Conformance Testing of Protocol Machines without Reset

by M. Yao, A. Petrenko and G.v. Bochmann

Publication # 861

Département d'informatique et de recherche opérationnelle

Université de Montréal

Fevrier 1993

Conformance Testing of Protocol Machines without Reset*

Mingyu Yao, Alexandre Petrenko** and Gregor v. Bochmann

Département d'informatique et de recherche opérationnelle
Université de Montréal, Montréal, Québec, Canada H3C 3J7

Abstract

In a number of test generation methods for conformance testing of communication protocols modeled by finite state machines, the *reliable reset* function has been assumed to be available in an implementation to be tested. In practice, however, the reliable reset may sometimes be difficult to realize and therefore this kind of test generation methods cannot be employed. In this paper, we propose an approach to the generation of test cases from protocol machines which can be specified by finite state machines possessing at least one Unique Input/Output sequence for each state. Our approach has been developed without the reliable reset assumption and it guarantees full fault coverage.

1. Introduction

* This research was supported by a grant from the Canadian Institute for Telecommunications Resaerch under the NCE program of the Government of Canada.

** On leave from the Institute of Electronics and Computer Science, Riga, Latvia.

In the recent years, the research for methods for test generation from *finite state machines* (FSMs) has received much attention in relation with conformance testing of communication protocols. This problem can be briefly stated in the following way.

A protocol specification is assumed to be available in the form of a FSM, while an implementation of the protocol is supposed to be a black-box (which theoretically can also be modeled by a FSM). A black-box implementation has some input and output ports through which one can give inputs (stimuli) to and observe corresponding outputs from the implementation, respectively. Thus, to decide whether an implementation is working properly with respect to the specification, one needs to derive, from the given FSM specification, one or more sequences of inputs (called test cases) which, when applied to the implementation under test (IUT), will enable us to infer solely from the observed outputs if the IUT is working as desired.

Currently, a number of methods have been proposed for the generation of test cases from finite state machine specifications. They have been developed in the context of hardware testing, software testing and conformance testing of communication protocols. These methods can be classified into two classes. One class of methods, including the W-method [Chow78, Vasi73], Wp-method [Fuji91], UIO-method [SaDa88], UIOv-method [Vuon89], CSP-method [Vuon90], Harmonized State Identification (HSI) method [Petr91] and Fault Function (FF) based method [Petr92], assume that the "reset" function is correctly implemented in an implementation under test. We call such a method a "*multiple testing approach*", since by using it, one usually generates from a given FSM specification a set of test cases, together forming a so-called a *test suite*. Each test case in the test suite is prefixed by the special "reset" symbol which guarantees that the test case is applied to the initial state of the IUT. This reliable reset assumption greatly simplifies the test generation problem and relatively efficient test suites can be obtained (in terms of the total length of the test cases in a test suite). However, as already pointed out by some researchers ([Fuji91], for instance), the reliable reset may in some cases be difficult to realize and therefore the multiple testing approaches cannot be applied.

The other class of methods, such as the T-method [Nait81], the DS-method [Gone70] and the so-called "optimization techniques" based on Unique Input/Output (UIO) sequences [Aho88, Shen89, Yang90 and Zhan92 etc.], do not assume the reliable reset function to be available in an IUT. We call such a method a "*single testing approach*", because by using such a method, one generates a single test case from a given FSM specification. The attractiveness of the single testing approach is that it is more applicable than a multiple testing approach because of the fact that the reliable reset is not required, although the length of a single test case generated by a

single testing approach could in some cases be much longer than the total length of the corresponding multiple test cases generated by a multiple testing approach.

Throughout this paper, our interest lies in the single testing approach to the test case generation problem. We will study in particular the case where a protocol specification is given as a FSM which has a UIO sequence for each of its states. In practice, most protocol machines do have short UIOs for its states [SaDa88], and therefore the class of FSMs which have UIO sequences has received much attention for test case generation. We will develop a new single testing approach for this class of FSMs.

The rest of the paper is organized as follows. In Section 2, we introduce some basic definitions which are mainly related to automata theory and are useful for the development of our single testing approach. A testing framework, within which our test case generation method will be presented, is defined in Section 3. Section 4 gives a brief review of the UIO based test case generation methods and a fault coverage problem related to the so-called optimization techniques is also pointed out there. Our single testing approach is then presented in Section 5. We will also briefly address the problem of incremental test case generation in Section 6. Finally in Section 7, we conclude the paper by discussing certain related issues.

2. Preliminaries

In this paper, the term "*finite state machine*" is used to specifically denote the model defined below.

Definition 2.1 (finite state machine)
A finite state machine is a 7-tuple $<S, X_s, Y_s, S_1, \delta_s, \lambda_s, D_s>$

- S: a set of n states $\{S_1, S_2, ..., S_n\}$ with S_1 as the initial state;
- X_s: a finite set of input symbols;
- Y_s: a finite set of output symbols;
- D_s: a specification domain which is a subset of $S \times X$
- δ_s: a transfer function δ_s: D_s --> S;
- λ_s: an output function λ_s: D_s --> Y. □

A FSM is called *completely specified*, iff $D_s = S \times X_s$. Otherwise it is called *partially* or *incompletely specified*. Since δ_s and λ_s are required to be functions, this FSM model is *deterministic*. That is, for each $(S_i, x) \in D_s$, there should be exactly one state $S_j \in S$ and exactly one output symbol $y \in Y_s$ such that $\delta_s(S_i, x) = S_j$ and $\lambda_s(S_i, x) = y$. In this case, we say there is a transition from state S_i to S_j with input x and output y. Such a transition is usually written as S_i -x/y-> S_j. A FSM can be given in a graph form, with the states and transitions of the FSM

represented by the vertices and arcs of the graph, respectively. An example of a graph representation of a FSM is given in Figure 1.

