Test Generation for Specifications Modeled by Input/Output Automata*

Q. M. Tan[†] and A. Petrenko[‡]

[†]Département d'IRO, Université de Montréal

C.P. 6128, Succ. Centre-Ville, Montréal, P.Q. H3C 3J7, Canada

[‡]CRIM, Centre de Recherche Informatique de Montréal

1081 McGill College Avenue, Montréal, P.Q. H3A 2N4, Canada

Abstract

In this paper, we consider conformance testing of communication systems modeled by I/O automata. A framework is proposed for testing I/O automata with full fault coverage for implementations with at most m states. The notion of state identification, which was originally defined in the realm of I/O finite state machines, is applied. Based on this notion, a test derivation algorithm is given for test suites which guarantee fault coverage. This algorithm is an analogue of the so-called FSM-based HSI-method.

Keywords

software testing, network protocols, conformance testing, test derivation, state identification, I/O automata

1 INTRODUCTION

Communication is a process in which a system interacts with its environment. In many real-life applications, communication is performed via *inputs* and *outputs*. For such a system, inputs are initialized by the environment and always enabled in the system for any interaction, while outputs are initialized by the system and cannot be blocked by the environment. In this context, when the system has its outputs and inputs at the same time, a choice mechanism is

*This work was supported by the NSERC grant OGPO194381

needed to decide which action among the outputs and inputs actions is executed; no communication can be blocked unless no outputs from the system and no inputs from the environment at all. Systems with such input/output interactions are called *I/O automata*.

The choice mechanism of I/O interactions can be implemented in a simple way such that a system and the environment are interacting in an alternative order. The alternative input/output interactions are usually assumed for so-called input/output finite state machines (FSMs). More general interactions with no distinction between inputs and outputs are called rendezvous. The rendezvous interactions are usually assumed for so-called labeled transition systems (LTSs).

The I/O automata is one of the important formalisms in the realm of conformance testing of communication systems, in particular, network protocols. There has been much work done on testing labeled transition systems [2, 9, 10] and finite state machines [3, 7, 4, 5]. However, for I/O automata, few studies [8, 6, 11, 12] have been done. It is argued that the input/output model is closer to the reality than the rendezvous model, while the alternative input/output model is only a limited case of I/O automata. Therefore, it is of both practical and theoretical interests to study conformance testing based on I/O automata.

A testing scenario for I/O automata was given in [8]. Conformance testing based on I/O automata was discussed in [6]. A test generation method was proposed in [11] for a quiescent trace preorder with unspecified behavior concerned. In [12], the refusal testing notion was further applied for test derivation. These existing methods usually use an exhaustive testing approach in order to prove the correctness of an implementation against its specification with respect to a particular conformance relation. Apparently, this approach is often impractical since it may involve a test suite of a big size or infinite behavior. Moreover, no fault coverage measure can be given for conformity of an implementation with its specification.

In this paper, we deal with test derivation for specifications modeled by I/O automata with guaranteed fault coverage for a well-defined fault model, which is defined as a set of mutants of a given specification with at most m states, where m is a known integer. I/O automata considered are transition-deterministic. We give in Section 2 the basic definitions and notations of I/O automata. In Section 3, a framework is proposed for testing I/O automata. In Section 4, a test derivation method, with the notion of state identification which was originally defined in the realm of FSMs, is given.

2 BASIC DEFINITIONS AND NOTATIONS

The starting point for conformance testing is a specification in some (formal) notation, an implementation given in the form of a black box, and a set of conformance requirements the implementation should satisfy. In this paper, I/O

notation	meaning
$(X \cup Y)^*$	set of sequences over $X \cup Y$; σ or $a_1 \dots a_n$ denotes such a sequence
$p - a_1 \dots a_n \rightarrow q$	there exist p_k , $1 \leq k < n$, such that $p - a_1 \rightarrow p_1 \dots p_{n-1} - a_n \rightarrow q$
$p - \sigma \rightarrow$	there exists q such that $p - \sigma \rightarrow q$
$p \not\rightarrow \sigma$	no q exists such that $p - \sigma \rightarrow q$
$out(p)$	$out(p) = \{a \in Y \mid p - a \rightarrow\}$
$p \text{ after } \sigma$	$p \text{ after } \sigma = \{q \in S \mid p - \sigma \rightarrow q\}$; $S \text{ after } \sigma = s_0 \text{ after } \sigma$
$Tr(p)$	$Tr(p) = \{\sigma \in (X \cup Y)^* \mid p - \sigma \rightarrow\}$; $Tr(S) = Tr(s_0)$

Table 1 Notations for I/O automata

automata are considered for specifications; implementations are also assumed to be described in the same model as its specification; the conformance requirements is supposed to be defined by a specific conformance relation—*trace equivalence*.

Definition 1 (*I/O automaton (IOA)*): An I/O automaton is a 4-tuple $\langle S, X, Y, \Delta, s_0 \rangle$, where:

- S is a finite non-empty set of states, $s_0 \in S$, is the initial state.
- X is a finite non-empty set of inputs.
- Y is a finite non-empty set of outputs.
- $\Delta \subseteq S \times (X \cup Y) \times S$ is a transition set, where $\forall p \in S$ and $\forall a \in X, \exists q \in S$ such that $(p, a, q) \in \Delta$.

An element $(p, a, q) \in \Delta$ is usually denoted by $p - a \rightarrow q$, and called a *transition*. If $a \in X$, it is called an *input transition*; otherwise $a \in Y$ and it is called an *output transition*. $p - a \rightarrow q$ also means that action a is enabled in state p . Note that I/O automata we give here are always input-enabled, i.e. completely specified.

