Modeling Basic LOTOS by FSMs for Conformance Testing

Q. M. Tan, A. Petrenko and G. v. Bochmann
Department d'IRO, Université de Montréal
C.P. 6128, Succ. Centre-ville, Montréal, H3C 3J7, Canada
E-mails: {tanq, petrenko, bochmann}@iro.umontreal.ca
Fax: (514)343-5834

Abstract

A challenging issue is the derivation of a finite test suite from a given LOTOS specification modeled by a labeled transition system (LTS) such that complete fault coverage is guaranteed for a certain class of implementations with respect to a particular conformance relation. It is shown in this paper that this problem can be solved by translating an LTS into an input/output finite state machine (FSM) for trace or failure semantics, respectively, and subsequently applying existing FSM-based methods for test derivation with complete fault coverage. It is also demonstrated that the obtained tests can be further optimized taking into account the specifics of the FSMs constructed from the LTSs.

1 INTRODUCTION

Conformance testing for communication protocols is one of the essential and challenging issues. Because of the complexity of protocols, it is generally accepted that formal techniques must be used. Amongst such formal description techniques (FDTs) are LOTOS, which is based on ***Labeled Transition Systems*** (LTSs), and SDL and Estelle, 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 [14].

Systematic approaches have been developed for protocol conformance testing and the generation of appropriate test suites based on the FSM model. Most work in this area is limited to completely specified, deterministic specifications [5, 8, 17, 13]. However, some recent research has addressed nondeterministic and partially specified specifications [11, 12]. A number of competing methods for deriving tests from FSMs that guarantee complete fault coverage for the given maximal number of states in implementations have been elaborated [14].

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. In the protocol engineering community, there has been much work done on the derivation of test suites from a given basic LOTOS, or corresponding LTS-based specification [3, 16, 19, 6], most of which deal with nondeterminism. Test derivation from LTSs in various semantics is currently an active research area.

Several attempts have been made to apply the ideas underlying the FSM-based methods to the LTS model [4, 7, 1]. In particular, this research is directed towards redefining the state identification and eventually the checking experiments in the LTS realm. [4] tries the UIO-based state identifiers which, as it is well known, do not always exist; [7] considers the characterization sets; and [1] introduces the state identification machines. However, in spite of these attempts, the problem of deriving a finite test suite with complete fault coverage from an arbitrary LTS for a given conformance relation remains open. Here we take another approach, initially outlined in [14]. It is suggested there 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 proper FSM constructed from the LTS in 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 two particular types of semantics, namely, trace and failure semantics. We formally show that LTS specifications can be modeled by proper input/output (I/O) FSMs in trace or failure semantics, and that complete test suites produced from the corresponding FSMs and then converted back to the LTS formalism can guarantee the given conformance relation when the number of multi-states [7] of any implementation is bounded by a known integer. We also demonstrate that the test suites can be further optimized taking into account the specifics of the FSMs derived from the LTSs.

In Section 2, we give basic definitions and notations of the FSM and LTS models and conformance relations. In Section 3, the FSM models for LTSs, in failure semantics and in trace semantics respectively, are defined, and the transformations are validated. In Section 4, the proposed FSM approach to test derivation from a given LTS specification is illustrated, and the optimization of the tests as well as other related problems are discussed.

2 BASIC DEFINITIONS AND NOTATIONS

In this section, we recall some basic definition and notations which are used to discuss both finite state machines and labeled transition systems [3, 12].

2.1 Finite State Machine

Definition 1 *Finite State Machine (FSM).* A 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.

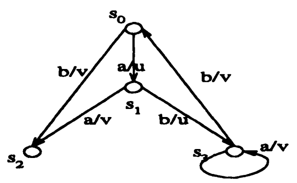


Figure 1: An FSM graph

- h is a behavior function, $h : D \to powerset(S \times Y)\backslash\{\emptyset\}$, where $D \subseteq S \times X$ and $\emptyset$ is the empty set. $(q, b) \in h(p, a)$ is also written $p{-}a/b{\to}q$, 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 $D = S \times X$ then the finite state machine is called completely specified or a *complete* FSM (CFSM), otherwise, it is partially specified or a *partial* FSM (PFSM). If $|h(p, a)| = 1$ for all $(p, a) \in D$ then the FSM is *deterministic* (DFSM); otherwise, it is *nondeterministic* (NFSM). Note that the finite state machines in the above definition may be *complete nondeterministic* (CNFSM) or *partial nondeterministic* (PNFSM).

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.

By convention, the traces of an FSM S are the sequences in $Tr(s_0)$, thus we also denote $Tr(\mathsf{S}) = Tr(s_0)$. In the following, we define two relations between FSMs, which are useful for test generation.

