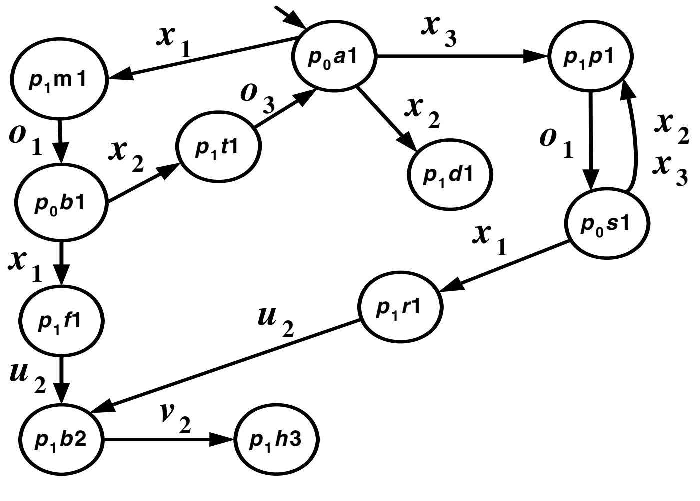

Fig. 5. IOTS *MAX* || *Context* || $IOTS^{Aug}_{u_2v_2}$

3.5 Translation of an Internal Test Suite

Since an internal test suite is a finite set, it can be translated by translating each of its test cases separately, as described in the previous subsection. To reduce the length of a resulting external test suite we have to select for each internal test *InTest* a shortest translation of *InTest.*

However, all the tests of an internal test suite *InTS* can be translated altogether. Given $IOTS_{InTS}$, we denote $IOTS^{Aug}_{InTS}$ the IOTS obtained from $IOTS_{InTS}$ by adding self-loops labeled with all $i \in I$ and $o \in O$ at every non-deadlock state. According to Theorem 1, the following statement holds.

Theorem 2. Given a context *Context* and an internal test suite *InTS*, the test suite *InTS* can be translated in the context, if and only if the IOTS (*MAX* || *Context* || $IOTS_{InTS})_{\downarrow U\cup V}$ is trace equivalent to $IOTS_{InTS}$. Moreover, if the test suite *InTS* has only reduced tests and can be translated in the context then the set of traces with the ($U\cup V$)-restriction *InTS* that take the IOTS *MAX* || *Context* || $IOTS^{Aug}_{InTS}$ into a deadlock state contains each set of reduced translations of the test suite *InTS*.

Due to Theorem 2, each subset of traces with the $(U \cup V)$-restriction *InTS* that take the IOTS *MAX* || *Context* || $IOTS_{InTest}^{Aug}$ into a deadlock state has the same fault detection power as the internal test suite *InTS* and is a translation of *InTS*.

Here we note the resulting translation of *InTS* (i.e. an external test suite) *Transl*(*InTS*) can have tests whose *I*-restrictions are prefixes of the same sequence over the alphabet *I*. According to our test architecture, for a tester it is sufficient to apply to the context longest *I*-restrictions of sequences of *Transl*(*InTS*). As an example, consider the internal test suite $\{u_2v_2, u_2v_1u_2v_1\}$. One of its translations is an external test suite $\{x_2u_2v_2, x_2u_2v_1u_2v_1\}$. When executing the external test suite $\{x_2u_2v_2, x_2u_2v_1u_2v_1\}$ that is a translation of the internal test suite $\{u_2v_2, u_2v_1u_2v_1\}$ the tester has to apply only the external input x_2 to the context and observe the obtained outputs.

4 Exhaustive External Test Suites

When an internal test suite cannot be translated throughout the given context, there may still exist another internal test suite that detects the same set of faulty implementations of *Emb* and can be translated in the given context. Therefore, given a fault domain $\Im(Emb)$, we would like to derive an internal test suite for *Emb* that can be translated in the given context to obtain a translation exhaustive in the fault domain $\Im(Con\text{-}Emb)$.

