Deriving Tests with Fault Coverage for Specifications in the Form of Labeled Transition Systems

Q. M. Tan,
A. Petrenko and
G. v. Bochmann

publication # 1073

Juin 1997

Deriving Tests with Fault Coverage for Specifications in the Form of Labeled Transition Systems

Q. M. Tan† A. Petrenko‡ G. v. Bochmann†

† D'IRO, Université of Montréal, Canada, {tanq, bochmann}@iro.umontreal.ca

‡ Centre de recherche informatique de Montréal, Canada, petrenko@crim.ca

Abstract

A challenging issue is the derivation of finite test suites with well-defined fault coverage for conformance testing of communication systems modeled by labeled transition systems (LTSs) with respect to a particular conformance relation. It is shown in this paper that this problem can be solved by translating an LTS specification into an input/output finite state machine (FSM) for trace or failure semantics, respectively, subsequently applying existing FSM-based methods for test derivation, and finally converting the obtained tests back to the LTS formalism. It is also demonstrated that the obtained tests can be optimized or the existing FSM-based methods can be adapted for generating optimized tests by taking into account the specifics of the FSMs which are obtained from the given LTSs.

Keywords: Protocol conformance testing, software testing, lebeled transition systems, formal modeling, fault coverage.

1 Introduction

Conformance testing is one of the essential and challenging issues in the development of communication systems, including network protocols and some kinds of software systems. Conformance testing is the activity of trying to find out whether an implementation conforms to its specification to gain confidence in the successful interactions of the implementation with its environment. Because of the complexity of systems , it is often proposed to use the formal techniques to support the automation of this activity. Amongst the formal

description techniques (FDTs) used for writing specifications are LOTOS [4], which is based on *Labeled Transition Systems* (LTSs), and SDL [2] and Estelle [8], which are based on the *Finite State Machine* (FSM) model. Much work on the derivation of tests from a given system specification has been done separately for the two models [3, 23].

FSMs and LTSs are often used in the analysis and design of various software systems to specify the "control structure". The control structure governs the actions of a program which affect its environment. When the interactions of a program with its environment are mainly concerned, the program can be considered to be a communication system. In addition to network protocols, these models are useful for describeing the behavior of compilers, real-time systems, embedded software and concurrent programs.

Systematic approaches have been developed for conformance testing of communication systems and the generation of appropriate test suites based on the FSM model. Most work in this area is limited to completely specified, deterministic specifications [10, 16, 27]. However, some recent research has addressed nondeterministic and partially specified specifications [22, 20, 21]. A number of competing methods for deriving tests from FSMs that guarantee fault coverage have been elaborated [23].

Compared to FSMs, LTSs are in some sense a more general descriptive model, since interactions of a specified system with its environment are usually considered rendezvous interactions making no distinction between input and output. LTSs are usually not completely specified; the unspecified interactions are not possible. There has been much work on testing theory and test derivation from LTS specifications [6, 7, 35, 26, 36, 31, 32, 19, 9, 14, 28]. However, most of the test derivation methods are based on exhaustive testing, where all the possible execution scenarios are carried out in order to prove the correctness of the implementation in respect to a given conformance relation. However, such exhaustive testing is often impractical since it may involve a test suite of infinite behavior. The approximation approach [26, 33], such as n-testers, which is proposed to solve this problem, provides no fault coverage measure for conformity of the implementation with its specification. Moreover, for the existing frameworks for testing LTSs [7, 32], it is difficult to define the verdicts for certain conformance relations, such as trace equivalence, nondeterministic reduction (**conf** [6] plus trace equivalence) [13] and failure equivalence [29].

A general framework for conformance testing of communication systems should be developed in such a way that the conformance relation is determined by the real conformance requirements, and a test suite should have finite behavior and ensure a "good" fault coverage. Several attempts have been made to apply the ideas underlying the FSM-based methods to the LTS model [9, 15, 1, 28], by redefining the state identification and eventually the checking experiments in the LTS realm. [9] tries the UIO-based state identifiers which, as is well known, do not always exist; [15] considers the characterization sets; and [1] introduces the state identification machines. However, these attempts did not solve the problem of deriving a finite test suite with complete fault coverage from an arbitrary LTS for a given conformance relation. In this paper we take another approach, initially outlined in [23]. It is suggested that tests for a given LTS specification and conformance relation could be obtained from tests directly generated by the existing FSM-based methods from a specific FSM which is constructed from the LTS according to the chosen semantics. This approach has the advantage of allowing reuse of existing FSM-based methods and testing tools for the LTS specifications. Evidently, the translation of an LTS into an FSM is semantic-driven and should be elaborated on a case-by-case basis. In this paper, we try to elaborate this general idea for the two particular types of semantics, namely, the trace and failure semantics. We formally show that LTS specifications can be modeled by proper input/output (I/O) FSMs in the trace or failure semantics, and that test suites with fault coverage produced from the corresponding FSMs and then converted back to the LTS formalism can guarantee the given conformance relation with corresponding fault coverage. We also demonstrate that the test suites can be further optimized taking into account the specific properties of the FSMs derived from LTSs.

We give in Section 2 basic definitions and notations of the FSM and LTS models, and common conformance relations used in the two models, respectively. In Section 3 a framework is proposed for finite testing of LTSs, and in Section 4 the verdict labeling functions for test cases with respect to the conformance relations are discussed. In Section 5, the FSM models for LTSs, in failure semantics and in trace semantics respectively, are defined, and the validity of these transformations are shown. In Section 6, the proposed approach to test derivation from a given LTS specification with respect to the conformance relations is illustrated, and the optimization of the test cases is discussed.

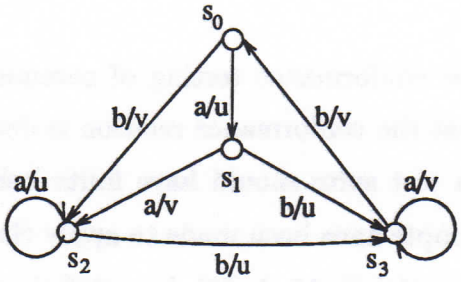


Figure 1: An FSM graph

2 Basic Definitions and Notions of Conformance

The starting point for conformance testing is a specification in some (formal) notation, an implementation given in the form of a black box, and a set of conformance requirements the implementation should satisfy. In this paper, two formal notations, which are *finite state machines* and *labeled transition systems*, respectively, are considered for specifications; implementations are also assumed to be described in the same model as its specification; the conformance requirements of a given specification is supposed to be defined by a specific conformance relation.

2.1 Finite State Machine

Definition 1 (*Finite State Machine (FSM)*): A completely specified finite state machine is a 5-tuple $< S, X, Y, h, s_0 >$, where:

- S is a finite set of states, and $s_0 \in S$, is the initial state.
- X is a finite set of inputs.
- Y is a finite set of outputs, and it may include Θ which represents the null output, that is, no output.
- h is a behavior function, $h : S \times X \rightarrow powerset(S \times Y) \backslash \{\emptyset\}$, where $\emptyset$ is the empty set. $(q, b) \in h(p, a)$ is also written $p - a/b \rightarrow q$, which is called a *transition* from p to q with the label a/b.

The behavior function defines the possible transitions of the machine. If $|h(p, a)| = 1$ for all $(p, a) \in S \times X$ then the FSM is *deterministic* (DFSM); otherwise, it is *nondeterministic* (NFSM). The FSM is also said to be an *observable* FSM (OFSM), if $|\{q \mid \forall (q, b) \in h(p, a)\}| \leq 1$ for all $p \in S$ and $(p, a) \in S \times X$. A DFSM is observable, but an OFSM may

notation	meaning
Γ	$X \times Y$, a set of input/output pairs; v denotes such a pair
Γ^*	set of sequences over Γ; γ or $v_1 \ldots v_n$ denotes such a sequence
$p-\varepsilon\rightarrow q$	$p = q$; ε is the empty sequence
$p-v_1 \ldots v_n\rightarrow q$	there exist $p_k, 1 \leq k < n$, such that $p-v_1\rightarrow p_1 \ldots p_{n-1}-v_n\rightarrow q$
$p-\gamma\rightarrow$	there exists q such that $p-\gamma\rightarrow q$
$p\not-\gamma\rightarrow$	no q exists such that $p-\sigma\rightarrow q$
$Tr(p)$	$Tr(p) = \{\gamma \in \Gamma^* \mid p-\gamma\rightarrow\}$; $Tr(\mathsf{S}) = Tr(s_0)$
γ^{in}	for $\gamma \in \Gamma^*$, $\gamma^{in} \in X^*$ is the input part of γ
$p-x_i \ldots x_n\rightarrow q$	there exists $\gamma \in \Gamma^*$ such that $p-\gamma\rightarrow q$ and $x_i \ldots x_n = \gamma^{in}$
$out(p, V)$	$out(p, V) = \{v \mid v^{in} \in V \wedge p-v\rightarrow\}$ for $V \subseteq X^*$

Table 1: Notations for finite state machines

still be nondeterministic. We note that any non-observable NFSM can be transformed into an equivalent observable NFSM (ONFSM) [20].

An FSM can also be represented by a directed graph in which the nodes are the states and each directed edge with a label is a transition linking two states, as shown in Figure 1. For the convenience of the presentation, we use $\mathsf{I}, \mathsf{P}, \mathsf{S}, \ldots$ to represent FSMs; $I, P, Q, \ldots$, for sets of states; $a, b, c, \ldots$, for inputs or outputs; and $i, p, q, s \ldots$, for states. Other notations are given in Table 1.

Conformance Relations

The following conformance relations are usually used to formalize the conformance requirements for conformance testing of the finite state machines. We say that an implementation M conforms to a specification S if the chosen conformance relation holds between M and S.

Definition 2 (*Reduction of nondeterminism*): The reduction relation between two states p and q in NFSMs, written $p \leq q$, holds iff $Tr(p) \subseteq Tr(q)$.
Given two FSMs S and M, we say that M is a reduction of S, written $\mathsf{M} \leq \mathsf{S}$, iff $m_0 \leq s_0$.

According to this conformance relation [23], all output sequences that are produced by an implementation in response to any input sequence should be described by its specification.

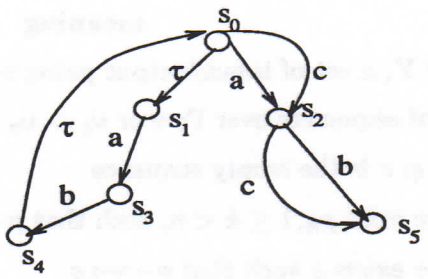


Figure 2: An LTS graph

Definition 3 (*Equivalence*): The equivalence relation between two states p and q in FSMs, written $p \sim q$, holds iff $Tr(p) = Tr(q)$.
Given two FSMs S and M, we say that M is equivalent to S, written $\mathsf{S} \sim \mathsf{M}$, iff $s_0 \sim m_0$.

The above definition of the equivalence relation is given in [20] for NFSMs, similar to that in [16, 10] for DFSMs. This conformance relation requires that an implementation has the same traces as its specification.

It can be shown that in the case of nondeterminism the reduction relation is a pre-order such that $\mathsf{M} \sim \mathsf{S}$ iff $\mathsf{M} \leq \mathsf{S}$ and $\mathsf{S} \leq \mathsf{M}$, and in the case of determinism, the reduction relation is the equivalence relation [24].

2.2 Labeled Transition System

Definition 4 (*Labeled transition system (LTS)*): A labeled transition system is a 4-tuple $< S, \Sigma, \Delta, s_0 >$, where:
- S is a finite non-empty set of states, $s_0 \in S$, is the initial state.
- Σ is a finite set of labels, called observable actions; $\tau \notin \Sigma$ is called an internal action.
- $\Delta \subseteq S \times (\Sigma \cup \{\tau\}) \times S$ is a transition set. An element (p, μ, q) is denoted by $p - \mu \rightarrow q$.

A state p is *unstable* if there exists $q \in S$ such that $p - \tau \rightarrow q \in \Delta$; otherwise it is *stable*. If there exists $p - \mu \rightarrow q \in \Delta$, p is said to be *active*; otherwise it is *inactive*. A stable LTS has no unstable states, whereas a unstable LTS has such states.

An LTS is said to be *nondeterministic* if it is unstable or there exist $p - a \rightarrow p_1, p - a \rightarrow p_2 \in \Delta$ but $p_1 \neq p_2$. In a *deterministic* LTS, the outgoing transitions of any state are uniquely labeled.

Here we also use $\mathsf{I}, \mathsf{P}, \mathsf{S}, \ldots$ to represent LTSs; $I, P, Q, \ldots$, for sets of states; $a, b, c, \ldots$, for actions; and $i, p, q, s \ldots$, for states. The notations are shown in Table 2 that are relevant to a given LTS, as introduced in [6].

An LTS can also be represented by a directed graph where nodes are states and labeled edges are transitions. An LTS graph is shown in Figure 2.

We can note that the $Ref(p, \sigma)$ includes all the sets of actions that may be refused by some state in p **after** σ. If a set A is refused, obviously, each set $B \subseteq A$ is refused as well. Thus, we may consider a minimal representation of $Ref(p, \sigma)$, denoted $\lceil Ref(p, \sigma) \rceil$, by deleting each element in $Ref(p, \sigma)$ that is a subset of another. Generally, for any set R, $\lceil R \rceil = R \backslash \{A \mid \exists B \in R \ (A \subset B)\}$.

We also denote $Acc(p, \sigma) = \wp(\Sigma) \backslash Ref(p, \sigma)$ and $Acc(p) = Acc(p, \varepsilon)$, where $\wp(\Sigma)$ is the power set of the set Σ. $Acc(p, \sigma)$ includes every set of the actions among which there exists at least one action that must be accepted by all the states in p **after** σ. We note that for any two sets of actions A and B, if $A \supseteq B$ then that B must be accepted implies that A must also be accepted. Therefore a minimal representation of $Acc(p, \sigma)$, denoted $\lfloor Acc(p, \sigma) \rfloor$, can be obtained by deleting each element in $Acc(p, \sigma)$ that is a superset of another. Formally, for any set R, $\lfloor R \rfloor = R \backslash \{A \mid \exists B \in R \ (A \supset B)\}$.

In the case of nondeterminism, after an observable action sequence, an LTS may enter a number of different states. In order to consider all these possibilities, a state subset (multi-state [15]), which contains all the states reachable by the LTS after this action sequence, is used.

Definition 5 (*Multi-state set*): The multi-state set of LTS S is the set $\Pi_S = \{S_i \subseteq S \mid \exists \sigma \in Tr(\mathsf{S}) \ (\mathsf{S} \textbf{ after } \sigma = S_i)\}$.

