A Framework for Conformance Testing of Systems Communicating through Rendezvous

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

Abstract

In this paper, a formal framework is first proposed for conformance testing of communication systems, which are modeled by labeled transition systems (LTSs), in a systematic and operational approach. In this framework, test cases are limited to deterministic processes with finite behavior and state labels; testing is a finite set of experiments where every test case is parallelly composed with an implementation under test; observations are action sequences, executed during the testing, from which the test verdict is drawn directly. The fault model and fault coverage criteria are introduced to measure the effectiveness of testing. Afterwards, based on this framework, for several common conformance relations, we present corresponding functions for the state labeling of test cases and upper bounds on the necessary sizes of test suites for obtaining complete fault coverage.

1. Introduction

One of the important issues in conformance testing of communication systems is to define a conceptual testing framework for systems communicating by rendezvous. Rendezvous does not distinguish explicitly between inputs and outputs, and 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. One of the specification formalisms for such systems is labeled transition systems (LTSs); it also serves as a semantic model for various specification languages, e.g., LOTOS [2], CCS [14], and CSP [9]. Since there are various criteria for conformance of LTSs, from which a dozen of conformance relations are introduced [20], in order to facilitate test generation, a formal testing framework should be defined to answer the following questions: how are test cases structured for a given conformance relation, what constitute observations in testing and how is the verdict assigned? Moreover, this framework should also take into account the current practice, that is, conformance testing as a finite activity which should provide a well-defined confidence that the implementation under test conforms to its specification.

Theories of conformance testing and methods for test derivation from LTSs have been developed in [3, 4, 20, 17, 21, 15, 19, 11, 7, 16, 18]. A fundamental framework was introduced in [4]. In this framework, the return status of testing (successful or fail exit, deadlock, etc.) are considered as observations, and the verdict obtained from a given test case depends only on whether or not the observations obtained during testing of the IUT are a subset of the observations expected from the specification. However, intuitively, it is clear that the interaction sequences observed during testing may be important for defining the verdict, and should in general not be ignored. In fact, it is difficult for the framework of [4] to define the verdict for certain conformance relations, such as trace equivalence, nondeterminism reduction (**conf** plus trace equivalence) [6], and failure equivalence. Furthermore, since test cases with infinite and nondeterministic behavior are allowed, no attempt is made in this framework to describe how to obtain the verdict in an operational way through finite experiments.

Another similar framework for conformance testing in the LTS formalism is drawn in [19] from the OSI Conformance Testing Methodology and Framework. In this framework, states of test cases are directly labeled with verdicts, and the verdict assignment is obtained from the verdicts returned during testing. Although this framework considers the action sequences observed during testing as observations, except for the **conf** relation, this framework does not answer how to structure test cases and assign the verdict for a given relation. Like the above framework, testing is treated as a correctness-proving process with respect to the given conformance relation. Therefore infinite testing is allowed and no fault coverage is considered. As a result, no estimation can be given for complexity of testing with guaranteed fault coverage.

In this paper, we propose a formal testing framework for the purpose of testing systems communicating through rendezvous in a systematic and operational approach. In our framework, test cases are limited to be deterministic and finite processes with state labels, and furthermore action sequences executed during testing are considered as observations. This results in a unified verdict assignment for several common conformance relations. Since testing is a finite set of experiments, the notions of fault model and fault coverage are introduced to measure the effectiveness of testing. Afterward, based on this framework, for each of the conformance relations, we present a state labeling procedure by which test cases in respect to the relation are formed, and an upper bound that guarantees full fault coverage, provided that the number of multi-states of any implementation is not more than a known integer.

In Section 2, we review basic definitions and notations. In Section 3, we give the testing framework. In Section 4, we present the state labeling functions. In Section 5, we discuss upper bounds of test suites for different conformance relations.

2. Basic Definitions and Notations of Conformance Testing

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

2.1. 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 $p{-}a{\rightarrow} p_1, p{-}a{\rightarrow} p_2 \in \Delta$ and $p_1 \neq p_2$. A *deterministic* LTS has no internal actions and the outgoing transitions of each 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.