As an example, consider a fault domain $\Im(Emb)$ of *Emb* (Figure 3a) that contains each IOTS with a behavior of a complete deterministic FSM with at most two states and an exhaustive test suite for *Emb* w.r.t. the fault domain $\Im(Emb)$. Such a test suite can be derived using the W-method [1,12] or its derivatives. The W-method provides an exhaustive test suite $E = \{u_2u_1, u_1u_1u_1, u_1u_2u_1\}$ as a set of input sequences over alphabet *U*. In order to transform this set into an internal test suite *InTS* for the IOTS *Emb* we proceed as follows. For each sequence $u_1...u_k$ of the set *E* we determine a corresponding trace $u_1v_1...u_kv_k$ of the embedded component *Emb*. Then, we append each prefix $u_1v_1...u_j$, $j \le k$, of the trace $u_1v_1...u_kv_k$ with all possible wrong internal outputs $v' \in V\backslash\{v_j\}$ and include the resulting sequences into the internal test suite *InTS*. In our example, we obtain $InTS = \{u_2v_2, u_2v_1u_1v_2, u_1v_2, u_1v_1u_1v_1, u_1v_1u_1v_2u_1v_2, u_1v_1u_2v_1, u_1v_1u_2v_2u_1v_2\}$, the *I*-restriction of this set is exactly the set *E*.

By direct inspection, one can assure that this test suite cannot be translated through the given context. The reason is that, for example, an internal test case $u_1v_1u_1v_1$ is not in the set of traces of the IOTS $(MAX \parallel Context)_{\downarrow U \cup V}$ and thus, cannot be executed in the given context.

Therefore, to derive an exhaustive internal test suite w.r.t. the above fault domain $\Im(Emb)$ that can be translated into an exhaustive external test suite w.r.t. the fault domain $\Im(Con\text{-}Emb)$, we have to consider only the behavior of the embedded component *Emb* for the sequences that can be executed in the given context. To this end, we define an IOTS a so called observable equivalent of *Emb*, by removing from it the sequences that cannot be executed with the given context.

Definition 7. Given IOTSs *MAX*, *Context* and *Emb*, the IOTS Eq_{Emb} is an *observable equivalent* of *Emb* if $Tr(Eq_{Emb}) = Tr(Emb) \cap Tr((MAX \parallel Context)_{\downarrow U \cup V})$.

Due to Definition 7 and Proposition 4, the observable equivalent Eq_{Emb} of an embedded component *Emb* can be derived as follows: $Eq_{Emb} = (MAX \parallel Context \parallel Emb)_{\downarrow U \cup V}$. For our working example, the observable equivalent is shown in Figure 6.

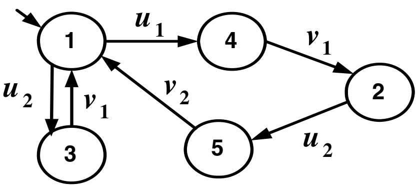

Fig. 6. Observable equivalent IOTS Eq_{Emb}

As a corollary to Theorem 2, the following statement holds.

Theorem 3. A reduced internal test $u_1v_1...u_kv_k \in (UV)^*\backslash Tr(Emb)$ can be translated in the given context if and only if the input sequence $u_1...u_k$ is a trace of the *U*-restriction of the IOTS Eq_{Emb}, i.e., $u_1...u_k \in (MAX \parallel Context \parallel Emb)_{\downarrow U}$.

If a reduced internal test *InTest* can be translated in the given context then every external translation of *InTest* has the same fault detection power as *InTest* (Proposition 3). According to Theorem 3, a reduced internal test $u_1v_1...u_kv_k$ can be translated if the internal input sequence $u_1...u_k$ is a trace of the *U*-restriction of the IOTS Eq_{Emb}. Therefore, the following two statements hold as corollaries to Theorem 3.

Corollary 2. Given a reduced internal test *InTest* such that the *U*-restriction of *InTest* is a trace of the IOTS $(Eq_{Emb})_{\downarrow U}$, the composition of the context and a faulty implementation $Imp \in \Im(Emb)$ is detected by the external test *Transl*(*InTest*) if and only if *Imp* is detected by *InTest*.