Let X* denote the set of all words constructed using an alphabet X. We use "ε" to represent the empty word. We also use, throughout this paper, the symbol "Θ" to represent the concatenation operation of two words in X*. However, this symbol is often simplified as the dot "·" or even omitted when no ambiguity arises.

We first introduce the notion of a "*defined input sequence*" for a state.

Definition 2.2 (defined input sequence)
Let $p = x_1x_2\cdots x_k \in X_s^*$, p is called a defined input sequence for state $S_i \in S$, if there exist k states $S_{i1}, S_{i2}, ..., S_{ik} \in S$ and an output sequence $q = y_1y_2...y_k \in Y_s^*$, such that there is a sequence of transitions

$$S_i \text{ -}x_1/y_1\text{->} S_{i1} \text{ -}x_2/y_2\text{->} S_{i2} \text{ --> ... --> } S_{ik-1} \text{ -}x_k/y_k\text{->} S_{ik} \qquad \textbf{(2-1)}$$

in the finite state machine. We use $\psi(S_i)$ to denote the set of all the defined input sequences for state S_i. □

A sequence of transitions such as (2-1) can be abbreviated as

$$S_i \text{ -}p/q\text{->} S_{ik}$$

which, when we do not care about the output sequence q, can be further simplified as

$$S_i \text{ -}p\text{->} S_{ik}$$

with the meaning that the FSM, when in state S_i and given an input sequence p, will enter state S_{ik}. The definitions of the transfer function δ_s and output function λ_s can be naturally extended to apply not only to single inputs, but also to sequences of inputs.

Definition 2.3 (extensions of transfer and output functions to input sequences)
Let $p = x_1x_2...x_k \in \psi(S_i)$ and ε be the empty word. Then,

$$\delta_s(S_i, \varepsilon) = S_i, \quad \delta_s(S_i, p) = \delta_s(\delta_s(S_i, p'), x_k)$$

$$\lambda_s(S_i, \varepsilon) = \varepsilon, \qquad \lambda_s(S_i, p) = \lambda_s(S_i, p').\lambda_s(\delta_s(S_i, p'), x_k)$$

where

$$p' = x_1x_2...x_{k-1}.$$ □

Definition 2.4 (compatible states and distinct states)
We say that S_i and S_j are *compatible* states if for : $p \in \psi(S_i) \cap \psi(S_j)$, $\lambda_s(S_i, p) = \lambda_s(S_j, p)$. Otherwise, they are called *distinct* states. □

We note that, according to the above definition, if $\psi(S_i) \cap \psi(S_j) = \phi$, then S_i is compatible with S_j. If the FSM happens to be completely specified, then the definition of compatible states given

above reduces to the definition of *equivalent states* as found in the literature (see for example, [Gill62, Koha78]).

Definition 2.5 (minimality)
A FSM is minimal if and only if no two states are compatible. □

Definition 2.6 (strongly connected FSM)
A FSM is *strongly connected* if, for any ordered pair of states $<S_i, S_j>$, there exists an input sequence p [$\psi(S_i)$ such that S_i -p->S_j. □

The concept of *Unique Input/Output sequence* (UIO), based on which a number of test sequence generation methods have been proposed, is formally defined in the following definition.

Definition 2.7 (UIO and UIS)
Let p be an input sequence defined for all states in the given FSM, that is p [$\psi(S_1)$ ($\psi(S_2)$ (··· ($\psi(S_n)$. Let $q = \lambda_s(S_i, p)$, the output sequence when p is applied to state S_i. Then we say that p/q is a *Unique Input/Output* (UIO) sequence for state S_i if

$$\lambda_s(S_i, p) \neq \lambda_s(S_j, p), \text{ for all } j \neq i.$$

And further we say that p is a *Unique Input Sequence* (UIS) for state S_i. □

A set of UIO sequences and the corresponding set of UISs for the three states of the FSM in Figure 1 are listed in Table 1.

According to their responses to some chosen input sequences, the states of a FSM can be partitioned into several blocks. Two states belong to a common block if and only if their responses to each of the chosen input sequences are the same. The following definition formally defines this concept.

Definition 2.8 (partitions induced by input sequences)
Let $W = \{ p_1, p_2, \cdots, p_k \}$ 1 $\psi(S_1)$ ($\psi(S_2)$ (··· ($\psi(S_n)$. The partition on the states induced by W, written as $\mathbf{P}_W$, is a set of subsets of states (blocks of states) $\mathbf{B}_1, \mathbf{B}_2, \ldots \mathbf{B}_m$, , such that

1) $\mathbf{B}_1$ " $\mathbf{B}_2$ " ··· " $\mathbf{B}_m = \{ S_1, S_2, \cdots, S_n \}$;
2) S_i, S_j [$\mathbf{B}_\neg$ iff $\lambda_s(S_i, p) = \lambda_s(S_j, p)$ for : p [W. □

It follows from the definition that any two different state blocks in the partition should be disjoint.

Let m be a positive integer and p be a word over an alphabet X. Then we use the notation p^m to represent the concatenation of p with itself for m times. That is

$$p^m = \underbrace{p.p.\cdots.p}_{\text{m times}}$$

With this convention, we define the following family of functions which will be used in the presentation of our test case generation method in Section 5.

$$\mathbf{F}_1\ (m;\ p) = p^m$$

and

$$\begin{aligned}\mathbf{F}_k\ &(m_1,\ m_2,\ ...,\ m_{k-2},\ m_{k-1},\ m_k;\ p_1,\ p_2,\ ...,\ p_{k-2},\ p_{k-1},\ p_k\ ) \\ &= [\ \mathbf{F}_{k-1}\ (m_1,\ m_2,\ ...,\ m_{k-2},\ 1;\ p_1,\ p_2,\ ...,\ p_{k-2},\ p_{k-1}\ )\ ]^{m_{k-1}} \\ &\quad \Theta\ \mathbf{F}_{k-1}\ (m_1,\ m_2,\ ...,\ m_{k-2},\ m_k\ ;\ p_1,\ p_2,\ ...,\ p_{k-2},\ p_k\ ) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{for} \quad k = 2,\ 3,\ 4,\ \cdots\end{aligned}$$

where $m,\ m_1,\ m_2,\ \cdots,\ m_k$ are any positive integers and $p,\ p_1,\ p_2,\ \cdots,\ p_k$ are any words over an alphabet.