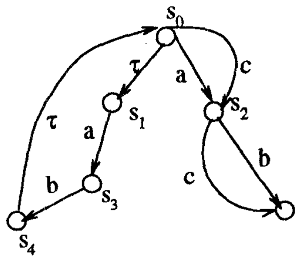

Figure 1. An LTS graph

The notations shown in Table 1 are relevant to a given LTS. 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. $Tr(s_0)$ is called the set of *traces* of S and $Ref(s_0, \sigma)$ the set of *refusals* of S after the observable action sequence σ.

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

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 [8]), 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 a set $\Pi_S = \{S_i \subseteq S \mid \exists \sigma \in Tr(s_0) \ (s_0\text{-}\mathbf{after}\text{-}\sigma = S_i)\}$.

Note that the empty sequence ε is supposed to be in Σ^*. Therefore $S_0 = s_0$-**after**-ε is in the multi-state set, 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 automata with trace-equivalence [10]. For Figure 1, the multi-state set is $\{\{s_0, s_1\}, \{s_2, s_3\}, \{s_2\}, \{s_0, s_1, s_4, s_5\}, \{s_5\}\}$. Obviously, each LTS has one and only one multi-state set.

Notations used for multi-states are shown in Table 2. In this table, $P, Q \subseteq S$, where S is the state set of the LTS S. Using the extended notations, it is easy to show that $Tr(s_0) = Tr(S_0)$ and s_0-**after**-$\sigma = S_0$-**after**-σ.

2.2. Conformance relations

There are different criteria for determining whether an implementation conforms to a specification. Such criteria can be formalized as conformance relations. Different

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

Table 1. Basic notations for labeled transition systems

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

Table 2. Additional notations for labeled transition systems

conformance relations have been proposed for comparing labeled transition systems [20]. We say that an implementation M conforms to a specification S if the chosen conformance relation holds between M and S. The following conformance relations are considered in this paper.

Definition 3 (*Conformance relations*):

Trace reduction ($\leq_t$): M $\leq_t$ S iff $Tr(m_0) \subseteq Tr(s_0)$.

Trace equivalence ($\approx_t$): M $\approx_t$ S iff $Tr(m_0) = Tr(s_0)$.

Failure reduction ($\leq_f$): M $\leq_f$ S iff $\forall \sigma \in \Sigma^*$ $Ref(m_0, \sigma) \subseteq Ref(s_0, \sigma)$.

Nondeterminism reduction ($\leq_n$): M $\leq_n$ S iff M $\approx_t$ S and M $\leq_f$ S.

Failure equivalence ($\approx_f$): M $\approx_f$ S iff M $\leq_f$ S and S $\leq_f$ M.

The trace reduction relation allows that the implementation M has the same or less traces than the specification S, whereas the trace equivalence relation requires that M has the same traces as S. The failure reduction relation further requires that everything of M is allowed by S, not only the traces but also the actions refused after any observable action sequence (deadlock). The failure equivalence not only states that everything that M does must be allowed by S, but also requires that everything prescribed by S should be implemented by M. The nondeterminism reduction relation accepts M if M is less nondeterministic than S, that is, M has the same traces as S, and equal or less deadlocks than S after any given trace. We note that for deterministic systems, the trace equivalence relation is the failure equivalence relation. Nondeterminism differentiates the two relations, so nondeterminism reduction corresponds to implementation choices for a nondeterministic specification. The trace reduction and equivalence relations belong to *trace semantics* [9], while the other three relations belong to *failure semantics* [5]. The trace and failure semantics only require a simple testing scenario in which a testing process is a communication process between the tester and the implementation under test.

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

3. Conformance Testing as Experiments

Conformance testing is a finite set of experiments, in which a set of test cases, derived from a given specification according to a given conformance relation, is applied by a tester or experimenter to the implementation under test (IUT). 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 a test experiment is defined by the test case used in this experiment. Thus a test case is a specification of behavior, which, like other specifications, can be represented as an LTS. 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 4 (*Test cases and test suite*): Given an LTS specification $\mathsf{S} =< S, \Sigma, \Delta, s_0 >$, a test case T of 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 $t_0 = \sigma \Rightarrow p$;
- $\ell : T \rightarrow \{\textbf{pass}, \textbf{fail}, \textbf{inconclusive}\}$ is a state labeling function.

