Testing Trace Equivalence for Labeled Transition Systems[1]

Q. M. Tan, A. Petrenko, G. Luo[2] and G. v. Bochmann
Department d'IRO, University of Montreal
C.P. 6128, Succ. Centre-Ville, Montreal, P.Q. H3C 3J7, Canada
e-mail:tanq@iro.umontreal.ca Fax:(514)343-5834

ABSTRACT: In this paper, we consider the problem on conformance testing of communication protocols which are modeled by labeled transition systems. The conformance requirements of specifications are represented as the trace equivalence relation and derived test suites have finite behavior and provide well-defined fault coverage. For this problem, we first give a testing framework and then, based on the state identification technique, we present a test generation method. The advantages of our method over other methods are that it not only ensures trace equivalence in a class of implementations whose state numbers are bounded by a known integer, but also requires no transformation of LTSs to input/output finite state machines.

KEYWORDS: Conformance relations, labeled transition systems, protocol conformance testing and software testing.

1 Introduction

One of important issues of conformance testing is to derive useful tests for communication protocols specified in labeled transition systems (LTSs), which serves as a semantic model for various specification languages, e.g., LOTOS, CCS, and CSP and assumes a rendezvous communication, i.e., communication between two processes occurs if both processes offer to interact on a particular action, and if the interaction takes place it occurs synchronously in both participating processes. Testing theories and methods for test derivation in the LTS formalism have been developed in [2, 13, 3, 1, 5]. 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 an infinite behavior; no fault coverage is measured for individual tests derived in [13]. Moreover, we believe that the **conf** relation alone is too weak as a criterion to accept an implementation, because only are the deadlocks that are

[1]This work was supported by the HP-NSERC-CITI Industrial Research Chair on Communication Protocols, Universite de Montreal

[2]G. Luo is with Bell Northern Research Ltd. P.O. Box 3511, Station C, Ottawa, Canada K1Y 4H7

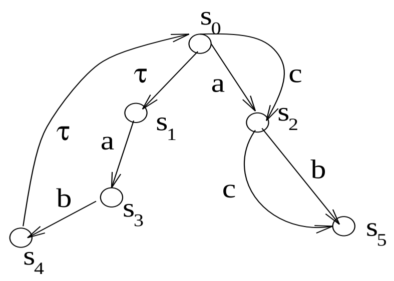

Figure 1: An LTS graph

implemented after the valid traces in the specification to be checked. Since this relation does not deal with invalid traces, this will lead to a trivial implementation M that has a single state with looping transitions labeled with all possible actions and conforms to any LTS specification S with the same alphabet with respect to **conf**. Thus even though an implementation is concluded being valid based on **conf**, another relation, such as *trace-equivalence*, has to be tested as well.

Conformance testing for protocols in the LTS formalism should be developed in such a way that the given conformance relation is determined by the real conformance requirements and a test suite has finite behavior and ensures fault coverage in a certain class of implementations. Several attempts have been made to apply the ideas underlying the FSM-based methods to the LTS model [6, 3, 1] for several relations. In particular, this research is directed towards redefining the state identification and eventually the checking experiments in the LTS realm for a given relation. [3] tries the UIO-based state identifiers [11], which, as it is well known, do not always exist; [6] considers the characterization sets [4]; 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. In [10, 12], another approach is taken, where an LTS is represented as an FSM model, the FSM method is applied, and then the derived tests are translated back into the LTS formalism. This approach has the advantage of allowing reuse of existing FSM-based methods and testing tools for the LTS specifications, but it requires several auxiliary transformations.

In this paper, we adapt the notion of the HSI–state identifiers [9, 8] in I/O FSMs to fit LTSs. Based on this, a test derivation method with complete fault coverage is presented for LTS specifications and the trace equivalence relation, provided that the number of states of any implementation is bounded by a known integer. Although in our method the notion of the HSI-state identifiers is used for state identification, other state identification techniques, such as that of the W– or Wp–method [4, 7], can be applied to LTSs in a similar way.

2 Basic Definitions and Notations

In this section, we review some basic definitions and notations which are related to testing LTSs.

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 $0 \leq k \leq n$ such that $p = p_0\!-\!\mu_1\!\rightarrow p_1 \ldots\!-\!\mu_n\!\rightarrow p_n = q$
$p\!=\!\varepsilon\!\Rightarrow q$	$p\!-\!\tau^n\!\rightarrow q$ $(1 \leq n)$ or $p = q$ (note: τ^n means n times τ)
$p\!=\!a\!\Rightarrow q$	there exist p_1, p_2 such that $p_1\!=\!\varepsilon\!\Rightarrow p_1\!-\!a\!\rightarrow p_2\!=\!\varepsilon\!\Rightarrow q$
$p\!=\!\sigma\!\Rightarrow q$	there exists p_k for $0 \leq k \leq n$ 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 $p\!=\!\sigma\!\Rightarrow q$
$p\!\neq\!\sigma\!\Rightarrow$	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 1: Notation for 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 $p_1 \neq p_2$. A *deterministic* LTS has no internal actions and the outgoing transitions of any state are uniquely labeled.

