11

Checking Experiments with Labeled Transition Systems for Trace Equivalence *

1*Q. M. Tan*†, *A. Petrenko*‡ *and G. v. Bochmann*†
†*Département d'IRO, Université de Montréal*
C.P. 6128, Succ. Centre-Ville, Montréal, (Québec) H3C 3J7, Canada
E-mail:(tanq,Bochmann)@iro.umontreal.ca Fax:(514)343-5834
‡*CRIM, Centre de Recherche Informatique de Montréal*
1801 Avenue McGill College, Montréal, (Québec) H3A 2N4, Canada
E-mail:petrenko@crim.ca Phone:(514)840-1234 Fax:(514)840-1244

Abstract

We apply the state identification techniques for testing communication systems which are modeled labeled by transition systems (LTSs). The conformance requirements of specifications are represented as the trace equivalence relation and derived tests have finite behavior and provide well-defined fault coverage. We redefine in the realm of LTSs the notions of state identification that were originally defined in the realm of input/output finite state machines (FSMs). Then we present the corresponding test generation methods and discuss their fault coverage.

Keywords

Conformance testing, formal description techniques, test generation, labeled transition systems, communication protocols

*This work was supported by the HP-NSERC-CITI Industrial Research Chair on Communication Protocols, Université de Montréal

Testing of Communicating Systems, M. Kim, S. Kang & K. Hong (Eds)
Published by Chapman & Hall

1 INTRODUCTION

One of the important issues of conformance testing is to derive useful tests for labeled transition systems (LTSs), which serve as a semantic model for various specification languages, e.g., LOTOS, CCS, and CSP. Testing theories and methods for test derivation in the LTS formalism have been developed in [2, 16, 11, 3, 13, 15]. In particular, a so-called **conf** relation and *canonical tester* [2] became the basis for a large body of work in this area.

Unfortunately, the canonical tester approach cannot be taken into account when test generation for real protocols is attempted. The canonical tester has infinite behavior whenever the specification describes an infinite behavior. Moreover, we believe that the **conf** relation alone is too weak as a criterion to accept an implementation. Since this relation does not deal with invalid traces, it allows for a trivial implementation which has a single state with looping transitions labeled with all possible actions, and such an implementation conforms to any LTS specification with the same alphabet with respect to the **conf** relation [14]. Thus even though an implementation is concluded being valid based on **conf**, another relation, such as *trace-equivalence*, has to be tested as well.

Observing and comparing traces of executed interactions is usual means for conformance testing of protocols, and in many cases it is required that an implementation should have the same traces as its specification. In particular, most existing protocols are deterministic, and in the case of determinism several other finer testing semantics, such as failure or failure trace, are reduced to trace semantics. Based on the notion of such experiments and the trace equivalence relation, a number of competing test derivation methods with fault coverage have been elaborated [8, 4, 12, 18, 7, 10, 9] for protocols in the formalism of input/output finite state machines (FSMs), many of which use the state identification techniques to obtain better fault coverage. Compared to FSMs, LTSs are in some sense a more general descriptive model which use rendezvous communication without distinction between input and output; there are various semantics determining whether an implementation conforms to a specification; most existing test derivation methods use the exhaustive testing approach in order to prove the correctness of the implementation in respect to a given conformance relation. Apparently, such an approach is often impractical since it may involve a test suite of infinite length. The approximation approach [11, 16], such as n-testers, which is proposed to solve this problem, provides no fault coverage measure for conformity of the implementation with its specification.

Several attempts have been made to apply the ideas underlying the FSM-based methods to the LTS model [6, 3, 13, 14] for several conformance relations. In particular, this research is directed towards redefining the notions of state identification in the LTS realm for a given relation. However, these attempts are limited to individual or informal applications of the notions of

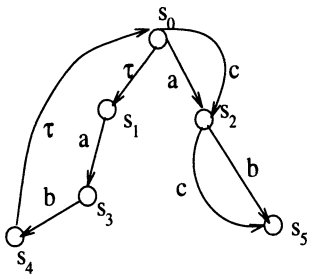

Figure 1 An LTS graph.

state identification underlying the FSM-based methods. In fact, the FSM-based notions can also be applied directly to the LTS model if an appropriate distinguishabilty of states is defined in the LTS model. Therefore, a systematic approach based on the notions of state identification can also be developed in the LTS model such that we could devise alternative and competing techniques that guarantee fault coverage, for constructing useful tests for protocols based on the LTS semantics.

In this paper, we redefine in the LTS model the notions of state identification which were originally used in the FSM realm for trace equivalence. Based on the adapted notions, the corresponding test derivation methods are presented, and it is shown that for an FSM-based method with a notion of state identification we can have a corresponding LTS-based method with a similar notion of state identification, and if the FSM-based method guarantees complete fault coverage then the LTS-analogue also guarantees complete fault coverage.