Definition 2 *Reduction.* The reduction relation between two states p and q in FSMs, written $p \leq q$, holds if and only if $Tr^{in}(q) \subseteq Tr^{in}(p)$ and for all $\gamma^{in} \in Tr^{in}(q)$: $Tr(p) \subseteq Tr(q)$.
Given two FSMs S and I, we say that I is a reduction of S, written $\mathsf{I} \leq \mathsf{S}$, if and only if $i_0 \leq s_0$.

notation	meaning
Γ	$X \times Y$, a set of input/output pairs; v denotes such a pair
Γ^*	set of sequences over Γ; γ denotes such a sequence
$p{=}\varepsilon{\Rightarrow}q$	$p = q$; ε is the empty sequence
$p{=}\gamma{\Rightarrow}q$	there exist p_k for $0 \leq k \leq n$ such that $p = p_0{-}v_1{\to}p_1 \ldots {-}v_n{\to}p_n = q$; $\gamma = v_1 \ldots v_k$
$p{=}\gamma{\Rightarrow}$	there exists q such that $p{=}\gamma{\Rightarrow}q$
$Tr(p)$	$Tr(p) = \{\gamma \in \Gamma^* \mid p{=}\gamma{\Rightarrow}\}$
γ^{in}	for $\gamma \in \Gamma^*$, $\gamma^{in} \in X^*$ is an input sequence obtained by deleting all outputs in γ
$Tr^{in}(p)$	$Tr^{in}(p) = \{\gamma^{in} \mid p{=}\gamma{\Rightarrow}\}$

Table 1: Notation for finite state machines

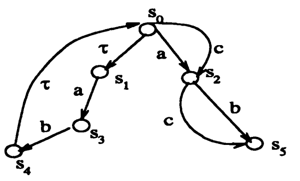


Figure 2: An LTS graph

This conformance relation is defined in [14] and requires that all output sequences that are produced by the implementation in response to all acceptable input sequences should be described by its specification. Any sequence in $Tr^{in}(s_0)$ is said to be an acceptable input sequence for the FSM S.

Definition 3 *Equivalence.* The equivalence relation between two states p and q in FSMs, written $p \sim q$, holds if and only if $Tr(p) = Tr(q)$.
Given two FSMs S and I with initial states s_0 and i_0 respectively, we say that I is equivalent to S, written S $\sim$ I, if and only if $s_0 \sim i_0$.

The above definition of equivalence is given in [12, 11] for CNFSMs and PNFSMs, similar to that in [8, 5] for deterministic and completely specified FSMs.

It can be shown that the reduction relation is a pre-order one and I $\sim$ S if and only if S $\leq$ I and I $\leq$ S.

Definition 4 *Observable FSMs (OFSMs).* An FSM S is said to be observable if and only if for all $p \in S$ and all $a/b \in \Gamma$, $|\{q \mid \forall(q,b) \in h(p,a)\}| \leq 1$.

In observable machines, a state and an I/O pair can uniquely determine at most one next state. However, OFSMs may still be nondeterministic in the sense that a state and an input can not determine a unique next state and a unique output. On the other hand, deterministic FSMs are observable. Any non-observable FSM can be transformed into an equivalent OFSM, in which each state corresponds to a subset of states in the original FSM. The test generation methods for ONFSMs can be found in [11, 12, 15].

2.2 Labeled Transition System

Definition 5 *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.

The notations are shown in Table 2 that are relevant to a given LTS, as introduced in

notation	meaning
Σ^*	set of sequences over Σ; σ denotes such a sequence
$p{-}\mu_1\ldots\mu_n{\rightarrow}q$	there exists p_k for $0 \le k \le n$ such that $p = p_0{-}\mu_1{\rightarrow}p_1\ldots{-}\mu_n{\rightarrow}p_n = q$
$p{=}\varepsilon{\Rightarrow}q$	$p{-}\tau^n{\rightarrow}q$ $(1 \le n)$ or $p = q$ (note: τ^n means n times τ)
$p{=}a{\Rightarrow}q$	there exist p_1, p_2 such that $p_1{=}\varepsilon{\Rightarrow}p_1{-}a{\rightarrow}p_2{=}\varepsilon{\Rightarrow}q$
$p{=}\sigma{\Rightarrow}q$	there exists p_k for $0 \le k \le n$ such that $p = p_0{=}a_1{\Rightarrow}p_1\ldots{=}a_n{\Rightarrow}p_n = q; \sigma = a_1\ldots a_n$
$p{=}\sigma{\Rightarrow}$	there exists q such that $p{=}\sigma{\Rightarrow}q$
$p{\not=}\sigma{\Rightarrow}$	no q exists such that $p{=}\sigma{\Rightarrow}q$
$out(p)$	$out(p) = \{a \in \Sigma \mid p{=}a{\Rightarrow}\}$
p **after** σ	p **after** $\sigma = \{q \in S \mid p{=}\sigma{\Rightarrow}q\}$
$Tr(p)$	$Tr(p) = \{\sigma \in \Sigma^* \mid p{=}\sigma{\Rightarrow}\}$

Table 2: Notation for labeled transition systems

[3]. 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. Similarly, $Tr(\mathsf{S}) = Tr(s_0)$, S **after** $\sigma = s_0$ **after** σ, and the sequences in $Tr(\mathsf{S})$ are called the *traces* of S.