3. A Testing Framework

The problem of conformance testing with the purpose of detecting faults in a black-box implementation is in general unsolvable unless it is dealt within a restricted framework [Moor56 and Gill62]. We propose here a framework within which our single approach for test case generation is presented.

3.1. Testing Assumptions

First of all, what we are considering in this framework is the so-called "testing a FSM implementation" [Ural91] problem which is formally defined as follows: given an FSM representation (specification) of a system (denoted henceforth as FSM_S) and an implementation of this FSM (denoted henceforth as FSM_I), we are asked to determine whether FSM_I conforms to FSM_S by testing FSM_I as a black-box. To solve this problem implies that we should

(1) formally define the conformance relation between FSM_I and FSM_S;
(2) generate from FSM_S a sequence TS_i of inputs and its expected sequence TS_o of outputs;
(3) apply TS_i to the input port of FSM_I;
(4) observe a sequence TS_a of actual outputs at the output port of FSM_I;
(5) compare TS_a with TS_o to determine the conformance of FSM_I to FSM_S.

(3) to (5) are mainly related to the actual execution of the test sequence and the analysis of test result whose discussion is outside the scope of this paper. To solve (1) and (2), we need to make certain assumptions. Let

$FSM_s = < \{S_1, S_2, ..., S_n\}, X_s, Y_s, S_1, \delta_s, \lambda_s, D_s >$

and

$FSM_I = < \{I_1, I_2, ..., I_u\}, X_I, Y_I, I_1, \delta_I, \lambda_I, D_I >$.

Then we assume throughout the rest of this paper that

(1) FSM_s is deterministic, strongly-connected and each of its states has a UIS;

(2) FSM_I is deterministic, completely specified and minimal;

(2) $u \leq n$, that is the number of states in FSM_I is not greater than that in FSM_S; and

(3) $X_s \{ X_I$, that is, all the input symbols of FSM_s should be included in the input set of FSM_I.

We note here in particular that we do not assume reliable reset to be available in FSM_I and that FSM_s need not be completely specified.

3.2. The Conformance Relation

We define the conformance relation, written as **CONF**, between FSM_I and FSM_s in the following way.

Definition 3.1 (equivalence of states in respect to a set of input sequences)
Let I_i be a state of the implementation FSM_I and S_j a state of the specification FSM_s. $V \{ \psi(S_j)$ is a set of input sequences. Then

$I_i -_V S_j$ if $\lambda_I (I_i, p) = \lambda_s(S_j, p)$, for : $p [V$. □

Definition 3.2 (conformance)
FSM_I **CONF** FSM_s iff $I_1 -_{\psi(S_1)} S_1$, where I_1 and S_1 are the initial states of FSM_I and FSM_s, respectively. □

We note that the conformance relation "**CONF**" is in fact the trace extension relation, that is, an implementation FSM_I conforms to a specification FSM_s if and only if FSM_I, when starting from its initial state I_1, can exhibit all the input/output traces specified for the initial state S_1 in FSM_s. This relation is also called "*quasi-equivalence*" in automata theory [Gill62].

Definition 3.3 (test case)
A test case is a sequence of inputs which should be of finite length and in $\psi(S_1)$. □

Definition 3.4 (test suite)
A test suite is a set of test cases. □

Definition 3.5 (pass of a test suite)
Let TS be a test suite. We say that a given implementation FSM_I passes the test suite, written FSM_I **pass** TS, iff $\lambda_I(I_1, p) = \lambda_S(S_1, p)$ for : p [TS. □

We can now define the concept of a "*complete test suite*".

Definition 3.6 (complete test suite)
Given a test suite TS and a specification FSM_S, we say that TS is a complete test suite for FSM_S with respect to the **CONF** relation if for any implementation FSM_I the following holds:
FSM_I **CONF** FSM_S iff FSM_I **pass** TS □

We note that in the case of a single testing approach, a test suite consists of a single test case and therefore we do not have to distinguish between a test suite and a single test case if no ambiguity arises.

4. Review of Test Generation Methods Based on UIO Sequences

A number of methods have been proposed for generating test cases from a finite state machine specification which has UIO sequences for its states. The UIO-method [SaDa88] and the UIOv-method [Vuon89] assume the reliable reset to be available in an IUT and therefore these two methods belong to the class of multiple testing approaches. The difference between these two methods is that the UIOv-method can guarantee full fault coverage (in the sense of Definition 3.6), while the UIO-method cannot.

The other methods ([Aho88], [Shen89], [Yang90] and [Zhan92] etc.) which are usually called "optimization techniques" are single testing approaches since they do not require the reliable reset to be provided by an IUT. These optimization techniques have been developed to generate a single test case from a specification FSM_S to check if *each specified transition* is correctly implemented in an implementation FSM_I . The general approach underlying these techniques is to ([Aho88])

(1) construct a test subsequence for each transition specified in FSM_S. A test subsequence is formed by the input symbol of the transition under test followed by the UIS for the ending state of that transition; and
(2) find a single optimal test case which traverses each of the test subsequences at least once, and if possible at most once by using the Rural Chinese Postman (RCP) tour problem.