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

From this definition, the behavior of a 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 can define $\ell(\sigma) = \ell(t)$ for $\{t\} = t_0\textbf{-after-}\sigma$.

The rendezvous interactions between a test case T and the IUT M can be formalized by the synchronization operator "||" of LOTOS, that is, $t_0 \mid\mid m_0$. When $t_0 \mid\mid m_0$ after an observable action sequence σ reaches a *deadlock*, that is, there exists state $p \in T \times M$ such that for all actions $a \in \Sigma$, $(t_0 \mid\mid m_0) = \sigma \Rightarrow p$ and $p \not= a \Rightarrow$, we say that this experiment completes a *test run*. In the light of the tester, each state of the test case T is characterized by the set of actions out of this state which are offered by the tester to the IUT. If this set is empty, we say that the test case has reached an inactive state; the other states are called active. The completion of a test run means that an inactive state is reached or the set of actions offered as next interactions is refused by the IUT.

Each test run produces an observable action sequence σ, which is a trace of both T and M; from T a unique label $\ell(\sigma)$ is determined by σ, which is called the verdict of the test run. We call σ (or σ and $\ell(\sigma)$ together) an *observation*. (We do not treat deadlock as observation since each test run ends as deadlock.)

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

Usually, LTSs are supposed to be nondeterministic. In order to test nondeterministic implementations, one usually makes the so-called *complete-testing assumption* which says that 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 that can be traversed by the test case [8, 13, 12]. Without such an assumption, no test suite can guarantee full fault coverage (in terms of conformance relations) for nondeterministic implementations. Therefore any experiment, in which M is tested by T, should include several test runs and lead to a complete set of observations:

$$\begin{aligned} Obs_{(\mathsf{T},\mathsf{M})} \;=\; & \{\sigma \in Tr(t_0) \mid \exists p \in T \times M, \forall a \in \Sigma : \\ & (t_0 \mid\mid m_0) = \sigma \Rightarrow p \not= a \Rightarrow\}. \end{aligned}$$

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

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

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

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. We here take the mutation approach [1], that is, we define the fault model to be a set $\mathcal{F}$ of all faulty LTS implementations considered. Based on $\mathcal{F}$, a test suite with complete fault coverage for a given LTS specification with respect to a given conformance relation can be defined as follows.

Definition 6 (*Complete test suite*): Given an LTS specification S, a fault model $\mathcal{F}$ and a conformance relation r, a test suite TS for S with respect to r is said to be complete, if for any M in $\mathcal{F}$, r holds between M and S if and only if M passes T for each T in TS.

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

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

4. State Labelings of Test Cases

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

4.1. Trace equivalence

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

Definition 7 (*Test cases for trace equivalence*): Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\approx_t$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0$-**after**-σ, the state labeling of T satisfies

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

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

Note that in T, σ leading to a state with **pass** is not a prefix of any other sequence in $Tr(s_0) \cap Tr(t_0)$.

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

A test case for the LTS given in Figure 1 in respect to trace equivalence is shown in Figure 2 (b).

4.2. Trace Reduction

For the trace reduction relation, a conforming implementation may contain any part of valid traces of its specification but no invalid traces. Thus for this relation the state labeling of test cases should be **fail** for the tail states of the sequences that are invalid traces of the specification; the other states are labeled with **inconclusive**.

Definition 8 (*Test cases for trace reduction*): Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\leq_t$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0$-**after**-σ, the state labeling of T satisfies

$$\ell_{\leq_t}(t_i) = \begin{cases} \textbf{fail} & \text{if } \sigma \notin Tr(s_0) \\ \textbf{inconclusive} & \text{otherwise.} \end{cases}$$

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

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

A test case for the LTS given in Figure 1 in respect to trace reduction is shown in Figure 2 (a).