Note that the empty sequence ε is in Σ^*. Therefore $S_0 = s_0$ **after** ε is in the multi-state set, and it is called the *initial multi-state*. The multi-state set can be obtained by a known algorithm which performs the deterministic transformation of a nondeterministic LTS with the trace-equivalence [18, 15, 9]. For Figure 1, the multi-state set is $\{\{s_0, s_1\}, \{s_2, s_3\}, \{s_2\}, \{s_0, s_1, s_4, s_5\}, \{s_5\}\}$. Obviously, each LTS has one and only one multi-state set.

notation	meaning
Σ^*	set of sequences over Σ; σ or $a_1 \ldots a_n$ denotes such a sequence
$p - \mu_1 \ldots \mu_n \rightarrow q$	there exists p_k, $1 \leq k < n$, such that $p - \mu_1 \rightarrow p_1 \ldots p_{n-1} - \mu_n \rightarrow q$
$p = \varepsilon \Rightarrow q$	$p - \tau^n \rightarrow q$ $(1 \leq n)$ or $p = q$ (note: τ^n means n times τ)
$p = a \Rightarrow q$	there exist p_1, p_2 such that $p_1 = \varepsilon \Rightarrow p_1 - a \rightarrow p_2 = \varepsilon \Rightarrow q$
$p = a_1 \ldots a_n \Rightarrow q$	there exists p_k, $1 \leq k < n$, such that $p = a_1 \Rightarrow p_1 \ldots p_{n-1} = a_n \Rightarrow q$
$p = \sigma \Rightarrow$	there exists q such that $p = \sigma \Rightarrow q$
$p \neq \sigma \Rightarrow$	no q exists such that $p = \sigma \Rightarrow q$
$init(p)$	$init(p) = \{a \in \Sigma \mid p = a \Rightarrow\}$
p **after** σ	p **after** $\sigma = \{q \in S \mid p = \sigma \Rightarrow q\}$; S **after** $\sigma = s_0$ **after** σ
$Tr(p)$	$Tr(p) = \{\sigma \in \Sigma^* \mid p = \sigma \Rightarrow\}$; $Tr(\mathrm{S}) = Tr(s_0)$
$Ref(p, \sigma)$	$Ref(p, \sigma) = \{A \subseteq \Sigma \mid \exists q \in p$ **after** $\sigma \; \forall a \in A \; (q \neq a \Rightarrow)\}$; $Ref(p) = Ref(p, \varepsilon)$; $Ref(\mathrm{S}, \sigma) = Ref(s_0, \sigma)$

Table 2: Basic notations for labeled transition systems

$Tr(P)$	$=$	$\bigcup_{(p \in P)} Tr(p)$
$init(P)$	$=$	$\bigcup_{(p \in P)} init(p)$
P **after** σ	$=$	$\bigcup_{(p \in P)} p$ **after** σ
$Ref(P, \sigma)$	$=$	$\bigcup_{(p \in P)} Ref(p, \sigma)$
$Ref(P)$	$=$	$\bigcup_{(p \in P)} Ref(p)$
$P = \sigma \Rightarrow Q$	$\Leftrightarrow$	$Q = \{q \in S \mid \exists p \in P \; (p = \sigma \Rightarrow q)\}$

Table 3: Additional notations for labeled transition systems

After any observable sequence, a nondeterministic system reaches a unique multi-state. Thus from the test perspective, it makes sense to identify multi-states, rather than single states. This viewpoint is reflected in the FSM realm by the presentation of a nondeterministic FSM specification as an observable FSM [20], in which each state is a subset of states of the non-observable FSM. The viewpoint is also reflected by the *refusal graphs* [14], in which a node corresponds to a multi-state.

Notations used for multi-states are shown in Table 3. In this table, $P, Q \subseteq S$, where S is the state set of LTS S. From the notations, it is easy to show that $Tr(s_0) = Tr(S_0)$ and s_0 **after** $\sigma = S_0$ **after** σ.

Conformance Relations

There are different criteria determining whether an implementation conforms to a specification [35]. Using different criteria, various conformance relations have been proposed for comparing labeled transition systems. Under the assumption that systems communicate by rendezvous, the following conformance relations are considered in this paper.

Definition 6 (*Conformance relations*): Given two LTSs M and S,

Trace equivalence ($\approx_t$):	$\mathsf{M} \approx_t \mathsf{S}$ iff $Tr(m_0) = Tr(s_0)$.
Failure reduction ($\leq_f$):	$\mathsf{M} \leq_f \mathsf{S}$ iff $\forall \sigma \in \Sigma^* \ Ref(m_0, \sigma) \subseteq Ref(s_0, \sigma)$.
Nondeterministic reduction ($\leq_n$):	$\mathsf{M} \leq_n \mathsf{S}$ iff $\mathsf{M} \approx_t \mathsf{S}$ and $\mathsf{M} \leq_f \mathsf{S}$.
Failure equivalence ($\approx_f$):	$\mathsf{M} \approx_f \mathsf{S}$ iff $\mathsf{M} \leq_f \mathsf{S}$ and $\mathsf{S} \leq_f \mathsf{M}$.

The trace equivalence relation requires that an implementation has the same traces as its specification. The failure reduction relation stipulates that everything of the implementation is allowed by its specification, not only the traces but also the actions refused after any observable action sequence (deadlock). The failure equivalence further states not only that everything the implementation does must be allowed by its specification, but also that everything prescribed by the specification should be implemented by the implementation. The nondeterministic reduction relation is the failure reduction relation with the trace equivalence. We note that for deterministic systems, the trace equivalence relation is the failure equivalence relation. Nondeterminism differentiates the two relations, so nondeterministic reduction corresponds to implementation choices for a nondeterministic specification. The trace equivalence relation belongs to *trace semantics* [17], while the other three relations belong to *failure semantics* [11].

The following relationships hold among these conformance relations: $\approx_f \Rightarrow \leq_n$; $\leq_n \Rightarrow \leq_f$; $\leq_n \Rightarrow \approx_t$; $\leq_f$ and $\approx_t \Rightarrow \leq_n$.

Note that there is a slight difference in the term "failure" between [11] and this paper. In [11], a failure means that the system stagnates in a state p with a set A of offered actions, that is, for all $\mu \in A \cup \{\tau\}$ $p \not\xrightarrow{\mu}$. In this paper this is weakened to $p \not\Rightarrow a \Rightarrow$ for all $a \in A$ [6], which says that communication of the system with the environment is blocked if A is offered in p (communication failure). Conformance testing, unlike verification, is to check the observable behavior of an implementation, taking no care of its internal actions or fairness. From this, the failure semantics in this paper equates with the semantics

based on *testing equivalence* [12], which is defined by observing the executions of a class of tests. We prefer characterizing a conformance relation in terms of properties of the system itself to defining it by using a class of tests.

3 Conformance Testing as Experiments

Conformance testing is a finite set of experiments, in which a set of test cases, derived from a specification according to a conformance relation, are applied by a tester or experimenter to the implementation under test (IUT), such that from the results of the execution of the test cases, it can be concluded whether or not the implementation conforms to the specification.

In testing systems that are modeled by FSMs, the testing framework is intuitive and simple. A test case is usually represented as an input sequence with the specified reactions, and testing is an exchange of input/output interactions between a tester and IUT, in which the test case is executed in such a way that the tester stimulates in turns an input to the IUT, observes the output of the IUT and compares it with the specified reaction(s). In the case of nondeterminism, for a given input, several possible output reactions may be specified, and the next input may depend on previous output. In the LTS formalism, however, since there are various conformance relations, and the rendezvous interactions make no distinction between input and output, a testing framework is needed to answer the following questions: How are test cases structured for a given conformance relation; what constitute observations; how is the verdict assigned after a finite amount of testing; and how can one obtain a well-defined confidence that the implementation under test conforms to its specification?

3.1 Test cases, Testing and Verdicts

The behavior of the tester during a test experiment is defined by the test case used in this experiment. Thus a test case is a specification of behavior, which, like other specifications, can be represented as an LTS [34]. An experiment should last for a finite time, so a test case should have no infinite behavior. Moreover, the tester should have certain control over the testing process, so nondeterminism in a test case is undesirable.

Definition 7 (*Test cases and test suite*): Given an LTS specification S $=< S, \Sigma, \Delta, s_0 >$, its test case T is a 5-tuple $< T, \Sigma_T, \Delta_T, t_0, \ell >$ where:

- $\Sigma_T \subseteq \Sigma$;
- $< T, \Sigma_T, \Delta_T, t_0 >$ is a deterministic, tree-structured LTS such that for each $p \in T$ there exists exactly one $\sigma \in \Sigma_T^*$ with $t_0 = \sigma \Rightarrow p$;
- $\ell : T \rightarrow \{\textbf{pass}, \textbf{fail}, \textbf{inconclusive}\}$ is a state labeling function.

A test suite for S is a finite set of test cases.

From this definition, the behavior of test case T is finite, since T and Σ are finite as defined. Moreover, a trace of T uniquely determines a single state in T, so we also write $\ell(\sigma) = \ell(t)$ for $\{t\} = t_0$ **after** σ.

The rendezvous interactions between a test case T and the IUT M can be formalized by the synchronization operator "||" of LOTOS. Let t and m be the current states of T and M, repectively, the test execution proceeds by following inference rules:

$$\begin{array}{|ll|} \hline m \text{-}\tau\text{-}\!\!\rightarrow m' & \vdash \quad t \parallel m \text{-}\tau\text{-}\!\!\rightarrow t \parallel m' \\ t \text{-}a\text{-}\!\!\rightarrow t', m \text{-}a\text{-}\!\!\rightarrow m', \ a \in \Sigma_T & \vdash \quad t \parallel m \text{-}a\text{-}\!\!\rightarrow t' \parallel m' \\ \hline \end{array}$$

When $t_0 \parallel m_0$ after an observable action sequence σ reaches a *deadlock*, that is, there exists a state $p \in T \times M$ such that for all actions $a \in \Sigma$, $t_0 \parallel m_0 = \sigma \Rightarrow p \neq a \Rightarrow$, we say that this experiment completes a *test run*. In the light of the tester, each state of the test case T is characterized by the set of actions out of this state which are offered by the tester to the IUT. If this set is empty, we say that the test case has reached an inactive state; the other states are active. The completion of a test run means that an inactive state is reached or the set of actions offered as next interactions are refused by the IUT.

Usually, a test case is designed to check some particular conformance requirement, sometimes called a *test purpose*. We define here the test purpose of a test case T, written $Pur(\mathsf{T})$, to be $Pur(\mathsf{T}) = \{\sigma \in Tr(t_0) \mid \ell(\sigma) = \textbf{pass}\}$. If $Pur(\mathsf{T}) = \emptyset$, then T should have at least one **fail** label and its purpose is to check that the IUT does not implement some specific unexpected behavior.

LTSs are generally supposed to be nondeterministic. In order to test nondeterministic implementations, one usually makes a so-called *complete-testing assumption:* it is possible,

by applying a given test case to the implementation a finite number of times, to exercise all possible execution paths of the implementation which are traversed by the test case [15, 21, 20]. Without such an assumption, no test suite can guarantee full fault coverage (in terms of conformance relations) for nondeterministic implementations. Therefore any experiment, in which M is tested by T, should include several test runs and lead to a complete set of observations $Obs_{(T,M)} = \{\sigma \in Tr(t_0) \mid \exists p \in T \times M, \forall a \in \Sigma\ (t_0 \parallel m_0 = \sigma \Rightarrow p \neq a \Rightarrow)\}$.

Based on $Obs_{(T,M)}$, which are the results of testing with the test case T, the success or failure of the testing needs to be concluded. The way a verdict is drawn from $Obs_{(T,M)}$ is the "verdict assignment" for T: $Obs_{(T,M)} \Rightarrow$ {**pass**, **fail**}. The verdict **pass** means success, which, intuitively, should mean that no unexpected behavior is found and the test purpose has been achieved. Therefore, we define the verdict assignment as follows:

Definition 8 (*Verdict assignment* v): Given an IUT M, a test case T, let $Obs_{fail} = \{\sigma \in Obs_{(T,M)} \mid \ell(\sigma) = \textbf{fail}\}$ and $Obs_{pass} = \{\sigma \in Obs_{(T,M)} \mid \ell(\sigma) = \textbf{pass}\}$,

$$v(Obs_{(T,M)}) = \begin{cases} \textbf{pass} & \text{if } Obs_{fail} = \emptyset \wedge Obs_{pass} = Pur(T) \\ \textbf{fail} & \text{otherwise.} \end{cases}$$

All observations $\sigma \in Obs_{(T,M)}$ and the corresponding labels $\ell(\sigma)$ together form a verdict for T. Single labels do not give such a verdict, so in this sense a state labeling function is not a verdict function [33].

3.2 Fault Model and Fault Coverage

The goal of conformance testing is to gain confidence in an implementation under test concerning its conformance with the specification. Increased confidence is normally obtained through time and effort spent in testing the implementation, which, however, are limited by practical and economical considerations. In order to have a more precise measure of the effectiveness of testing, a fault model and fault coverage criteria [3] are introduced. We here take the mutation approach [3], that is, we define the fault model to be a set $\mathcal{F}$ of all faulty LTS implementations considered. Based on $\mathcal{F}$, a test suite with complete fault coverage for a given LTS specification with respect to a given conformance relation can be defined as follows.

Definition 9 (*Complete test suite*): Given an LTS specification S, a conformance relation r and a fault model $\mathcal{F}$, a test suite TS for S w.r.t. r is said to be complete over $\mathcal{F}$, if for any M in $\mathcal{F}$, r holds between M and S iff M passes T for each T in TS.

In the following, we consider a particular fault model of the form $\mathcal{F}(m)$ which consists of all LTS implementations over the alphabet of the LTS specification S and with at most m multi-states, where m is a known integer. We say that a test suite is m-complete for a given specification if it is complete over the fault model $\mathcal{F}(m)$.

A complete test suite guarantees that for any implementation M in $\mathcal{F}$, if M passes all test cases, it is a conforming implementation of the given specification with respect to the given conformance relation, and any faulty implementation in $\mathcal{F}$ is detected by failing at least one test case in the test suite.

The similar notions of fault models and complete test suites are usually also used in conformance testing of the FSM model [3]. In the FSM formalism, for an FSM implementation M and a test case or *test sequence* $\rho \in X^*$, the set of observations $Obs_{(\rho,\mathrm{M})} = \{\gamma \in \Gamma^* \mid \gamma \in Tr(m_0) \wedge \gamma^{in} = \rho\}$, and thus, given an FSM specification S, we say that M passes ρ, if $Obs_{(\rho,\mathrm{M})} = Obs_{(\rho,\mathrm{S})}$ for the equivalence relation or $Obs_{(\rho,\mathrm{M})} \leq Obs_{(\rho,\mathrm{S})}$ for the reduction relation.