This general approach can be enhanced by multiple UIOs and overlapping ([Shen89], [Yang90] and [Zhan92]) to obtain an even shorter test case. However, these optimization techniques cannot guarantee full fault coverage, that is, a single optimal test case generated in such a way can sometimes fail to detect certain faults which should be detected.

As a counter-example, let us consider the FSM specification given in Figure 1. We use the set of UISs given in Table 1 to form the transition test subsequences n_{t1}, n_{t2}, n_{t3}, n_{t4}, n_{t5} and n_{t6} given in Table 2. By using the above general approach, we generate a single test case which is also given in Table 2. It is easy to verify that this test case traverses each of the six transition test subsequences once. However, a faulty implementation modeled by the FSM given in Figure 2 can still pass this test case. (The same problem also exists for the multiple testing approach "UIO-method" [SaDa88]. Actually, the specification FSM in Figure 1 and the implementation FSM in Figure 2 have been used in [Vuon89] to show that the UIO-method cannot guarantee full fault coverage.)

The reason that this sort of problems may happen is that a unique input sequence derived from a given specification may no longer be a unique input sequence in a faulty implementation [Vuon89]. As in the above counter-example, the UIS "ba" for state S_3 in Figure 1 is no longer a UIS for the corresponding state I_3 in Figure 2, since states I_1 and I_3 respond identically to this input sequence.

5. UIOG - A Single Testing Approach Based on UIO Sequences with Guaranteed Fault Coverage

Our single testing approach improves the so-called optimization techniques by adding a *verification* part in a single test case. In other words, a test case generated from a given specification with our method consists of two functional parts called the "verification" part and the "transition checking" part. The transition checking part is formed from a set of *transition checking subsequences*. The verification part is designed to verify if all the UISs derived from the specification are still valid in the IUT. This is achieved by using two kinds of subsequences, namely *state verification subsequences* and *UIS verification subsequences*. The state verification subsequences are used to verify if the IUT has the same number of states as the specification and if so, the UIS verification subsequences will ensure that each UIS is applied to all the states of the IUT and therefore its validity as a UIS in the IUT is verified. These verified UISs are then used in the transition checking part to verify if each transition is correctly implemented. In this way, our approach guarantees to generate a complete single test case (see Definition 3.6) from FSM_s for the **CONF** relation.

5.1. The Basic Form of the UIOG Method

Let

$$\alpha_1,\ \alpha_2, \cdots, \alpha_n$$

be the unique input sequences for the n states S_1, S_2, ..., S_n, respectively, in the specification

$$FSM_s = < \{S_1, S_2, ..., S_n\}, X_s, Y_s, S_1, \delta_s, \lambda_s, D_s >.$$

Intuitively, by using shorter UISs we will in general generate a shorter test case. Therefore, we propose here to use the shortest UISs, although this is not a necessary requirement. In the following we will first give the procedures for deriving three types of subsequences, and then explain how to generate a test case from these subsequences. It should be noted that each subsequence is designed to have a specific starting state, that is, it should be applied to the designated state in the specification FSM_s.

Design of the State Verification Subsequences

For each state in the specification, a state verification subsequence should be designed which uses that state as the starting state. The n state verification subsequences together will verify if the IUT has the same number of distinct states as the specification and if so, a one-to-one correspondence can be established between the states of the specification and the states of the implementation. We can achieve this by designing these subsequences in the following way.

Since according to our definition of unique input sequences (Definition 2.7),

$$\{ \alpha_1, \alpha_2, \cdots, \alpha_n \} \; [\; \psi(S_1) \; V \; \psi(S_2) \; V \cdots \; V \; \psi(S_n)$$

and further the specification FSM_s is assumed to be strongly connected, the following so-called *simple looping sequences* β_{ij}'s are guaranteed to exist:

$$\beta_{ij} = \alpha_j.T(Q_{ij}, S_i), \quad \text{for } i = 1, 2, \cdots, n \text{ and } 1 \le j \le i.$$

such that

(1) S_i is the starting state for β_{ij};

(2) α_j is a prefix of β_{ij} ; and

(3) $T(Q_{ij}, S_i)$ is a *shortest* input sequence such that β_{ij} leads FSM_s from state S_i back to state S_i, that is,

$$S_i \text{ --}\alpha_j\text{--> } Q_i \text{ --}T(Q_{ij}, S_i)\text{--> } S_i .$$

Let us consider in particular the simple looping sequences $\beta_{11}, \beta_{22}, ..., \beta_{nn}$. Since α_i is a prefix of β_{ii} and α_i is a UIS for state S_i, β_{ii} is also a UIS for state S_i. We will call $\beta_{11}, \beta_{22}, ..., \beta_{nn}$ *Looping Unique Input Sequences* (LUIS).

Further, we define

$$W_i = \{ \alpha_1, \alpha_2, \cdots, \alpha_i \} \quad \text{for } i = 1, 2, ..., n \qquad \textbf{(5-1)}$$

These n sets of input sequences induce n partitions, written as $\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_{n-1}, \mathbf{P}_n$, on the state set $\{ S_1, S_2, ..., S_n \}$. Let m_i be the cardinality of $\mathbf{P}_i$, that is, the number of blocks in $\mathbf{P}_i$.