Corollary 3. Given an internal test suite *InTS* with reduced tests such that the *U*-restriction of each test of *InTS* is a trace of the IOTS $(Eq_{Emb})_{\downarrow U}$, the composition of the context and a faulty implementation $Imp \in \Im(Emb)$ is detected by the external test suite *Transl-InTS* if and only if *Imp* is detected by *InTS*.

There is a special case when $Tr(Eq_{Emb})_{\downarrow U} = U^*$. This means that the context has a behavior of a complete FSM and any internal test case can be translated.

Corollary 4. Given the observable equivalent Eq_{Emb} of the embedded component, if $Tr(Eq_{Emb})_{\downarrow U} = U^*$ then each reduced internal test can be translated through the given context.

Here we note that the notion of the observable equivalent is close to the notion of the embedded equivalent in [10]. However, in that work, the observable equivalent is derived under the assumption that the internal channels are not observable; in fact, the construction refines a so-called conforming part of the embedded component *Emb* restricting it to alphabets of *Emb*.

According to Corollary 3, internal test suites are derived from the specification of the embedded component that has a behavior of a partial deterministic FSM. Then an internal test suite for the embedded component can be derived, using the State

Counting (SC) method in [8], exhaustive w.r.t. the fault model <*Spec*, ≤, *FD*>, where *Spec* is a partial FSM, ≤ is the quasi-equivalence relation, called weak conformance in [13], and *FD* is the set of all possible implementation FSMs with a restricted number of states.

Applied to the partial FSM that is encoded as the IOTS Eq_{Emb}, the SC-method returns a set E of internal (over the alphabet U) input sequences. In order to transform this set into an internal test suite *InTS* we again for each sequence $u_1...u_k$ of the set E, determine a corresponding trace $u_1v_1...u_kv_k$ of the embedded component *Emb*, append each prefix $u_1v_1...u_j$, $j \leq k$, of the trace $u_1v_1...u_kv_k$ with all possible wrong internal outputs $v' \in V\backslash\{v_j\}$ and include the resulting sequences into the internal test suite *InTS*.

Consider the observable equivalent IOTS Eq_{Emb} of *Emb* in Figure 6. The IOTS Eq_{Emb} has a behavior of a partial FSM with two states. If we consider the fault domain $\Im(Emb)$ of all IOTSs that have a behavior of a complete deterministic FSM with at most two states, then we can derive, using the SC-method or the method in [13], the test suite $\{u_1v_2,\ u_1v_1u_2v_1,\ u_1v_1u_2v_2u_2v_2,\ u_2v_2,\ u_2v_1u_2v_2\}$ exhaustive w.r.t. the fault domain $\Im(Emb)$ and quasi-equivalence relation. The corresponding external tests are $\{x_2u_1v_2,\ x_2u_1v_1u_2v_1,\ x_1o_1x_1u_2v_2,\ x_1o_1x_1u_2v_1o_1x_1u_2v_2\}$ and according to Theorem 3, this external test suite is exhaustive w.r.t. the fault domain $\Im(Con\text{-}Emb)$.

Another approach for test derivation from the embedded equivalent is mutant-based testing. A mutant may model certain suspected faults, which have to be tested for their presence. The approach is based on the enumeration of mutants of the embedded component *Emb* and finding external tests that kill these mutants. To this end, given a mutant $Imp \in \Im(Emb)$, we consider the IOTS $Imp \parallel Eq_{Emb}$. We first note that the observable equivalent Eq_{Emb} does not deadlock, since each IOTS *Context* and *Emb* has a behavior of a complete FSM. Secondly, given $Imp \in \Im(Emb)$, *Imp* is not trace equivalent to *Emb* if and only if the IOTS $Imp \parallel Emb$ deadlocks. If the IOTS $Imp \parallel Eq_{Emb}$ does not deadlock then the mutant IOTS *Imp* is a conforming implementation of *Emb*. Otherwise, each trace of *Imp* such that its U-restriction takes the IOTS $(Imp \parallel Eq_{Emb})\downarrow_U$ to a deadlock state is an internal test that detects a faulty implementation *Imp* and this internal test can be translated through the given context.