The fault models for an FSM are $\mathcal{F}_{fsm}$, which is a set of all faulty FSM implementations considered, and $\mathcal{F}_{fsm}(m)$, which consists of all FSM implementations over the alphabet of S and with at most m states in their observable forms [20]. Thus, similar to the above definition of a complete test suite for an LTS, we can define a complete test suite and an m-complete test suite for an FSM.

4 State Labelings of Test Cases

Given a specification S and a conformance relation, the state labeling function of test cases T must be "sound", that is, for any implementation M, if the relation holds between M and S, then M passes T. In the following, we present the state labeling functions for the conformance relations discussed in this paper.

Trace Equivalence

In the context of trace equivalence, a conforming implementation should have the same traces as a given specification. Therefore each test case specifies certain sequences of actions, which are either valid or invalid traces of the specification, to verify that an IUT has implemented the valid ones and not any of the invalid ones. If such a sequence specified in a test case is implemented in the IUT, then there must exist a test run such that the sequence is observed. If the observed sequence is a valid trace, a **pass** verdict should be assigned to this test run, which implies that the state after the sequence in the test case should be labeled with **pass**; no conclusion could be made if a test run completes before the end of the sequence, so the tail states of all the proper prefixes of the sequence should be labeled with **inconclusive**. On the other hand, if the observed sequence is an invalid trace, a **fail** verdict should be assigned to this test run, which implies that the state after the sequence in the test case should be labeled with **fail**. Based on this reasoning, we conclude that all test cases for trace equivalence must be of the following form:

Definition 10 *Test cases for trace equivalence.* Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\approx_t$, if, for $\sigma \in Tr(t_0)$, $\{t_i\} = t_0$ **after** σ, the state labeling of T satisfies

$$\ell_{\approx t}(t_i) = \begin{cases} \textbf{pass} & \sigma \in Tr(s_0) \wedge init(t_i) \cap init(s_0 \textbf{ after } \sigma) = \emptyset \\ \textbf{fail} & \sigma \in Tr(t_0) \backslash Tr(s_0) \\ \textbf{inconclusive} & \text{otherwise.} \end{cases}$$

A test suite for S w.r.t. $\approx_t$ is a set of test cases for S w.r.t. $\approx_t$.

Note that in T, σ leading to a state with **pass** is not a prefix of any other sequence in $Tr(\mathsf{S}) \cap Tr(\mathsf{T})$.

Proposition 1 *Given a test case* T *for* S *w.r.t.* $\approx_t$*, for any LTS* M*, if* M $\approx_t$ S*, then* M *passes* T.

A test case for the LTS given in Figure 2 in respect to trace equivalence is shown in Figure 3 (a).

Failure Equivalence

In the context of failure equivalence, a conforming implementation and its specification

should have the same refusal set after any observable action sequence. Therefore each test case for this relation should be designed such that out of each of its states a certain set of actions is specified as the set of actions offered by the tester in the corresponding testing interaction. (Note that an empty set is assumed in an inactive state.) It is expected that, in any interaction, the IUT may refuse the offered set iff the offered set may also be refused by the specification after the same sequence.

If the IUT implements any sequence in a test case, there must exist a test run such that this sequence is observed; furthermore if the set of actions offered in the last interaction of this test run is a set in the refusal set of the specification after the sequence, then a **pass** verdict should be assigned to this test run, which implies that the state after the sequence in the test case should be labeled with **pass**. On the other hand, if the offered set of actions is not in the refusal set of the specification after the sequence, then a **fail** verdict should be given to this test run, that is, the state after the sequence in the test case should be labeled with **fail**. Based on this reasoning, we conclude that all test cases for failure equivalence must be of the following form:

Definition 11 (*Test cases for failure equivalence*): Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\approx_f$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0$ **after** σ, the state labeling of T satisfies

$$\ell_{\approx_f}(t_i) = \begin{cases} \textbf{pass} & \text{if } init(t_i) \in Ref(s_0 \textbf{ after } \sigma) \\ \textbf{fail} & \text{otherwise.} \end{cases}$$

A test suite for S w.r.t. $\approx_f$ is a set of test cases for S w.r.t. $\approx_f$.

Note that if σ is not is a valid trace of S, then $Ref(s_0$ **after** $\sigma)$ is an empty set, and from this t_i is labeled with **fail** because no matter whether $init(t_i)$ is an empty set or not, $init(t_i)$ is not in the empty set $Ref(s_0$ **after** $\sigma)$.

Proposition 2 *Given a test case* T *for* S *w.r.t.* $\approx_f$, *for any LTS* M, *if* M $\approx_f$ S, *then* M *passes* T.

A test case for the LTS given in Figure 2 in respect to failure equivalence is shown in Figure 3 (d).

Failure Reduction

For the failure reduction relation, a conforming implementation may be any implementation whose refusal set, after a given action sequence, is a subset of the refusal set of its specification after the same sequence. Thus for this relation, we only need to check the unspecified deadlocks in a given implementation. For a test case for this relation, a state should be labeled with **fail** if the set of actions out of it is not in the refusal set of the specification after the sequence to the state; otherwise with **inconclusive.**

Definition 12 (*Test cases for failure reduction*): Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\leq_f$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0$ **after** σ, the state labeling of T satisfies

$$\ell_{\leq f}(t_i) = \begin{cases} \textbf{fail} & \text{if } init(t_i) \notin Ref(s_0 \textbf{ after } \sigma) \\ \textbf{inconclusive} & \text{otherwise.} \end{cases}$$

A test suite for S w.r.t. $\leq_f$ is a set of test cases for S w.r.t. $\leq_f$.

Proposition 3 *Given a test case* T *for* S *w.r.t.* $\leq_f$, *for any LTS* M, *if* $\mathsf{M} \leq_f \mathsf{S}$, *then* M *passes* T.

A test case for the LTS given in Figure 2 in respect to failure reduction is shown in Figure 3 (b).

Nondeterministic Reduction

Since nondeterministic reduction is the combination of failure reduction and trace equivalence, the state labeling of test cases for nondeterministic reduction can be obtained by combining the corresponding state labeling functions of test cases for these two relations.

Definition 13 (*Test cases for nondeterministic reduction*): Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\leq_n$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0$ **after** σ, the state labeling of T satisfies

$$\ell_{\simeq f}(t_i) = \begin{cases} \textbf{pass} & \text{if } init(t_i) \in Ref(s_0 \textbf{ after } \sigma) \wedge init(t_i) \cap init(s_0 \textbf{ after } \sigma) = \emptyset \\ \textbf{fail} & \text{if } init(t_i) \notin Ref(s_0 \textbf{ after } \sigma) \\ \textbf{inconclusive} & \text{otherwise.} \end{cases}$$

A test suite for S w.r.t. $\leq_n$ is a set of test cases for S w.r.t. $\leq_n$.

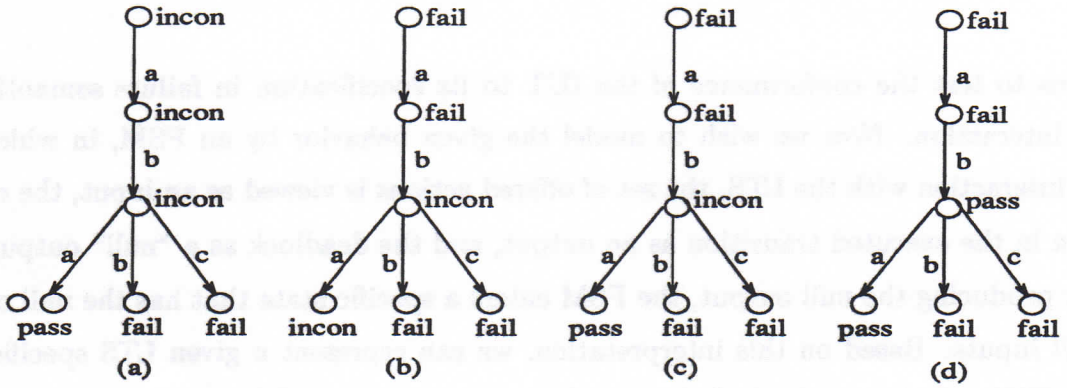


Figure 3: Test cases for different relations: (a) $\approx_t$, (b) $\leq_f$, (c) $\leq_n$ and (d) $\approx_f$

From the above definition, we can note that label **pass** is designated to check trace equivalence, while label **fail** to check the failure reduction. $init(t_i) \in Ref(s_0 \textbf{ after } \sigma)$ implies $\sigma \in Tr(s_0)$ and $init(t_i) \cap init(s_0 \textbf{ after } \sigma) = \emptyset$ implies that sequence σ is not a prefix of another sequence in $Tr(s_0) \cap Tr(t_0)$.

Proposition 4 *Given a test case* T *for* S *w.r.t.* $\leq_n$*, for any LTS* M*, if* $\mathsf{M} \leq_n \mathsf{S}$*, then* M *passes* T.

A test case for the LTS given in Figure 2 in respect to nondeterministic reduction is shown in Figure 3 (c). We note that in this case we have chosen the four test cases in Figure 3 to have the same underlying LTS, but clearly they have different state labelings.

5 Transforming LTSs to FSMs

We focus in this section on how to represent the behavior specified by a given LTS, based on trace semantics or failure semantics, respectively, using an FSM model.

5.1 General Idea

In the context of conformance testing, an LTS IUT is viewed as a black box, which, in each interaction, chooses autonomously one action from a set of offered actions to execute a transition, or it blocks all the actions [35]. According to the LOTOS semantics, no further action can be executed after the deadlock occurs. Under the assumption that at least one action is offered in each interaction, we have $2^{|\Sigma|} - 1$ possible sets of offered

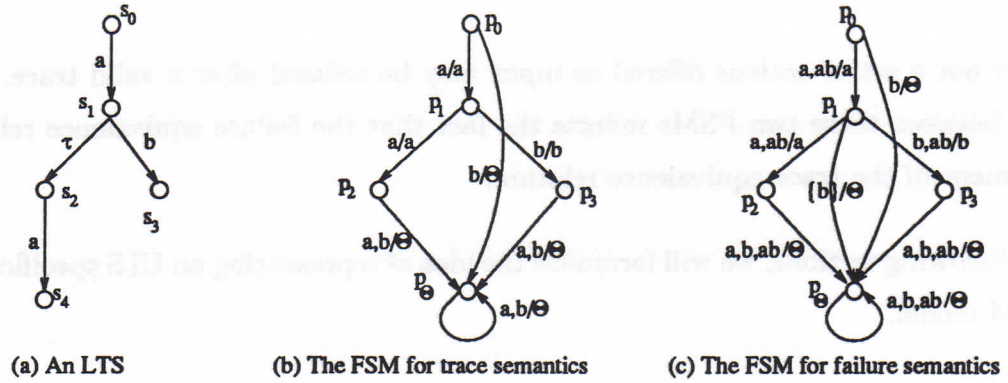
actions to test the conformance of the IUT to its specification in failure semantics for each interaction. Now we wish to model the given behavior by an FSM, in which, for each interaction with the LTS, the set of offered actions is viewed as an input, the chosen action in the executed transition as an output, and the deadlock as a "null" output [23]. After producing the null output, the FSM enters a specific state that has the null output for all inputs. Based on this interpretation, we can represent a given LTS specification as an FSM, which models the behavior of the corresponding LTS in trace semantics or in failure semantics, respectively.

In the case that we are only interested in trace semantics, all relevant properties can be tested by offering single actions. We therefore assume a simplified FSM model, which defines the behavior only for single action offers, thereby reducing the number of inputs from $2^{|\Sigma|}-1$ to $|\Sigma|$. For the case that the environment offers several actions simultaneously, we assume that a demon chooses arbitrarily one of the offered actions for execution by the FSM. The deadlock properties of the system are not completely modeled. Therefore the implementation may deadlock before the end of a possible test case. We consider this as an inconclusive test result, and as usual for nondeterministic systems, the test should be repeated.

Trace Semantics

Given an LTS, we wish to construct an FSM that produces as output all of the traces of the LTS and signals by the null output Θ that the given input action cannot form a valid trace of the LTS. As a simple example, Figure 4 (a) shows an LTS specification and (b) its corresponding FSM representation in trace semantics. In (a), the LTS has the alphabet set $\Sigma = \{a, b\}$. In (b), the FSM has the input set $X = \Sigma$, the output set $Y = \{a, b, \Theta\}$; and each transition is labeled with an input/output pair, in which the output is either the same as the input or Θ. For example, a/a means that when a is offered, a can be executed, and b/Θ means that when action b is offered, nothing but deadlock can be observed. To keep the picture clear, label $a, b/\Theta$ corresponds to the pairs a/Θ and b/Θ.

The transformation from an LTS to the FSM involves the mapping of the LTS multi-states onto the FSM states. In the above example, $\{s_0\}$ is mapped to p_0, $\{s_1, s_2\}$ to p_1, $\{s_3\}$ to p_3 and $\{s_4\}$ to p_2.



(a) An LTS (b) The FSM for trace semantics (c) The FSM for failure semantics

Figure 4: Representation of an LTS using the FSM model

The sink state s_Θ in our FSM model represents the situation of the corresponding LTS after any deadlock has occurred and before a reset is applied. Once the deadlock is detected, the tester has to stop the current test run, regardless of whether it has been completed successfully [6]. This is modeled in our FSM by the sink state s_Θ and all transitions to/from s_Θ which output the null output Θ.

Failure Semantics

We can also construct from a given LTS such an FSM that not only produces as output all the traces of the LTS, but also signals by the null output Θ that certain sets of actions on its input form a refusal set of the LTS after a given trace. An example of the FSM representation for the LTS in Figure 4 (a) in the failure semantics is shown in Figure 4 (c). This FSM has the input set $X = \{\{a\}, \{b\}, \{a, b\}\}$ and the output set $Y = \{a, b, \Theta\}$; and each transition is labeled with an input/output pair, in which the output is either an action in the input or Θ. For example, $\{a, b\}/a$, in which $\{a, b\}$ is the set of offered actions and a is the action that is chosen for execution. If the output is Θ, then a deadlock may be observed for the set of offered actions. (Note that also for a clear picture, in the figure the labels $a, ab/a$ represent $\{a\}/a$ and $\{a, b\}/a$, respectively.) The mapping from the multi-states of the LTS to the states of the FSM as well as the sink state s_Θ are the same as for the trace semantics.

It can be shown that, given an LTS, the FSM constructed for trace semantics is a deterministic submachine of the FSM for failure semantics. Both machines have the same states. The trace FSM only determines whether or not an input action can form a valid trace for the corresponding LTS. So does the failure FSM, and it also indicates

whether or not a set of actions offered as input may be refused after a valid trace. The difference between these two FSMs reflects the fact that the failure equivalence relation is a refinement of the trace equivalence relation.

In the following sections, we will formalize the idea of representing an LTS specification by an FSM model.

5.2 Trace Finite State Machines

The FSM model for a given LTS specification in trace semantics, called the corresponding *trace finite state machine* (TFSM), is defined as follows.