Then the "*state verification subsequence* γ_k" for state S_k can be given as follows:

$$\gamma_k = \mathbf{F}_k\,(n\text{-}m_1\text{+}2,\ n\text{-}m_2\text{+}2,\ \ldots,\ n\text{-}m_{k-1}\text{+}2,\ 1;\ \beta_{k1},\ \beta_{k2},\ \ldots,\ \beta_{k,k-1},\ \alpha_k) \quad \text{for } k = 1, 2, \cdots, n$$

which, if written explicitly, will be

$$\begin{aligned}
&\gamma_1 = \alpha_1 \\
&\gamma_2 = \beta_{21}^{n-m_1+2}\alpha_2 \\
&\gamma_3 = (\beta_{31}^{n-m_1+2}\beta_{32})^{n-m_2+2}(\beta_{31}^{n-m_1+2}\alpha_3) \\
&\gamma_4 = ((\beta_{41}^{n-m_1+2}\beta_{42})^{n-m_2+2}(\beta_{41}^{n-m_1+2}\beta_{43}))^{n-m_3+2}((\beta_{41}^{n-m_1+2}\beta_{42})^{n-m_2+2}(\beta_{41}^{n-m_1+2}\alpha_4)) \\
&\ \vdots \\
&\ \vdots
\end{aligned} \qquad \textbf{(5-2)}$$

Theorem 5.1

Let TS_s be an input sequence which begins with γ_1, that is, the state verification subsequence for the initial state S_1, and then traverses *in any order* all the other n - 1 state verification subsequences $\gamma_2, \gamma_3, \ldots, \gamma_n$ in **(5-2)** at least once. If an implementation FSM_I passes TS_s, then

(1) for each state S_i, we can find in the implementation a state denoted (without loosing generality) as I_i, such that $\alpha_1, \alpha_2, \cdots, \alpha_i$ are applied to I_i and its responses to these input sequences are the same as those of S_i, that is

$$S_i -_{W_i} I_i \qquad \text{for } i = 1, 2, \ldots, n$$

where $W_1, W_2, \ldots, W_n$ are given in **(5-1)** ;

(2) $I_1, I_2, \ldots, I_n$ are distinct states and therefore the mapping

$$\Phi: \{ S_1, S_2, \ldots, S_n \} \dashrightarrow \{ I_1, I_2, \ldots, I_n \}$$

defined by

$$\Phi(S_i) = I_i \qquad \text{for } i = 1, 2, \ldots, n$$

is one-to-one. □

The proof of this Theorem is given in the Appendix.

Design of the UIS Verification Subsequences

To verify that the UISs $\alpha_1, \alpha_2, \cdots, \alpha_{n-1}, \alpha_n$ derived for the states in the specification are still UISs for the corresponding states in the implementation, it is necessary that all the UISs $\alpha_1, \alpha_2, \cdots, \alpha_{n-1}, \alpha_n$ are applied to each state in the implementation. However, we have seen in Theorem 5.1 that the state verification subsequences ensure that $\alpha_1, \alpha_2, \cdots, \alpha_i$ are applied to I_i. Therefore what remains to do is to also apply $\alpha_{i+1}, \alpha_{i+2}, \cdots, \alpha_n$ to I_i, for i = 1, 2, ..., n. We note that α_k can distinguish S_k from all other n-1 states in the specification. However, to distinguish S_k from another particular state, say S_j ($j \neq k$), it may not be necessary to use the whole sequence α_k.

Instead, a prefix of α_k will often suffice. This is the reason that we first generate the following so-called *distinguishing words* α_{ij}'s which should be calculated for i = 1, 2, ..., n and $1 \le j \le i$:

(1) α_{ij} (i > j) is the *shortest* prefix of α_i such that $\lambda_s(S_i, \alpha_{ij}) \neq \lambda_s(S_j, \alpha_{ij})$; and

(2) $\alpha_{ii} = \alpha_i$.

Then we can construct the *UIS verification subsequences* $\mu_{k\neg}$'s as follows:

$$\mu_{k\neg} = \beta_{kk}.\alpha_{\neg k}, \quad \text{for} \quad k = 1, 2, ..., n \quad \text{and} \quad k \le \neg \le n \qquad \textbf{(5-3)}$$

where β_{kk}'s are the simple looping sequences generated before. We note that the starting state of $\mu_{k\neg}$ is S_k.

Theorem 5.2

Let TS_u be an input sequence which begins with γ_1, that is, the state verification subsequence for the initial state S_1, and then traverses *in any order* all the other n - 1 state verification subsequences $\gamma_2, \gamma_3, ..., \gamma_n$ in **(5-2)** and all the UIS verification subsequences $\mu_{k\neg}$'s in **(5-3)** at least once. If an implementation FSM_I passes TS_u, then

(1) the LUISs $\beta_{11}, \beta_{22}, ..., \beta_{nn}$ are still LUISs in the implementation FSM_I; and (2) the UISs $\alpha_1, \alpha_2, \cdots, \alpha_n$ are still UISs in the implementation FSM_I. □

The proof follows directly from Theorem 5.1. Details are given in the Appendix.

Design of the Transition Checking Subsequences

Finally, for each transition t of the form

$$S_i \text{ -x/y-> } S_j$$

in FSM_s, we generate its transition checking subsequence as follows

$$\nu_t = \beta_{ii}.x.\alpha_j \qquad \textbf{(5-4)}$$

where β_{ii} and α_j are used to verify the correctness of the starting state and the ending state of the transition. The starting state of ν_t is S_i. Then we have the following theorem.

Theorem 5.3

Let TS be an input sequence which begins with γ_1, that is, the state verification subsequence for the initial state S_1, and then traverses *in any order* all the other n - 1 state verification subsequences $\gamma_2, \gamma_3, ..., \gamma_n$ in **(5-2)**, all the UIS verification subsequences $\mu_{k\neg}$'s in **(5-3)** and all the transition checking subsequences in **(5-4)** at least once (note that overlapping is allowed). Then TS is a complete single test case for the given specification FSM_s with respect to the

conformance relation **CONF**.

□

The proof of this theorem is given in the Appendix.