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.

The notations shown in Table 1 are relevant to a given LTS, as introduced in [2]. In this paper we use $\mathsf{M}, \mathsf{P}, \mathsf{S}, \ldots$ to represent LTSs; $M, P, Q, \ldots$, for sets of states; $a, b, c, \ldots$, for actions; and $i, p, q, s \ldots$, for states. Additionally, we also denote $Tr(\mathsf{S})$ by $Tr(s_0)$, S-**after**-σ by s_0-**after**-σ, and the sequences in $Tr(\mathsf{S})$ are called the *traces* of S.

3 Conformance Testing

The starting point for conformance testing is a specification in some (formal) notation, an implementation given in the form of a black box, and the conformance requirements 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; an conformance relation, called *trace equivalence*, is used to formalize the conformance requirements. We say that an implementation M conforms to a specification S if M is trace-equivalent to S.

Definition 2 (*Trace equivalence*): The trace equivalence relation between two states p and q, written $p \approx_{\mathbf{tr}} q$, holds if and only if $Tr(p) = Tr(q)$.
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 $\mathsf{M} \approx_{\mathbf{tr}} \mathsf{S}$, if only if $m_0 \approx_{\mathbf{tr}} s_0$.

Conformance testing is a finite set of experiments, in which a set of test cases, 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 used 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.

Definition 3 (*Test cases and test suite*): Given an LTS specification $\mathsf{S} = < S, \Sigma, \Delta, s_0 >$, a test case 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 $t_0 {=}\sigma{\Rightarrow} p$;
- $\ell : T \to \{\mathbf{pass}, \mathbf{fail}, \mathbf{inconclusive}\}$ 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 T and Σ are finite. Moreover, a trace of T uniquely determines a single state in T, so we define $\ell(\sigma) = \ell(t)$ for $\{t\} = \mathsf{T}$-**after**-σ.

The interactions between a test case T and the IUT M can be formalized by the synchronization operator "$\|$" of LOTOS, that is, $\mathsf{T} \| \mathsf{M}$. When $\mathsf{T} \| \mathsf{M}$ 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 $a \in \Sigma$, $\mathsf{T} \| \mathsf{M} {=}\sigma{\Rightarrow} p$ and $p {\not=} a {\Rightarrow}$, we say that this experiment completes a *test run*.

Usually, LTSs are supposed to be nondeterministic. 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, 8]. Therefore any experiment, in which M is tested by T, should include several test runs and lead to a complete set of observations $Obs_{(\mathrm{T,M})} = \{\sigma \in Tr(\mathsf{T}) \mid \exists p \in T \times M, \forall a \in \Sigma \ ((\mathsf{T} \| \mathsf{M}) {=}\sigma{\Rightarrow} p {\not=} a {\Rightarrow})\}$.

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. The verdict **pass** means success, which, intuitively, should mean that no unexpected behavior is found and the test purpose has been achieved. If we define *the test purpose* of T, written $Pur(\mathsf{T})$, to be $Pur(\mathsf{T}) = \{\sigma \in Tr(\mathsf{T}) \mid \ell(\sigma) = \mathbf{pass}\}$, then the conclusion can be drawn as follows.

Definition 4 (*Verdict assignment v*): Given an IUT M, a test case T, let $Obs_{fail} = \{\sigma \in Obs_{(\mathrm{T,M})} \mid \ell(\sigma) = \mathbf{fail}\}$ and $Obs_{pass} = \{\sigma \in Obs_{(\mathrm{T,M})} \mid \ell(\sigma) = \mathbf{pass}\}$,

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

Given a test suite TS, we say that M passes TS if and only if for all $\mathsf{T} \in TS$ $v(Obs_{(\mathrm{T,M})}) = \mathbf{pass}$.

4 Test Generation

In this section, we first present a reference model and then a test generation algorithm for a given LTS specification with respect to trace equivalence.

4.1 Trace Observable System

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, the set of the different states, rather than the single states, is used to define the transition checking and state identification [6]. The viewpoint is reflected in the FSM realm by the presentation of a nondeterministic FSM specification as an observable FSM [8], in which each state is a subset of states of the non-observable FSM. The viewpoint is also reflected by the *refusal graphs* [5], in which a node also corresponds to a subset of states. To test trace equivalence, we do not need refusal sets in the refusal graphs. If these sets of a refusal graph are dropped and trace-equivalent nodes are merged then we have a so-called trace observable system for test generation.