An I/O automaton is said to be *transition-deterministic* if there exist no $p - a \rightarrow q_1, p - a \rightarrow q_2 \in \Delta$ where $q_1 \neq q_2$. An I/O automaton is also said to be *output-deterministic* if there exist no states enabling two or more different outputs, i.e. there are no $p - y_1 \rightarrow q_1, p - y_2 \rightarrow q_2 \in \Delta$ where $y_1, y_2 \in Y$ and $y_1 \neq y_2$. In this paper, we only address the transition-deterministic IOAs. Note that the transition-deterministic I/O automata can still be non-deterministic, because when there may be two or more outputs enabled in a state, which output is executed is not determined.

Given two sequences σ_1, σ_2 , we use $\sigma_1.\sigma_2$ to represent the concatenation of σ_1 and σ_2 . Given two sets of sequences V_1, V_2 , we use $V_1 @ V_2$ to represent the concatenation of V_1 and V_2 , that is, $V_1 @ V_2 = \{\sigma_1.\sigma_2 \mid \sigma_1 \in V_1 \wedge \sigma_2 \in V_2\}$. We also use $pref(\sigma)$ to denote a set of prefixes of σ and $pref(V)$ to denote a set of prefixes of all sequences in V . We also use V^m to represent the concatenation of set V of sequences with itself for m times. Formally, $V^m = V @ V^{m-1}$ and $V^0 = \{\varepsilon\}$.

Figure 1 An I/O automaton

Let $S = \langle S, X, Y, \Delta, s_0 \rangle$ be an I/O automaton and $p, q \in S$, the conventional notations shown in Table 1 are used in this paper. Furthermore for the convenience of the presentation, we always use $I, P, S, \dots$ to represent I/O automata; $I, P, Q, \dots$ for sets of states; $a, b, c, \dots$ for inputs or output; $x, x_1, \dots$ for inputs; $y, y_1, \dots$ for outputs; $i, p, q, s, \dots$ for states. The set $Tr(p)$ is called the set of *traces* for p .

An IOA can also be represented by a directed graph where nodes are states and labeled edges are transitions. An IOA graph, which describes a slide window of size 2, is shown in Figure 1. In this IOA, where $p1$ and $p2$ are the input actions, g is the output action. In the initial state s_0 , the two buffers in the window are empty, in which $p1$ puts a package with the sequence number i to the current buffer and $p2$ puts a package with the sequence number $i + 1$ to the next buffer. The arrival of packages might not be in order, but g must get a package from the window in turn. Once a package is removed from the current buffer, the buffer becomes the next empty buffer (i.e. the pointer j to the current buffer now is $mod_2(j + 1)$). In the state s_3 the window is full.

Definition 2 (*Trace equivalence*): The trace equivalence relation between two states i and s , written $i = s$, holds iff $Tr(i) = Tr(s)$.

Given two IOAs I and S , we say that I is trace-equivalent to S , written $I = S$, iff $i_0 = s_0$.

The trace equivalence relation requires that an implementation has the same traces of interactions as its specification.

3 CONFORMANCE TESTING

Conformance testing is a finite set of experiments, in which a set of test cases, derived from a specification according to the 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.

Definition 3 (*Test cases and test suite*): Given an IOA $S = \langle S, X, Y, \Delta, s_0 \rangle$, a test case for S is a 3-tuple $\langle \rho, \Omega, \ell \rangle$ where

- $\rho \in (X \cup Y)^* @ Y$;
- $\Omega = (pref(\rho) \setminus \{\rho\}) @ Y$ is a set of all possible observations while executing this test case.
- $\ell : \Omega \rightarrow \{\text{pass}, \text{fail}, \text{inconclusive}\}$ is an acceptance function.

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

Before a test case T , where $\rho = a_1.a_2 \dots a_n$, is applied, the IUT M is in the initial state m_0 . Starting with a_1 , a_i is applied in turn until a_n , if no exception occurs. Let currently M be in state p and meanwhile a_i be applied, the I/O interaction between T and M can be defined as shown in the following table.

I/O	Available Transitions	Execution	Test Process
$a_i \in X$	$p - a_i \rightarrow q$, no $p - y \rightarrow$	a_i	proceed
$a_i \in X$	$p - a_i \rightarrow q$, $p - y \rightarrow$	a_i	proceed
$a_i \in X$	$p - a_i \rightarrow q$, $p - y \rightarrow$	y	exception
$a_i \in Y$	$p - a_i \rightarrow q$	a_i	proceed
$a_i \in Y$	$p - y \rightarrow q$, ($y \neq a_i$)	y	exception
$a_i \in Y$	no $p - y \rightarrow q$		exception(deadlock)

Once an exception occurs at a_i , the test execution is terminated, and the exception function $\ell(a_i)$ tells the result of this test run. The deadlock is a specific exception, which may be detected by a time mechanism. A test case should be designed to be "sound", i.e. no deadlock or **fail** exception should happen if a test suite is applied to a valid implementation. A reliable reset also is assumed to exist always in any IUT, which brings the IUT from any state to the initial state such that a next test run can be executed.