2 LABELED TRANSITION SYSTEMS

Definition 1 (*Labeled transition system (LTS)*): A labeled transition system is a 4-tuple $< S, \Sigma, \Delta, s_0 >$, where

- S is a finite 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 transitions set. $(p, \mu, q) \in \Delta$ is denoted by $p - \mu \rightarrow q$.

An LTS is said to be *nondeterministic* if it has some transition labeled with τ or there exist $p - a \rightarrow p_1, p - a \rightarrow p_2 \in \Delta$ but $p_1 \neq p_2$; otherwise it is *deterministic* LTS.

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 1.

Given an LTS $\mathsf{S} =< S, \Sigma, \Delta, s_0 >$, the conventional notations are shown in Table 1, as introduced in [2]. In this paper we use $\mathsf{M}, \mathsf{P}, \mathsf{S}, \ldots$ to represent

Table 1 Basic notations for labeled transition systems.

notation	meaning
Σ^*	set of sequences over Σ; σ or $a_1 \ldots a_n$ for such a sequence
$p-\mu_1 \ldots \mu_n \rightarrow q$	$\exists\, p_k$, $1 \leq k < n$, 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)
$p=a \Rightarrow q$	$\exists\, p_1, p_2$ such that $p=\varepsilon \Rightarrow p_1-a \rightarrow p_2=\varepsilon \Rightarrow q$
$p=a_1 \ldots a_n \Rightarrow q$	$\exists\, p_k$, $1 \leq k < n$, such that $p=a_1 \Rightarrow p_1 \ldots p_{n-1}=a_n \Rightarrow q$
$p=\sigma \Rightarrow$	$\exists\, q$ 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
$Tr(p)$	$Tr(p) = \{\sigma \in \Sigma^* \mid p=\sigma \Rightarrow\}$; $Tr(\mathrm{S}) = Tr(s_0)$

LTSs; $M, P, Q, \ldots$, for sets of states; $a, b, c, \ldots$, for actions; and $i, p, q, s \ldots$, for states. The sequences in $Tr(p)$ are called the *traces* of S for p.

Given $V \subseteq \Sigma^*$, we denote $Pref(V) = \{\sigma_1 \in \Sigma^* \mid \exists \sigma_2 \in \Sigma^* \ (\sigma_1.\sigma_2 \in V)\}$. Given $V_1, V_2 \subseteq \Sigma^*$, we denote $V_1@V_2 = \{\sigma_1.\sigma_2 \mid \sigma_1 \in V_1 \wedge \sigma \in V_2\}$. We also write $V^n = V@V^{n-1}$ for $n > 0$ and $V^0 = \{\varepsilon\}$.

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 [6]), which contains all the states reachable by the LTS after this action sequence, is used.

Definition 2 (*Multi-state set*): The multi-state set of LTS S is the set $\Pi_S = \{S_i \subseteq S \mid \exists \sigma \in \Sigma^* \ (s_0\text{-}\mathbf{after}\text{-}\sigma = S_i)\}$.

Note that $S_0 = s_0$-**after**-ε is in Π_S and 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 automaton with trace equivalence [6]. For Figure 1, $\{\{s_0, s_1\}, \{s_2, s_3\}, \{s_2\}, \{s_0, s_1, s_4, s_5\}, \{s_5\}\}$ is the multi-state set. Obviously, each LTS has one and only one multi-state set.

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 as an observable FSM [9], in which each state is a subset of states of the non-observable FSM.

3 CONFORMANCE TESTING

3.1 Conformance Relation

The starting point for conformance testing is a specification in some notation, an implementation given in the form of a black box, and a conformance

criterion that the implementation should satisfy. In this paper, the notation of the specification is the LTS formalism; the implementation is assumed to be described in the same model as its specification; a conformance relation, called *trace equivalence*, is used as the conformance criterion. We say that an implementation M conforms to a specification S if M is trace-equivalent to S.

Definition 3 (*Trace equivalence*): The trace equivalence relation between two states p and q, written $p \approx q$, holds iff $Tr(p) = Tr(q)$.
Given two LTSs S and M with initial states s_0 and m_0 respectively, we say that M is trace-equivalent to S, written $\mathsf{M} \approx \mathsf{S}$, iff $m_0 \approx s_0$.

We say that two states are *distinguishable* in trace semantics if they are not trace-equivalent. For any two states that are not trace-equivalent we can surely find a sequence of observable actions, which is a trace one of the two states, not both, to distinguish them. We also say that an LTS is *reduced* in trace semantics if all of its states are distinguishable in trace semantics.

3.2 Testing Framework

Conformance testing is a finite set of experiments, in which a set of test cases, usually derived from a specification according to a given conformance relation, is 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.

The behavior of the tester during testing is defined by the applied test case. Thus a test case is a specification of behavior, which, like other specifications, can be represented as an LTS. 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 [14, 17].