Definition 5 (*Trace observable system (TOS)*): Given an LTS $\mathsf{S} =< S, \Sigma, \Delta, s_0 >$, an LTS $\overline{\mathsf{S}} =< \overline{S}, \Sigma, \overline{\Delta}, \overline{s}_0 >$ is said to be the trace observable system of S, if there exists a mapping $\psi : \Pi \rightarrow \overline{S}$, where $\Pi = \{S_i \subseteq S \mid \exists \sigma \in Tr(s_0)\ (s_0\text{-}\mathbf{after}\text{-}\sigma = S_i)\}$, such that

- $\overline{s}_0 = \psi(S_0)$ where $S_0 = \{s \in S \mid s_0 {=}\varepsilon{\Rightarrow} s\}$;
- $\overline{s}_i {-}a{\rightarrow} \overline{s}_j \in \overline{\Delta}$ for $\overline{s}_i, \overline{s}_j \in \overline{S}$ and $\psi(S_i) = \overline{s}_i$ iff there exist $S_i, S_j \in \Pi$ such that $S_j = \{s \in S \mid \exists p \in S_i\ (p {=} a {\Rightarrow} s)\}$ and $\psi(S_j) = \overline{s}_j$;
- $\psi(S_i) = \psi(S_j)$ iff $\bigcup_{p \in S_i} Tr(p) = \bigcup_{q \in S_j} Tr(q)$.

From the above definition, the TOS $\overline{\mathsf{S}}$ of S is deterministic and trace-equivalent to S; and furthermore, none of states in $\overline{\mathsf{S}}$ are trace-equivalent, that is, $\overline{\mathsf{S}}$ is *minimal*. For the LTS in Figure 1, its TOS is given in Figure 2. For any LTS, there are several existing algorithms and tools [3] for the TOS through determinization and minimization.

4.2 Algorithm

Given any LTS S, the TOS $\overline{\mathsf{S}}$ models all its observable behavior in trace semantics. Therefore, we assume that LTS specifications for test generation are given in the form of the trace observable systems. To present our method, we need the following notion for state identification, which is adapted from the FSM model [8].

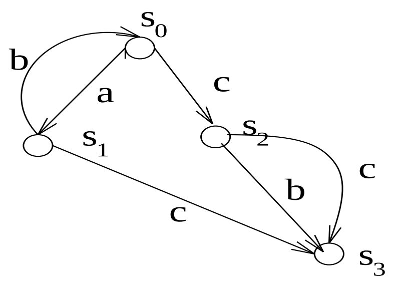

Figure 2: A corresponding trace observable system of Figure 1

Given a set of sequences $V \in \Sigma^*$, we use the notation $Pref(V)$ to represent all prefixes of sequences in V. Formally, $Pref(V) = \{\sigma_1 \mid \exists \sigma_2 \in \Sigma^* \ (\sigma_1.\sigma_2 \in V)\}$.

Definition 6 (*A tuple of harmonized state identifiers* $< W_0, W_1, \ldots, W_{n-1} >$): Given a TOS $\overline{\mathsf{S}}$ with n states, $< W_0, W_1, \ldots, W_{n-1} >$ is said to be a tuple of harmonized state identifiers of $\overline{\mathsf{S}}$, if, for $i, j = 0, 1, \ldots, n-1, i \neq j$, $W_i, W_j \subseteq \Sigma^*$ and there exists $\sigma \in Pref(W_i) \cap Pref(W_j)$ such that $\sigma \in Tr(\overline{s}_i) \oplus Tr(\overline{s}_j)$, where $Tr(\overline{s}_i) \oplus Tr(\overline{s}_j) = (Tr(\overline{s}_i) \cup Tr(\overline{s}_j)) \backslash (Tr(\overline{s}_i) \cap Tr(\overline{s}_j))$.

According to this definition, there exists a tuple of harmonized state identifiers for $\overline{\mathsf{S}}$ because none of states in $\overline{\mathsf{S}}$ are trace-equivalent. W_i is a harmonized identifier of state $\overline{s}_i$. The harmonized identifier catches the following property: *for any different state $\overline{s}_j$, there exists a sequence in $Pref(W_i) \cap Pref(W_j)$ such that it is a valid trace of* S *in only one of the two states*, that is, $\overline{s}_i$ is distinguished from $\overline{s}_j$ by the sequence.

As an example, for the LTS in Figure 2, we can obtain the harmonized state identifiers $W_0 = \{a, b\}, W_1 = \{b.a\}, W_2 = \{b.a\}, W_3 = \{a, b\}$. We only consider W_0: a is used to distinguish $\overline{s}_0$ from $\overline{s}_3$, so a is also in W_3; b is used to distinguish $\overline{s}_0$ from $\overline{s}_1$ and $\overline{s}_2$, so W_1 and W_2 have $b.a$ where b is its prefix.

We use "." to represent the concatenation of two sets of sequences. Formally, assuming $V_1, V_2 \subseteq \Sigma^*$, *the concatenation of sets*, $V_1.V_2$, is defined as a set $\{\sigma_1.\sigma_2 \mid \sigma_1 \in V_1 \wedge \sigma \in V_2\}$. We also write $V^n = V.V^{n-1}$ for $n > 0$ and $V^0 = \{\varepsilon\}$.