4.3. Failure Equivalence

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

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

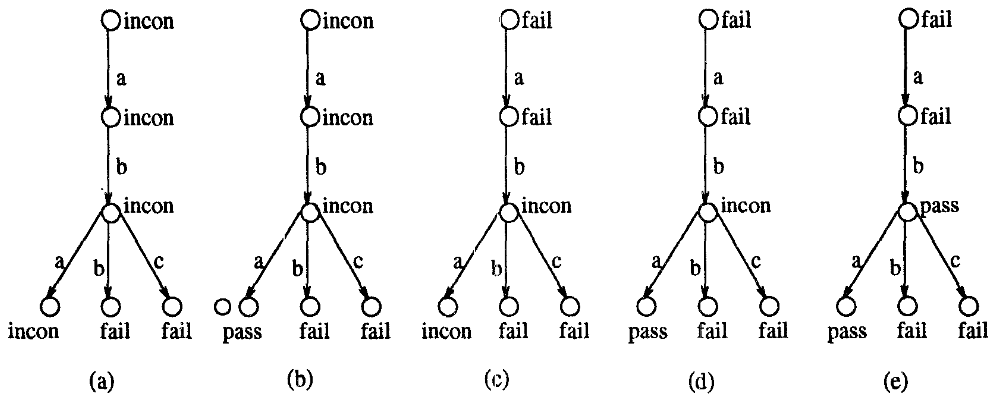

Figure 2. Test cases for (a) $\leq_t$, (b) $\approx_t$, (c) $\leq_f$, (d) $\leq_n$ and (e) $\approx_f$

Definition 9 (*Test cases for failure equivalence*): Given an LTS specification S, a test case T is said to be a test case for S w.r.t. $\approx_f$, if, for all $\sigma \in Tr(t_0)$ and $\{t_i\} = t_0$-**after**-σ, the state labeling of T satisfies

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

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

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

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

A test case for the LTS given in Figure 1 in respect to failure equivalence is shown in Figure 2 (e).

4.4. Failure Reduction

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

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

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

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

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

A test case for the LTS given in Figure 1 in respect to trace reduction is shown in Figure 2 (c).

4.5. Nondeterminism Reduction

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

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

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

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

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

Proposition 5 *Given a test case* T *for* S *w.r.t.* $\leq_n$, *for any LTS* M, *if* M $\leq_n$ S, *then* M *passes* T.

A test case for the LTS given in Figure 1 in respect to nondeterminism reduction is shown in Figure 2 (d). We note that in this case the five test cases in Figure 2 to have the same underlying LTS, but clearly they have different state labelings.

5. Complexity of Testing

In this section, we discuss how to estimate the complexity of testing with complete fault coverage in the class of LTS implementations with at most m multi-states from a given LTS specification, using our testing framework, with respect to any of the conformance relations considered in this paper.

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 $V^0 = \{\varepsilon\}$.

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

In order to perform testing, we assume that the underlying LTS of each test case is a T*-system*, which is formed from a sequence of observable actions σ and a set of observable actions B. The sequence σ may bring the IUT to a desired multi-state from the initial state, so that the transitions or refusals from that multi-state could be checked using the set B.

Definition 12 (T*-system*): A T-system is an LTS formed from a sequence $\sigma = a_1 a_2 \ldots a_n \in \Sigma^*$ and a set $B \subseteq \Sigma$ according to the LOTOS expression $a_1; a_2; \ldots; a_n[]_{(b_j \in B)} b_j$, written $[\sigma \bullet B]$.

A T-system $[\sigma \bullet B]$ is a deterministic and tree-structured LTS. Note that σ may be the empty sequence ε and B may be the empty set $\emptyset$, for example, we have $[\varepsilon \bullet \{a\}] = [a \bullet \emptyset]$. In Figure 2, the underlying LTS of all the test cases is the T-system $[a.b \bullet \{a, b, c\}]$.

Given a multi-state P of the specification, we also denote $blk(P) = \Sigma \backslash out(P)$ and $Acc(P) = powerset(\Sigma) \backslash Ref(P)$. Any action in $blk(P)$ must be blocked at P, while for any set of actions in $Acc(P)$ there exists at least one action in it that must be accepted at P. Similar to the minimal representation of refusal sets introduced in Section 2.1, we also consider a minimal representation of $Acc(P)$, denoted by $\lfloor Acc(P) \rfloor$, since for any two sets of actions A and B in $Acc(P)$, if $A \supseteq B$ then the fact that an action in B must be accepted implies that an action in A must also be accepted. Generally, for any set R, $\lfloor R \rfloor = R \backslash \{A \mid \exists B \in R\, (A \supset B)\}$.