Definition 4 (*Test cases and test suite*): Given an LTS specification $\mathsf{S} = < S, \Sigma, \Delta, s_0 >$, a test case T for S 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 $t_0 = \sigma \Rightarrow p$;
- $\ell : T \rightarrow \{\mathbf{pass}, \mathbf{fail}, \mathbf{inconclusive}\}$ is a state labeling function.

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

From this definition, the behavior of test case T is finite, since it has no cycles. Moreover, a trace of T uniquely determines a single state in T, so we define $\ell(\sigma) = \ell(t)$ for $\{t\} = t_0\text{-}\mathbf{after}\text{-}\sigma$.

The interactions between a test case T and the IUT M can be formalized by the composition operator "||" of LOTOS, that is, T || M. 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 $a \in \Sigma$, $t_0 \parallel m_0 = \sigma \Rightarrow p$ and $p \neq a \Rightarrow$, we say that this experiment completes a *test run.* In order to start a new test run, a global reset is always assumed in our testing framework.

In order to test nondeterministic implementations, one usually makes the 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 [6, 9]. Therefore any experiment, in which M is tested by T, should include several test runs and lead to a complete set of observations $Obs_{(\mathrm{T,M})} = \{\sigma \in Tr(t_0) \mid \exists p \in T \times M, \forall a \in \Sigma\ ((t_0 \parallel m_0) = \sigma \Rightarrow p \neq a \Rightarrow)\}$. Note that for deterministic systems, such as most of real-life protocols, there is no need for this assumption.

Based on $Obs_{(\mathrm{T,M})}$, the success or failure of testing needs to be concluded. The way a verdict is drawn from $Obs_{(\mathrm{T,M})}$ is the *verdict assignment* for T. A pass verdict means success, which, intuitively, should mean that no unexpected behavior is found and the test purpose has been achieved; otherwise, the verdict should be a fail verdict. If we define $Pur(\mathsf{T}) = \{\sigma \in Tr(t_0) \mid \ell(\sigma) = \textbf{pass}\}$ for *the test purpose* of T, then the conclusion can be drawn as follows.

Definition 5 (*Verdict assignment*): Given an IUT M, a test case T, let $Obs_{fail} = \{\sigma \in Obs_{(\mathrm{T,M})} \mid \ell(\sigma) = \textbf{fail}\}$ and $Obs_{pass} = \{\sigma \in Obs_{(\mathrm{T,M})} \mid \ell(\sigma) = \textbf{pass}\}$,

$$\begin{cases} \mathsf{M} \text{ passes } \mathsf{T} & \text{iff } Obs_{fail} = \emptyset \wedge Obs_{pass} = Pur(\mathsf{T}) \\ \mathsf{M} \text{ fails } \mathsf{T} & \text{otherwise.} \end{cases}$$

Given a test suite *TS*, we also denote that M passes *TS* iff for all $\mathsf{T} \in TS$ M passes T, and M fails *TS* otherwise.

3.3 State Labelings of Test Cases

Given a specification S, a test case T should be "sound", that is, for any implementation M, if M and S are trace-equivalent, then M passes T.

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. The purpose of a test case is to verify that an IUT has implemented the valid ones and not any of the invalid ones. Accordingly, we conclude that all test cases for trace equivalence must be of the following form [15]:

Definition 6 (*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. $\approx$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0\textbf{-after-}\sigma$, the state labeling of T satisfies

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

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

From this definition, we have the following proposition [15]: Given a test case T for S w.r.t. $\approx$, for any LTS M, if M $\approx$ S, then M passes T.

Since in trace semantics test cases for S are represented as valid or invalid traces of S, given a sequence $\sigma \in \Sigma^*$, let $\sigma = a_1.a_2 \ldots .a_n$, a test case T for S w.r.t. $\approx$ can be obtained by constructing an LTS T = $t_0 - a_1 \rightarrow t_1 \ldots t_{n-1} - a_n \rightarrow t_n$ and then labeling T according to Definition 6. A sequence that is used to form a test case is also called a *test sequence.*

3.4 Fault Model and Fault Coverage

The goal of conformance testing is to gain confidence in the correct functioning of the implementation under test. Increased confidence is normally obtained through time and effort spent in testing the implementation, which, however, is 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 [1] are introduced, which usually take the mutation approach [1], that is, a fault model is defined as a set of all faulty LTS implementations considered. Here we consider a particular fault model $\mathcal{F}(m)$ which consists of all LTS implementations over the alphabet of the specification S and with at most m multi-states, where m is a known integer. Based on $\mathcal{F}(m)$, a test suite with complete fault coverage for a given LTS specification with respect to the trace equivalence relation can be defined as follows.