Definition 14 *Trace finite state machine w.r.t. LTS.* Given an LTS S=$< S, \Sigma, \Delta, s_0 >$, a trace finite state machine w.r.t. S, is a finite state machine $\mathsf{P}_T =< P, X, Y, h, p_0 >$, such that:

- $X = \Sigma$.
- $Y \backslash \{\Theta\} = \Sigma$, where Θ represents the null output.
- P is a finite state set, and the sink state s_Θ is in P.
- Let Π_S be the multi-state set of S. There exists a one-to-one mapping $\psi : \Pi_S \rightarrow P \backslash \{s_\Theta\}$ and for all $S_i \in \Pi_S$ and all $a \in X$,
  - $(\psi(S_j), a) \in h(\psi(S_i), a)$ iff $S_i = a \Rightarrow S_j$;
  - $(s_\Theta, \Theta) \in h(\psi(S_i), a)$ iff $a \in \Sigma \backslash init(S_i)$;
  - $\{(s_\Theta, \Theta)\} = h(s_\Theta, a)$.

According to the definition, it is possible to construct from the LTS S the corresponding TFSM P_T. Figure 4 (b) is an example of the TFSM with respect to the LTS in Figure 4 (a). From the above definition, it can be seen that all transitions in the TFSM are labeled with a pair of the form "a/a" or "b/Θ". Furthermore, each trace of the TFSM is a sequence of pairs of the form "a/a", possibly followed by a sequence of one or several pairs "b/Θ". It is implied that once the first Θ occurs, the TFSM enters the special sink state s_Θ, and outputs Θ for any subsequent input.

Given an action sequence $\sigma \in \Sigma^*$, we use $pt(\sigma)$ to represent an input/output sequence in which both of its input part and output part are σ. Formally, we define $pt(\sigma.a) = pt(\sigma).a/a$ and $pt(\varepsilon) = \varepsilon$. TFSMs have the following properties.

Proposition 5 *Any TFSM is a deterministic FSM.*

Proposition 6 *Given an LTS* S *and its corresponding TFSM* P_T*, for all* $\sigma \in \Sigma^*$ *and all* $\gamma = pt(\sigma) \in \Gamma^*$, $\sigma \in Tr(s_0)$ *iff* $\gamma \in Tr(p_0)$.

Proposition 6 comes directly from the definitions of TFSMs and the multi-state set. This proposition shows the way in which an I/O FSM models the behavior of an LTS in trace semantics. The TFSM and its corresponding LTS exhibit identical behavior: any action sequence is a trace of the LTS iff it is accepted and produced by its TFSM. On the other hand, since the TFSM is completely specified, any action sequence that is not a trace of the LTS corresponds to a trace of TFSMs with Θ outputs.

Accordingly, the trace equivalence relation in LTSs directly corresponds to the equivalence relation in FSMs, as stated by the following theorem.

Theorem 1 *For any given two LTSs* R, S *and their corresponding TFSMs* P_T, Q_T, $\mathsf{R} \approx_t \mathsf{S}$ *iff* $\mathsf{P}_T \sim \mathsf{Q}_T$.

By virtue of Theorem 1, the test cases for the TFSM model can be used to test the LTS implementations with respect to their specifications for the trace equivalence relation. Now it becomes clear that the methods based on DFSMs [10, 16, 22, 27] are fully applicable to derive test cases from LTS specifications through TFSMs. However, the obtained test cases should be further transformed to test cases in the LTS context, because LTSs have a different, i.e. rendezvous, interface to interact with their environment. We explain this transformation in Section 6.

5.3 Failure Finite State Machines

In this section, we present the FSM model for a given LTS specification in failure semantics. It is similar to the TFSM construction, and is called a *failure finite state machine*, or FFSM. In the FFSM, sets of actions, along with single actions, are treated as inputs.

Definition 15 *Failure finite state machine w.r.t. LTS.* Given an LTS S=$< S, \Sigma, \Delta, s_0 >$, a failure finite state machine w.r.t. S, is a finite state machine $\mathsf{P}_F =< P, X, Y, h, p_0 >$, such that:

- $X = \wp(\Sigma)\backslash\{\emptyset\}$.
- $Y\backslash\{\Theta\} = \Sigma$, where Θ represents the null output.
- P is a finite state set, and the sink state s_Θ is in P.
- Let Π_S be the multi-state set of S. There exists a one-to-one mapping $\psi : \Pi_S \rightarrow P\backslash\{s_\Theta\}$ and for all $S_i \in \Pi_S$ and all $A \in X$,
  - $(\psi(S_j), a) \in h(\psi(S_i), A)$ iff $a \in A$ and $S_i = a \Rightarrow S_j$, or
  - $(s_\Theta, \Theta) \in h(\psi(S_i), A)$ iff $A \in Ref(S_i)$;
  - $\{(s_\Theta, \Theta)\} = h(s_\Theta, A)$.

Figure 4 (c) shows an example of the FFSM with respect to the LTS in Figure 4 (a). From the above definition, it can be seen that all transitions in the FFSM are labeled with a pair of the form "A/a" where $a \in A \subseteq \Sigma$, or "A/Θ". Similar to the TFSM, each trace of the FFSM is a sequence of pairs of the form "A/a", possibly followed by a sequence of one or several pairs "A/Θ"; and once the first Θ occurs, the FFSM also enters the state s_Θ, and outputs Θ for any subsequent input.

The following properties hold between multi-states S_i of an LTS S and the corresponding states p_i of the FFSM P_F: $\{a \in \Sigma \mid \exists x \in X\ (p_i - x/a \rightarrow)\} = init(S_i)$ and $\{x \subseteq \Sigma \mid \exists x \in X\ (p_i - x/\Theta \rightarrow)\} \cup \{\emptyset\} = Ref(S_i)$. Therefore, for each state p of P_F, we define:

$$\begin{aligned} init(p) &= \{a \in \Sigma \mid \exists x \in X\ (p - x/a \rightarrow)\} \\ Ref(p) &= \{A \subseteq \Sigma \mid \exists x \in X\ (p - A/\Theta) \rightarrow\} \cup \{\emptyset\}. \end{aligned}$$

Similarly, for $\sigma \in \Sigma^*$, we also have $pf(\sigma)$ represent any input and output sequence in which the output part is σ and the input part contains σ. Formally, $pf(\sigma.a) = pf(\sigma).x/a$ and $pf(\varepsilon) = \varepsilon$, where $a \in x$ and $x \in X$.

Proposition 7 *In any FFSM* P_F, *let* $p, q \in P$, $x_1, x_2 \in X$ *and* $y \in Y$,

(1) *if* $y = \Theta$ *and* $x_2 \subseteq x_1$ *then* $p - x_1/y \rightarrow q$ *implies* $p - x_2/y \rightarrow q$;

(2) *if* $a \in x_1 \cap x_2$ *then* $p - x_1/y \rightarrow q$ *iff* $p - x_2/y \rightarrow q$;

(3) *if* $x_1 \in \lceil Ref(p) \rceil$ *then* $init(p) \cup x_1 = \Sigma$.

In FFSMs certain transitions are implied by others. The transition $p - \{a\}/a \rightarrow q$ implies exactly $2^{|\Sigma|-1}$ transitions with the same output a for all the inputs that contain a, and the transition $p - x/\Theta \rightarrow p_\Theta$ implies exactly $2^{|x|} - 1$ transitions with the same output

Θ for all the inputs that are subsets of x. The proposition also tells that in any state, the output complement is always refused.

Proposition 8 *Any FFSM is an observable nondeterministic FSM.*

Unlike a TFSM, an FFSM is nondeterministic if its corresponding LTS is nondeterministic.

Proposition 9 *Given an LTS* S *and its corresponding FFSM* P_F*, for all* $\sigma \in \Sigma^*$ *and all* $\gamma = pf(\sigma) \in \Gamma^*$,

(1) $\sigma \in Tr(s_0)$ *iff* $\gamma \in Tr(p_0)$;

(2) *if there exists* $p \in P$ *such that* $p_0 - \gamma \rightarrow p$ *then* $Ref(p) = Ref(s_0, \sigma)$.

This proposition shows the way in which an I/O FSM models the behavior of an LTS in failure semantics. The FFSM and its corresponding LTS exhibit identical behavior: a set A may be refused after trace σ by the LTS iff its FFSM may produce output Θ once A is applied after trace $pf(\sigma)$.

Accordingly, the failure equivalence and reduction relations in LTSs directly correspond to the equivalence and reduction relations in FSMs, as stated by the following theorem.

Theorem 2 *For any given two LTSs* R, S *and their corresponding FFSMs* P_F, Q_F,

(1) $\mathsf{R} \leq_f \mathsf{S}$ *iff* $\mathsf{P}_F \leq \mathsf{Q}_F$;

(2) $\mathsf{R} \approx_f \mathsf{S}$ *iff* $\mathsf{P}_F \sim \mathsf{Q}_F$.

Since the nondeterministic reduction relation is composed of the trace equivalence and failure reduction relations, it corresponds to the equivalence and reduction relations, respectively in TFSMs and FFSMs.

Corollary 1 *For any given two LTSs* R, S *and their corresponding TFSMs* P_T, Q_T *and FFSMs* P_F, Q_F, *we have* $\mathsf{R} \leq_n \mathsf{S}$ *iff* $\mathsf{P}_T \sim \mathsf{Q}_T$ *and* $\mathsf{P}_F \leq \mathsf{Q}_F$.

By virtue of Theorem 2, the test cases for the FFSM model can be used to test the LTS implementations with respect to their specification for the conformance relations in failure semantics. The existing NFSM-based methods for the reduction relation [24, 25] or equivalence relation [20, 21] can be exploited to derive relevant test cases from LTS

specifications through the FFSMs. Like in the case of trace testing, the obtained test cases should be further transformed so that they can be executed through the rendezvous interface of LTSs.

6 Test Generation

It follows from the results of the previous section that the derivation of an m-complete test suite from an LTS specification can be performed by transforming the specification into an TFSM or FFSM, according to the given conformance relation, applying an existing method to it, and then converting the obtained test cases back to the LTS formalism. The approach is illustrated in the following by several examples.

6.1 Testing Trace Equivalence

Given an LTS specification S and a fault model $\mathcal{F}$, in trace semantics, S can be transformed into an TFSM P_T and the set $\mathcal{F}_{tfsm}$ of TFSMs corresponding to $\mathcal{F}$ can be treated as a fault model for P_T. In particular, if the fault model $\mathcal{F}(m)$ for S is given, the fault model for P_T is the set $\mathcal{F}_{tfsm}(m)$ of TFSMs with at most m states and with the same alphabet as P_T. Since any TFSM is a DFSM, an existing DFSM-based method can be applied directly to the TFSM to obtain a test suite. Usually, existing DFSM-based methods assume a fault model $\mathcal{F}_{dfsm}(m)$ which is a set of DFSMs with at most m states and with the same alphabet as the given DFSM specification. Obviously, $\mathcal{F}_{tfsm}(m) \subseteq \mathcal{F}_{dfsm}(m)$, and hence if the obtained test suite completely covers $\mathcal{F}_{dfsm}(m)$ then it also completely covers $\mathcal{F}_{tfsm}(m)$.

Using an Existing Method

From the definition of the corresponding TFSM, we can get the TFSM shown in Figure 5 (a) for the LTS specification S of Figure 2. This TFSM is not minimal, so it is transformed into its minimal form, shown in Figure 5 (b), as usually required by an existing method. We choose the W-method [10] and apply it the TFSM.

According to the W–method, a complete test suite TS over $\mathcal{F}_{dfsm}(4)$ is constructed in the following way: $TS = Q@(\{\varepsilon\} \cup X)@W$, where Q is a state cover, in which for each state p_i there exists an input sequence leading to p_i, $Q@(\{\varepsilon\} \cup X)$ is the transition cover

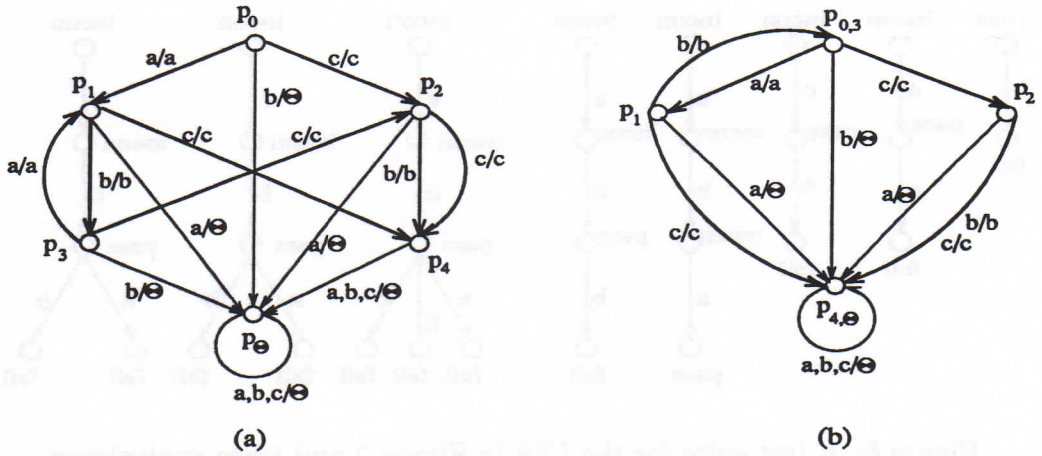


Figure 5: An example for test generation using TFSM

and W is a characterization set. From the TFSM in Figure 5 we may select $Q = \{\varepsilon, a, b, c\}$ and $W = \{a, b.a\}$. The resulting test suite is $\{a.b.a, b.b.a, c.b.a, a.a.a, a.a.b.a, a.b.b.a, a.c.a, a.c.b.a, b.a.a, b.a.b.a, b.b.b.a, b.c.a, b.c.b.a, c.a.a, c.a.b.a, c.b.b.a, c.c.a, c.c.b.a\}$. This test suite is also complete over $\mathcal{F}_{tfsm}(4)$.

In a TFSM, the Θ for an input implies Θ for all subsequent inputs. From this we can note that there is a certain redundancy in the above test suite. For example, the suffix $a.a$ of test case $b.a.a$ is not necessary because of the Θ for the first input b. According to the LTS semantics, if b cannot form a valid trace of S, then $b.a.a$ cannot do it either. These suffixes can be removed and the resulting test cases still constitute a complete test suite for the TFSM over $\mathcal{F}_{tfsm}(4)$: $\{a.b.a, b, c.b.a, a.a, a.b.b, a.c.a, a.c.b, c.a, c.b.b, c.c.a, c.c.b\}$.