We present in the following a set of propositions which explain how to select a set of LTSs in a worst-case scenario and make it an m-complete test suites for a given LTS specification S and for each of the implementation relations presented in this paper. These propositions make use of several sets of T-systems which are defined for given integers m and n as follows.

$$\begin{aligned}
TS_I &= \{[\sigma \bullet \{a\}] \mid \sigma \in Pref(\Sigma^{mn-1}) \wedge S_0 = \sigma \Rightarrow S_i \\
&\quad \wedge a \in blk(S_i)\} \\
TS_{II} &= \{[\sigma \bullet \{a\}] \mid \sigma \in Pref(\Sigma^{mn-1}) \wedge S_0 = \sigma \Rightarrow S_i \\
&\quad \wedge a \in out(S_i)\} \\
TS_{III} &= \{[\sigma \bullet A] \mid \sigma \in Pref(\Sigma^{mn-1}) \wedge S_0 = \sigma \Rightarrow S_i \\
&\quad \wedge A \in \lfloor Acc(S_i) \rfloor\} \\
TS_{IV} &= \{[\sigma \bullet A] \mid \sigma \in Pref(\Sigma^{mn-1}) \wedge S_0 = \sigma \Rightarrow S_i \\
&\quad \wedge A \in \lceil Ref(S_i) \rceil\}
\end{aligned}$$

Proposition 6 *Given an LTS specification* S *with* n *multi-states, the set* TS_I *with the state labeling* $\ell_{\trianglelefteq}$ *forms an* m*-complete test suite for* S *with respect to* $\leq_t$.

Proposition 7 *Given an LTS specification* S *with* n *multi-states, the set* $TS_I \cup TS_{II}$ *with the state labeling* $\ell_{\approx_t}$ *forms an* m*-complete test suite for* S *with respect to* $\approx_t$.

Proposition 8 *Given an LTS specification* S *with* n *multi-states, the set* $TS_I \cup TS_{III}$ *with the state labeling* $\ell_{\leq_f}$ *forms an* m*-complete test suite for* S *with respect to* $\leq_f$.

Proposition 9 *Given an LTS specification* S *with* n *multi-states, the set* $TS_I \cup TS_{II} \cup TS_{III}$ *with the state labeling* $\ell_{\simeq_f}$ *forms an* m*-complete test suite for* S *with respect to* $\leq_n$.

Proposition 10 *Given an LTS specification* S *with* n *multi-states, the set* $TS_{III} \cup TS_{IV}$ *with the state labeling* $\ell_{\approx_f}$ *forms an* m*-complete test suite for* S *with respect to* $\approx_f$.

The derivation of the above test suites only takes into account the alphabet, the number of multi-states and the properties prescripted by the corresponding relations. If we consider the length of a test case as the total length of all the sequences from the initial state to an inactive state and the length of a test suite is the total length of all its test cases, then we can give an upper bound $mn|\Sigma|^{mn}$ for the length of the smallest m-complete test suite for an LTS specification with n multi-states in trace semantics, and an upper bound $mn2^{|\Sigma|}|\Sigma|^{mn-1}$ in failure semantics.

It is possible to derive complete test suites far below the upper bounds, if the concept of state identification is applied:

leading the IUT to a multi-state, checking a transition or deadlock and verifying the tail state of the transition with a pre-selected set of state identifiers. Such a test generation method has been proposed in [16] for trace equivalence. We will explore the other relations in future work.

Unlike some existing methods [15, 17, 18], in which infinite behavior is approximated by finite behavior, our approach provides a fixed fault coverage for the confidence in the correct functioning of an implementation in conformity with its specification.

6. Conclusion

LTSs are the basic semantics for the LOTOS Language and other specification formalisms. A testing framework has been presented for conformance testing of communication systems in the LTS formalism. In this framework, testing is a finite set of experiments that ensures a certain fault coverage. A test experiment is run by observing all possible traces of synchronous interactions between an implementation under test and a test case which is a finite and deterministic process with state labeling, and a verdict assignment function is used to draw a conclusion from the observed results.