In this paper, the implementations are modeled by I/O automata, which may not be output-deterministic. In order to test nondeterministic implementations, one usually makes a so-called *complete-testing assumption*: it is possible, by applying a given test case to the implementation a finite number of times, to exercise all possible execution paths of the implementation which are traversed by the test case. Without such an assumption, no test suite can guarantee full fault coverage (in terms of conformance relations) for nondeterministic implementations. Therefore, the test application should include several test runs and lead to a complete set of observations $Obs_{(T,M)} = \Omega \cap Tr(m_0)$.

Based on $Obs_{(T,M)}$, the success or failure of testing needs to be concluded. The way a verdict is drawn from $Obs_{(T,M)}$ is the *verdict assignment* for T . Intuitively, the success should mean that no unexpected behavior is observed and the test purpose has been achieved. If we define the *test purpose* of T , written $Pur(T)$, to be $Pur(T) = \{\sigma \in \Omega \mid \ell(\sigma) = \text{pass}\}$ then the conclusion can be drawn as follows.

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

M passes T if $Obs_{fail} = \emptyset \wedge Obs_{pass} = Pur(T)$

M fails T otherwise.

The goal of conformance testing is to gain confidence in an implementation under test concerning its conformance with the specification. Increased confidence is normally obtained through time and effort spent in testing the implementation, which, however, are limited by practical and economical considerations. In order to have a more precise measure of the effectiveness of testing, a fault model and fault coverage criteria [1] are introduced. Like in the FSM realm, we also take the mutation approach [1] to define the fault model in the IOA realm to be a set $\mathcal{F}(m)$ of all transition-deterministic IOAs with at most m states and the same alphabets. Based on $\mathcal{F}(m)$, a test suite with complete fault coverage for an IOA specification can be defined as follows.

Definition 5 (*Complete test suite*): Given an IOA S and a fault model $\mathcal{F}$, a test suite TS for S is said to be complete over $\mathcal{F}(m)$, if for any M in $\mathcal{F}(m)$, $M = S$ iff M passes T for each T in TS .

A complete test suite guarantees that for any implementation M in $\mathcal{F}(m)$, if M passes all test cases, it is a *conforming* or *valid* implementation of the given specification, and any faulty implementation in the fault model is detected by failing at least one test case.

In the context of trace equivalence, a conforming implementation should have the same traces as a specification. Therefore each test case specifies a sequence of inputs and outputs, which is either valid or invalid trace for the specification, to verify that an implementation has implemented the valid one and not the invalid one. If such a sequence is implemented in the implementation, then there must exist a test run such that the sequence is observed. If the sequence is a valid trace, a **pass** verdict should be assigned to this test run, which implies that 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 all the proper prefixes of the sequence should be labeled with **inconclusive**. On the other hand, if the sequence is an invalid trace, a **fail** verdict should be assigned to this test run, which implies the sequence in the test case should be labeled with **fail**. Based on this reasoning, we conclude that all test cases must be of the following form:

Definition 6 (*Acceptance function*): Given an IOA specification S and $T = \langle \rho, \Omega, \ell \rangle$, the acceptance function $\ell(\sigma)$, $\sigma \in \Omega$, satisfies:

$$\ell(\sigma) = \begin{cases} \text{pass} & (\sigma \in Tr(s_0) \cap pref(\rho)) \wedge \\ & (\sigma.a \in pref(\rho) \rightarrow \sigma.a \notin Tr(s_0)) \\ \text{fail} & \sigma \notin Tr(s_0) \\ \text{inconclusive} & \text{otherwise.} \end{cases}$$

Figure 2 Test case for I/O sequence $p1.p2.g.g$

Obviously, if $\ell(\sigma) = \mathbf{fail}$, all the sequences in Ω which have σ as a prefix are labeled with **fail**. On the other hand, if $\ell(\sigma) = \mathbf{pass}$, all the sequences in Ω which are prefixes of σ are labeled with **inconclusive** and all the sequences in Ω which have σ as a proper prefix cannot be labeled with **pass** any more.

Theorem 1 *Given a test case T for IOA S , for any IOA I , if $I = S$, then I passes T .*

A test case with I/O sequence $p1.p2.g.g$ for the IOA given in Figure 1 is shown in Figure 2. Note that in the figure Ω is represented as a tree.

4 TEST GENERATION

We intend to apply the notion of state identification, which was originally defined in the realm of FSMs, for the IOA-based test derivation. In this section, we discuss how the notion is adapted to the realm of IOAs. Based on this, a test derivation method with full fault coverage is presented for an IOA specification.

4.1 State Identification in Specifications

Similar to the case of FSMs, in order to identify states in a given IOA specification, first the specification is required to have certain testability properties, one of which is the so-called reducibility.

Definition 7 (*Distinguishable states of an IOA*): Two states m and s of an IOA are said to be distinguishable if there exists $\sigma \in (X \cup Y)^*$, such that $m' = m \text{ after } \sigma$, $s' = s \text{ after } \sigma$ and $out(m') \neq out(s')$.

Obviously, two distinguishable states are not trace equivalent. This means that for two distinguishable states, there exists a sequence of inputs and outputs, such that it is a trace for one of the two states, but it is not a trace for the other.

Definition 8 (*Reduced IOA*): An IOA is said to *reduced* if all its states are pairwise distinguishable.

Here we assume that the given IOA specification S is in the reduced form and has n states $s_0, s_1, \dots, s_{n-1}$.

There are several state identification notions that may be borrowed from the FSM-based testing. Here we only present the so-called *harmonized state identifiers* [5] for the IOA-based testing. The other notions of state identification could be defined similarly, as we did for test derivation based on labeled transition systems in [9].