In general, if any test suite for a given TFSM is complete over a certain faul model, that is a set of TFSMs, then removing the suffixes of test sequences that follow the first input with the output "Θ" results in a test suite which is still complete over the same fault model. In order to state this in a formal way, we use $pref(V)$ to represent all prefixes of sequences in V, i.e. $pref(V) = \{\gamma_1 \mid \gamma_1 \in \Gamma^* \wedge \gamma_1.\gamma_2 \in V\}$. The following theorem gives the validity of this simplification of TFSM test cases.

Theorem 3 *Given a TFSM specification* P_T *and a fault model* $\mathcal{F}_{tfsm}$*, if TS is a complete test suite for* P_T *w.r.t.* $\sim$ *over* $\mathcal{F}_{tfsm}$ *then* $TS' = \{\rho.b \in pref(TS) \mid (pt(\rho.b) \in Tr(s_0) \wedge \rho.b \in TS) \vee pt(\rho).b/\Theta \in Tr(s_0)\}$ *is also a complete test suite for* P_T *w.r.t.* $\sim$ *over* $\mathcal{F}_{tfsm}$.

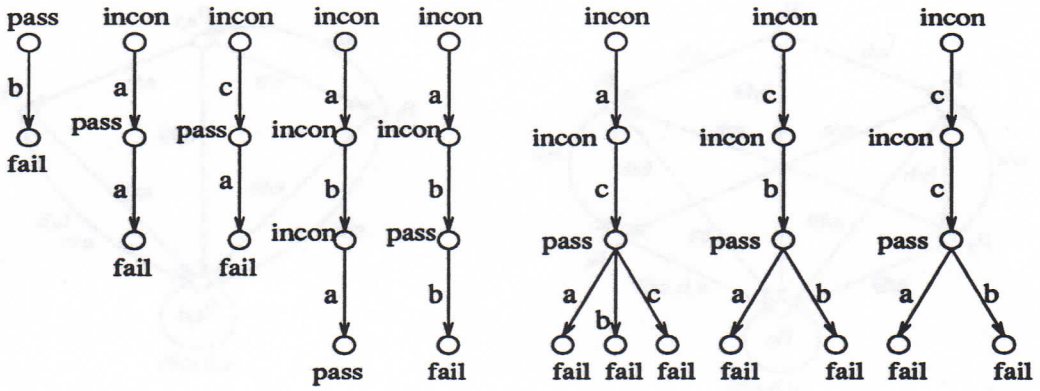


Figure 6: A test suite for the LTS in Figure 2 and trace equivalence

Modifying an Existing Method

Another solution to the redundancy problem is to modify an existing method in such a way that the sink state is excluded from the computation. The null output Θ distinguishes the sink state from the others; furthermore, in the LTS semantics it represents the IUT in the deadlock, so it is not necessary to check the transitions which leave this state. Accordingly, if an existing method with complete coverage over $\mathcal{F}_{dfsm}(m)$ is to be modified, according to Theorem 3, then the resulting method will ensure a test suite completely covering $\mathcal{F}_{tfsm}(m)$.

We demonstrate the modification of the HSI–method [22] and apply it again to the TFSM in Figure 5 (b) as an example. Let us write $X^n = X@X^{n-1}$ for $n > 0$ and $X^0 = \{\varepsilon\}$, according to the HSI-method, for an FSM with n states ($n \leq m$), $TS = \bigcup_{i=0}^{n-1}(TC_i@W_i)$ is a complete test suite over $\mathcal{F}_{dfsm}(m)$, where $TC_i = \{\rho \in Q@(\bigcup_{k=0}^{m-n+1} X^k) \mid p_0 - \rho \rightarrow p_i\}$ and the W_i are harmonized state identifiers for the states p_i. We modify the method in the following way:

(1) We separate the sink state from any other states that may be equivalent to it. In the example TFSM, p_4 is equivalent to the sink state p_Θ, but we keep p_4 and p_Θ separately.

(2) We compute a state cover Q which includes no sequence leading to the sink state. For the example TFSM, we choose $Q = \{\varepsilon, a, a.c, c\}$.

(3) We compute an HSI W_i for every state, except for the sink state. For the example TFSM, we choose $W_{0,3} = \{a, b\}, W_1 = \{b.a\}, W_2 = \{b.a\}$ and $W_4 = \{a, b\}$.

(4) Let p_{n-1} be the sink state, compute TC_i:

$$TC_i = \begin{cases} \{\rho \in Q@(\bigcup_{k=0}^{m-m+1} X^k) \mid p_0 - \rho \rightarrow p_i\} & 0 \leq i \leq n-2 \\ \{\rho.x \in Q@(\bigcup_{k=0}^{m-m+1} X^k) \mid \exists p_j \neq p_i \ (p_0 - \rho \rightarrow p_j - x \rightarrow p_i)\} & i = n-1. \end{cases}$$

Considering $m = n$, for the example TFSM, we obtain $TC_{0,3} = \{\varepsilon, a.b\}$, $TC_1 = \{a\}$, $TC_2 = \{c\}$, $TC_4 = \{a.c, c.b, c.c\}$ and $TC_\Theta = \{b, a.a, a.c.a, a.c.b, a.c.c, c.a\}$.

(5) We Construct a test suite TS without identifying the sink state, that is,

$$TS = TC_{n-1} \cup (\bigcup_{i=0}^{n-2} (TC_i @ W_i)).$$

We note that the sequences in TC_{n-1} lead to s_Θ, so no state identifier is appended to these sequences. The resulting test suite for the example TFSM is $\{b, a.a, c.a, a.b.a, a.b.b, a.c.a, a.c.b, a.c.c, c.b.a, c.b.b, c.c.a, c.c.b\}$, which is complete over $\mathcal{F}_{tfsm}(4)$.

Back to the LTS Formalism

From a test sequence for the corresponding TFSM of a given LTS specification, we can easily design a corresponding test case in the LTS formalism with respect to trace equivalence by converting the sequence into a corresponding LTS and subsequently labeling the LTS according to Definition 10, as follows:

Algorithm 1 Test Conversion from TFSM to LTS

Input: *A test sequence ρ for an TFSM* P_T.

Output: *A deterministic and tree-structured LTS* T.

Let $\rho = a_1.a_2 \ldots .a_n$. *Construct an LTS* $\mathsf{T} = t_0 - a_1 \rightarrow t_1 \ldots t_{n-1} - a_n \rightarrow t_n$.

An LTS test suite obtained by transforming the above test suite for the TFSM in Figure 5 (b) is shown in Figure 6. This test suite can be used to test implementations of the LTS specification in Figure 2 with respect to trace equivalence. The following theorem guarantees that this test suite is a 4-complete test suite for the LTS specification.

Theorem 4 *Given an LTS specification* S *and a fault model* $\mathcal{F}$, *the corresponding TFSM* P_T *and fault model* $\mathcal{F}_{tfsm}$, *if a test suite for* P_T *w.r.t.* $\sim$ *is complete over* $\mathcal{F}_{tfsm}$ *then the test suite obtained by Algorithm 1 and Definition 10 is complete for* S *w.r.t.* $\approx_t$ *over* $\mathcal{F}$.

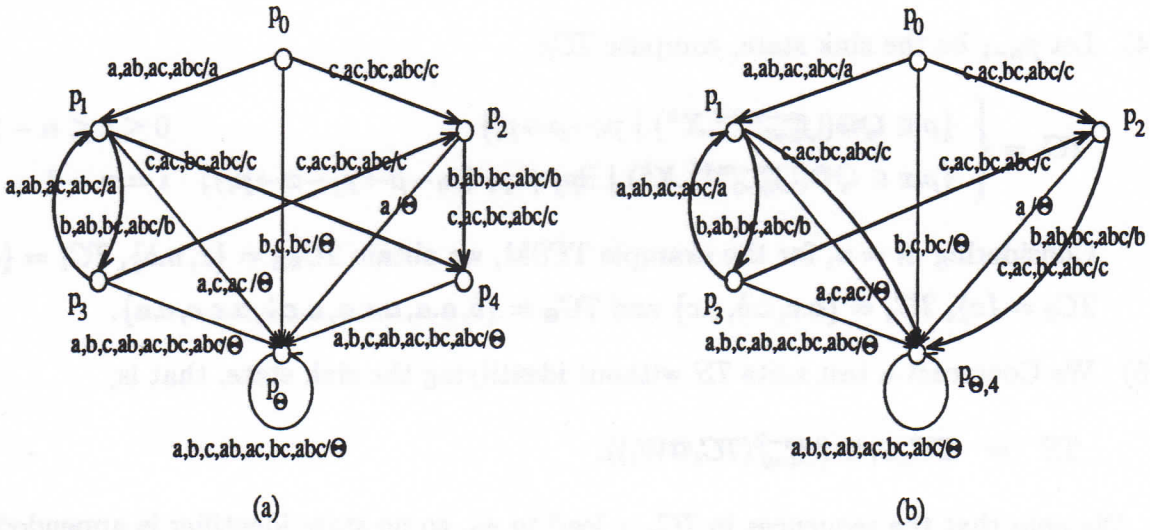


Figure 7: An example for test generation using FFSM

6.2 Testing Failure Equivalence

The traditional transition checking approach in FSMs relies on the general transition fault model, according to which every transition can be mutated independently of the others [5]. However, this is not the case for the fault model we are considering in testing of LTSs, in which mutants are really LTSs but are viewed as FFSMs. It has been stated in Proposition 7 that certain transitions of an FFSM can be implied by others. Implied transitions in an FFSM should not be treated as completely independent for test derivation. The dependency among transitions of the FFSM would be fully neglected if an existing test derivation method is applied to the FFSM in a straightforward manner, and hence any resulting test suite with complete fault coverage would definitely be redundant. Consider two transitions $p_0 - \{a\}/a \rightarrow p_1$ and $p_0 - \{a, b\}/a \rightarrow p_1$ in the FFSM in Figure 4 (c) as an example; once the first transition is checked there is no need to check the second. The reason is that if $p_0 - \{a, b\}/\Theta \rightarrow s_\Theta$ is implemented then it implies $p_0 - \{a\}/\Theta \rightarrow s_\Theta$.

Proposition 10 *Given two states p and q of an FFSM and let $V = \lfloor Acc(p) \rfloor \cup \lceil Ref(p) \rceil$, we have $out(p, X) = out(q, X)$ iff $out(p, V) = out(q, V)$.*

We note that X is the input set of the FFSM, while $\lfloor Acc(p) \rfloor \cup \lceil Ref(p) \rceil$) is a subset of X. The proposition states that the subset V, instead of the whole input set X, can be used to cover all the transitions out of state p. For example, for the FFSM shown in Figure 7 (a), which corresponds to the LTS specification S of Figure 2, we have

States p_i	p_0	p_1	p_2	p_3	p_4	p_Θ
$\lfloor Acc(p_i) \rfloor$	$\{a\}$	$\{b\}$	$\{b\},\{c\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\lceil Ref(p_i) \rceil$	$\{b,c\}$	$\{a,c\}$	$\{a\}$	$\{a,b,c\}$	$\{a,b,c\}$	$\{a,b,c\}$

Therefore one can derive a test suite for an LTS specification S with respect to failure semantics, based on an existing NFSM-based method for the corresponding FFSM in a way similar to TFSMs – by directly applying the method and subsequently removing redundancy in the test suite, or by modifying the method to avoid the redundancy. Proposition 10 characterizes the existing redundancy.

In order to derive a complete test suite over a given fault model $\mathcal{F}$ from a given LTS specification S with respect to failure equivalence, S can be transformed into an FFSM P_F and the set $\mathcal{F}_{ffsm}$ of FFSMs corresponding to $\mathcal{F}$ can be treated as a fault model for P_F. In particular, if the fault model $\mathcal{F}(m)$ is given, the fault model for P_F is the set $\mathcal{F}_{ffsm}(m)$ of FFSMs with at most m states and with the same alphabet as P_F. Existing NFSM-based methods usually assume a fault model $\mathcal{F}_{nfsm}(m)$, which is a set of NFSMs (of the observable form) with at most m states and with the same alphabet as the given specification. Obviously, $\mathcal{F}_{ffsm}(m) \subseteq \mathcal{F}_{nfsm}(m)$. Therefore if any existing method covering $\mathcal{F}_{nfsm}(m)$ is applied to P_F, then the resulting test suite is also complete over $\mathcal{F}_{ffsm}(m)$. In the following, we take the FFSM in Figure 7 (a) as an example to illustrate our approach.

Using an Existing Method

The FFSM in Figure 7 (a) is not minimal, so it is transformed into its minimal form, shown in Figure 7 (b). We can apply the W-method [20] to the minimal FFSM to derive a 5-complete test suite. We select a state cover $Q = \{\varepsilon, a, c, a.b\}$, and a characterization set $W = \{ac\}$ (Note that $a.b$ represents $\{a\}.\{b\}$ and ac represents $\{a,c\}$). The transition cover is $\{\varepsilon, a, b, c, ab, ac, bc, abc, a.a, a.b, a.c, a.ab, a.ac, a.bc, a.abc, c.a, c.b, c.c, c.ab, c.ac, c.bc, c.abc, a.b.a, a.b.b, a.b.c, a.b.ab, a.b.ac, a.b.bc, a.b.abc\}$ and the resulting test suite can be computed according to $Q@(\{\varepsilon\} \cup X)@W$ and it completely covers $\mathcal{F}_{ffsm}(5)$.

As we have discussed, there is much redundancy in this test suite. From Proposition 10 and the fact that the sink state in FFSMs does not require any identification and the transitions which leave this state do not require checking, the redundancy can be removed using the following algorithm.

Algorithm 2 Redundancy Removal from an FFSM Test Suite

Input: *A test suite TS_I derived from an FFSM P_F using an existing FSM-based method.*

Output: *A test suite TS_O.*

For each $\rho \in TS_I$, construct $\mathcal{K} = \{\gamma \in Tr(p_0) \mid \gamma^{in} = \rho\}$ and for each $\gamma \in \mathcal{K}$, do the following steps (assuming that γ has the form $x_1/y_1.x_2/y_2 \ldots x_n/y_n$ and corresponds to the transitions $p_0 - x_1/y_1 \rightarrow p_1 \ldots p_{n-1} - x_n/y_n \rightarrow p_n$):

Step 1: *From $i = 1$ to $i = n$ or until $y_i = \Theta$ form a new test sequence ρ' (initially $\rho' = \varepsilon$) as follows:*

1. *In $\lfloor Acc(p_{i-1}) \rfloor \cup \lceil Ref(p_{i-1}) \rceil$ find an x'_i such that:*
   (a) *$x'_i \in \lceil Ref(p_{i-1}) \rceil$ and $x_i \subseteq x'_i$,*
   (b) *$x'_i \in \lfloor Acc(p_{i-1}) \rfloor$, $y_i \in x'_i$ and $x'_i \subseteq x_i$, or*
   (c) *$y \in x'_i$.*
2. *If $x_i \in Acc(p_{i-1})$ and $(x_i \backslash init(p_{i-1})) \neq \emptyset$, find a x''_i in $\lceil Ref(p_{i-1}) \rceil$ such that $(x_i \backslash init(p_{i-1})) \subseteq x''_i$, form $\rho'.x''_i$ and add it to TS_O.*
3. *$\rho' = \rho'.x'_i$.*

Step 2: *Add the resulting ρ' to TS_O. Delete any sequence in TS_O if it is a prefix of another sequence.*

Using the above algorithm, we can obtain from the above test suite for the FFSM shown in Figure 7 (b) the following test suite with the same coverage over $\mathcal{F}_{ffsm}(5)$ and obviously a smaller size: $\{bc, a.ac.abc, bc.b.abc, bc.c.abc, a.b.abc.ac, a.b.abc.a, a.b.abc.c\}$. The following theorem gives the validity of Algorithm 2.