For several common conformance relations, such as trace reduction, trace equivalence, failure reduction, nondeterminism reduction and failure equivalence, we have given the corresponding functions for the state labeling of test cases. Moreover, under the assumption that the number of multi-states in any implementation is bound by a known integer, we have established upper bounds on the necessary sizes of test suites to obtain full fault coverage for these relations, which give an idea on how complex testing can be in worst-case situations. However, for practical purposes, more optimized test derivation methods with guaranteed fault coverage, such as [16], have yet to be established for most of these conformance relations.

Acknowledgments:

This work was supported by a Strategic Research Grant from the Natural Science and Engineering Research Council of Canada.

Appendix

In this appendix we give the proof of all the propositions presented in this paper.

Proposition 1 Proof: According to Definition 7, if there exists $\sigma \in Obs_{(\mathsf{T},\mathsf{M})}$ and $\ell_{\approx_t}(\sigma) =$ **fail** then $\sigma \in Tr(m_0)\backslash Tr(s_0)$; if there exists $\sigma \in Pur(\mathsf{T})$ but $\sigma \notin Obs_{(\mathsf{T},\mathsf{M})}$, then $\sigma \in Tr(s_0)\backslash Tr(m_0)$. The both cases contradict M $\approx_t$ S. So M passes T with respect to $\approx_t$.

Proposition 2 Proof: Similar to the proof of Proposition 1.

Proposition 3 Proof: According to Definition 9, if there exists $\sigma \in Obs_{(\mathsf{T},\mathsf{M})}$ and $\ell_{\approx_f}(\sigma) =$ **fail**, then $out(t_0\textbf{-after-}\sigma) \in Ref(m_0,\sigma)\backslash Ref(s_0,\sigma)$; if there exists $\sigma \in Pur(\mathsf{T})$ but $\sigma \notin Obs_{(\mathsf{T},\mathsf{M})}$, then $out(t_0\textbf{-after-}\sigma) \in Ref(m_0,\sigma)\backslash Ref(s_0,\sigma)$. The both cases contradict M $\approx_f$ S. So M passes T w.r.t. $\approx_f$.

Proposition 4 Proof: Similar to the proof of Proposition 3.

Proposition 5 Proof: Similar to the proof of Proposition 3.

Proposition 6 Proof: $\Rightarrow$ Proposition 2.
$\Leftarrow$ Since any LTS implementation M is assumed to have no more than m multi-states, and S has exactly n multi-states, there are no more than mn different pairs of multi-states for M and S. If M has a trace that is not in $Tr(s_0)$, that is, M $\not\leq_t$ S, then there must exist $\sigma_i.a \in \Sigma^*$ such that $\sigma_i \in Tr(m_0) \cap Tr(s_0)$, $\sigma_i.a \in Tr(m_0)\backslash Tr(s_0)$ and $|\sigma_i| \leq mn - 1$. Let $S_0 = \sigma_i \Rightarrow S_i$, then $a \in blk(S_i)$. Thus we have $[\sigma_i \bullet \{a\}] \in TS_I$ such that $\ell_{\leq_t}(\sigma_i.a) =$ **fail** and $\sigma_i.a \in Obs_{(\mathsf{T},\mathsf{M})}$. This implies $Obs_{fail} \neq \emptyset$. According to the verdict v, M fails T.

Proposition 7 Proof: $\Rightarrow$ Proposition 1.
$\Leftarrow$ There are no more than mn different pairs of multi-states for M and S. If M $\not\approx_t$ S, then there must exist $\sigma_i.a \in \Sigma^*$ such that $\sigma_i \in Tr(m_0) \cap Tr(s_0)$, $\sigma_i.a$ is in $Tr(m_0)\backslash Tr(s_0)$ or $Tr(s_0)\backslash Tr(m_0)$, and $|\sigma_i| \leq mn - 1$. Thus we have $\mathsf{T} = [\sigma_i \bullet \{a\}] \in TS_I \cup TS_{II}$. If $\sigma_i.a \in Tr(m_0)\backslash Tr(s_0)$, then $\sigma_i.a \in Obs_{\mathsf{T},\mathsf{M}}$ and $\ell_{\approx_t}(\sigma_i.a) =$ **fail**; if $\sigma_i.a \in Tr(s_0)\backslash Tr(m_0)$, then $\sigma_i.a \notin Obs_{(\mathsf{T},\mathsf{M})}$ and $\ell_{\approx_t}(\sigma_i.a) =$ **pass**. This implies that $Obs_{fail} \neq \emptyset$, or $Obs_{pass} \neq Pur(\mathsf{T})$. According to the verdict v, M fails T.