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.

There are different criteria for determining whether an implementation conforms to its LTS specification [14]. In this paper, we use the following implementation relations as the criteria of the LTS conformance.

Definition 6 *Trace equivalence.* The trace equivalence relation between two LTSs S and I, written $\mathsf{S} =_{\mathbf{tr}} \mathsf{I}$, holds if and only if $Tr(\mathsf{S}) = Tr(\mathsf{I})$.

The above definition of trace equivalence corresponds to the equivalence relation between specifications and implementations in the FSM formalism [5, 8, 11]. The relation requires that a conforming implementation has the same set of traces as its specification.

Definition 7 *Refusal function.* The refusal function of an LTS S, $Ref : S \times \Sigma^* \to powerset(powerset(\Sigma))$, is defined at each $p \in S$ for each σ in Σ^* by $Ref(p, \sigma) = \{A \subseteq \Sigma \mid \exists q \in p \textbf{ after } \sigma\ (\forall a \in A, q{\not=}a{\Rightarrow})\}$.

The refusal function gives the set of all the action sets which may be refused by the LTS after a given trace σ is executed from a given state p. $Ref(p, \sigma)$ is called a *refusal set* at p after σ. Similarly, the refusal set of the LTS after σ is that of its initial state after σ, so $Ref(s_0, \sigma) = Ref(\mathsf{S}, \sigma)$. $Ref(p, \varepsilon)$ is called a *state refusal set* of state p. The notation $Ref(p, \varepsilon)$ may be simplified into $Ref(p)$.

Definition 8 *Failure reduction.* The failure reduction relation between two LTSs S and I, written I **red** S, holds if and only if for all $\sigma \in \Sigma^*$ $Ref(\mathsf{I}, \sigma) \subseteq Ref(\mathsf{S}, \sigma)$.

The failure reduction relation [3] between the specification S and its implementation I requires that everything that I does must be allowed by S. The definition states that for any sequence σ in Σ^*, if σ is a trace of I then it is also a trace of S; and after σ is applied, if an action set A may be refused by I then A may also be refused by S.

$P{=}\sigma{\Rightarrow}Q \;=_{\mathbf{def}}\; \forall q \in Q\; \exists p \in P\; (p{=}\sigma{\Rightarrow}q)$					
$Tr(P)$	$=$	$\bigcup_{(p\in P)} Tr(p)$	$out(P)$	$=$	$\bigcup_{(p\in P)} out(p)$
$Ref(P,\sigma)$	$=$	$\bigcup_{(p\in P)} Ref(p,\sigma)$	$Ref(P)$	$=$	$\bigcup_{(p\in P)} Ref(p)$
P **after** σ	$=$	$\bigcup_{(p\in P)} p$ **after** σ			

Table 3: Extended notations for labeled transition systems

Definition 9 *Failure equivalence (or testing equivalence).* The failure equivalence relation between two LTSs S and I, written S $=_{\mathbf{te}}$ I, holds if and only if for each σ in $\Sigma^*, Ref(\mathsf{S},\sigma) = Ref(\mathsf{I},\sigma)$.

Obviously, S $=_{\mathbf{te}}$ I implies that S **red** I and I **red** S. Therefore, the failure equivalence relation not only states that everything that I does must be allowed by S, but also requires that everything prescribed by S should be implemented by I.

If a set A is refused after σ, obviously, each $B \subseteq A$ is refused as well. Thus, we may consider a minimal representation of the refusal functions of LTSs, denoting $Ref^{min}(p,\sigma)$, by deleting each element in $Ref(p,\sigma)$ that is a subset of another. Generally, for a set of sets R, $R^{min} = R\backslash\{A \mid \exists B \in R\; (A \subset B)\}$.

In the case of nondeterminism, after an observable action sequence, an LTS may enter one of a number of different states. In order to consider all possibilities, a state subset (multi-state [7]), which contains all the states which are reachable by the LTS after this action sequence, is used.

Definition 10 *Multi-state set.* The multi-state set of LTS S is a set $\Pi_S = \{S_i \subseteq S \mid \exists\sigma \in Tr(\mathsf{S})\; (\mathsf{S}\ \mathbf{after}\ \sigma = S_i)\}$.

Note that the empty sequence ε is supposed to be in Σ^*. $S_0 = s_0$ **after** ε belongs to the multi-state set, and is called the *initial multi-state*. The multi-state set can be obtained by a known algorithm which performs a deterministic transformation of a nondeterministic automaton using the trace-equivalence [10, 7, 4]. For Figure 2, 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.