The Formation of a Test Case

According to Theorem 5.3, a single test case should begin with γ_1 and traverse in any order all the other subsequences at least once, so it can be generated by directly using the Rural Chinese Postman tour based algorithm developed by Aho et al [Aho88]. The only difference is that in [Aho88] only one kind of subsequences, namely the transition test subsequences, should be traversed, while here three kinds of subsequences need to be traversed with a specific requirement that γ_1 should be a prefix of the generated test case. We note that, by requiring $\gamma_1 = \alpha_1$ to be at the beginning of the single test case, we ensure that the test case checks that the IUT is correctly initialized [Yann91]. That is, the generated test case can detect the fault that the IUT is in a wrong initial state.

Although the order in which the subsequences (except the state verification subsequence γ_1) appear in a test case is not important, in practice we do hope to find a particular order so that an optimal (that is, the shortest) test case can be obtained. We also point out here that further reduction on the length of a test case can be achieved if those subsequences are overlapped whenever possible. However, we do not elaborate this technique in detail here.

Example

As an example, let us consider the specification FSM given in Figure 1. We use the set of UISs listed in Table 1 and follow our approach to generate a single test case. Table 3a lists some intermediate calculations. The generated three types of subsequences are given in Table 3b. In order to form a test case from these subsequences, we can first construct an augmented graph as in Figure 4 from the original specification FSM given in Figure 1. In Figure 4, a bold arc from state S_i to state S_j is added to represent a subsequence starting from state S_i and ending at state S_j. Then we find a path in this augmented graph which begins with the bold arc with the label γ_1 and covers all the other bold arcs at least (and if possible at most) once. Those transitions belonging to the original specification FSM are used only when necessary to serve as "bridges" for combining those bold arcs. As already pointed out, we can also use overlapping to obtain a shorter test case. Table 3c gives a test case of length 53. In particular, this single test case will detect the faulty implementation given in Figure 2.

5.2. An Improvement on the Basic UIOG Method

Our method presented in Section 5.1 is applicable to any specification FSM as long as each of its states has a unique input sequence. However, this method can be further improved, under certain conditions, for the state verification subsequences and the UIS verification subsequences. In the following, we present an improved version of the basic method for the case that the UISs for some states are prefixes of the UIS for another given state.

Let

$$\alpha_1,\ \alpha_2, \cdots, \alpha_n$$

be the unique input sequences for the n states $S_1, S_2, ..., S_n$, respectively, in the specification

$$FSM_s = < \{S_1, S_2, ..., S_n\}, X_s, Y_s, S_1, \delta_s, \lambda_s, D_s >.$$

Without loosing generality, let h be the maximum integer such that $\alpha_1,\ \alpha_2, \cdots, \alpha_{h-1}$ are prefixes of α_h. We note that if α_h is applied to a state in the IUT, then $\alpha_1,\ \alpha_2, \cdots, \alpha_{h-1}$ are also applied to that state automatically and therefore need not to be applied separately. Based on this observation, we are able to simplify the state and UIS verification subsequences.

Let us denote

$$n' = n - h + 1$$
$$S'_i = S_{i+h-1} \quad \text{for} \quad i = 1, 2, ..., n - h + 1$$
$$\alpha'_i = \alpha_{i+h-1} \quad \text{for} \quad i = 1, 2, ..., n - h + 1$$

For this set of n - h + 1 states $S'_1, S'_2, ..., S'_{n-h+1}$ and their corresponding UISs $\alpha'_1, \alpha'_2, ..., \alpha'_{n-h+1}$, we can follow the method presented in Section 5.1 to generate n - h + 1 state verification subsequences

$$\gamma'_1,\ \gamma'_2, ...,\ \gamma'_{n-h+1}$$

and a set of UIS verification subsequences

$$\mu'_{k\neg} \quad \text{for } \neg \geq k \text{ and } k = 1, 2, \cdots, n - h \text{ and } \neg = 1, 2, \cdots, n - h + 1.$$

Then the set of n state verification subsequences for the original n states $S_1, S_2, ..., S_n$ of FSM_s can be given as follows:

$$\gamma_i = \alpha_i \quad \text{for} \quad i = 1, 2, ..., h - 1$$
$$\gamma_i = \gamma'_{i-h+1} \quad \text{for} \quad i = h, h + 1, h + 2, ..., n. \qquad \textbf{(5-5)}$$

To give all the UIS verification subsequences for the original n states $S_1, S_2, ..., S_n$ of FSM_s, we need to generate the following additional simple looping sequences:

$$\beta_{ii} = \alpha_i.T(Q_i, S_i), \quad \text{for } i = 1, 2, \cdots, h - 1$$

and the following additional distinguishing words

$$\alpha_{ij} \quad \text{for } i = h + 1, h + 2, ..., n \text{ and } j = 1, 2, ..., h - 1.$$

Then the UIS verification subsequences for the original n states $S_1, S_2, ..., S_n$ of FSM_s can be given as follows:

$\mu_{kk} = \beta_{kk}.\alpha_k$, for k = 1, 2, ..., h - 1

$\mu_{k\neg} = \beta_{kk}.\alpha_{\neg k}$, for k = 1, 2, ..., h - 1 and $\neg$ = h + 1, h + 2, ..., n **(5-6)**

$\mu_{k\neg} = \mu'_{k-h+1,\neg-h+1}$ for $\neg \geq k$, k = h, h + 1, ..., n - 1 and $\neg$ = h, h + 1, h + 2, ..., n

The transition checking subsequences can be constructed in the same way as given in Section 5.1.

Then we can follow exactly the same procedure as given in Section 5.1 to form a test case by finding an input sequence which traverses each state verification subsequence in **(5-5)**, each UIS verification subsequences in **(5-6)** and each transition checking subsequences **(5-4)**.

Let us again consider the example specification given in Figure 1 and its UIS set given in Table 1. We see here that $\alpha_1 = a$ is a prefix of $\alpha_2 = aa$ and therefore we can use the improved UIOG method to generate a test case. Table 4a shows the new state verification subsequences and UIS verification subsequences, while Table 4b gives a test case of length 31 (which is more than 40% shorter than the one given in Table 3c).