Definition 9 (*harmonized state identifiers (HSI)*): $\langle H_0, H_1, \dots, H_{n-1} \rangle$ is said to be a set of harmonized state identifiers for an IOA S , if

- $H_i \subseteq (X \cup Y)^* @ Y$ for $i = 0, 1, \dots, n-1$;
- For each pair of different states $s_i \neq s_j$, there exists $\sigma \in \text{pref}(H_i) \cap \text{pref}(H_j)$ such that $\sigma \in \text{Tr}(s_i) \oplus \text{Tr}(s_j)$, where the operator $\oplus$ is denoted to be $A \oplus B = (A \setminus B) \cup (B \setminus A)$.

H_i is said to be a harmonized state identifier for state s_i . The harmonized state identifier for s_i captures the following property: for any other state s_j , there exists a sequence σ_i in $\text{Pref}(H_i)$ that distinguishes s_i from s_j and σ_i is also in $\text{Pref}(H_j)$.

Harmonized state identifiers always exist for any reduced IOA. As an example, for the IOA in Figure 1, we can choose the harmonized state identifiers $H_0 = \{g, p1.g.g\}$, $H_1 = \{g.g\}$, $H_2 = \{g, p1.g.g\}$, $H_3 = \{g.g\}$. Considering H_0 : g is used to distinguish s_0 from s_1 and s_3 , so g is also in $\text{pref}(H_1)$ and $\text{pref}(H_3)$; $p1.g.g$ is used to distinguish s_0 from s_2 , so H_2 has $p1.g.g$.

4.2 State identification in implementations

Given an IOA specification S with n states and an IOA implementation I with m states, similar to the FSM-based testing, there are the two phases for the IOA-based testing. In the first phase, the state identification facility is applied to I to check if it can also properly identify the states in I . Once I passes the first phase, we can in the second phase test whether each transition and its tail state are correctly implemented.

In order to perform the first testing phase, proper transfer sequences are needed to bring I from the initial state to those particular states in I 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 I . Since a reliable reset is assumed, we can guarantee this in the following way: after a sequence in H_i is applied, the implementation I is reset to the initial state, and brought to 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 sequence σ such that $s_0 - \sigma \rightarrow s_i$, we can use $\langle N_0, N_1, \dots, N_{n-1} \rangle$ to cover all states of I , where

$$N_i = \{\sigma \in Q @ ((X \cup Y)^0 \cup (X \cup Y)^1 \cup \dots \cup (X \cup Y)^{m-n}) \mid s_0 - \sigma \rightarrow s_i\} \quad (1)$$

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

$$TS_1 = \bigcup_{i=0}^n N_i @ H_i \quad (2)$$

Using TS_1 , we can make test cases for the IOA S for the first testing phase according to Definitions 3 and 6. We call any I/O sequence that is used to make a test case for an IOA a *test sequence*.

After l passes all test cases made by TS_1 , the states of l are partitioned into n groups, each of which is mapped by a state of S , written $f(s_k)$, $0 \leq k \leq n-1$. For all states i_j in the same group $f(s_k)$, if $\sigma \in H_k$ then $\sigma \in Tr(i_j)$, and for two states $i_k \in f(s_k)$ and $i_j \in f(s_j)$, there exists $\sigma \in H_i$ such that $\sigma \in Tr(i_i) \oplus Tr(i_j)$.

In the second phase of testing, for testing a given transition $s_i - a \rightarrow s_j$ in S , it is necessary to first bring l into each state $i_k \in f(s_i)$, then apply a at this state and see if a can be executed; moreover, let l be in i_l after a is executed, it is necessary to check that $i_l \in f(s_j)$ which should be verified by H_j . If any undefined output transition out of s_i has been implemented in l , an exception might occur, i.e. an unexpected output is observed. If a defined output transition out of s_i has not been implemented in l , a deadlock exception might occur. In the case of any **inconclusive** exception, testing is repeated.

Obviously, N_i may be used to bring l to any state $i_k \in f(s_i)$. Using this state cover, we can obtain a transition cover $\langle E_0, E_1, \dots, E_{n-1} \rangle$ for the implementation, where

$$E_i = \{ \sigma \in \bigcup_{k=0}^{n-1} (N_k @ (X \cup Y)) \mid s_0 - \sigma \rightarrow s_i \} \quad (3)$$

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

$$TS_2 = \bigcup_{i=0}^{n-1} (E_i \setminus N_i) @ H_i \quad (4)$$

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

$$TS' = TS_1 \cup TS_2 = \left(\bigcup_{i=0}^{n-1} N_i @ H_i \right) \cup \left(\bigcup_{i=0}^{n-1} (E_i \setminus N_i) @ H_i \right) = \bigcup_{i=0}^{n-1} E_i @ H_i \quad (5)$$

We have seen that the above process of checking experiments for IOAs is an analogue to the checking experiments for FSMs [10]. It is expected that a test suite which is derived from the IOA S based on the above process

is complete for the fault model $\mathcal{F}(m)$. Below we formally present the test derivation algorithm.

4.3 Test selection algorithm

Following the testing process given in the above subsection, we have the following test generation algorithm.

Algorithm 1 *Test Generation Algorithm:*

Input: An IOA S with n states in the reduced form and an upper bound m on the number of the states of IOA implementations. ($n \leq m$.)

Output: A test suite TS .

Step 1: Find a tuple of harmonized state identifiers $\{H_0, H_1, \dots, H_{n-1}\}$ from S .