In the following, we give the test generation algorithm for a given TOS and trace equivalence. This algorithm derives a test suite with complete fault coverage in a class of LTS implementations in which the number of states in the TOS of each implementation is bounded by a known integer. Thus, in order to apply this algorithm, the user must previously give an estimate on this upper bound. The upper bound for implementations is similarly required in existing test generation methods in the FSM realm. Estimating the bound is an intuitive process based on the knowledge of the given specification and implementation. In the simplest case, one may assume that this bound is equal to the number of states of the specification.

Test Generation Algorithm:

Input: A TOS $\overline{\mathsf{S}}$ and the upper bound m on the number of the states in the TOSs of all LTS implementations.

Output: A test suite TS.

Step 1: Let the number of states in $\overline{\mathsf{S}}$ be n ($n \leq m$). Find a tuple of harmonized state identifiers $\{W_0, W_1, \ldots, W_{n-1}\}$ from $\overline{\mathsf{S}}$.

Step 2: Construct a minimal set of sequences $Q \subseteq \Sigma^*$ such that
$$\forall \overline{s}_i \in \overline{S}\ \exists \sigma \in Q\ (\overline{s}_0 = \sigma \Rightarrow \overline{s}_i).$$

Step 3: Construct the set P such that
$$P = \{\sigma.a \in Q.(\textstyle\bigcup_{i=0}^{m-n+1} \Sigma^i) \mid \overline{s}_0 = \sigma \Rightarrow \overline{s}_i \neq a \Rightarrow\}$$

Step 4: Construct a tuple of the sets $\{R_0, R_1, \ldots, R_{n-1}\}$ such that
$$R_i = \{\sigma \in Q.(\textstyle\bigcup_{i=0}^{m-n+1} \Sigma^i) \mid \overline{s}_0 = \sigma \Rightarrow \overline{s}_i\}.$$

Step 5: Construct the set T of action sequences such that
$$T = (\textstyle\bigcup_{i=0}^{n-1} R_i.W_i) \cup P.$$

Step 6: Construct TS by transforming each sequence $a_1.a_2 \ldots .a_k$ in T into a corresponding LTS $p_0 - a_1 \rightarrow p_1 \ldots p_{k-1} - a_k \rightarrow p_k$ and applying the following state labeling:

$$\ell(p_i) = \begin{cases} \textbf{inconclusive} & i < j \\ \textbf{pass} & i = j \\ \textbf{fail} & \text{otherwise.} \end{cases}$$

where $1 \leq j \leq k$ such that $a_1.a_2 \ldots .a_j \in Tr(\overline{s}_0)$ but $a_1.a_2 \ldots .a_j.a_{j+1} \notin Tr(\overline{s}_0)$.

This method resembles the HSI-method [8] for non-deterministic FSMs, which was originally developed for deterministic FSMs [9]. We intuitively explain the validity of this method. Q is a cover of all states in $\overline{\mathsf{S}}$; $R_0 \cup R_1 \ldots \cup R_{n-1}$ is intended to be a cover of all the transitions in the IUT allowed by the specification, while P is intented to be a cover of all the transitions forbidden by the specification. If a specified transition is implemented, the state identification is needed to check the tail state of the transition. If a specified transition is missed, there exists a sequence in $R_0 \cup R_1 \ldots \cup R_{n-1}$ which can not be observed, and the test case corresponding to this sequence will fail the IUT. If an unspecified transition is implemented, the IUT will fail the test case corresponding to a sequence in P and no state identification is required, unlike to testing partially specified FSMs. We note that in FSM testing, transition checking is considered only for the inputs allowed by the specification and the state identification includes the tail states of all these transitions.

Theorem Given an LTS specification $\overline{\mathsf{S}}$ of the TOS form and any LTS implementation M. Suppose $n \leq m$ where n is the number of states of $\overline{\mathsf{S}}$, and m is the upper bound on the number of states in the TOS of M. Let TS be the test suite derived from $\overline{\mathsf{S}}$ using the test generation algorithm. We have M passes TS if and only if $\overline{\mathsf{S}} \approx_{\mathbf{tr}} \mathsf{M}$.