Certainly, further improvements can be made on our UIOG method. For instance, if the UISs α_1, α_2, ···, α_n can be divided into several groups such that all the UISs in a group are prefixes of another UIS in the same group, then further simplifications on the state verification and UIS verification subsequences can be made. However, due to limited space of this paper, we do not discuss these simplifications in detail here.

It is not difficult to see that in the extreme case when all the n UISs α_1, α_2, ···, α_n for the n states of a specification FSMs are the same, our approach reduces to the DS-method [Gone70], although in a slightly different form.

6. Incremental Test Generation

An existing system may need to be modified or extended due to the new demands and requirements from users and/or its environment [Erra92]. For instance, a communication protocol may have to be extended to incorporate new operations (behaviors) so that new services can be provided to the users. Two approaches are possible to the generation of test suites for the extended system. The first one is to take the extended system as a *new system* and apply a test case generation method to this new system. The second approach, which we call *incremental test generation*, takes as a starting point a test suite already generated for the system before behavior extensions are made. This test suite is then extended so that the newly added system behaviors will also be tested. Apparently, the incremental test generation approach is more effective since

it reuses the existing test suite rather than starting the test generation procedure from the beginning.

Another foreseeable application of incremental test generation is in the testing of object-oriented systems. In an object-oriented system, the components called objects are usually organized into object classes. An object class is a set of objects, which are called its instances. An object class definition specifies a set of allowable behaviors that each object instance in that class may exhibit. Furthermore, the inheritance mechanism allows one to define a new class (called *subclass*) from existing classes (called *superclasses*). The subclass can inherit the behaviors defined for its superclasses. Therefore, if the tests exist for testing the objects of the superclasses, then these existing tests can be reused [YaBo93] in the generation of the tests for testing the objects of the subclass.

In the following, we will use our UIOG method presented in Section 5 to show how to incrementally generate a test case for a finite state machine.

Let

$$M_1 = < \{S_1, S_2, ..., S_n\}, X_s, Y_s, S_1, \delta_s, \lambda_s, D_s >$$

be a specification FSM which satisfies the assumptions given in Section 3, and

$$\alpha_1, \alpha_2, \cdots, \alpha_n$$

be the unique input sequences for the n states S_1, S_2, ..., S_n, respectively. Then we can use the UIOG method to generate from M_1 a single test case denoted as TS_1. Further, let $\beta_{11}, \beta_{22}, ..., \beta_{nn}$ be the looping unique input sequences derived during the generation of TS_1 (see Section 5.1).

Suppose M_2 is a FSM obtained by adding only additional transitions to M_1. That is, no additional states are added and therefore, M_2 has exactly the same set of states as M_1 and $\alpha_1, \alpha_2, \cdots, \alpha_n$ are still the unique input sequences for the n states S_1, S_2, ..., S_n of M_2. Then we can incrementally generate for M_2 a test case (denoted as TS_2) from TS_1 as follows.

For each newly added transition t, say

$$S_i \text{ -x/y-> } S_j,$$

an additional transition checking subsequence is designed

$$\nu_t = \beta_{ii}.x.\alpha_j.$$

Then TS_2 can be constructed as a sequence of inputs which begins with TS_1 and then traverses in any order all the additional transition checking subsequences for the newly added transitions.

For example, let us consider the FSM given in Figure 3 which has been obtained from the FSM in Figure 1 by adding two new transitions t7 and t8 under an additional input "c". The additional transition checking subsequences ν_{t7} and ν_{t8} are given in Table 5. A test case for the FSM in

Figure 3 is also given in Table 5 which starts with the test case in Table 4b and traverses ν_{t7} and ν_{t8} exactly once.

For the case that additional states are added (and therefore additional transitions are added in order to keep M_2 strongly connected), the problem of incrementally generating TS_2 becomes much more complicated, since

(1) the newly added states may prevent α_1, α_2, ···, α_n from being UISs for S_1, S_2, ..., S_n in M_2;

(2) the newly added states, say S_{n+1}, S_{n+2}, ..., may have no UISs.

Therefore, the problem of incrementally generating test cases remains open when additional states are added.

7. Discussions and Conclusions

The single testing approach UIOG that we have presented in this paper is an improvement on the so-called optimization techniques. In order to guarantee full fault coverage, our UIOG method imposes more constraints (three types of subsequences) on a test case. Therefore, the length of a test case generated by the UIOG method is in general longer than a test case generated by an optimization technique ([Aho88], for instance). For example, the test case given in Table 4b is about 40% longer than the test case given in Table 2. Also, we have seen that, without the reliable reset assumption, certain portions of a test case have to be repeated several times in order to make sure that the same state in the implementation is reached at different points of a test case. Therefore, the length of a single test case generated by our UIOG method is in general longer than the total length of the test cases in a test suite generated by a multiple testing approach, say the UIOv method [Vuon89]. For instance, a test suite is given in Table 6 which is generated by the UIOv method for the FSM in Figure 1. The total length of the test cases in this test suite is 22 (including the reset symbols "r"). This means the single test case in Table 4b is about 30% longer than the test suite in Table 6. However, as we have already pointed out, our UIOG method is more applicable than the UIOv method, since the reliable reset assumption is not required by the UIOG method.

Hennie [Henn64] proposed a single testing approach for the generation of test cases for a class of finite state machines which have characterization sets (often called W-sets). Therefore, his method can be applied whenever our UIOG is applicable, since if α_1, α_2, ···, α_n are UISs for the n states S_1, S_2, ..., S_n of a given FSM, then { α_1, α_2, ···, α_n } is a W-set for that FSM. However, by using Hennie's method, one often obtains a test case which is prohibitively long. On the other hand, the UIOG method normally gives a much shorter test case.