Theorem 5 *If a test suite TS_I is complete for P_F w.r.t. $\sim$ over $\mathcal{F}_{ffsm}$, then the test suite TS_O obtained by Algorithm 2 is also complete for P_F w.r.t. $\sim$ over $\mathcal{F}_{ffsm}$.*

Modifying an Existing Method

Because of Proposition 10, it is sufficient to check only the transitions covered by the input set $\lfloor Acc(p) \rfloor \cup \lceil Ref(p) \rceil$ for each state p. Moreover, as in the case of TFSMs, the sink state does not require identification and the transitions which leave this state are excluded from checking. Accordingly, if an existing method with complete coverage over

$\mathcal{F}_{nfsm}(m)$ is modified in such a way, then the resulting method will provide a test suite with complete coverage over $\mathcal{F}_{ffsm}(m)$.

We here demonstrate the modification of the GWp-method [20] and use the FFSM in Figure 7 (a) as an example. There are the following five steps:

(1) We separate the sink state from any other states which may be equivalent to it. In the example FFSM, p_4 is equivalent to the sink state p_Θ, but we keep p_4 and p_Θ separately.

(2) We compute a state cover Q, in which, for each state p_i except the sink state, there exists $\rho \in Q$ such that $p_0 - \rho \rightarrow p_i$. For the example FFSM, we choose $Q = \{\varepsilon, a, c, a.bc\}$.

(3) We compute a characterization set W and a tuple of partial characterization set W_i for each state, except for the sink state. We here choose $W = \{ac\}$ and let $W_i = W$ for the example FFSM.

(4) We use $\lfloor Acc(p) \rfloor \cup \lceil Ref(p) \rceil$ to compute a transition cover. In general, given any FFSM P_F with n states ($n \leq m$), the transition cover can be computed recurrently, starting with $TC^{(0)} = Q$ and for $k \leq m - n + 1$, by

$$TC^{(k)} = \{\rho.x \mid \rho \in TC^{(k-1)} \wedge (\exists\, p \in p_0 \textbf{ after } \rho\ (p \neq p_\Theta \wedge x \in \lfloor Acc(p) \rfloor \cup \lceil Ref(p) \rceil))\}$$

Let p_{n-1} be the sink state p_Θ, we further compute

$$TC_i = \begin{cases} \{\rho \in TC^{(m-n+1)} \backslash TC^{(m-n)} \mid p_0 - \rho \rightarrow p_i\} & 0 \leq i \leq n-2 \\ \{\rho \in TC^{(m-n+1)} \mid p_0 - \rho \rightarrow p_i\} & i = n-1. \end{cases}$$

Considering $m = n$ for the example FFSM, we obtain the transition cover $TC^{(1)} = \{\varepsilon, bc, a, a.ac, a.b, c.a, c.b, c.c, a.bc.abc\}$ and $TC_\Theta = \{bc, a.ac, c.a, a.bc.abc\}$.

(5) We construct a test suite TS without identifying the sink state, that is,

$$TS = TC_{n-1} \cup (TC^{(m-n)} \backslash TC_{n-1}) @ W \cup (\bigcup_{i=0}^{n-2} TC_i @ W_i).$$

The resulting test suite for the example FFSM is $\{ac, bc.ac, a.ac.ac, a.b.ac, c.a, c.b.ac, c.c.ac, a.b.abc.ac\}$, which completely covers $\mathcal{F}_{ffsm}(5)$. Note that in the transition cover, only $c.a$ uniquely leads to the sink state, and thus W or W_i is not appended to it.

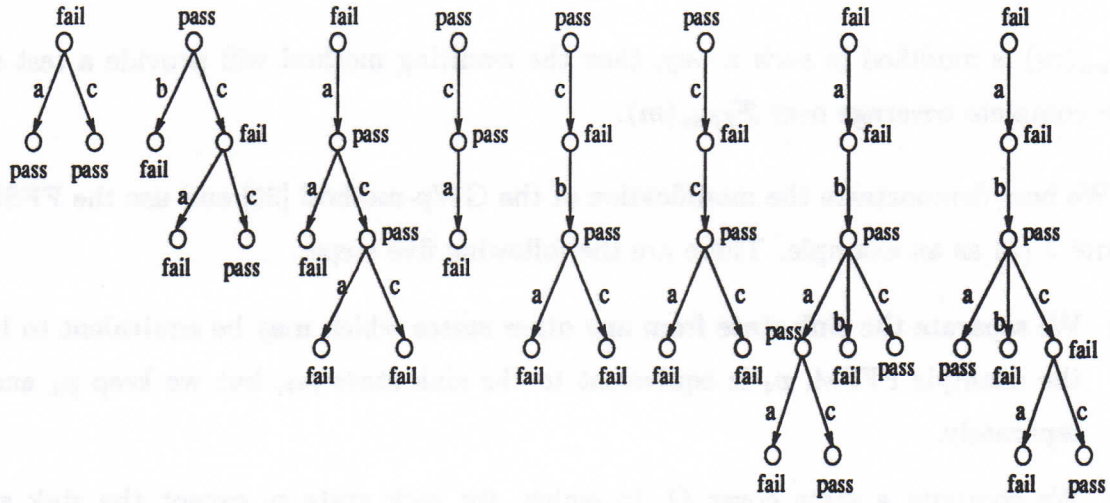


Figure 8: A test suite for the LTS in Figure 2 and failure equivalence

Back to the LTS Formalism

In order to obtain a test case for an LTS specification in respect to failure equivalence from a test sequence for the corresponding FFSM, similar to the case of testing trace equivalence, the test sequence is first converted into a set of deterministic and tree-structured LTSs, and secondly the resulting LTSs are labeled according to Definition 11. The conversion is performed in the following algorithm:

Algorithm 3 Test Conversion from FFSM to LTS

Input: *A test sequence ρ for an FFSM P_F.*

Output: *A set of deterministic and tree-structured LTSs.*

Construct $\mathcal{K} = \{\gamma \in Tr(p_0) \mid \gamma^{in} = \rho\}$. For each $\gamma \in \mathcal{K}$, do the following steps (assuming that γ has the form $x_1/y_1.x_2/y_2 \ldots x_n/y_n$):

- *Construct an LTS T with only the initial state t_0. Let $t = t_0$.*
- *From $i = 1$ to $i = n$ or until $y_i = \Theta$, for each $a \in x_i$, add a new transition $t \text{-} a \rightarrow t_j$ to T; $t = t_k$ if $t \text{-} x_i \rightarrow t_k$.*

An LTS test suite obtained by transforming the above test sequences for the FFSM in Figure 7 (b) is shown in Figure 8. This test suite can be used to test implementations of the LTS specification in Figure 2 with respect to failure equivalence. The following theorem

guarantees that such a test suite is complete for the specification and the conformance relation.

Theorem 6 *Given an LTS specification* S *and a fault model* $\mathcal{F}$*, the corresponding FFSM* P_F *and fault model* $\mathcal{F}_{ffsm}$*, if a test suite for* P_F *w.r.t.* $\sim$ *is complete over* $\mathcal{F}_{ffsm}$ *then the test suite obtained by Algorithm 3 and Definition 11 is complete for* S *w.r.t.* $\approx_f$ *over* $\mathcal{F}$*.*

6.3 Testing the Other Relations

Using the FFSM model and an NFSM-based method for the reduction relation, such as the method given in [24], we can derive test suites for LTS specifications with respect to the failure reduction relation in a similar way as discussed in the above subsection. Algorithm 2 can still be used to remove redundancy in the FFSM test cases obtained by applying the existing method; Algorithm 3 can still be used to convert the FFSM test sequences to corresponding LTSs, but they are labeled according to Definition 12. The adaptation of the NFSM-based method is also similar.

Since the nondeterministic reduction relation consists of the trace equivalence relation and the failure reduction relation, the test derivation for an LTS specification with respect to this relation can be obtained by transforming the LTS specification to a TFSM and an FFSM, respectively, and then applying an existing FSM-based method for equivalence to the TFSM and an existing FSM-based method for reduction to the FFSM. The obtained two test suites can be merged into one test suite and then be converted to the LTS formalism using Algorithm 3. The resulting LTS are labeled according to Definition 13.

7 Conclusion

Labeled transition systems (LTSs) are the basic semantics for the LOTOS language and other specification formalisms. This paper deals with test suite development from a specification given in the LTS formalism. We have introduced a framework for testing LTSs with respect to several common conformance relations through finite experiments. Afterwards we have shown that in the context of trace semantics, LTSs can be represented equivalently by an input/output finite state machine model (FSM) – the trace finite state machines (TFSMs); and in the context of failure semantics, by the failure finite state

machines (FFSMs). The benefit of this transformation is that the problem of deriving a conformance test suite for an LTS can be transferred into the realm of the FSM model, where the test derivation theory has been elaborated for several decades and several testing tools have already been constructed [30].

Trace FSMs are deterministic FSMs, so the existing methods for FSMs can be applied to the TFSMs directly for the derivation of test suites to check the corresponding LTSs with trace equivalence. An example is presented which illustrates the process of test derivation from an LTS specification for trace equivalence, by transforming the LTS into a TFSM and subsequently applying the W–method. The removal of redundant tests is discussed. A slight modification of the HSI–method for TFSMs is also proposed to avoid the redundancy of tests.

Failure FSMs are observable, nondeterministic FSMs, so the existing methods for such machines can be applied to the FFSMs for the derivation of test suites to check the corresponding LTSs in failure semantics. However, since certain transitions in the FFSMs may not require testing according to the LTS semantics, an adaptation of the existing methods to FFSMs is needed to avoid redundancy. Two examples of test derivation for failure equivalence, the direct application of the GWp-method to an FFSM and a slight modification of this method, are used to illustrate the process of test derivation for LTS specifications in respect to failure semantics. Meanwhile, we also give the algorithms for redundancy removal from the FFSM tests obtained by applying an existing method, and the conversion of the FFSM tests to the LTS formalism.

A number of existing FSM-based methods provide complete fault coverage for a well-known fault model [3] which is the set of mutants with at most m states, where m is a known integer. An identical fault model has also been considered in the LTS realm. Therefore, for a given LTS specification and any of the conformance relations discussed in this paper, the transformation approach will produce a test suite with complete fault coverage if the chosen FSM-based method guarantees such fault coverage.

Though our transformation approach is encouraging, much work still remains to be done. Other conformance relations on LTSs in trace or failure semantics, such as **conf** [6] and completed trace equivalence [35], are left for further study. We also note that, due to

nondeterminism, the test execution may be blocked unexpectedly before the designated transition or deadlock for a test case is checked; therefore the test case may have to be executed repeatedly until the test purpose can be achieved. It is therefore important to select appropriate preambles and state identifiers which minimize the nondeterminism and reduce the number of repeated test executions. Another problem is the derivation of tests with fault coverage for LTS specifications according to the so-called failure trace semantics, for which testing may proceed after a refusal by applying a different offer. This is contrary to failure semantics where a refusal is the end of a test run.

Acknowledgments

This work is supported by a strategic grant from the Natural Sciences and Engineering Research Council of Canada.

Appendix

In this appendix, we give the proofs for the theorems and propositions included in the paper.

Proposition 1 Proof: M passes T, according to Definition 10, then there exists $\sigma \in Obs_{(T,M)}$ and $\ell_{\approx t}(\sigma) = \textbf{fail}$, which imply $\sigma \in Tr(m_0) \backslash Tr(s_0)$, or there exists $\sigma \in Pur(T)$ and $\sigma \notin Obs_{(T,M)}$, which imply $\sigma \in Tr(s_0) \backslash Tr(m_0)$. The both cases further imply $M \not\approx_t S$. Thus M passes T with respect to $\approx_t$.

Proposition 2 Proof: M passes T, according to Definition 11, then there exists $\sigma \in Obs_{(T,M)}$ and $\ell_{\approx f}(\sigma) = \textbf{fail}$, which imply $init(t_0 \textbf{ after } \sigma) \in Ref(m_0, \sigma) \backslash Ref(s_0, \sigma)$, or there exists $\sigma \in Pur(T)$ and $\sigma \notin Obs_{(T,M)}$, which imply $init(t_0 \textbf{ after } \sigma) \in Ref(m_0, \sigma) \backslash Ref(s_0, \sigma)$. The both cases further imply $M \not\approx_f S$. Thus M passes T with respect to $\approx_f$.

Proposition 3 Proof: Similar to the proof of Proposition 2.

Proposition 4 Proof: Similar to the proof of Proposition 2.

Proposition 5 Proof: Suppose there exist $p_k, p_i, p_j \in P, a \in X$ such that $p_k - a/a \rightarrow p_i$, $p_k - a/y \rightarrow p_j$ and $p_i \neq p_j$ where $y = a$ or Θ. From the TFSM definition, $y = \Theta$ is

impossible, so we have $y = a$. Let $p_k = \psi(S_k), p_i = \psi(S_i), p_j = \psi(S_j)$, $S_k = \mathsf{S}$ **after** σ' and $\sigma = \sigma'.a$. Obviously, if $p_i \neq p_j$, then $S_i \neq S_j$ because ψ is a one-to-one mapping. However, it is impossible since S **after** $\sigma = S_i = S_j$ by the definition of the multi-state set. Therefore P_T is deterministic.

Theorem 1 Proof: (1). $\Rightarrow$. Since $\mathsf{R} \approx_t \mathsf{S}$, for any $R_i = r_0$ **after** σ and $S_i = s_0$ **after** σ, $\sigma \in \Sigma^*$, $init(R_i) = init(S_i)$; and furthermore from Proposition 5, for any $\gamma \in \Gamma^*$ and $\gamma = pt(\sigma)$ there is $p_0 - \gamma \rightarrow p$ in P_T iff there is $q_0 - \gamma \rightarrow q$ in Q_T. $init(R_i) = init(S_i)$ means also that $\{x \in X \mid p - x/\Theta \rightarrow\} = \{x \in X \mid q - x/\Theta \rightarrow\}$ where $p = \psi(R_i)$ and $q = \psi(S_i)$. Thus it is shown that a set of all the traces of P_T which have at most one "x/Θ" is equal to a set of the traces of Q_T which have at most one "x/Θ". Furthermore, any trace with two or more pairs of the form "x/Θ" is implied by its prefix with only one "x/Θ", thus $Tr(p_0) = Tr(q_0)$, that is, $\mathsf{M}' \sim \mathsf{S}'$.