As an example, consider the faulty implementation Imp_1 (Figure 3b) of the embedded component *Emb* (Figure 3a). The composition $Imp_1 \parallel Eq_{Emb}$ is similar to the Eq_{Emb} in Figure 6; only state labels are renamed 11, 22, 33, 44, and 55. Since the composition $Imp_1 \parallel Emb$ does not deadlock, the faulty implementation Imp_1 cannot be detected through the given context, and thus Imp_1 is a conforming implementation (in the given context). As another example, consider the faulty implementation Imp_2 which is similar to Imp_1 of Fig. 3b except that the transition connecting states 5 and 1 has the label v_1 instead of v_2. The composition $Imp_2 \parallel Emb$ deadlocks after the trace $u_1v_1u_2$ and thus Imp_2 can be detected through the given context.

5 Conclusions

In this paper, we proposed an approach for translating internal tests derived for a component embedded within a modular system into external tests of the system. The system is represented as two complete deterministic communicating finite state machines, an embedded component machine to be tested and a context machine that represents the remaining part of the system. The context is assumed to be fault free and the interactions between the component machines are observable. Also, in this paper, we established necessary and sufficient conditions for an internal test (suite) to be translated in the given context. If a test cannot be translated, we demonstrated another test with the guaranteed fault detection power could be determined (if such a test exists) that can be translated in the given context. In our future work, we intend to generalize the fault translation approach elaborated in this paper for communicating finite state machines to input output transition systems.

References

1. T. S. Chow, "Test design modeled by finite-state machines", *IEEE Trans. SE*, vol. 4, no.3, pp. 178-187, 1978.
2. K. El-Fakih and N. Yevtushenko, "Fault propagation by equation solving", *Proc. of the IFIP* 24th *International Conference on Formal Techniques for Networked and Distributed Systems*, Madrid, Spain, LNCS 3235, pp. 185-198, 2004.
3. J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, N.Y., 1979.
4. C. Jard, T. Jéron, L. Tanguy, and C. Viho, "Remote testing can be as powerful as local testing", *Proc. of the IFIP Joint Intl. Conf. Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification* (*FORTE XII / PSTV XIX*), volume 156 of *IFIP Conference Proceedings*, Beijing, China, Oct. 5-8, Kluwer, pp. 25–40, 1999.
5. L. P. Lima, "A pragmatic method to generate test sequences for embedded systems", Ph.D. Thesis, *Institute National des Telecommunications*, Evry, France, 1998.
6. L. P Lima and A. R. Cavalli, "A pragmatic approach to generating test sequences for embedded systems", *Proc. of the* 10th *International Workshop on Testing of Communicating Systems*, pp: 125-140, 1997.
7. A. Petrenko and N. Yevtushenko, "Testing faults in embedded components", *Proc. of the* 10th *International Workshop on Testing of Communicating Systems*, pp. 272-287, 1997.
8. A. Petrenko and N. Yevtushenko, "Testing from partial deterministic FSM specifications", *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1154-1165, 2005.
9. A. Petrenko, N. Yevtushenko, and G. v. Bochmann, "Fault models for testing in context", *Proc. International Conference on Formal Techniques for Networked and Distributed Systems*, pp. 125-140, 1996.
10. A. Petrenko, N. Yevtushenko, G. v. Bochmann, and R. Dssouli, "Testing in context: framework and test derivation", *Computer communications*, Vol. 19, pp. 1236-1249, 1996.
11. J. Tretmans and L. Verhaard, "A queue model relating synchronous and asynchronous communication", In R. J. Linn, Jr. and M. Ü. Uyar, eds., *Proc. of the IFIP TC6/WG6.1* 12th *Intl. Symp. Protocol Specification, Testing and Verification*, volume C-8 of *IFIP Transactions*, Lake Buena Vista, Florida, USA, pp. 131–145, 1992.
12. M. P. Vasilevskii, "Failure diagnosis of automata", translated from Kibernetika, no.4, pp. 98-108, 1973.
13. M. Yannakakis and D. Lee, "Testing finite state machines", *Proc. of the* 23rd *Annual ACM Symposium on Theory of Computing*, New Orleans, Louisiana, pp. 476-485, 1995.