As said before, in the case of nondeterminism, after an observable action sequence, different states in a corresponding multi-state may be reached. Thus from the test perspective it makes sense to define the transition checking and state identification on multi-states, rather than single states. The viewpoint is reflected in the FSM realm by the presentation of a nondeterministic FSM specification as an observable FSM, in which each state is a subset of states of the non-observable FSM. The viewpoint is also reflected by the *refusal graphs* [6].

Next, we extend some of the above notations to a subset of states, as shown in Table 3. From the extended notations, we can directly derive the following proposition:

Proposition 1 *Given LTSs* S *and* I *with the initial multi-states* S_0 *and* I_0,

1. $Tr(\mathsf{S}) = Tr(S_0)$;
2. S **after** σ $=$ S_0 **after** σ;
3. $Ref(\mathsf{S},\sigma) = Ref(S_0,\sigma)$;
4. $Ref(\mathsf{S},\sigma) = Ref(S_i)$ *if* S **after** $\sigma = S_i$;

3 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.

3.1 General Idea

In the context of conformance testing, an LTS implementation under test (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 [9]. 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 [14]. 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 failure semantics, respectively.

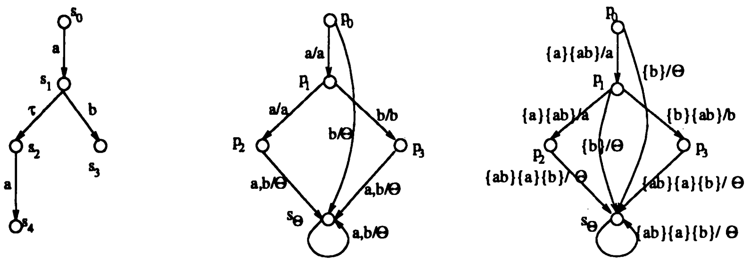
In the case that we are only interested in trace semantics, all relevant properties can be tested by offering single actions. Therefore we assume a simplified FSM model in this case which defines the behavior only for single action offers, thereby reducing the number of inputs from $2^{|\Sigma|} - 1$ 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 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 3 (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.

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



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

Figure 3: Representation of an LTS using the FSM model

completed successfully [3]. 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 3 (a) in failure semantics is shown in Figure 3 (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. The mapping from multi-states of the LTS to states of the FSM as well as the sink state s_Θ are the same as for 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 failure equivalence is a refinement of trace equivalence.

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

3.2 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 11 *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 P =$< P, X, Y, h, p_0 >$, 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 $a \in X$,
  - $(\psi(S_j), a) \in h(\psi(S_i), a)$ if and only if $S_i = a \Rightarrow S_j$;
  - $(s_\Theta, \Theta) \in h(\psi(S_i), a)$ if and only if $a \in \Sigma \backslash out(S_i)$;
  - $\{(s_\Theta, \Theta)\} = h(s_\Theta, a)$.

According to the definition, it is possible to construct from the LTS S the corresponding TFSM P, which is a complete FSM. Figure 3 (b) is an example of the TFSM w.r.t. the LTS in Figure 3 (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 $evo_t(\sigma)$ to represent an input/output sequence such that both of its input part and output part are σ. Formally, we define $evo_t(\varepsilon) = \varepsilon$; and $evo_t(\sigma.a) = evo_t(\sigma).a/a$. TFSMs have the following properties.

Proposition 2 *Any TFSM is deterministic and completely specified.*

Proof: See [18].

Proposition 3 *Given an LTS* S *and its corresponding TFSM* P*, for all $\sigma \in \Sigma^*$ and all $\gamma = evo_t(\sigma) \in \Gamma^*$, $\sigma \in Tr(\mathsf{S})$ if and only if $\gamma \in Tr(\mathsf{P})$.*

Proposition 3 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 if and only if 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* S, I *and their corresponding TFSMs* S′, I′, $\mathsf{I} =_{\mathrm{tr}} \mathsf{S}$ *if and only if* $\mathsf{I}' \sim \mathsf{S}'$.

Proof: See [18].

By virtue of Theorem 1, the tests 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 CDFSMs [5, 8, 13, 17] are fully applicable to derive tests from LTS specifications. However, the tests derived from the TFSM model should be transformed to tests in the LTS context, because LTSs have a different, i.e. rendez-vous, interface to interact with their environment. We explain this transformation in Section 4.