Definition 7 (*Complete test suite*): Given an LTS specification S and the fault model $\mathcal{F}(m)$, a test suite *TS* for S w.r.t. $\approx$ is said to be complete, if for any M in $\mathcal{F}(m)$, M $\approx$ S iff M passes *TS*.

We also say that a test suite is *m-complete* for S if it is complete for S in respect to the fault model $\mathcal{F}(m)$. A complete test suite guarantees that for any implementation M in the context of $\mathcal{F}(m)$, if M passes all test cases, it must be a conforming implementation, and any faulty implementation in $\mathcal{F}(m)$ must be detected by failing at least one test case in the test suite.

4 STATE IDENTIFICATION IN SPECIFICATIONS

Similar to the case of FSMs, in order to identify states in a given LTS specification, at first the specification is required to have certain testability properties, two of which are the so-called reducibility and observability.

4.1 Trace Observable System

Definition 8 (*Trace observable system (TOS)*): Given an LTS S, a deterministic LTS $\overline{\mathsf{S}}$ is said to be the trace observable system corresponding to S, if $\overline{\mathsf{S}} \approx$ S and $\overline{\mathsf{S}}$ is reduced in trace semantics.

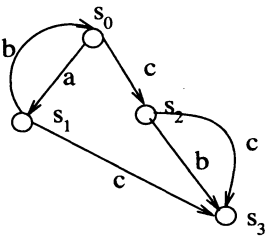

Figure 2 A corresponding trace observable system of Figure 1.

From the above definition, the TOS $\overline{S}$ of S is deterministic, reduced and trace-equivalent to S; moreover, the TOS $\overline{S}$ is unique for all LTSs trace-equivalent to S. There are the algorithms and tools that transform a given LTS into its TOS form [3]. For the LTS in Figure 1, the TOS is given in Figure 2.

In the context of trace semantics, for any LTS, the corresponding TOS models all its observable behavior. Therefore, for test generation, any LTS considered can be assumed to be in the TOS form.

4.2 State identification Facilities

There are the following facilities of state identification which can be adapted from the FSM model to the LTS model. Here we assume that the given LTS specification S is in the TOS form that has n states $s_0, s_1, \ldots s_{n-1}$, where s_0 is the initial state.

Distinguishing Sequence

Given an LTS S, we say that an observable sequence distinguishes two states if the sequence is a trace for one of the two states, but not for both. A *distinguishing sequence* for S is an observable sequence that distinguishes any two different states. Formally, $\sigma \in \Sigma^*$ is a distinguishing sequence of S if for all $s_i, s_j \in S, i \neq j$, there exists $\sigma' \in Pref(\sigma)$ such that $\sigma' \in Tr(s_i) \oplus Tr(s_j)$, where $A \oplus B = (A \backslash B) \cup (B \backslash A)$.

There are LTSs in the TOS form without any distinguishing sequence. As an example, the LTS in Figure 2 has no distinguishing sequence.

Unique Sequences

A *unique sequence* for a state is an observable sequence that distinguishes the given state from all others. Formally, $\sigma_i \in \Sigma^*$ is a unique sequence for $s_i \in S$, if, for all $s_j \in S, i \neq j$, there exists $\sigma_i' \in Pref(\sigma_i)$ such that $\sigma_i' \in Tr(s_i) \oplus Tr(s_j)$. Let S have n states, a tuple of unique sequences $< \sigma_0, \sigma_1, \ldots, \sigma_{n-1} >$ is said be set of unique sequences for S. If there exists $\sigma \in \Sigma^*$ such that $\sigma_i \in Pref(\sigma)$, for $0 \leq i \leq n-1$, then σ is a distinguishing sequence. The notion of unique sequences, also called unique event sequences in [3], corresponds to that of FSM-based UIO sequences [12].

For the LTS in Figure 2, we may choose $< a, b.a, b.a, c >$ as its unique sequences. Note that unique sequences do not always exist. For example, if the transition $s_2 - c \rightarrow s_3$ in Figure 2 is deleted, then no unique sequence exists for s_3 in the resulting LTS.

Characterization Set

If a set of observable sequences, instead of a unique distinguishing sequence, is used to distinguish all the states of S, we have a so-called *characterization set* for S. A characterization set for S is a set $W \subseteq \Sigma^*$ such that for all $s_i, s_j \in S, i \neq j$, there exists $\sigma_i \in Pref(W)$ such that $\sigma_i \in Tr(s_i) \oplus Tr(s_j)$.

There exists a characterization set W for any S in the TOS form. For the LTS in Figure 2, we may choose $W = \{a, b.a\}$.

Partial Characterization Set

A tuple of sets of observable sequences $< W_0, W_1, \ldots, W_{n-1} >$ is said to be *partial characterization sets*, if, for all $s_i \in S, 0 \leq i \leq n-1$, and for all $s_j \in S, i \neq j$, there exists $\sigma_i \in Pref(W_i)$ such that $\sigma_i \in Tr(s_i) \oplus Tr(s_j)$. The notion of partial characterization sets correspond to the notion of partial UIO sequences in [5].