Step 2: Construct a minimal set $Q \subseteq (X \cup Y)^*$ such that $\forall s_i \in S \exists \sigma \in V (s_0 - \sigma \rightarrow s_i)$.

Step 3: Construct the sets $\langle E_0, E_1, \dots, E_{n-1} \rangle$ such that $E_i = \{\sigma \in Q @ ((X \cup Y)^0 \cup (X \cup Y)^1 \cup \dots \cup (X \cup Y)^{m-n+1}) \mid s_0 - \sigma \rightarrow s_i\}$.

Step 4: Construct the set of test sequences

$$TS' = \bigcup_{i=0}^{n-1} E_i @ H_i$$

Step 5: Construct a test case T for each $\rho \in TS'$:

- compute Ω according to Definition 3,
- label each $\sigma \in \Omega$ according to Definition 6.

All the resulting test cases constitute the test suite TS .

The above method is an analogue of the FSM-based HSI-method [5]. The following theorem guarantees that a test suite obtained with the above method has complete fault coverage for the fault model $\mathcal{F}(m)$.

Theorem 2 *Given an IOA specification S in the reduced form and the fault model $\mathcal{F}(m)$, a test suite obtained with Algorithm 1 is complete for S .*

If we do not consider the optimization of state cover and state identification, for given n and m , the complexity of this algorithm is effective. The size of a test suite, i.e. the sum of the lengths of all I/O sequences obtained with the above test generation algorithm, is less than $n^3 |X \cup Y|^{m-n+1}$.

For the other alternatives of state identification, such as W -set [3, 4] and UIO-sequences [7], the corresponding test generation methods could be devised similarly for IOAs. In [9, 10], we presented a list of the test generation methods based on the different notions of state identification for labeled transition systems.

4.4 Example

We use the IOA in Figure 1 which describes a slide window specification of size 2 as an example to the application of the test generation algorithm.

Figure 3 Test suite for a slide window specification

We assume the fault model to be $\mathcal{F}(4)$, i.e. implementations considered are transition-deterministic IOAs with at most 4 states and with the alphabet $X = \{p1, p2\}$ and $Y = \{g\}$. This IOA specification is reduced, and it has HSI identifiers

$$H_0 = \{g, p1.g.g\}, H_1 = \{g.g\}, H_2 = \{g, p1.g.g\}, H_3 = \{g.g\}.$$

If we chose $Q = \{\varepsilon, p1, p2, p1.p2\}$ as a state cover for the specification, then we have

$$\begin{aligned} E_0 &= \{\varepsilon, p1.g\}, & E_1 &= \{p1, p1.p1, p1.p2.g\} \\ E_2 &= \{p2, p2.p2\}, & E_3 &= \{p1.p2, p2.p1, p1.p2.p1, p1.p2.p2\} \end{aligned}$$

and further a set of test sequences

$$TS' = \{g, p1.g.g, p1.g.p1.g.g, p1.p1.g.g, p1.p2.g.g.g, p2.g, p2.p1.g.g, p2.p2.g, p2.p2.p1.g.g, p1.p2.p1.g.g, p1.p2.p2.g.g\}.$$

Note that in TS' we remove those sequences which are prefixes of the others.

Using TS' , we can constitute a test suite for the slide window specification, as shown in Figure 3. This test suite is complete for $\mathcal{F}(4)$.

4.5 Test optimization

From the example in the above subsection, we can see that in the resulting test suite, even though we have removed those test sequences which are prefixes of other test sequences, there are still some test cases which are implied by others. For example, the test case with the test sequence g is implied by any

other test cases, because its possible observation set Ω is contained in the possible observation set Ω' of another test case. The following algorithm is used to remove the test cases which are implied by other test cases.

Algorithm 2 *Test Optimization Algorithm*

Input: A test suite TS for an IOA specification.

Output: A optimized test suite.

Step1 For each test case $T \in TS$, let ρ be its test sequence, if there exists another test case TS' with the possible observation set Ω' such that $\rho \in \Omega'$, remove T from TS .

Step2 Return the resulting TS as the output.

The following theorem guarantees that, if the given test suite is complete, then the test suite obtained with Algorithm 2 is also complete.

Theorem 3 *Given a test suite TS for an IOA specification S , if TS is complete then the test suite obtained with Algorithm 2 is also complete for S .*

For the test suite derived in the example in the above subsection, after the test optimization algorithm is applied, the resulting test suite has the set of test sequences:

$$TS' = \{p1.g.p1.g.g, p1.p1.g.g, p1.p2.g.g.g, p2.p1.g.g, p2.p2.p1.g.g, p1.p2.p1.g.g, p1.p2.p2.g.g\}.$$

The four test cases with the test sequences $g, p1.g.g, p2.g$ and $p2.p2.g$ are removed by the algorithm.

5 CONCLUSION

This paper deals with conformance testing of communication systems which are specified in the formalism of I/O automata. The I/O automaton model is a more powerful model than the I/O finite state machine model (FSM), which is extensively used in conformance testing of communication systems. A framework has been introduced for testing transition-deterministic I/O automata for trace equivalence. In this framework, test cases, test execution, verdict assignment and fault model are defined. The fault model is a set of mutants of a given specification with at most m states, where m is a known integer. The notion of state identification, called harmonized state identifiers (HSI), which was originally defined in FSMs, has been introduced for test derivation for I/O automata. Based on the state identification technique, a test generation method with complete fault coverage, which is an analogue of the FSM-based HSI-method, has been presented.