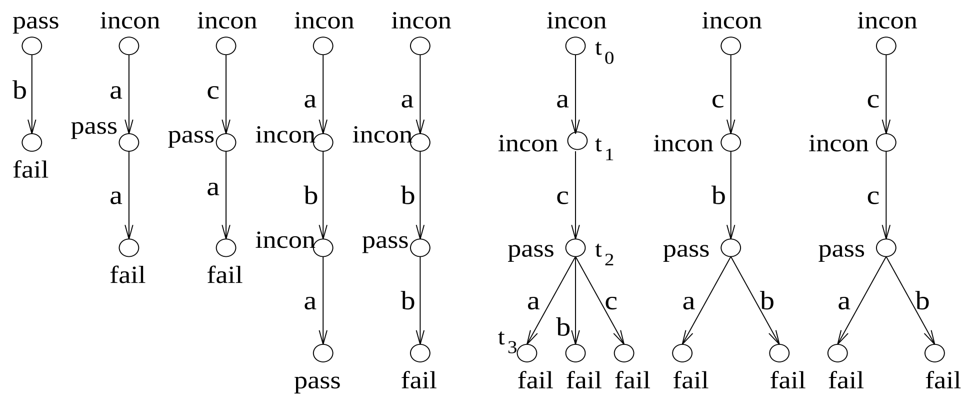

Figure 3: A test suite for the LTS specification in Figure 1

In other words, we claim that the algorithm yields a finite test suite with complete fault coverage in the sense that it detects any trace–nonequivalent implementation, provided that the number of states in the TOS of this implementation is not more than m.

As an example, assuming the TOS of any implementation does not have more states than the specification given in Figure 2, we derive a test suite, which checks trace equivalence with respect to this specification as well as to the specification in Figure 1 and guarantees full fault coverage, as follows.

Intermediate results:

	$\overline{s}_0$	$\overline{s}_1$	$\overline{s}_2$	$\overline{s}_3$
W_i	$a,\ b$	$b.a$	$b.a$	$a,\ b$
Q	ε	a	c	$a.c$
R_i	$\varepsilon,\ a.b$	a	c	$a.c,\ c.b,\ c.c$
$P = \{b,\ a.a,\ c.a,\ a.c.a,\ a.c.b,\ a.c.c\}$				

The set T of test sequences: $\{b,\ a.a,\ c.a,\ a.b.b,\ a.b.a,\ a.c.a,\ a.c.b,\ a.c.c,\ c.b.a,\ c.b.b,\ c.c.a,\ c.c.b\}$. The resulting test suite *TS* is given in Figure 3.

Considering a test case $t_0 \text{—}a\!\rightarrow t_1 \text{—}c\!\rightarrow t_2 \text{—}a\!\rightarrow t_3$, where sequence $a.c.a$ is not a trace of the specification but its prefix $a.c$ is. According to trace equivalence, $a.c.a$ should not be implemented and $a.c$ must be implemented, so t_3 is labeled with **fail** and t_2 with **pass**. Due to nondeterminism of implementations, several test runs are needed to obtain all possible observations for the application of this test case, and t_0 and t_1 are labeled with **inconclusive** for possible deadlocks. In testing, if $a.c$ is observed but $a.c.a$ not, the test purpose is achieved and a **pass** verdict is given to the IUT for this test case. On the other hand, if $a.c.a$ is observed or $a.c$ does not occur, this means that an invalid trace is implemented or a valid trace is not implemented, so a **fail** verdict is given.

Similarly, we could also use the ideas of the W- and Wp-methods [4, 7] for test generation of LTSs with respect to trace equivalence with the same fault coverage power. In fact, the union of harmonized state identifiers for an LTS can treated as a characterization set W for the LTS, in which for any two states there exists a sequence such that one of its prefixes is a trace of either of the two states, not both. However, such a W may contain the sequences whose suffixes are not necessary for identification of some states; thus it follows that the test cases derived may have certain redundancy [12]. For example, a W set for the LTS in Figure 2 includes $b.a$, in which the suffix a is not necessary to identify the initial state s_0 because b should be blocked in the corresponding state for any conforming implementation. The given method does not produce the redundancy since the harmonized state identifiers do not contain such suffixes.

5 Conclusion

LTSs are the basic semantics for the LOTOS language and other specification formalisms. We presented in this paper a method for generating test cases from a specification given in the LTS formalism with respect to the so-called trace equivalence relation. For I/O machines, several existing methods can be used to derive finite test suites with guaranteed fault coverage for trace equivalence, but they are not directly applicable to LTSs. The existing methods based on LTSs for the trace equivalence either do not assure fault coverage, or require a transformation from LTSs to I/O FSMs. In the method proposed in this paper, such transformation is not required and the derived test suites have complete fault coverage, provided that the state number of the implementations is bounded by a known integer.

In our method the notion of the HSI state identifiers is used for state identification. Other state identification techniques, such as the notion of a characterization set, in which for any two states of a given LTS there exists a sequence of observable actions such that one of its prefixes is is a trace of either of the two states, not both, can also be used in a similar way.

Appendix

In this appendix we give the proof of the Theorem. First we recall the basic assumptions for Test Generation Algorithm and introduce several notations to help the proof, then we prove a series of lemmas which lead to the Theorem.