3.3 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 constructuion, 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 12 *Failure finite state machine w.r.t. LTS.* Given an LTS $\mathsf{S} = < S, \Sigma, \Delta, s_0 >$, a failure finite state machine w.r.t. S, is a finite state machine $\mathsf{P} = < P, X, Y, h, p_0 >$, such that:

- $X = powerset(\Sigma) \backslash \{\emptyset\}$.
- $Y \backslash \{\Theta\} = \Sigma$, 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 $\psi : \Pi_S \to P \backslash \{s_\Theta\}$ and for all $S_i \in \Pi_S$ and all $A \in X$,
  - $(\psi(S_j), a) \in h(\psi(S_i), A)$ if and only if $a \in A$ and $S_i = a \Rightarrow S_j$, or
  - $(s_\Theta, \Theta) \in h(\psi(S_i), A)$ if and only if $A \in Ref(S_i)$;
  - $\{(s_\Theta, \Theta)\} = h(s_\Theta, A)$.

Figure 3 (c) shows an example of the FFSM w.r.t. the LTS in Figure 3 (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 $a \in A$, or "B/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 "B/Θ"; and once the first Θ occurs, the FFSM also enters the state s_Θ, and outputs Θ for any subsequent input.

It can be easily shown that $\{a \in \Sigma | p_i{-}A/a{\to}\} = out(S_i)$ and $\{A \subseteq \Sigma | p_i{-}A/\Theta{\to}\} \cup \{\emptyset\} = Ref(S_i)$ for each state p_i in P, where S_i is a multi-state in its LTS S and $p_i = \psi(S_i)$. Therefore we define $out(p) = \{a \in \Sigma \mid p{-}A/a{\to}\}$ and $Ref(p) = \{A \subseteq \Sigma \mid p{-}A/\Theta{\to}\} \cup \{\emptyset\}$. Similarly, for $\sigma \in \Sigma^*$, we also define $evo_f(\sigma)$: $evo_f(\varepsilon) = \varepsilon$; and $evo_f(\sigma.a) = evo_f(\sigma).A/a$ where $a \in A$ and $A \in X$. The properties of FFSMs are expressed by the following propositions.

Proposition 4 *For any FFSM* P*, for all $p \in P$ and all $B \in Ref^{min}(p)$ $out(p) \cup B = \Sigma$.*

Proof: See [18].

From $\Sigma = out(p) \cup B$ we get $\Sigma \backslash out(p) \subseteq B$, which means that in any state, the output complement is always refused.

Proposition 5 *Any FFSM is observable and completely specified.*

Proof: See [18].

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

Proposition 6 *Given an LTS* S *and its corresponding FFSM* P*, for all $\sigma \in \Sigma^*$ and all $\gamma = evo_f(\sigma) \in \Gamma^*$, if there exists $p \in P$ such that $p_0 = \gamma \Rightarrow p$, then $Ref(p) = Ref(\mathsf{S}, \sigma)$.*

Proof: See [18].

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 if and only if its FFSM may produce output Θ once A is applied after trace $evo_f(\sigma)$.

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

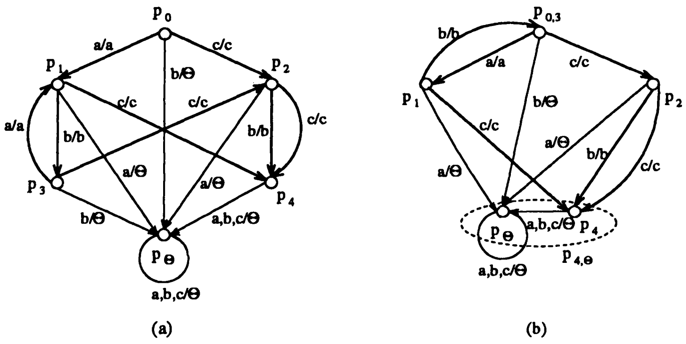


Figure 4: An example for test generation

theorem.

Theorem 2 *For any given two LTSs* S, I *and their corresponding FFSMs* S′, I′, (1) I **red** S *if and only if* $I' \leq S'$. (2) $I =_{te} S$ *if and only if* $I' \sim S'$.

Proof: See [18].

By virtue of Theorem 2, the tests for the FFSM model can be used to test the LTS implementations with respect to their specification for the conformance relations of failure semantics. The existing methods for CNFSMs and the reduction relation [15] or equivalence relation [11, 12] can be exploited to derive relevant tests from LTS specifications through the FFSMs. Like in the case of trace testing, the tests obtained should be translated into tests that obey a rendez-vous interface of LTSs.

However, it should be noted that the failure FSMs constitute a specific subclass of FSMs that have the following peculiarity from the test perspective: *Certain transitions are implied by others and may not require testing according to the LTS semantics.* (see Section 4.2.) This observation suggests that the test derivation methods based on the FSM model should be modified for FFSMs, rather than directly applied.

4 TEST GENERATION

4.1 Testing trace equivalence

It follows from the results of the previous section that the derivation of a finite test suite with complete fault coverage from an LTS specification with respect to trace equivalence can be solved by transforming the specification into an TFSM, applying a CDFSM-based method to it, and then converting the obtained tests back to the LTS formalism. The approach is illustrated by the following examples.