Proposition 8 Proof: $\Rightarrow$ Proposition 4.
$\Leftarrow$ If M $\not\leq_f$ S, that is, there exists $\sigma \in \Sigma^*$ such that $Ref(m_0,\sigma) \not\subseteq Ref(s_0,\sigma)$, then $\sigma \in Tr(m_0)\backslash Tr(s_0)$ or $\sigma \in Tr(m_0) \cap Tr(s_0)$, because $Ref(m_0,\sigma) \subseteq Ref(s_0,\sigma)$ otherwise. Note that for any LTS P, $Ref(\mathsf{P},\sigma) = \emptyset$ if $\sigma \notin Tr(\mathsf{P})$. For $\sigma \in Tr(m_0)\backslash Tr(s_0)$, similar to the proof of Proposition 6, there must exist $\mathsf{T} = [\sigma_i \bullet \{a\}] \in TS_I$ such that $\sigma_i.a \in Obs_{\mathsf{T},\mathsf{M}}$ and $\ell_{\leq_f}(\sigma_i.a) =$ **fail**. For $\sigma \in Tr(m_0) \cap Tr(s_0)$, since there are no more than mn different pairs of multi-states for M and S, there exists σ_i of length $\leq mn - 1$ such that if $(M_0, S_0) = \sigma \Rightarrow (M_i, S_i)$ then $(M_0, S_0) = \sigma_i \Rightarrow (M_i, S_i)$. Since $Ref(m_0,\sigma) \not\subseteq Ref(s_0,\sigma)$, there exists $A \subseteq \Sigma, A \neq \emptyset$ such that $A \in Ref(M_i)\backslash Ref(S_i)$. Obviously $A \in Acc(S_i)$, so there exists $B \in \lfloor Acc(S_i) \rfloor$ such that $B \subseteq A$. $B \subseteq A$ also implies $B \in Ref(M_i)$. Thus there exists $\mathsf{T} = [\sigma_i \bullet B] \in TS_{III}$

such that $\ell_{\approx_f}(\sigma_i) = \textbf{fail}$ and $\sigma_i \in Obs_{(\mathrm{T,M})}$. In the both cases, M fails T.

Proposition 9 Proof: $\Rightarrow$ Proposition 5.
$\Leftarrow$ M $\not\leq_n$ S implies that M $\not\leq_f$ S or M $\leq_t$ S and M $\leq_f$ S. If M $\not\leq_f$ S, similar to Proposition 8, we can prove that there exits T $\in TS_I \cup TS_{III}$ such that $Obs_{fail} \neq \emptyset$. If M $\leq_t$ S and M $\leq_f$ S, similar to Proposition 7, we can prove that there exists T $\in TS_I \cup TS_{II}$ such that $Obs_{pass} \neq Pur(\mathrm{T})$. In the both cases, M fails T.