Given an LTS specification S and an LTS implementation M, we assume in the following:
(1) All states of S and M are reachable from the initial state s_0 and i_0, respectively.
(2) $\overline{\mathsf{S}}$ is the corresponding trace observable system of S and has n states with $n > 1$.
(3) $\overline{\mathsf{M}}$ is the corresponding trace observable system of M and has at most m states with $m \geq n$.
(4) $\overline{s}_i, \overline{s}_j, \overline{s}_k, \overline{s}_l$ and $\overline{m}_i, \overline{m}_j, \overline{m}_k, \overline{m}_l$ represent the states of $\overline{\mathsf{S}}$ and $\overline{\mathsf{M}}$, respectively.
(5) A tuple of state identifiers $\{W_0, W_1, \ldots, W_{n-1}\}$.
(6) Sets Q, T and the test suite TS, which are defined in Test Generation Algorithm.

Definition 7 *V–equivalence.* Given a set $V \subseteq \Sigma^*$, The V–equivalence relation between two states p and q, written $p \approx_V q$, holds if and only if for all $\sigma \in Pref(V)$, $\sigma \in Tr(p) \Leftrightarrow \sigma \in Tr(q)$. Given two LTSs S and M with initial states s_0 and m_0 respectively, we say that M is V-equivalent to S, written $\mathsf{S} \approx_V \mathsf{M}$, if only if $s_0 \approx_V m_0$.

notation	meaning
$[\overline{s}_i, \overline{m}_i] - a \rightarrow [\overline{s}_j, \overline{m}_j]$	For $a \in \Sigma, \overline{s}_i - a \rightarrow \overline{s}_j$ and $\overline{m}_i - a \rightarrow \overline{m}_j$
$[\overline{s}_i, \overline{m}_i] = \sigma \Rightarrow [\overline{s}_j, \overline{m}_j]$	For $\sigma \in \Sigma^*, \overline{s}_i = \sigma \Rightarrow \overline{s}_j$ and $\overline{m}_i = \sigma \Rightarrow \overline{m}_j$
$[\overline{s}_i, \overline{m}_i]$-**after**-$V$	given a pair of states $[\overline{s}_i, \overline{m}_i] \in \overline{S} \times \overline{M}$, and a set $V \subseteq \Sigma^*$ $[\overline{s}_i, \overline{m}_i]$-**after**-$V = \{[\overline{s}_j, \overline{m}_j] \mid \forall \sigma \in Pref(V)\ [\overline{s}_i, \overline{m}_i] = \sigma \Rightarrow [\overline{s}_j, \overline{m}_j]\}$
D	$D = [\overline{s}_0, \overline{m}_0]$-**after**-$\Sigma^*$
D_r	$D_r = \{[\overline{s}_i, \overline{m}_j] \in D \mid \overline{s}_i \approx_{W_i} \overline{m}_j\}$
$\overline{\Sigma}^k$	$\overline{\Sigma}^k = \bigcup_{i=0}^{k} \Sigma^i$

Lemma 1 *For $V \subseteq \Sigma^*$, assume $|[\overline{s}_0, \overline{m}_0]$-**after**-$V| \geq k$. If $|D| > k$, then $|[\overline{s}_0, \overline{m}_0]$-**after**-$V.\overline{\Sigma}^1| \geq k+1$; if $|D| \leq k$, then $[\overline{s}_0, \overline{m}_0]$-**after**-$V.\overline{\Sigma}^1 = [\overline{s}_0, \overline{m}_0]$-**after**-$V$.*

Proof:
(I) To prove that the lemma holds when $|D| > k$.
The lemma holds when $|[\overline{s}_0, \overline{m}_0]$-**after**-$V| > k$. Consider the case that $|[\overline{s}_0, \overline{m}_0]$-**after**-$V| = k$.

(1)	$\lvert D\rvert > k$ and $\lvert[\overline{s}_0, \overline{m}_0]$-**after**-$V\rvert = k$	hypothesis
(2)	$[\overline{s}_0, \overline{m}_0]$-**after**-$V \subseteq D$	definition of D
(3)	$\exists [\overline{s}_k, \overline{m}_k] \in D \backslash [\overline{s}_0, \overline{m}_0]$-**after**-$V$	(1),(2)
	$\exists [\overline{s}_i, \overline{m}_i] \in [\overline{s}_0, \overline{m}_0]$-**after**-$V$	(1)
	$\exists \sigma \in Pref(V)\ \exists \sigma.a \in \Sigma^*([\overline{s}_0, \overline{m}_0] = \sigma \Rightarrow [\overline{s}_i, \overline{m}_i] - a \rightarrow [\overline{s}_k, \overline{m}_k])$	(2)
(4)	$[\overline{s}_k, \overline{m}_k] \in [\overline{s}_0, \overline{m}_0]$-**after**-$V.\overline{\Sigma}^1 \backslash [\overline{s}_0, \overline{m}_0]$-**after**-$V$	(3)
(5)	$[\overline{s}_0, \overline{m}_0]$-**after**-$V.\overline{\Sigma}^1 \geq k+1$	(4).