Obviously, since the given S is in the TOS form, in other words, none of its two states are trace-equivalent, there exist partial characterization sets for S. We also note that the union of all partial characterization sets for S is a characterization set for S. For the LTS in Figure 2, we may choose $< \{a\}, \{b.a\}, \{b.a\}, \{a, b\} >$ as its partial characterization sets.

Harmonized State Identifiers

A tuple of sets of observable sequences $< H_0, H_1, \ldots, H_{n-1} >$ is said to be a set of *harmonized state identifiers* for S, if it is a tuple of partial characterization sets for S and for $i, j = 0, 1, \ldots, n-1, i \neq j$, there exists $\sigma \in Pref(H_i) \cap Pref(H_j)$. H_i also is said to be a harmonized identifier for $s_i \in S$. The harmonized identifier for s_i captures the following property: for any different state s_j, there exists a sequence σ_i in $Pref(H_i)$ that distinguishes s_i from s_j and σ_i is also in $Pref(H_j)$.

Harmonized state identifiers always exist, just as partial characterization sets do. As an example, for the LTS in Figure 2, we can choose the harmonized state identifiers $H_0 = \{a, b\}, H_1 = \{b.a\}, H_2 = \{b.a\}, H_3 = \{a, b\}$.

5 STATE IDENTIFICATION IN IMPLEMENTATIONS

Similar to FSM-based testing, we assume that the given implementation is an LTS M whose set of all possible actions is limited to the set of actions Σ of the specification S (the correct interface assumption [1]). We also have a reliable reset, such that the state entered when this implementation is started or after the reset is applied is the initial state (the reliable reset assumption). In the case of nondeterminism, it makes no sense to identify single states of

M, so M is also assumed to be a TOS, in which each multi-state consist of a single state. For this reason, we require that S is in the TOS form, so that a state identification facility can be developed from S and also can be used to identify the states of M.

In order to identify the states of the implementation M, the number of states of M is also assumed to be bound by a known integer m. Therefore, M is also a mutant according to the fault model $\mathcal{F}(m)$.

Similar to FSM-based testing [7], there are also the two phases for LTS-based testing. In the first phase, the used state identification facility is applied to M to check if it can also properly identify the states in M. Once M passes the first phase, we can in the second phase test whether each transition and its tail state are correctly implemented. We present the structure of tests for the two phases using harmonized state identifiers as an example. In order to perform the first testing phase, proper transfer sequences are needed to bring M from the initial state to those particular states in M to which H_i should be applied. Moreover, it should be guaranteed that all the sequences in H_i are applied to the same particular state in M. Since a reliable reset is assumed, we can guarantee this in a way similar to FSM based testing: after a sequence in H_i is applied, the implementation M is reset to the initial state, and brought into the same particular state by the same transfer sequence,and then another sequence in H_i is applied. This process is repeated until all the sequences are applied.

Accordingly, let Q be a *state cover for* S, i.e. for each state s_i of S, there exists exactly one input sequence σ in Q such that $s_0 - \sigma \rightarrow s_i$, similar to FSM based testing, we can use $< N_0, N_1, \ldots N_{n-1} >$ to cover all states of M (a *state cover for* M), where

$$N_i \doteq \{\sigma \in Q @ (\Sigma^0 \cup \Sigma_1 \cup \ldots \cup \Sigma^{m-n}) \mid s_0 = \sigma \Rightarrow s_i\}$$

and construct a set of test sequences to be executed by M from the initial state in the first testing phase as follows:

$$TS_1 = \bigcup_{i=0}^{n} N_i @ H_i$$

Inituitively, sequences of the sets N_i are used to reach n required states, as well as all possible $(m-n)$ additional states in M. Harmonized state identifiers H_i are applied to identify all states in M. In order to execute a given sequence $\sigma = a_1.a_2 \ldots a_k$ from the initial state m_0, we can convert σ into an LTS $t_0 - a_1 \rightarrow t_2 \ldots - a_k \rightarrow t_k$ and then compose this LTS with M in parallel composition $t_0 \parallel m_0$. Due to nondeterminism, it is possible that this run ends before the final action of this sequence is executed. Several runs are needed to exercise all the possible paths of M that can be traversed by this sequence (the complete testing assumption).

Using TS_1, we can make test cases for LTS S for the first testing phase by transforming the sequences in TS_1 into the corresponding LTSs as above and then labeling the LTSs according to Definition 6. In the following, this transforming and labeling process is always implied if we say that a test suite is obtained from a given set of test sequences.

After TS_1 is successfully executed, all the states of M which execute all traces of H_k are grouped in the same group $f(s_k)$, where $0 \leq k \leq n-1$.