Proposition 10 Proof: $\Rightarrow$ Proposition 3.
$\Leftarrow$ If M $\not\approx_f$ S, then there exists $\sigma \in \Sigma^*$ such that $Ref(m_0, \sigma) \neq Ref(s_0, \sigma)$, that is, there exists $A \subseteq \Sigma$ such that $A \in Ref(m_0, \sigma) \backslash Ref(s_0, \sigma)$ or $A \in Ref(s_0, \sigma) \backslash Ref(m_0, \sigma)$. Since there are no more than mn different pairs of multi-states for M and S, there exists σ_i of length $\leq mn - 1$ such that (1) if $(M_0, S_0) = \sigma \Rightarrow (M_i, S_i)$ then $(M_0, S_0) = \sigma_i \Rightarrow (M_i, S_i)$, or (2) if $\sigma \notin Tr(m_0) \cap Tr(s_0)$ then, let $\sigma_j.a \in Pref(\sigma)$ such that $\sigma_j.a \notin Tr(m_0) \cap Tr(s_0)$ but $\sigma_j \in Tr(m_0) \cap Tr(s_0)$, if $(M_0, S_0) = \sigma_j \Rightarrow (M_i, S_i)$ then $(M_0, S_0) = \sigma_i \Rightarrow (M_i, S_i)$. At first let us consider the case (1). If $A \in Ref(M_i) \backslash Ref(S_i)$, similar to the proof of Proposition 8, there exists T $\in TS_{III}$ such that $Obs_{fail} \neq \emptyset$. If $A \in Ref(S_i) \backslash Ref(M_i)$, then there must exist $B \in \lceil Ref(S_i) \rceil$ such that $B \supseteq A$. Thus there exists $T = [\sigma_i \bullet B] \in TS_{IV}$ such that $\ell_{\approx_f}(\sigma_i) = \textbf{pass}$. Since $A \notin Ref(M_i)$, $B \notin Ref(M_i)$ and furthermore $\sigma_i \notin Obs_{(\mathrm{T,M})}$. This is $Obs_{pass} \neq Pur(\mathrm{T})$. Secondly we consider the case (2). If $a \in out(M_i) \backslash out(S_i)$, then $\{a\} \in Ref(S_i)$, and from this there exists $A \in \lceil Ref(S_i) \rceil$ such that $\{a\} \subseteq A$. Therefore we have T $= [\sigma_i \bullet A] \in TS_{IV}$. Since $a \notin out(S_i)$, $Ref(s_0, \sigma_i.a) = \emptyset$ and from this $\ell_{\approx_f}(\sigma_i.a) = \textbf{fail}$. Furthermore since $a \in out(M_i)$, $\sigma.a \in Obs_{(\mathrm{T,M})}$. This is $Obs_{fail} \neq \emptyset$. If $a \in out(S_i) \backslash out(M_i)$, then there must exist $\{a\} \in Ref(S_i)$ or $\{a\} \in Acc(S_i)$, and from this there exists $A \in \lceil Ref(S_i) \rceil$ such that $\{a\} \subseteq A$ or $A \in \lfloor Acc(S_i) \rfloor$ such that $A = \{a\}$. Therefore we have $T = [\sigma_i \bullet A] \in TS_{III} \cup TS_{IV}$ such that $\ell_{\approx_f}(\sigma_i.a) = \textbf{pass}$. Since $a \notin out(M_i)$, $\sigma_i.a \notin Obs_{(\mathrm{T,M})}$. This is $Obs_{pass} \neq Pur(\mathrm{T})$. Therefore, M fails T.

References

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

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

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

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

[5] Brookes S. D. and et al. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[6] K. Drira, and et al. Testability of a communicating system through an environment. In *4th International Joint Conf. on Theory and Practice of Software Development*, 329–341, 1993.

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

[8] S. Fujiwara and G. v. Bochmann. Testing nonterministic finite state machine with fault coverage. In *IFIP 4th International Workshop on Protocol Test Systems*, 267–280, 1991.

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

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

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

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

[13] 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*, 91–106, 1994.

[14] R. Milner. *A Calculus of Communicating Systems (Lecture Notes in Computer Science, vol. 92)*. McGraw-Hill Computer Science Series, New York, 1980.

[15] D. H. Pitt and D. Freestone. The derivation of comformance tests from LOTOS specifications. *IEEE Trans. Softw. Eng.*, SE-16(12):1337–1343, 1990.

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

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

[18] J. Tretmans. Testing labelled transition systems with inputs and outputs. In *IFIP 8th International Workshop on Protocol Test Systems*, France, 1995.

[19] J. Tretmans, P. Kars, and E. Brinskma. Protocol conformance testing: A formal perspective on ISO IS-9646. In *IFIP 4th International Workshop on Protocol Test Systems*, 131-142, 1991.

[20] R. J. van Glabbeek. The linear time-branching time spectrum. *Lecture Notes on Computer Science (456)*, 278–297, 1990.

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

Proceedings of the 26th Annual International Symposium on

SESSION B5

Design

Fault-Tolerant Computing — 1996

Chair

Algirdas Avižienis
USA