(II) To prove that the lemma holds when $|D| \leq k$.

(1)	$\lvert D\rvert \leq k$ and $\lvert[\overline{s}_0, \overline{m}_0]$-**after**-$V\rvert = k$	hypothesis
(2)	$[\overline{s}_0, \overline{m}_0]$-**after**-$V \subseteq D$	definition of D
(3)	$[\overline{s}_0, \overline{m}_0]$-**after**-$V.\overline{\Sigma}^1 = [\overline{s}_0, \overline{m}_0]$-**after**-$V$	(1),(2).

Lemma 2 *Assume $\overline{s}_0 \approx_Q \overline{m}_0$. If $|D| > m$, then $|[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n}| \geq m$; and if $|D| \leq m$, then $[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n} = D$.*

Proof:
(I) To prove that the lemma holds when $|D| > m$.

(1)	$\overline{s}_0 \approx_Q \overline{m}_0$ and $\lvert D\rvert > m$	hypothesis
(2)	$\lvert[\overline{s}_0, \overline{m}_0]$-**after**-$Q\rvert \geq n$	initially connected $\overline{\mathsf{S}}$, (1)
(3)	$\lvert[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n}\rvert \geq m$	Lemma 1, (1),(2).

(II) It is evident from Lemma 1 when $|D| \leq m$.

Lemma 3 *If $\overline{s}_i \approx_{W_i} \overline{m}_k$ and $\overline{s}_j \approx_{W_j} \overline{m}_k$, then $i = j$.*

Proof:

(1)	For $V \subseteq \Sigma^*, \overline{s}_i \approx_V \overline{m}_k \Leftrightarrow \overline{s}_i \approx_{Pref(V)} \overline{m}_k$	evident
(2)	$\overline{s}_i \approx_{W_i} \overline{m}_k$ and $\overline{s}_j \approx_{W_j} \overline{m}_k$	hypothesis
(3)	$\overline{s}_i \approx_{Pref(W_i)} \overline{m}_k$ and $\overline{s}_j \approx_{Pref(W_j)} \overline{m}_k$	(1),(2)
(4)	$i \neq j$	assumption
(5)	$\exists \sigma \in Tr(\overline{s}_i) \oplus Tr(\overline{s}_j) \cap Pref(W_i) \cap Pref(W_j)$	definition of W_i, (4)
(6)	let $\sigma \in Tr(\overline{s}_i)$, then $\sigma \in Tr(\overline{m}_k)$	(3)
(7)	$\sigma \in Tr(\overline{s}_j)$	(3),(6)
(8)	$i = j$	(6),(7)$\not\Leftrightarrow Tr(\overline{s}_i) \oplus Tr(\overline{s}_j)$.

Lemma 4 $\lvert D_r \rvert \le m$.

Proof:

(1)	$\lvert\overline{M}\rvert \le m$	hypothesis
(2)	$\lvert D_r\rvert > m$	assumption
(3)	$\exists[\overline{s}_i, \overline{m}_k], [\overline{s}_j, \overline{m}_k](i \neq j, \overline{s}_i \approx_{W_i} \overline{m}_k \wedge \overline{s}_j \approx_{W_j} \overline{m}_k)$	(1),(2)
(4)	$\lvert D_r\rvert \le m$.	(3)$\not\Leftrightarrow$Lemma 3.

Lemma 5 *If* $\overline{s}_0 \approx_T \overline{m}_0$*, then* $[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n} = D_r = D$.

Proof:

(I) To prove that the lemma holds when $\lvert D\rvert \le m$.

(1)	$\lvert D\rvert \le m$	hypothesis
(2)	$\overline{s}_0 \approx_T \overline{m}_0$	hypothesis
(3)	$\overline{s}_0 \approx_Q \overline{m}_0$	(2)
(4)	$[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n} = D$	(1),(3),Lemma 2
(5)	$\forall[\overline{s}_i, \overline{m}_j] \in [\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n} (\overline{s}_i \approx_{W_i} \overline{m}_j)$ (2)	
(6)	$D = Dr$	(4),(5),definition of D_r

(II) To prove that the lemma holds when $\lvert D\rvert > m$.

(1)	$\lvert D\rvert > m$	assumption
(2)	$\overline{s}_0 \approx_T \overline{m}_0$	hypothesis
(3)	$[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n+1} \subseteq D$	definition of D
(4)	$\forall[\overline{s}_i, \overline{m}_j] \in [\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n+1} (\overline{s}_i \approx_{W_i} \overline{m}_j)$	(2)
(5)	$[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n+1} \subseteq D_r$	(3),(4),definition of D_r
(6)	$\lvert[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n+1}\rvert \ge m+1$	(1),(2),Lemma 2,Lemma 1
(7)	$\lvert D_r\rvert \ge m+1$	(3),(4)
(8)	$\lvert D\rvert \le m$	(5)$\not\Leftrightarrow$Lemma 4
(9)	$[\overline{s}_0, \overline{m}_0]$-**after**-$Q.\overline{\Sigma}^{m-n} = D_r = D$	(6),Lemma 2.