Although we only introduced one notion of state identification, the other notions of state identification, such as a W-set and UIO-sequences, could also be

defined similarly, and thus a set of useful and competing test derivation methods with fault coverage could be developed for testing specifications modeled by I/O automata.

6 REFERENCES

- [1] G. v. Bochmann and A. Petrenko. (1994) Protocol testing: Review of methods and relevance for software testing. In *ACM 1994 Intern. Symp. on Software Testing and Analysis*, 109–124.
- [2] E. Brinksma. (1988) A theory for the derivation of tests. In *IFIP PSTV VIII*, 63–74.
- [3] T. S. Chow. (1978) Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187.
- [4] S. Fujiwara et al. (1991) Test selection based on finite state models. *IEEE Transactions on Software Engineering*, SE-17(6):591–603.
- [5] A. Petrenko. (1991) Checking experiments with protocol machines. In *IFIP 4th IWPTS*, 83–94.
- [6] M. Phalippou. (1993) Executable testers. In *IFIP 6th IWPTS*, 35–50.
- [7] K. Sabnani and A. T. Dahbura. (1988) A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285–297.
- [8] R. Segala. (1993) Quiescence, fairness, testing, and the notion of implementation. In *CONCUR'93*.
- [9] Q. M. Tan, A. Petrenko, and G. v. Bochmann. (1997) Checking experiments with labeled transition systems for trace equivalence. In *IFIP 10th IWTCS*.
- [10] Q. M. Tan. (1998) On conformance testing of systems communicating by rendezvous. Ph.D. thesis, University of Montreal, Canada.
- [11] J. Tretmans. (1995) Testing labelled transition systems with inputs and outputs. In *IFIP 8th IWPTS*.
- [12] L. Heerink, and J. Tretmans. (1997) Refusal testing for classes of transition systems with inputs and outputs. In *FORTE X/PSTV XVI*.

7 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, PQ, Canada for the Ph.D. degree in conformance testing on communication protocols. From 1984 to 1992, he was a lecturer in the Department of Computer Science of Chongqing University. Now he is also working with Claremont Technology Inc. Canada.

Alexandre Petrenko 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.

APPENDIX

We here only give the proof of Theorem 2. Given an IOA specification S and an IOA implementation M , we assume the following:

- (1) All states of S and M are reachable from the initial state s_0 and m_0 , respectively.
- (2) $\bar{S}$ is the reduced form of S and has at most n states with $n > 1$.
- (3) $\bar{M}$ is the reduced form of M and has at most m states with $m \geq n$, i.e. $\bar{M} \in \mathcal{F}(m)$.
- (4) $\bar{s}_i, \bar{s}_j, \bar{s}_k, \bar{s}_l$ and $\bar{m}_i, \bar{m}_j, \bar{m}_k, \bar{m}_l$ represent the states of $\bar{S}$ and $\bar{M}$, respectively.
- (5) A tuple of harmonized state identifiers $\{H_0, H_1, \dots, H_{n-1}\}$.
- (6) Q is a state cover for $\bar{S}$.
- (7) TS is a test suite obtained with Algorithm 1 and TS' is the set of test sequences for TS .

Definition 10 *V-equivalence*. Given a set $V \subseteq \Sigma^*$, the V -equivalence relation between two states p and q , written $p =_V q$, holds if and only if for all $\sigma \in \text{pref}(V)$, $\sigma \in \text{Tr}(p) \Leftrightarrow \sigma \in \text{Tr}(q)$.

Given two IOAs S and M with initial states s_0 and m_0 respectively, we say that M is V -equivalent to S , written $S =_V M$, if only if $s_0 =_V m_0$.

notation	meaning
$[\bar{s}_i, \bar{m}_i] - a \rightarrow [\bar{s}_j, \bar{m}_j]$	For $a \in \Sigma$, $\bar{s}_i - a \rightarrow \bar{s}_j$ and $\bar{m}_i - a \rightarrow \bar{m}_j$
$[\bar{s}_i, \bar{m}_i] - \sigma \rightarrow [\bar{s}_j, \bar{m}_j]$	For $\sigma \in \Sigma^*$, $\bar{s}_i - \sigma \rightarrow \bar{s}_j$ and $\bar{m}_i - \sigma \rightarrow \bar{m}_j$
$[\bar{s}_i, \bar{m}_i]$ after V	given a pair of states $[\bar{s}_i, \bar{m}_i] \in \bar{S} \times \bar{M}$, and a set $V \subseteq \Sigma^*$ $[\bar{s}_i, \bar{m}_i]$ after $V = \{[\bar{s}_j, \bar{m}_j] \mid \exists \sigma \in \text{pref}(V) ([\bar{s}_i, \bar{m}_i] - \sigma \rightarrow [\bar{s}_j, \bar{m}_j])\}$
D	$D = [\bar{s}_0, \bar{m}_0]$ after Σ^*
D_r	$D_r = \{[\bar{s}_i, \bar{m}_j] \in D \mid \bar{s}_i = H_i, \bar{m}_j\}$
$\bar{\Sigma}^k$	$\bar{\Sigma}^k = \bigcup_{i=0}^k \Sigma^i$

Lemma 1 For $V \subseteq \Sigma^*$, assume $|[\bar{s}_0, \bar{m}_0] \text{ after } V| \geq k$. If $|D| > k$, then

$|\bar{s}_0, \bar{m}_0| \text{ after } V.\bar{\Sigma}^1| \geq k+1$; if $|D| \leq k$, then $|\bar{s}_0, \bar{m}_0| \text{ after } V@ \bar{\Sigma}^1 = |\bar{s}_0, \bar{m}_0| \text{ after } V$.