In the second phase of testing, for testing a given defined transition $s_i - a \rightarrow s_j$ in S, it is necessary to first bring M into each state $m_k \in f(s_i)$, then apply a at this state to see if a can be executed; moreover, let M be in m_l after a is executed, it is necessary to check that $m_l \in f(s_j)$ which should be verified by H_j. (Note that due to nondeterminism, m_k may really be a multi-state, the action that is expected to check may not be executed in a time, so the above process should be tried several times.) On the other hand, we should further check if any undefined transition out of s_i has been implemented in M, i.e. for each $b \in \Sigma$, if $s_i \not- b \rightarrow$, then check that $m_k = b \Rightarrow$ does not exist. Because if $m_k - b \rightarrow$ exists, M is surely an invalid implementation, so it is not necessary to verify the tail state after b is executed.

Obviously, N_i may be used to bring M to any state $m_k \in f(s_i)$. Using this state cover, we can obtain a *valid transition cover* $< E_0, E_1, \ldots E_{n-1} >$, where

$$E_i = \{\sigma \in \bigcup_{k=0}^{n-1} (N_k @ \Sigma) \mid s_0 = \sigma \Rightarrow s_i\}$$

which covers all transitions that should be present in any conforming implementation, and an *invalid transition cover* $\overline{E}$,

$$\overline{E} = \{\sigma.a \in \bigcup_{k=0}^{n-1} (N_k @ \Sigma) \mid \exists s_i \in S \ (s_0 = \sigma \Rightarrow s_i \neq a \Rightarrow)\}$$

which covers all transitions that should be absent in any conforming implementation.

Next, H_i is used to verify the tail states of reached after each sequence in E_i. Excluding the transitions that have already been tested in the first testing phase, we can construct the set of test sequences for the second testing phase as follows:

$$TS_2 = \overline{E} \cup (\bigcup_{i=0}^{n-1} (E_i \backslash N_i) @ H_i)$$

We conclude that the set of test sequences is expressed as follows, by combining the two sets of test sequences for the first and second testing phases:

$$TS = TS_1 \cup TS_2 = \overline{E} \cup (\bigcup_{i=0}^{n-1} E_i @ H_i)$$

We have seen that the above process is an analogue of the checking experiments for the FSM model, except that invalid transitions need to be tested although their tail states need not to be verified. Similarly, it is expected that a test suite which is derived from S based on the above process is complete with respect to trace equivalence for $\mathcal{F}(m)$. In the next section, we present the test generation methods, based on the facilities presented in Section 4.2.

6 TEST GENERATION

6.1 Methods

Based on the above state identification techniques, we have a number of methods for constructing a set *TS* of test sequences for a given LTS specification S and with certain fault coverage for $\mathcal{F}(m)$. Let S be given in the TOS form with n states. We can obtain the state cover for implementation $< N_0, N_1, \ldots N_{n-1} >$, the valid transition cover for implementation $< E_0, E_1, \ldots E_{n-1} >$ and the invalid transition cover for implementation $\overline{E}$ as presented in the above section. Let $E = \bigcup_{i=0}^{n-1} E_i$ and $N = \bigcup_{i=0}^{n-1} N_i$.

The DS-method
Similar to the FSM-based DS-method [8], we use a distinguishing sequence σ for S to form a test suite for S, as follows.

$$TS = E@\{\sigma\} \cup \overline{E} \quad (1)$$

Theorem 1 *Given an LTS specification* S *in the TOS form and a distinguishing sequence* σ *for* S*, the test suite obtained from TS as given in (1) is* m*-complete for* S *w.r.t.* $\approx$.

Unlike the traditional FSM-based DS-method, the LTS-based DS-method does not construct a single test sequence since a reliable reset exists. It seems that, in case of deadlock, the reset is the only way to continue test execution.

The US-method
Let $< \sigma_0, \sigma_1, \ldots, \sigma_{n-1} >$ be a set of unique sequences for S, then a test suite for S, which is an analogue of that derived by the FSM-based UIO-method [12], can be formed as

$$TS = (\bigcup_{i=0}^{n-1} E_i@\{\sigma_i\}) \cup \overline{E} \quad (2)$$

As a specific case, unique sequences might be prefixes of the same (distinguishing) sequence. For the same reason explained in relation with the DS-method, the US-method does not combine unique sequences using the rural Chinese postman tour algorithm to obtain an optimal single test sequence.

Since unique sequences do not always exist, partial characterization sets can be used instead of unique sequences. This corresponds to the improvement on the UIO-method in [5]. Although partial characterization sets exist for any LTS in the TOS form, like the US-method, the improvement can not guarantee that a derived test suite is m-complete.

A similar method borrowing the notion of UIO sequences in the FSM model is proposed in [3], in which unique sequences are called unique event sequences. This method does not check invalid transitions, so it may not cover a fault where an undefined transition has been implemented.