From the definition of the corresponding TFSM, we can get the TFSM shown in Figure 4 (a) for the LTS specification S of Figure 2. This TFSM is not minimal, so it is transformed into its minimal form P, shown in Figure 4 (b), as required by the W-method [5].

Let any LTS implementation of the LTS specification S be viewed as a TFSM and let the number of states in an equivalent minimal form of the TFSM be not more than the number of states in P. According to the W–method, the set of input sequences of a complete test suite TS is constructed in the following way: $TS^{in} = Q.(\{\varepsilon\} \cup \Sigma).W$, where Q is a state cover, $Q.(\{\varepsilon\} \cup \Sigma)$ is a transition cover and W is a characterization set. From the TFSM in Figure 4 we may select $Q = \{\varepsilon, a, b, c\}$ and $W = \{a, b.a\}$. The resulting test suite is as follows.

$\{a/a.b/b.a/a,\ b/\Theta.b/\Theta.a/\Theta,\ c/c.b/b.a/\Theta,\ a/a.a/\Theta.a/\Theta,\ a/a.a/\Theta.b/\Theta.a/\Theta,$
$a/a.b/b.b/\Theta.a/\Theta,\ a/a.c/c.a/\Theta,\ a/a.c/c.b/\Theta.a/\Theta,\ b/\Theta.a/\Theta.a/\Theta,\ b/\Theta.a/\Theta.b/\Theta.a/\Theta,$
$b/\Theta.b/\Theta.b/\Theta.a/\Theta,\ b/\Theta.c/\Theta.a/\Theta,\ b/\Theta.c/\Theta.b/\Theta.a/\Theta,\ c/c.a/\Theta.a/\Theta, c/c.a/\Theta.b/\Theta.a/\Theta,$
$c/c.b/b.b/\Theta.a/\Theta,\ c/c.c/c.a/\Theta,\ c/c.c/c.b/\Theta.a/\Theta\}$

Since all implementations of P are assumed to be TFSMs, in which the Θ for an input implies Θ for all subsequent inputs, there is a certain redundancy in this test suite. For example, the suffix $a/\Theta.a/\Theta$ of test case $b/\Theta.a/\Theta.a/\Theta$ is not necessary because of the Θ for the first input b. According to the LTS semantics, if b can not form a valid trace of S, then $b.a.a$ can not do it either. These suffixes can be removed and the resulting tests still constitute a complete test suite of the TFSM:

$\{a/a.b/b.a/a,\ b/\Theta,\ c/c.b/b.a/\Theta,\ a/a.a/\Theta,\ a/a.b/b.b/\Theta,\ a/a.c/c.a/\Theta,\ a/a.c/c.b/\Theta,$
$c/a.a/\Theta,\ c/c.b/b.b/\Theta,\ c/c.c/c.a/\Theta,\ c/c.c/c.b/\Theta\}$

In general, for any complete test suite of a given TFSM [2] w.r.t. a certain class of TFSMs, after removing the suffixes of tests that follow the first pair "b/Θ", the resulting test suite is also complete w.r.t. the same class. In order to state this in a formal way, we use $pref(TS)$ to represent all prefixes of tests in TS, i.e. $pref(TS) = \{\gamma_1 \mid \gamma_1 \in \Gamma^* \wedge \gamma_1.\gamma_2 \in TS\}$. The following theorem gives the validity of the simplification of tests.

Theorem 3 *Given a TFSM S, if TS is a complete test suite w.r.t. a certain class of TFSMs then* $TS' = \{\gamma \in pref(TS) \mid \exists \sigma.b \in \Sigma^*((\gamma = evo_t(\sigma.b) \wedge \gamma \in TS) \vee \gamma = evo_t(\sigma).b/\Theta)\}$ *is also a complete test suite w.r.t. the same class.*

Proof: See [18].

Another solution to the redundancy problem is to modify the existing method in such a way that the sink state is excluded from the computation. The null output Θ will distinguish the sink state from others; and 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.

As an example, we consider again the TFSM of Figure 4 (b), p_4 is equivalent to the sink state s_Θ, but we keep it separately. Exculding the sink state, we use the harmonized identifiers [13] $H = \{\{a, b\}, \{b.a\}, \{b.a\}, \{a, b\}\}$ rather than the characterization set W, because the W set also causes redundancy. A state cover is now $\{\varepsilon, a, a.c, c\}$. From this we obtain the transition cover $T = \{\varepsilon, a, b, c, a.a, a.b, a.c, a.c.a, a.c.b, a.c.c, c.a, c.b, c.c\}$. In the transition cover, b, $a.a$, $a.c.a$, $a.c.b$, $a.c.c$ and $c.a$ lead to s_Θ, so no identifier is needed to check the tail state. Thus, the resulting test suite is:

$\{b/\Theta,\ a/a.a/\Theta,\ c/a.a/\Theta,\ a/a.b/b.a/a,\ a/a.b/b.b/\Theta,\ a/a.c/c.a/\Theta,\ a/a.c/c.b/\Theta,$
$a/a.c/c.c/\Theta,\ c/c.b/b.a/\Theta,\ c/c.b/b.b/\Theta,\ c/c.c/c.a/\Theta,\ c/c.c/c.b/\Theta\}$

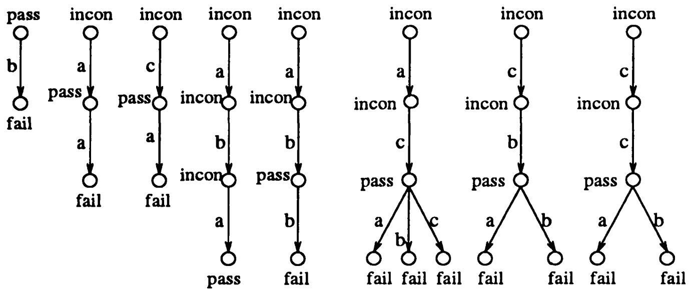


Figure 5: A test suite for the LTS specification in Figure 2

The test cases for the TFSM model can be transformed for the LTS testing by converting each action sequence into a corresponding LTS with state verdicts. Let $s_0 - a_1 \rightarrow s_1 \ldots s_{n-1} - a_n \rightarrow s_n$ be an LTS corresponding to action sequence $a_1.a_2 \ldots .a_n$, which outputs the first Θ at a_k in the TSFM model, $1 \leq k \leq n+1$. (Note that if there is no Θ output for this sequence, then k is assumed to be $n+1$) We have the state verdicts as follows.

$$s_i = \begin{cases} \textbf{pass} & i = k-1 \\ \textbf{fail} & i \geq k \\ \textbf{inconclusive} & \text{otherwise} \end{cases}$$

An LTS test suite obtained by transforming the above test suite of the TFSM in Figure 4 (b) is shown in Figure 5. This test suite can be used to test implementations of the LTS specification in Figure 2 with respect to trace equivalence.

4.2 Failure FSMs

In an abstract I/O FSM, every transition is usually completely independent of others. However, this is not the case for an FFSM, as we mentioned before; certain transitions are implied by others. Consider, for example, a transition from a multi-state S_i labeled by action a, in a given LTS. In the FFSM, such a transition yields exactly $2^{|\Sigma|-1}$ transitions with the same output a from the corresponding state. These transitions have different inputs denoting all the supersets of a and are implied by a single transition labeled by $\{a\}/a$.

Implied transitions in an FFSM should not be treated as completely independent for test derivation. The traditional transition checking approach relies on the general transition fault model, according to which any transition can be mutated independently of the others [2]. 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 $p_0 - \{ab\}/a \rightarrow p_1$ in the FFSM in Figure 3 (c) as an example, where once the first is checked there is no need to check the second. The reason is that if $p_0 - \{ab\}/\Theta \rightarrow s_\Theta$ is implemented then it implies $p_0 - \{a\}/\Theta \rightarrow s_\Theta$.

Another distinctive feature of FFSMs comes from the closure property of refusal sets. Consider a refusal set $Ref(S_i)$, $|Ref(S_i)| > 1$, of multi-state S_i in the given LTS. This set creates exactly $2^{|Ref(S_i)|-1}$ transitions in the corresponding FFSM. Again, there is no need to test all these transitions separately. Checking transitions corresponding to the minimal refusal set $Ref^{mim}(S_i)$ is sufficient.

Finally, similar to TFSMs, in FFSMs, the sink state does not require any identification and the transitions which leave this state do not require checking.

It follows from the above analysis that one can also derive tests for the LTS with respect to failure semantics based on the existing FSM-based methods for FFSMs in a way similar to TFSMs – by directly applying an existing method for a test suite and subsequently removing redundancy in it, or by modifying the existing method to avoid the redundancy. However, since the implications among transitions in FFSMs raise a new redundancy of tests, a further research is needed for removing or avoiding this redundancy. This is our work in progress.

5 CONCLUSION

LTSs are the basic semantics for LOTOS and other specification formalisms. This paper deals with test suite development from a specification given in the LTS formalism. We have shown that in the context of trace semantics, LTSs can be represented equivalently by an input/output FSM model – 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 a number of testing tools have been constructed already.

Trace FSMs are deterministic, completely specified FSMs, so the existing methods for CDFSMs 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, completely specified, nondeterministic FSMs, so the existing testing methods for CNFSMs can be applied to the FFSMs for the derivation of test suites to check the corresponding LTSs in the failure semantics. However, since certain transitions in FFSMs may not require testing according to the LTS semantics, an adaptation of the existing methods to FFSMs is needed to avoid redundancy. Our work in progress deals with this problem.