Proof:

(I) To prove that the lemma holds when $|D| > k$.

The lemma holds when $|\bar{s}_0, \bar{m}_0| \text{ after } V| > k$. Consider $|\bar{s}_0, \bar{m}_0| \text{ after } V| = k$.

- (1) $|D| > k$ and $|\bar{s}_0, \bar{m}_0| \text{ after } V| = k$ hypothesis
- (2) $|\bar{s}_0, \bar{m}_0| \text{ after } V \subseteq D$ definition of D
- (3) $\exists [\bar{s}_k, \bar{m}_k] \in D \setminus |\bar{s}_0, \bar{m}_0| \text{ after } V$ (1),(2)
- $\exists [\bar{s}_i, \bar{m}_i] \in |\bar{s}_0, \bar{m}_0| \text{ after } V$ (1)
- $\exists \sigma \in \text{pref}(V) \exists \sigma.a \in \Sigma^*$
- $([\bar{s}_0, \bar{m}_0] - \sigma \rightarrow [\bar{s}_i, \bar{m}_i] - a \rightarrow [\bar{s}_k, \bar{m}_k])$ (2)
- (4) $[\bar{s}_k, \bar{m}_k] \in |\bar{s}_0, \bar{m}_0| \text{ after } V@ \bar{\Sigma}^1 \setminus |\bar{s}_0, \bar{m}_0| \text{ after } V$ (3)
- (5) $|\bar{s}_0, \bar{m}_0| \text{ after } V@ \bar{\Sigma}^1 \geq k+1$ (4).

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

- (1) $|D| \leq k$ and $|\bar{s}_0, \bar{m}_0| \text{ after } V| = k$ hypothesis
- (2) $|\bar{s}_0, \bar{m}_0| \text{ after } V \subseteq D$ definition of D
- (3) $|\bar{s}_0, \bar{m}_0| \text{ after } V@ \bar{\Sigma}^1 = |\bar{s}_0, \bar{m}_0| \text{ after } V$ (1),(2).

Lemma 2 Assume $\bar{s}_0 =_Q \bar{m}_0$. If $|D| > m$, then $|\bar{s}_0, \bar{m}_0| \text{ after } Q@ \bar{\Sigma}^{m-n}| \geq m$; and if $|D| \leq m$, then $|\bar{s}_0, \bar{m}_0| \text{ after } Q@ \bar{\Sigma}^{m-n} = D$.

Proof:

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

- (1) $\bar{s}_0 =_Q \bar{m}_0$ and $|D| > m$ hypothesis
- (2) $|\bar{s}_0, \bar{m}_0| \text{ after } Q| \geq n$ initially connected $\bar{S}$, (1)
- (3) $|\bar{s}_0, \bar{m}_0| \text{ after } Q@ \bar{\Sigma}^{m-n}| \geq m$ Lemma 1, (1),(2).

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

Lemma 3 If $\bar{s}_i =_{H_i} \bar{m}_k$ and $\bar{s}_j =_{H_j} \bar{m}_k$, then $i = j$.

Proof:

- (1) For $V \subseteq \Sigma^*$, $\bar{s}_i =_V \bar{m}_k \Leftrightarrow \bar{s}_i =_{\text{pref}(V)} \bar{m}_k$ evident
- (2) $\bar{s}_i =_{H_i} \bar{m}_k$ and $\bar{s}_j =_{H_j} \bar{m}_k$ hypothesis
- (3) $\bar{s}_i =_{\text{pref}(H_i)} \bar{m}_k$ and $\bar{s}_j =_{\text{pref}(H_j)} \bar{m}_k$ (1),(2)
- (4) $i \neq j$ assumption
- (5) $\exists \sigma \in (Tr(\bar{s}_i) \oplus Tr(\bar{s}_j)) \cap \text{pref}(H_i) \cap \text{pref}(H_j)$ definition of H_i , (4)
- (6) let $\sigma \in Tr(\bar{s}_i)$, then $\sigma \in Tr(\bar{m}_k)$ (3)
- (7) $\sigma \in Tr(\bar{s}_j)$ (3),(6)
- (8) $i = j$ (6),(7) $\nRightarrow Tr(\bar{s}_i) \oplus Tr(\bar{s}_j)$.

Lemma 4 $|D_r| \leq m$.

Proof:

- (1) $|\overline{M}| \leq m$ hypothesis
- (2) $|D_r| > m$ assumption
- (3) $\exists [\overline{s}_i, \overline{m}_k], [\overline{s}_j, \overline{m}_k] (i \neq j, \overline{s}_i =_{H_i} \overline{m}_k \wedge \overline{s}_j =_{H_j} \overline{m}_k)$ (1),(2)
- (4) $|D_r| \leq m$ (3) $\nrightarrow$ Lemma 3
(definition of H_i).

Lemma 5 *If $\overline{s}_0 =_{TS'} \overline{m}_0$, 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 $|D| \leq m$.

- (1) $|D| \leq m$ hypothesis
- (2) $\overline{s}_0 =_{TS'} \overline{m}_0$ hypothesis
- (3) $\overline{s}_0 =_Q \overline{m}_0$ (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 $Q @ \overline{\Sigma}^{m-n} (\overline{s}_i =_{H_i} \overline{m}_j)$ (2)
- (6) $D = D_r$ (4),(5), definition of D_r .