The Uv-method

In order to obtain an m-complete test suite, the US-method can be improved such that

$$TS = N@(\bigcup_{i=0}^{n-1} \sigma_i) \cup (\bigcup_{i=0}^{n-1} (E_i \backslash N_i)@\{\sigma_i\}) \cup \overline{E} \quad (3)$$

Theorem 2 *Given an LTS specification* S *in the TOS form and a set of unique sequences* $< \sigma_0, \sigma_1, \ldots, \sigma_{n-1} >$ *for* S*, the test suite obtained from TS as given in (3) is m-complete for* S *w.r.t.* $\approx$.

The Uv-method usually drives a test suite of length larger than the US-method. However, unlike the US-method, it guarantees full fault coverage. The Uv-method corresponds to the FSM-based UIOv-method [18].

The W-method

Given a characterization set W for S, we form a test suite for S by the following formula. This is an LTS-analogue of the FSM-based W-method [4].

$$TS = E@W \cup \overline{E} \quad (4)$$

Theorem 3 *Given an LTS specification* S *in the TOS form and a characterization set* W *for* S*, the test suite obtained from TS as given in (4) is m-complete for* S *w.r.t.* $\approx$.

We note that in the case that $|W| = 1$, the W-method is the DS-method.

The Wp-method

Let W be a characterization set for S and $< W_0, W_1, \ldots, W_{n-1} >$ be partial characterization sets for S, similar to the FSM-based Wp-method [7], the Wp-method uses the following test sequences to form a test suite for S

$$TS = N@W \cup (\bigcup_{i=0}^{n-1} (E_i \backslash N_i)@W_i) \cup \overline{E} \quad (5)$$

Theorem 4 *Given an LTS specification* S *in the TOS form, a characterization set* W *and partial characterization sets* $< W_0, W_1, \ldots, W_{n-1} >$ *for* S*, the test suite obtained from TS as given in (5) is m-complete for* S *w.r.t.* $\approx$.

Obviously, the Wp-method derives usually a test suite of length smaller than the W-method because $W_i \subseteq W$. We note that the Uv-method is a specific case of the Wp-method, in which the union $\bigcup_{i=0}^{n-1} \sigma_i$ is a characterization set and $< \{\sigma_0\}, \{\sigma_1\}, \ldots, \{\sigma_{n-1}\} >$ are partial characterization sets.

The HSI-method

Let $< H_0, H_1, \ldots, H_{n-1} >$ be harmonized state identifiers for S, similar to the FSM-based HSI-method [10, 9], The HSI-method follows completely the approach presented in the above section to form a test suite for S.

$$TS = (\bigcup_{i=0}^{n-1} E_i@H_i) \cup \overline{E} \tag{6}$$

Theorem 5 *Given an LTS specification* S *in the TOS form and harmonized state identifiers* $< H_0, H_1, \ldots, H_{n-1} >$ *for* S, *the test suite obtained from TS as given in (6) is m-complete for* S *w.r.t.* $\approx$.

Since the union $\bigcup_{i=0}^{n-1} H_i$ is a characterization set, the HSI-method usually derives a test suite of length smaller than the W-method.

6.2 Examples

Assuming that the specification is given in Figure 2, with the HSI-method, we can derive a 4-complete test suite, which checks trace equivalence for this specification, as well as to the specification in Figure 1, as follows.

	s_0	s_1	s_2	s_3
State Identifiers H_i	a, b	$b.a$	$b.a$	a, b
State Cover Q	ε	a	c	$a.c$
Valid Transition Cover E_i	ε, $a.b$	a	c	$a.c$, $c.b$, $c.c$
Invalid Transition Cover $\overline{E} = \{b,\ a.a,\ c.a,\ a.c.a,\ a.c.b,\ a.c.c\}$				

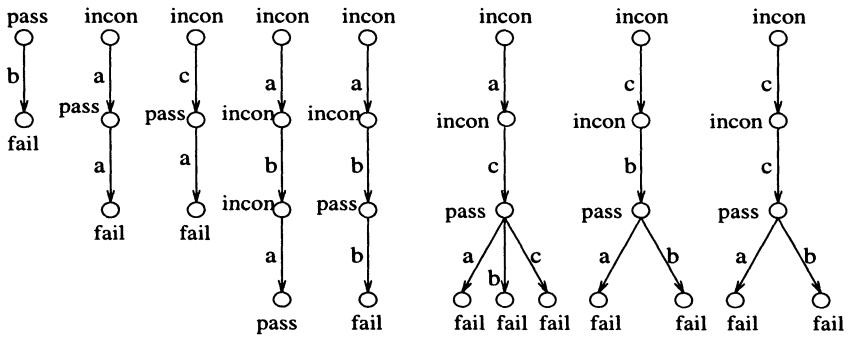

Figure 3 A complete test suite for the LTS specification in Figure 2.1.