Lemma 6 *If* $\overline{s}_0 \approx_T \overline{m}_0$*, then* $\overline{s}_0 \approx_{\mathbf{tr}} \overline{m}_0$.

Proof:

(1)	$\overline{s}_0 \approx_T \overline{m}_0$	hypothesis
(2)	$\forall[\overline{s}_i, \overline{m}_i] \in D\ \exists\sigma \in Q.\overline{\Sigma}^{m-n}\ ([\overline{s}_0, \overline{m}_0] = \sigma \Rightarrow [\overline{s}_i, \overline{m}_i])$	(1),Lemma 5
(3)	$\overline{s}_i \approx_\Sigma \overline{m}_i$	(1)
(4)	$not(\overline{s}_0 \approx_{\mathbf{tr}} \overline{m}_0)$	assumption
(5)	$\exists a \in \Sigma\ \exists[\overline{s}_i, \overline{m}_i] \in D\ not(\overline{s}_i \approx_{\{a\}} \overline{m}_i))$	(4)
(6)	$\overline{s}_0 \approx_{\mathbf{tr}} \overline{m}_0$	(5)$\not\Leftrightarrow$(3).

Lemma 7 M *passes TS if and only if* $\overline{\mathsf{S}} \approx_{\mathbf{tr}} \mathsf{M}$.

Proof: For each $\sigma \in T$, we obtain a test case be T such that $\ell(\sigma_i) = \mathbf{pass}$ where $\sigma = \sigma_i.\sigma_j$ and $\sigma_i \in Tr(\mathsf{S})$. If $\overline{\mathsf{S}} \approx_{\mathbf{tr}} \mathsf{M}$, then $\sigma_i \in Obs_{(\mathrm{T},\mathrm{M})}$ and any $\sigma' \in Obs_{(\mathrm{T},\mathrm{M})}$ implies $\sigma' \in Pref(\sigma_i)$. Thus $v(Obs_{(\mathrm{T},\mathrm{M})}) = \mathbf{pass}$.

On the other hand, if $\overline{\mathsf{S}} \approx_{\mathbf{tr}} \mathsf{M}$ does not hold, from Lemma 6, there exists $\sigma \in T$ such that $\sigma \in Tr(\mathsf{M}) \backslash Tr(\mathsf{S})$ or $\sigma \in Tr(\mathsf{S}) \backslash Tr(\mathsf{M})$. For the former, T has $\ell(\sigma) = \mathbf{fail}$ and $\sigma \in Obs_{(\mathrm{T},\mathrm{M})}$; for the latter, $\ell(\sigma) = \mathbf{pass}$ but $\sigma \notin Obs_{(\mathrm{T},\mathrm{M})}$. Thus $v(Obs_{(\mathrm{T},\mathrm{M})}) = \mathbf{fail}$.

References

[1] J. Arkko. On the existence and production of state identification machines for labeled transition systems. In R. L. Tennecy, P. D. Amer, and M. U. Uyar, editors, *IFIP Formal Description Techniques VI*, pages 351–365, 1993.

[2] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *IFIP Protocol Specification, Testing, and Verification VIII*, pages 63–74. North-Holland, 1988.

[3] A. R. Cavalli and S. U. Kim. Automated protocol conformance test generation based on formal methods for LOTOS specifications. In G.v. Bochmann, R. Dssouli, and A. Das, editors, *IFIP 5th International Workshop on Protocol Test Systems*, pages 212–220. North-Holland, 1992.

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

[5] K. Drira, P. Azema, and F. Vernadat. Refusal graphs for conformance tester generation and simplification: a computational framework. In A. Danthine, G. Leduc, and P. Wolper, editors, *IFIP Protocol Specification, Testing, and Verification XIII*. North-Holland, 1994.

[6] 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. North-Holland, 1991.

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

[8] 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, Japan, 1994.

[9] A. Petrenko. Checking experiments with protocol machines. In J. Kroon, J. Heijink, and E. Brinksma, editors, *IFIP 4th International Workshop on Protocol Test Systems*, pages 83–94. North-Holland, 1991.

[10] 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, Pau, France, 1993.

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

[12] Q. M. Tan, A. Petrenko, and G. v. Bochmann. Modeling basic LOTOS by FSMs for conformance testing. In *IFIP Protocol Specification, Testing, and Verification XIIII*, Poland, 1995.

[13] J. Tretmans. Test case derivation from LOTOS specifications. In S. T. Vuong, editor, *IFIP 2th International Conf. on Formal Description Techniques for Distributed Sysytems and Communication Protocols*, pages 345–359. North-Holland, 1990.