(II) To prove that the lemma holds when $|D| > m$.

- (1) $|D| > m$ assumption
- (2) $\overline{s}_0 =_{TS'} \overline{m}_0$ 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 $Q @ \overline{\Sigma}^{m-n+1}$
 $(\overline{s}_i =_{H_i} \overline{m}_j)$ (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) $||[\overline{s}_0, \overline{m}_0]$ after $Q @ \overline{\Sigma}^{m-n+1}|| \geq m + 1$ (1),(2), Lemma 2, Lemma 1
- (7) $|D_r| \geq m + 1$ (3),(4)
- (8) $|D| \leq m$ (5) $\nrightarrow$ Lemma 4
- (9) $[\overline{s}_0, \overline{m}_0]$ after $Q @ \overline{\Sigma}^{m-n} = D_r = D$ (6), Lemma 2.

Lemma 6 *If $\overline{m}_0 =_{TS'} \overline{s}_0$, then $\overline{m}_0 =_{Tr(\overline{s}_0)} \overline{s}_0$.*

Proof:

- (1) $\overline{m}_0 =_{TS'} \overline{s}_0$ 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) $\overline{m}_i =_{out(\overline{s}_i)} \overline{s}_i$ (1)
- (4) $not(\overline{m}_0 =_{Tr(\overline{s}_0)} \overline{s}_0)$ assumption
- (5) $\exists [\overline{s}_i, \overline{m}_i] \in D \exists y \in out(\overline{s}_i) not(\overline{m}_i =_{\{y\}} \overline{s}_i)$ (4)
- (6) $\overline{m}_0 =_{Tr(\overline{s}_0)} \overline{s}_0$ (5) $\nrightarrow$ (3).

Lemma 7 *If $\overline{m}_0 =_{TS'} \overline{s}_0$ and $\overline{m}_0 \neq \overline{s}_0$, then there exist $T \in TS$ and $\sigma \in \Omega$ such that $\ell(\sigma) = \text{fail}$ and $\sigma \in Tr(\overline{m}_0)$.*

Proof:

- (1) $\overline{s}_0 =_{TS'} \overline{m}_0$ 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) $\overline{m}_i =_{out(\overline{s}_i)} \overline{s}_i$ (1)

- (4) $\bar{m}_0 \neq \bar{s}_0$ hypothesis
- (5) $\exists y \in out(\bar{m}_i) \setminus out(\bar{s}_i) \exists [\bar{s}_i, \bar{m}_i] \in D \exists \sigma' \in prep(TS')$
 $([\bar{s}_0, \bar{m}_0] - \sigma' \rightarrow [\bar{s}_i, \bar{m}_i] \wedge \sigma'.y \in Tr(\bar{m}_0))$ (1),(3),(4)
- (6) $\exists T \in TS (\sigma' \in \rho \wedge \sigma'.y \in \Omega)$ (5), Definition 3
- (7) $\ell(\sigma'.y) = \mathbf{fail}$ (6), Definition 6

Lemma 8 $\bar{M}$ passes TS iff $\bar{m}_0 = \bar{s}_0$.

Proof:

- (1) $\bar{m}_0 = \bar{s}_0 \implies \bar{M}$ passes TS Theorem 1
- (2) $\bar{M}$ passes TS hypothesis
- (3) $\bar{m}_0 \neq \bar{s}_0$ assumption
- (4) 1) $not(\bar{m}_0 =_{TS'} \bar{s}_0)$ or 2) $\bar{m}_0 =_{TS'} \bar{s}_0 \wedge \bar{m}_0 \neq \bar{s}_0$ (3)
- (5) $not(\bar{m}_0 =_{TS'} \bar{s}_0)$ (4) 1)
- (6) $\exists \sigma \in TS' (\sigma \in Tr(\bar{m}_0) \setminus Tr(\bar{s}_0) \text{ or } \sigma \in Tr(\bar{s}_0) \setminus Tr(\bar{m}_0))$ Lemma 6
- (7) $\exists T \in TS (\sigma = \rho)$ T made by σ
- (8) $\ell(\sigma) = \mathbf{fail} \wedge \sigma \in Obs_{(T, \bar{M})}$ or $\ell(\sigma) = \mathbf{pass} \wedge \sigma \notin Obs_{(T, \bar{M})}$ Definition 6, (6)
- (9) $\bar{m}_0 =_{TS'} \bar{s}_0 \wedge \bar{m}_0 \neq \bar{s}_0$ (4) 2)
- (10) $\exists T \in TS \wedge \sigma \in \Omega (\ell(\sigma) = \mathbf{fail} \wedge \sigma \in Obs_{(T, \bar{M})})$ Lemma 7
- (11) $\bar{M}$ fails T , i.e. $\bar{M}$ fails TS Definition 4,(8),(10)
- (12) $\bar{m}_0 = \bar{s}_0$ (11) $\nRightarrow$ (2).

Theorem 2 M passes TS iff $m_0 = s_0$.

Proof:

- (1) $s_0 = \bar{s}_0, m_0 = \bar{m}_0$ reduced forms of S and M
- (2) $\bar{M}$ passes TS iff $\bar{m}_0 = \bar{s}_0$ Lemma 8
- (3) M passes TS iff $\bar{M}$ passes TS (1)
- (4) $m_0 = s_0$ iff $\bar{m}_0 = \bar{s}_0$ (1)
- (5) M passes TS iff $m_0 = s_0$ (3),(4).