$TS = \{b,\ a.a,\ c.a,\ a.b.b,\ a.b.a,\ a.c.a,\ a.c.b,\ a.c.c,\ c.b.a,\ c.b.b,\ c.c.a,\ c.c.b\}$. The corresponding test cases are shown in Figure 3.

7 CONCLUSION

In this paper, we have redefined, in the LTS model, the notions of state identification, which were originally defined in the formalism of input/output finite state machines (FSMs). Then we presented corresponding test derivation methods for specifications given in the LTS formalism that derive finite tests with fault coverage for trace equivalence. Note that the existing FSM-based methods are not directly applicable to LTSs, because LTSs assume rendezvous interactions making no distinction between inputs and outputs.

The notions of state identification in the LTS realm are distinguishing sequence, unique sequences, characterization set, partial characterization sets and harmonized state identifiers. The test generation methods based on these techniques are the DS-method, the US-method, the Uv-method, the W-method, the Wp-method and HSI-method. Among these methods, the DS-method, Uv-method, the W-method, the Wp-method and the HSI-method guarantee complete fault coverage.

8 REFERENCES

[1] G. v. Bochmann and A. Petrenko. Protocol testing: Review of methods and relevance for software testing. (1994) *ACM 1994 Intern. Symp. on Soft. Testing and Analysis*, 109–124.

[2] E. Brinksma. A theory for the derivation of tests. (1988) *PSTV VIII*, 63–74.

[3] A. R. Cavalli and S. U. Kim. Automated protocol conformance test generation based on formal methods for LOTOS specifications. (1992) *5th IWPTC*, 212–220.

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

[5] W. Chun and P. D. Amer. Improvements on UIO sequence generation and partial UIO sequences. (1992) *PSTV XII*, 245–259.

[6] S. Fujiwara and G. v. Bochmann. Testing nonterministic finite state machine with fault coverage. (1991) *4th IWPTS*, 267–280.

[7] S. Fujiwara et al. Test selection based on finite state models. (1991) *IEEE Trans. on Soft. Engi.*, SE-17(6):591–603.

[8] F. C. Hennie. Fault-detecting experiments for sequential circuits. (1964) *5th Symp. on Switching Circuit Theory and Logical Design.*

[9] G. Luo, et al. Selecting test sequences for partially-specified nondeterministic finite machines. (1994) *7th IWPTS*, 91–106.

[10] A. Petrenko. Checking experiments with protocol machines. (1991) *4th IWPTS*, 83–94.

[11] D. H. Pitt and D. Freestone. The derivation of conformance tests from LOTOS specifications. (1990) *IEEE Trans. on Soft. Engi.*, SE-16(12):1337–1343.

[12] K. Sabnani and A. T. Dahbura. A protocol test generation procedure. (1988) *Comput. Net. and ISDN Syst.*, 15(4):285–297.

[13] Q. M. Tan, A. Petrenko, and G. v. Bochmann. Modeling basic LOTOS by FSMs for conformance testing. (1995) *PSTV XV*, 137–152.

[14] Q. M. Tan, et al. Testing trace equivalence for labeled transition systems. (1995) TR 976, Dept.of I.R.O., University of Montreal, canada.

[15] Q. M. Tan, A. Petrenko, and G. v. Bochmann. A framework for conformance testing of systems communicating through rendezvous. (1996) *IEEE 26th Intern. Symp. on Fault-Tolerant Computing*, 230–238.

[16] J. Tretmans. *A Formal Approach to Conformance Testing.* (1992) Ph.D. thesis, Hengelo, Netherlands.

[17] J. Tretmans. Testing labelled transition systems with inputs and outputs. (1995) *8th IWPTS*, 461–476.

[18] S. T. Vuong and et al. The UIOv-method for protocol test sequence generation. (1990) *2th IWPTS*, 203–225.

9 BIOGRAPHY

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, 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. Now he is also working with Claremont Technology Inc. Canada.

Alexandre Petrenko received the Dipl. degree in electrical and computer engineering from Riga Polytechnic Institute and the Ph.D. in computer science from the Institute of Electronics and Computer Science, Riga, USSR. In 1996, he has joined CRIM, Centre de Recherche informatique de Montréal, Canada. He is also an adjunct professor of the Université de Montréal, where he was a visiting professor/researcher from 1992 to 1996. From 1982 to 1992, he was the head of a research department of the Institute of Electronics and Computer Science in Riga. From 1979 to 1982, he was with the Networking Task Force of the International Institute for Applied Systems Analysis (IIASA), Vienna, Austria. His current research interests include high-speed networks, communication software engineering, formal methods, conformance testing, and testability.

Gregor v. Bochmann (M'82-SM'85) received the Dipl. 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).