Acknowledgments

This work was supported by the Hewlett-Packard-NSERC-CITI Industrial Research Chair on Communication Protocols, Université de Montréal. The authors would like to thank S. A. Ezust for comments.

References

[1] J. Arkko. (1993) On the existence and production of state identification machines for labeled transition systems. In *IFIP Formal Description Techniques VI* (ed. R. L. Tenney, et al), 351–365.

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

[3] E. Brinksma. (1988) A theory for the derivation of tests. In *IFIP Protocol Specification, Testing, and Verification VIII* (ed. S. Aggarwal and K. Sabnani), 63–74.

[4] A. R. Cavalli and S. U. Kim. (1992) Automated protocol conformance test generation based on formal methods for LOTOS specifications. In *the IFIP 5th International Workshop on Protocol Test Systems* (ed. G.v. Bochmann, et al), 212–220.

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

[6] K. Drira, P. Azema, and F. Vernadat. (1994) Refusal graphs for conformance tester generation and simplification: a computational framework. In *IFIP Protocol Specification, Testing, and Verification XIII* (ed. A. Danthine, et al), 257-272, 1994.

[7] S. Fujiwara and G. v. Bochmann. (1991) Testing nonterministic finite state machine with fault coverage. In *the IFIP 4th International Workshop on Protocol Test Systems* (ed. J. Kroon, et al), 267–280.

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

[9] R. J. Glabbeek. (1990) The linear time-branching time spectrum. *Lecture Notes on Computer Science*, 14(1):25–59.

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

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

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

[13] A. Petrenko. (1991) Checking experiments with protocol machines. In *the IFIP 4th International Workshop on Protocol Test Systems* (ed. J. Kroon, et al), 83–94.

[14] A. Petrenko, G. v. Bochmann, and R. Dssouli. (1993) Conformance relations and test derivation. In *the IFIP 6th International Workshop on Protocol Test Systems* (ed. O. Rafig), 91–106.

[15] A. Petrenko, N. Yevtushenko, and G. v. Bochmann. (1994) Experiments on nondeterministic systems for the reduction relation. Technical Report 932, Dept. of I.R.O., University of Montreal.

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

[17] K. Sabnani and A. T. Dahbura. (1988) A protocol test generation procedure. *Com-*

puter Networks and ISDN Systems, 15(4):285–297.

[18] Q. M. Tan, A. Petrenko, and G. v. Bochmann (1995) Modeling Basic LOTOS by FSMs for Conformance Testing. Technical Report 958, Dept. of I.R.O., University of Montreal.

[19] J. Tretmans. (1990) Test case derivation from LOTOS specifications. In *the IFIP 2th International Conf. on Formal Description Techniques for Distributed Sysytems and Communication Protocols* (ed. S. T. Vuong), 345–359.

Biographies

Qiang-Ming Tan received the B.S. degree and the M.S degree in computer science from Chongqing University, Chongqing, China, in 1982 and 1984, respectively. Since 1993, he has been with the Université de Montréal, PQ, Canada for the Ph.D. degree in conformance testing on communication protocols. From 1984 to 1992, he was a lecturer in the Department of Computer Science of Chongqing University.

Alexandre Petrenko received the Dipl. degree in electrical and computer engineering from Riga Polytechnic Institute in 1970 and the Ph.D. in computer science from the Institute of Electronics and Computer Science, Riga, USSR, in 1974. Since 1992, he has been with the Université de Montréal, PQ, Canada. From 1982 to 1992, he was the head of a research department of the Institute of Electronics and Computer Science, Riga, Latvia. From 1979 to 1982, he was with the Networking Task Force of the International Institute for Applied Systems Analysis (IIASA), Vienna, Austria. From 1969 to 1979, he was a researcher and the head of a research department of the Institute of Electronics and Computer Science, Riga, Latvia. His current research interests include communication software engineering, protocol engineering, conformance testing, and testability.

Gregor v. Bochmann (M'82-SM'85) received the Diploma degree in physics from the University of Munich, Munich, West Germany, in 1968 and the Ph.D. degree from McGill University, Montréal, P.Q., Canada, in 1971. He has worked in the areas of programming languages, compiler design, communication protocols, and software engineering and has published many papers in these areas. He holds the Hewlett-Packard-NSERC-CITI chair of industrial research on communication protocols in Université de Montréal, Montréal. His present work is aimed at design methods for communication protocols and distributed systems. He has been actively involved in the standardization of formal description techniques for OSI. From 1977 to 1978 he was a Visiting Professor at the Ecole Polytechnique Fédérale, Lausanne, Switzerland. From 1979 to 1980 he was a Visiting Professor in the Computer Systems Laboratory, Stanford University, Stanford, CA. From 1986 to 1987 he was a Visiting Researcher at Siemens, Munich. He is presently one of the scientific directors of the Centre de Recherche Informatique de Montréal (CRIM).