$\Leftarrow$. From Proposition 6, if $\mathsf{P}_T \sim \mathsf{Q}_T$, that is, $Tr(p_0) = Tr(q_0)$, then $Tr(r_0) = Tr(s_0)$.

Proposition 7 Proof: (1) and (2) are evident from the FFSM definition.

(3) Let P_F is the FFSM of LTS S. From Definition 15, for each node $p \in P, p \neq s_\Theta$, there exists $S_i \in \Pi_S$ such that $p = \psi(S_i)$ and $\lceil Ref(S_i) \rceil = \lceil Ref(p) \rceil$. Here we denote $blk(s) = \Sigma \backslash \{a \in \Sigma \mid s - a \rightarrow\}$ for a stable state s of S and $blk(s) = \emptyset$ if s is unstable. Besides, for the transitions $s - \tau^n \rightarrow t$ $(1 \leq n)$, if $s = t$, we say that it is a τ-Cycle.

If there are no τ-Cycles in S, then $Ref(S_i) = \bigcup_{(s \in S_i)} \wp(blk(s))$. Note that $blk(s) = \emptyset$ for unstable s. Let $s \in S_i$ be stable and $blk(s)$ be in $\lceil Ref(S_i) \rceil$. From the definition of $init(p)$, $init(S_i) = init(p)$, $init(p) \supseteq init(s)$. So, $\Sigma = init(p) \cup blk(s)$ for each $blk(s) \in \lceil Ref(p) \rceil$.

If there are τ-Cycles in S_i, then each of the τ-Cycles can be collapsed into a single state with failure equivalence. Let W be the state set in one of the τ-Cycles, w be the state after collapsing W. If w is unstable, none of elements in $Ref(S_i)$ are determined by W. If w is stable, then $\lceil Ref(W) \rceil = \{blk(w)\}$. Thus for each $B \in \lceil Ref(S_i) \rceil$ ($\lceil Ref(p) \rceil$), there exists $s \in S_i$ where s is a stable state or s is in a W. Similar to the case without τ-Cycles, $init(p) \supseteq init(s)$. So, $\Sigma = init(p) \cup B$.

For the sink state s_Θ, $init(s_\Theta) = \emptyset$ and $\lceil Ref(s_\Theta) \rceil = \{\Sigma\}$, Proposition 7 also holds.

Proposition 8 Proof: From Proposition 7, for an FFSM P_F, $\Sigma = init(p) \cup x'$ for each state $p \in P$ if $x' \in \lceil Ref(p) \rceil$. As a result, for all $x \in X$ and all $p \in P$, since $X = \wp(\Sigma) \backslash \emptyset$,

$a \in x$ for $a \in init(p)$, or $x \subseteq x'$ for $x' \in \lceil Ref(p) \rceil$. From Definition 15, the transition $s-x/a\rightarrow$ or $s-x/\Theta\rightarrow s_\Theta$ is specified in P_F.

Suppose there exist $p_k, p_i, p_j \in P$, $x \in X$ and $y \in Y$ such that $p_k-x/y\rightarrow p_i$, $p_k-x/y\rightarrow p_j$ and $p_i \neq p_j$. Obviously $y \neq \Theta$ because, from the FFSM definition, $y = \Theta$ implies $p_i = p_j = p_\Theta$. Let $p_k = \psi(S_k), p_i = \psi(S_i), p_j = \psi(S_j)$, $S_k = \mathsf{S}$ **after** σ' and $\sigma = \sigma'.y$. Obviously, if $p_i \neq p_j$, then $S_i \neq S_j$ because ψ is a one-to-one mapping. However, it is impossible since S **after** $\sigma = S_i = S_j$ from the definition of the multi-state set. So P_F is observable.

Proposition 9 Proof: We first prove that any $pf(\sigma) \in Tr(p_0)$ iff $\sigma \in Tr(s_0)$.

(1). For each trace $\sigma \in Tr(s_0)$, let transitions $s_0 = a_1 \Rightarrow s_1 \ldots s_{n-1} = a_n \Rightarrow s_n$ have σ, $1 \leq n$. Obviously, for each s_i, $0 \leq i < n$, there exists $S_0, S_1, \ldots, S_n \in \Pi_S$ such that $s_0 \in S_0, s_1 \in S_2, \ldots, s_n \in S_n$. Therefore there exist transitions $\psi(S_0)-pf(\sigma)\rightarrow \psi(S_n)$ in P_F. On the other hand, If there exists γ in $Tr(p_0)$ where $\gamma = pf(\sigma)$, Similarly, because ψ is a one-to-one mapping, σ is in $Tr(s_0)$.

(2). If there exists $\gamma \in Tr(p_0)$ such that $\gamma = pf(\sigma), \sigma \in \Sigma^*$, let $p_0-\gamma\rightarrow p_i$, then from the above proof, $S_i = s_0$ **after** σ and $p_i = \psi(S_i)$. Furthermore since $Ref(s_0, \sigma) = Ref(S_i)$, from the definition of the $Ref(p_i)$, $Ref(s_0, \sigma) = Ref(p_i)$.

Theorem 2 Proof: (1). $\Rightarrow$. Since $\mathsf{R} \leq_f \mathsf{S}$, that is, $Ref(r_0, \sigma) \subseteq Ref(s_0, \sigma)$, from the proof of Proposition 9, for any $\gamma \in \Gamma^*$ where $\gamma = pf(\sigma)$ if $\gamma \in Tr(p_0)$ then $\gamma \in Tr(q_0)$ and further if there is $p_0-\gamma\rightarrow p$ in P_F then there exists $q_0-\gamma\rightarrow q$ in Q_F and $Ref(p) \subseteq Ref(q)$. It is shown that a set of all the traces of P_T which have at most one "x/Θ" is a subset of a set of all the traces of Q_T which have at most one "x/Θ". Furthermore, any trace with two or more pairs of the form "x/Θ" is implied by its prefix with only one "x/Θ", thus, $Tr(p_0) \subseteq Tr(q_0)$, that is, $\mathsf{P}_F \leq \mathsf{Q}_F$.

$\Leftarrow$. If $\mathsf{P}_F \leq \mathsf{Q}_F$, then for any $\sigma \in \Sigma^*$, let $\gamma = pf(\sigma)$, if there exists $p_0-\gamma\rightarrow p$ in P_F then there exists $q_0 = \gamma \Rightarrow q$ in Q_F and $Ref(p_0) \subseteq Ref(q_0)$, which, from Proposition 9, mean that $Ref(r_0, \sigma) \subseteq Ref(s_0, \sigma)$ for $\sigma \in Tr(r_0)$. On the other hand, for $\sigma \notin Tr(r_0)$, $(\sigma \notin Tr(s_0))$, $Ref(r_0, \sigma) = \emptyset$ $(Ref(s_0, \sigma) = \emptyset)$. Thus $\mathsf{R} \leq_f \mathsf{S}$ holds.

(2). $\Rightarrow$. From the definition, if $\mathsf{R} \approx_f \mathsf{S}$ then $\mathsf{R} \leq_f \mathsf{S}$ and $\mathsf{S} \leq_f \mathsf{R}$. From the above proof, $\mathsf{P}_F \leq \mathsf{Q}_F$ and $\mathsf{Q}_F \leq \mathsf{P}_F$, that is , $\mathsf{P}_F \sim \mathsf{Q}_F$.

$\Leftarrow$. Similarly, if $\mathsf{P}_F \sim \mathsf{Q}_F$ then $\mathsf{P}_F \leq \mathsf{Q}_F$ and $\mathsf{Q}_F \leq \mathsf{P}_F$. From the above proof, $\mathsf{R} \leq_f \mathsf{S}$ and $\mathsf{S} \leq_f \mathsf{R}$, that is, $\mathsf{M} \approx_f \mathsf{S}$.

Theorem 3 Proof: $pref(TS)$ is a complete test suite iff TS is complete. Let $pref(TS) = TS_1 \cup TS_2$, where $TS_1 = pref(TS')$ and $TS_2 = TS \backslash TS_1$. Obviously TS_2 is a set of the sequences in $pref(TS)$ by which P_T produces the two or more "Θ"s. If there exists any TFSM M in $\mathcal{F}_{tfsm}$ that is not equivalent to P_T, then there exists γ^{in} in $pref(TS)$ which M fails (i.e. $\gamma \notin Tr(m_0) \cap Tr(p_0)$). If $\gamma^{in} \in TS_1$ then $\gamma^{in} \in pref(TS')$. If $\gamma^{in} \in TS_2$, then there exists $\sigma.b$ in TS_1 which is a prefix of γ^{in} such that $pt(\sigma).b/\Theta \in Tr(s_0)$. From this, M fails $\sigma.b$ because if $pt(\sigma).b/\Theta \in Tr(m_0)$ $(Tr(s_0))$ then for any γ_t of the form "b/Θ", $pt(\sigma).b/\Theta.\gamma_t \in Tr(m_0)$ $(Tr(s_0))$. Thus $pref(TS')$ is a complete test suite, i.e. TS' is complete.

Theorem 4 Proof: Let TS_{tfsm} be the test suite for the TFSM P_T and TS_{lts} be the converted test suite for the LTS S. From Proposition 1, any $\mathsf{T} \in TS_{lts}$ is sound. For any $\mathsf{M} \in \mathcal{F}$ we have $\mathsf{M}' \in \mathcal{F}_{tfsm}$ and from Theorem 1 if $\mathsf{M} \not\approx_t \mathsf{S}$ then $\mathsf{M}' \not\approx \mathsf{P}_T$. Since TS_{tfsm} is complete over $\mathcal{F}_{tfsm}$, there exists $\gamma^{in} \in TS_{tfsm}$ such that 1) $p_0 - \gamma \rightarrow$ and $m'_0 \not\rightarrow \gamma \rightarrow$ or 2) $p_0 - \gamma \rightarrow$ and $m'_0 \not\rightarrow \gamma \rightarrow$. From Proposition 6, This corresponds to $s_0 - \gamma^i \rightarrow$ and $m_0 \not\rightarrow \gamma^{in} \rightarrow$ or $s_0 - \gamma^{in} \rightarrow$ and $m_0 \not\rightarrow \gamma^{in} \rightarrow$. Let $\mathsf{T} \in TS_{lts}$ correspond to γ^{in}. For T, according to Definition 10, the case 1) implies $\gamma^{in} \in Pur(\mathsf{T})$ and $\gamma^{in} \notin Obs_{(\mathsf{T},\mathsf{M})}$ and the case 2) implies $\ell_{\approx t}(\gamma^{in}) = \textbf{fail}$ and $\gamma^{in} \in Obs(\mathsf{M}, \mathsf{T})$. According to Definition 8, this says that M fails T.

Proposition 10 Proof: $\Rightarrow$. It is evident because $V \subseteq X$.

$\Leftarrow$. Assume that $out(p, X) \neq out(q, X)$. It happens in either of the cases: 1) $p - x/y \rightarrow$ and $q \not\rightarrow x/y \rightarrow$ or 2) $p \not\rightarrow x/y \rightarrow$ and $q - x/y \rightarrow$. Consider the case 1). If $y = \Theta$ then $p - x/\Theta \rightarrow$ implies $x \in Ref(p)$, and therefore there exists $x' \in \lceil Ref(p) \rceil$ such that $x \subseteq x'$. $x' \in Ref(p)$ implies $p - x'/\Theta \rightarrow$ and, since $out(p, V) = out(q, V)$, $q - x'/\Theta \rightarrow$. From Proposition 7, $q - x'/\Theta \rightarrow$ and $x \subseteq x'$ imply $q - x/\Theta \rightarrow$. This contradicts $q \not\rightarrow x/\Theta \rightarrow$. If $y \neq \Theta$, let $y = a$, then $p - x/a \rightarrow$ imply $a \in x$ and $p - \{a\}/a \rightarrow$. Therefore if $\{a\} \in Ref(p)$ then there exists $x' \in \lceil Ref(p) \rceil$ such that $\{a\} \subseteq x'$; otherwise $\{a\} \in \lfloor Acc(p) \rfloor$ and $x' = \{a\}$. From Proposition 7, $p - x/a \rightarrow$ implies $p - x'/a \rightarrow$ and, since $out(p, V) = out(q, V)$, $q - x'/a \rightarrow$. Furthermore, since $a \in x \cap x'$, from Proposition 7 $q - x'/a \rightarrow$ implies $q - x/a \rightarrow$. This contradicts $q \not\rightarrow x/a \rightarrow$.

Consider the case (2. If $y = \Theta$ then $p \not\xrightarrow{x/\Theta}$ implies $x \in Acc(p)$, and therefore there exists $x' \in \lfloor Acc(p) \rfloor$ such that $x' \subseteq x$. From Proposition 7, $q \xrightarrow{x/\Theta}$ implies $q \xrightarrow{x'/\Theta}$ and, since $out(p, V) = out(q, V)$, $p \xrightarrow{x'/\Theta}$. This contradicts $x' \in Acc(p)$. If $y \neq \Theta$, let $y = a$, then $p \not\xrightarrow{x/a}$ implies $\{a\} \in Ref(p)$ and $p \not\xrightarrow{\{a\}/a}$; from this there exists $x' \in \lceil Ref(p) \rceil$ such that $\{a\} \subseteq x'$. On the other hand, $q \xrightarrow{x/a}$ implies $a \in x$; furthermore since $a \in x'$, from Proposition 7, $q \xrightarrow{x'/a}$. Since $out(p, V) = out(q, V)$, we have $p \xrightarrow{x'/a}$ and, from Proposition 7, $p \xrightarrow{\{a\}/a}$. This contradicts $p \not\xrightarrow{\{a\}/a}$.

Lemma 1 *In any FFSM* P, *let* $p, q \in P, x \in X$ *and* $y \in Y$, *if* $p \xrightarrow{x/y} q$ *then there exists* $x' \in \lfloor Acc(p) \rfloor \cup \lceil Ref(p) \rceil$ *such that* $p \xrightarrow{x'/y} q$.

Proof: See the proof of Prosistion 10 in which $p \xrightarrow{x/y}$ is treated as $p \xrightarrow{x/y} q$.

Theorem 5 Proof: Since TS_I is given to be complete, for $\mathsf{M} \in \mathcal{F}_{ffsm}$, if $\mathsf{M} \sim \mathsf{P}_F$ does not hold, then there must exist $\rho = x_1 \ldots x_{i-1}.x_i \in pref(TS_I)$ such that $p_0 \xrightarrow{x_1/y_1 \ldots x_{i-1}/y_{i-1}} p_{i-1}$, $m_0 \xrightarrow{x_1/y_1 \ldots x_{i-1}/y_{i-1}} m_{i-1}$ such that $y_j \neq \Theta$ for $1 \leq j < i$ and 1) $p_{i-1} \xrightarrow{x_i/y_i}$ and $m'_{i-1} \not\xrightarrow{x_i/y_i}$ or 2) $p_{i-1} \not\xrightarrow{x_i/y_i}$ and $m'_{i-1} \xrightarrow{x_i/y_i}$. (That an FFSM outputs the first Θ implies that the FFSM subsequently outputs Θ and stays in the sink state.)

According to Algorithm 2 and Lemma 1, we have $\rho' = x'_1 \ldots x'_{i-1}.x'_i \in pref(TS_O)$, $p_0 \xrightarrow{x'_1/y_1 \ldots x'_{i-1}/y_{i-1}} p_{i-1}$ and $m_0 \xrightarrow{x'_1/y_1 \ldots x'_{i-1}/y_{i-1}} m_{i-1}$. If $y_i \neq \Theta$, considering case 1, then $y_i \in x_i$ since $p_{i-1} \xrightarrow{x_i/y_i}$. According to Algorithm 2 and Lemma 1, $y_i \in x'_i$; furthermore from Proposition 7, $p_{i-1} \xrightarrow{x'_i/y_i}$, and $m'_{i-1} \not\xrightarrow{x'_i/y_i}$ because if $m'_{i-1} \xrightarrow{x'_i/y_i}$ then $m'_{i-1} \xrightarrow{x_i/y_i}$. Now consider case 2) for $y_i \neq \Theta$. $y_i \in x_i$ since $m_{i-1} \xrightarrow{x_i/y_i}$, and $y_i \in x_i \backslash init(p_{i-1})$ since $p_{i-1} \not\xrightarrow{x_i/y_i}$. Obviously, there exists $x'_i \in \lceil Ref(p_{i-1}) \rceil$ such that $(x_i \backslash init(p_{i-1})) \subseteq x'_i$. $y_i \in x'_i$, so from Proposition 7, $p_{i-1} \not\xrightarrow{x'_i/y_i}$ and $m'_{i-1} \xrightarrow{x'_i/y_i}$.

Consider case 1) for $y_i = \Theta$. Since $p_{i-1} \xrightarrow{x_i/\Theta}$, $x_i \in Ref(p_{i-1})$. From this there exists $x'_i \in \lceil Ref(p_{i-1}) \rceil$ such that $x_i \subseteq x'_i$. Thus, $p_{i-1} \xrightarrow{x'_i/\Theta}$, and $m'_{i-1} \not\xrightarrow{x'_i/\Theta}$ because from Proposition 7 if $m'_{i-1} \xrightarrow{x'_i/\Theta}$ then $m'_{i-1} \xrightarrow{x_i/\Theta}$. Consider case 2) for $y_i = \Theta$. Since $p_{i-1} \not\xrightarrow{x_i/\Theta}$, $x_i \in Acc(p_i$. From this there exists $x'_i \in \lfloor Acc(p) \rfloor$ such that $x'_i \subseteq x_i$ and for any $a \in x'_i$, $p_{i-1} \xrightarrow{x_i/a}$. Thus $p_{i-1} \not\xrightarrow{x'_i/\Theta}$ and, from Proposition 7, $m'_{i-1} \xrightarrow{x'_i/\Theta}$.

Lemma 2 *Given a test case* T *for an LTS specification* S *w.r.t.* $\approx_f$, *for any* $\sigma_i \in Tr(t_0) \cap Tr(s_0)$, *there exists* $\sigma_j \in Pur(\mathsf{T})$ *such that* σ_i *is a prefix of* σ_j.

Proof: Let $t = t_0$ **after** σ_j, according to Definition 11, t is labeled with **fail** only if $init(t) \notin Ref(\sigma_j, s_0)$. We only consider $\sigma_j \in Tr(s_0)$, because if $\sigma_j \notin Tr(s_0)$, since $\sigma_i \in Tr(s_0)$ is a prefix of σ_j, then there exists a prefix σ_k of σ_j such that $\sigma_k \in Tr(s_0)$ and σ_i is a prefix of σ_k. At first let $\sigma_j = \sigma_i$. If $\ell_{\approx f}(t) =$ **fail** then $init(t) \notin Ref(\sigma_j, s_0)$. This implies $init(t) \neq \emptyset$ and for all $a \in init(t)$ $\sigma_j.a \in Tr(s_0)$. Let $\sigma_j = \sigma_j.a$. If $\ell_{\approx f}(t) =$ **fail** then repeat the above process. Since any $\sigma \in Tr(t_0)$ is a finite sequence, it is impossible for the above process to repeat for ever, so there must exist such a $\sigma_j \in Tr(t_0)$ that $\ell_{\approx f}(t) =$ **pass.**

Theorem 6 Proof: Let TS_{ffsm} be the test suite for the FFSM P_F and TS_{lts} be the converted test suite for the LTS S. From Proposition 2, any $\mathsf{T} \in TS_{lts}$ is sound. Assume that $\mathsf{M} \approx_f \mathsf{S}$ does not hold. For any $\mathsf{M} \in \mathcal{F}$ we have $\mathsf{M}' \in \mathcal{F}_{ffsm}$ and from Theorem 2, if $\mathsf{M} \approx_f \mathsf{S}$ does not hold then $\mathsf{M}' \sim \mathsf{P}_F$ does not hold. Since TS_{ffsm} is complete over $\mathcal{F}_{ffsm}$, there must exist $x_1 \ldots x_{i-1}.x_i \in pref(TS_{ffsm})$ such that $p_0 - x_1/y_1 \ldots x_{i-1}/y_{i-1} \rightarrow p_{i-1}$, $m'_0 - x_1/y_1 \ldots x_{i-1}/y_{i-1} \rightarrow m'_{i-1}$ such that $y_j \neq \Theta$ for $1 \leq j < i$, and 1) $p_{i-1} - x_i/y_i \rightarrow$ and $m'_{i-1} \not\rightarrow x_i/y_i \rightarrow$ or 2) $p_{i-1} \not\rightarrow x_i/y_i \rightarrow$ and $m'_{i-1} - x_i/y_i \rightarrow$. Let $\sigma = y_1 \ldots y_{i-1}$ and $\mathsf{T} \in TS_{ffsm}$ be the corresponding test case.

Consider case 1). If $x_i \in Ref(p_{i-1})$ then there exists $p_{i-1} - x_i/\Theta \not\rightarrow$, thus since $m'_{i-1} \not\rightarrow x_i/\Theta \not\rightarrow$, $x_i \in Acc(m'_{i-1})$. From Proposition 9, $x_i \in Ref(p_{i-1})$ also implies $x_i \in Ref(s_0, \sigma)$ and hence according to Definition 11, $\sigma \in Pur(\mathsf{T})$. Moreover, from Proposition 9 and the definition of $Acc(p)$, $x_i \in Acc(m'_{i-1})$ also implies $x_i \notin Ref(m_0, \sigma)$ and hence $\sigma \notin Obs_{(\mathrm{T},\mathrm{M})}$. According to Definition 8, M fails T.

Otherwise, $x_i \in Acc(p_{i-1})$. From the FFSM definition, $p_{i-1} - x_i/y_i \rightarrow$ implies $y_i \in X_i$; furthermore according to Algorithm 3 $\sigma.y_i \in Tr(\mathsf{T})$. Therefore from Lemma 2 there exists $\sigma' \in Pur(\mathsf{T})$ such that $\sigma.y_i$ is a prefix of σ'. On the other hand, from Proposition 9, $m'_{i-1} \not\rightarrow x_i/y_i \rightarrow$ implies $\sigma.y_i \notin Tr(m_0)$ and hence $\sigma' \notin Obs_{(\mathrm{T},\mathrm{M})}$. According to Definition 8, M also fails T.

Consider case 2). If $x_i \in Ref(p_{i-1})$ then there exists $p_{i-1} - x_i/\Theta \rightarrow$. Thus $y_i \neq \Theta$. This again implies $x_i \in Acc(m'_{i-1})$. As we have proved, M fails T.

If $x_i \notin Ref(p_{i-1})$, then from Proposition 9 $x_i \notin Ref(s_0, \sigma)$ and furthermore from Definition 11 $\ell_{\approx f}(\sigma) =$ **fail**. If $y_i = \Theta$ then $x_i \in Ref(m'_{i-1})$ and furthermore from Proposition 9 $x_i \in Ref(\sigma, m_0)$. This implies $\sigma \in Obs_{(\mathrm{T},\mathrm{M})}$ and according to Definition 8, M also

fails T. If $y_i \neq \Theta$ then $m'_{i-1} - x_i/y_i \rightarrow$ implies $y_i \in x_i$ and $\sigma.y_i \in Tr(m_0)$ from Proposition 9, that is, $\sigma.y_i \in Obs_{(T,M)}$; moreover $p_{i-1} \not\rightarrow x_i/y_i \rightarrow$ implies $\sigma.y_i \notin Tr(s_0)$. According to Algorithm 3, $\sigma.y_i \in Tr(T)$ and from Definition 11, $\ell_{\approx f}(\sigma.y_i) =$ **fail**. Therefore according to Definition 8, M also fails T.

References

[1] J. Arkko. On the existence and production of state identification machines for labeled transition systems. In *IFIP Formal Description Techniques VI*, pages 351–365, 1993.

[2] F. Belina and D. Hogrefe. The CCITT–specification and description language SDL. *Computer Networks and ISDN Systems*, 16(4):331–341, 1989.

[3] G. v. Bochmann and A. Petrenko. Protocol testing: Review of methods and relevance for software testing. In *Proceeding of the ACM 1994 International Symposium on Software Testing and Analysis*, pages 109–124, Seattle USA, 1994.

[4] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[5] G. v. Bochmann, A. Petrenko, and M. Y. Yao. Fault coverage of tests based on finite state models. In *IFIP 7th International Workshop on Protocol Test Systems*, pages 91–106, Japan, 1994.

[6] E. Brinksma. A theory for the derivation of tests. In *IFIP Protocol Specification, Testing, and Verification VIII*, pages 63–74, 1988.

[7] E. Brinksma and et al. A formal approach to conformance testing. In *IFIP 2th International Workshop on Protocol Test Systems*, pages 349–363, 1990.

[8] S. Budkowski and P. Dembinski. Introduction to ESTELLE: A specification language for distributed systems. *Computer Networks and ISDN Systems*, 14(1):3–23, 1987.

[9] A. R. Cavalli and S. U. Kim. Automated protocol conformance test generation based on formal methods for LOTOS specifications. In *IFIP 5th International Workshop on Protocol Test Systems*, pages 212–220, 1992.

[10] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.

[11] Brookes S. D., Hoare C. A. R., and Roscoe. A theory of communicating sequencetial processes. *JACM*, 31(3):560–599, 1984.

[12] R. De Nicola and M. Hennessy. Testing equivalences for processes. In *Theoretical Computer Science (34)*, pages 83–133, 1984.

[13] K. Drira, P. Azema, B. Soulas, and A.M. Chemali. Testability of a communicating system through an environment. In *4th International Joint Conf. on Theory and Practice of Software Development*, pages 329–341, 1993.

[14] K. Drira, P. Azema, and F. Vernadat. Refusal graphs for conformance tester generation and simplification: a computational framework. In *IFIP Protocol Specification, Testing, and Verification XIII*, pages 257–272, 1994.

[15] S. Fujiwara and G. v. Bochmann. Testing nonterministic finite state machine with fault coverage. In J. Kroon, J. Heijink, and E. Brinksma, editors, *IFIP 4th International Workshop on Protocol Test Systems*, pages 267–280, 1991.

[16] S. Fujiwara et al. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, SE-17(6):591–603, 1991.

[17] C. R. A. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[18] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill Computer Science Series, New York, 1970.

[19] G. Leduc. Conformance relation, associated equivalence and new canonical tester in LOTOS. In *IFIP Protocol Specification, Testing, and Verification XI*, pages 249–264, 1991.

[20] G. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, SE-20(2):149–162, 1994.

[21] G. Luo, A. Petrenko, and G. v. Bochmann. Selecting test sequences for partially-specified nondeterministic finite machines. In *IFIP 7th International Workshop on Protocol Test Systems*, pages 91–106, 1994.

[22] A. Petrenko. Checking experiments with protocol machines. In *IFIP 4th International Workshop on Protocol Test Systems*, pages 83–94, 1991.

[23] A. Petrenko, G. v. Bochmann, and R. Dssouli. Conformance relations and test

derivation. In *IFIP 6th International Workshop on Protocol Test Systems*, pages 91–106, 1993.

[24] A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Testing Deterministic Implementations from Nondeterministic Specification. In *IFIP 9th International Workshop on Protocol Test Systems*, pages 79–94, 1996.

[25] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das. Nondeterministic state machines in protocol conformance testing. In *IFIP 6th International Workshop on Protocol Test Systems*, pages 363–378, 1993.

[26] D. H. Pitt and D. Freestone. The derivation of comformance tests from LOTOS specifications. *IEEE Transactions on Software Engineering*, SE-16(12):1337–1343, 1990.

[27] K. Sabnani and A. T. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285–297, 1988.

[28] Q. M. Tan, A. Petrenko, and G. v. Bochmann. Modeling basic LOTOS by FSMs for conformance testing. In *IFIP Protocol Specification, Testing, and Verification XV*, pages 137–152, 1995.

[29] Q. M. Tan, A. Petrenko, and G. v. Bochmann. A framework for conformance testing of systems communicating through rendezvous. In *26th IEEE International Symposium on Fault-Tolerant Computing*, pages 230-238, 1996.

[30] Q. M. Tan, A. Petrenko, and G. v. Bochmann. A test generation tool for specifications in the form of state machines. In *IEEE International Conference on Communications – 96*, Dallas, USA, 1996.

[31] J. Tretmans. Test case derivation from LOTOS specifications. In *IFIP 2th International Conf. on Formal Description Techniques for Distributed Sysytems and Communication Protocols*, pages 345–359, 1990.

[32] J. Tretmans, P. Kars, and E. Brinskma. Protocol conformance testing: A formal perspective on ISO IS-9646. In *IFIP 4th International Workshop on Protocol Test Systems*, 1991.

[33] J. Tretmans. A formal approach to conformance testing. Ph.D. thesis, Twente University, 1992.

[34] J. Tretmans. Testing labelled transition systems with inputs and outputs. In *IFIP*

8th International Workshop on Protocol Test Systems, pages 461–476, 1995.

[35] R. J. van Glabbeek. The linear time-branching time spectrum. *Lecture Notes on Computer Science*, 458:278–297, 1990.

[36] C. D. Wezeman. The CO-OP method for compositional derivation of conformance testers. In *IFIP Protocol Specification, Testing, and Verification IX*, pages 145–158, 1990.