Hsieh [Hsie71] proposed a single testing approach for a class of finite state machines which are called n/2 resolvable machines. His approach is both *efficient* (in terms of the length of a test case) and *effective* (in terms of its guarantee for fault coverage). However, the class of n/2 resolvable FSMs is not the same as the class of FSMs which have UIOs for all the states. That is, a FSM which has UIOs for all its states may not be necessarily n/2 resolvable. Therefore, Hsieh's approach cannot be applied to such a FSM, while the UIOG method can be applied.

In Section 5, the UIOG method has been presented for the case that only one UIO (or UIS) is used for each state. Actually it can easily be generalized to the case where multiple UIOs are used for each state. Multiple UIOs have been proposed to increase the chances of overlapping and therefore to reduce the length of a test case [Shen89, Yang90 and Zhan92]. With our approach, in order to guarantee full fault coverage, we need to construct n additional UIS verification subsequences for each additional UIS of a state. A test case should also traverse these additional UIS verification subsequences at least once and therefore the length of the verification part of the test case is in general increased. Therefore it is not clear how much one can benefit with the UIOG method from the use of multiple UISs.

It has been suggested for the so-called optimization techniques, that a signature [SaDa88] be used for a state which has no UIO, or in the case that the length of the UIO is longer than the signature. In the UIOG method, it is required that each state of the given FSM has a UIO sequence. It remains to be seen whether the UIOG method can be generalized to allow for signatures.

References

[Aho88] A. Aho, et al., "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours", Proc. of PSTV'88, 1988.

[Chow78] T.S. Chow, “Test Design Modeled by Finite-State Machines”, IEEE Trans. SE-4, 3, 1978.

[Erra92] M. Erradi, et al., "Dynamic Extension of Object-Oriented Distributed System Specifications", Int. Workshop on Feature Interactions in Telecommunications Software Systems, St Petersburg, Florida, USA, Dec. 3-4, 1992.

[Fuji91] S. Fujiwara, et al., "Test Selection Based on Finite State Models", IEEE Trans. SE-17, No. 6, June 1991, pp.591-603.

[Gill62] A. Gill, "Introduction to the Theory of Finite-State Machines", McGraw-Hill Book Company, Inc., 1962, pp. 207.

[Gone70] G. Gonenc, “A Method for the Design of Fault Detection Experiments”, IEEE Trans. Computers, Vol. C-19, June 1970, pp. 551-558.

[Henn64] F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits", Proc. of the 5th Annual Symposium on Switching Circuits Theory and Logical Design, Nov. 11-13, 1964

[Hsie71] E. P. Hsieh, "Checking Experiments for Sequential Machines", IEEE Trans. Computers, Vol. C-20, No. 10, Oct. 1971, pp. 1152-1166.

[Koha78] Z. Kohavi, "Switching and Finite Automata Theory", New York, McGraw-Hill, 1978, pp. 658.

[Moor56] E. F. Moore, "Gedanken-Experiments on Sequential Machines", Automata Studies, Princeton University Press, Princeton, New Jersey, 1956.

[Nait81] S. Naito and M. Tsunoyama, “Fault Detection for Sequential Machines by Transition-Tours”, Proc. of FTCS, 1981, pp. 238-243.

[Petr91] A.F. Petrenko, “Checking Experiments with Protocol Machines”, Proc. of the 4th Int. Workshop on Protocol Test Systems, 1991.

[Petr92] A.F. Petrenko and N. Yevtushenko, "Test Generation for a FSM with a Given Type of Implementation Errors", Proc. of PSTV'92, Florida, USA, June 92,

[SaDa88] K.K. Sabnani and A.T. Dahbura, “A Protocol Test Generation Procedure”, Computer Networks and ISDN Systems, Vol. 15, No. 4, 1988, pp. 285-297.

[Shen89] Y. N. Shen, et al., "Protocol Conformance Testing Using Multiple UIO Sequences", Proc. of PSTV'89, 1989.

[Ural91] H. Ural, "Formal Methods for Test Sequence Generation", Computer Communications, Vol. 15, No. 5, June 1992, pp. 311-325.

[Vasi73] M. P. Vasilevskii, "Failure Diagnosis of Automata", translated from Kibernetika, No.4, July-August, 1973, pp. 98-108.

[Vuon89] S.T. Vuong, et al., “The UIOv-method for Protocol Test Sequence Generation”, Proc. of the 2nd Int. Workshop on Protocol Test Systems, Berlin, Germany, Oct. 3-6, 1989.

[Vuon90] S.T. Vuong and K.C. Ko, “A Novel Approach to Protocol Test Sequence Generation”, Proc. of GlobalCOM’90, 1990.

[YaBo93] M. Yao and G. v. Bochmann, "Testing for a Conformance Relation Based on Acceptance", Accepted by TAPSOFT'93 (FASE), Orsay, France, April 13-17, 1993.

[Yang90] B. Yang and H. Ural, "Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlapping", Computer Communication Review, No. 4, 1990, pp. 118-125.

[Yann91] M. Yannakakis, "Testing Finite State Machines", in Proceedings of the 23d Annual ACM Symposium on Theory of Computing, New Orleans, Louisiana, 1991, pp 476-485.

[Zhan92] L. D. Zhang, et al., "A Further Optimization Technique for Conformance Testing Based on Multiple UIO Sequences", Proc. of IWPTS' 92, Sept. 1992.

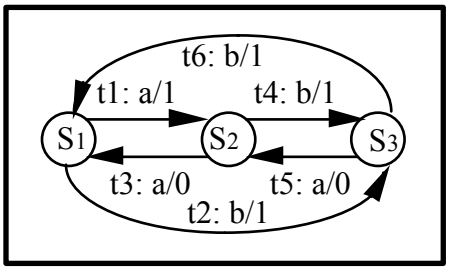

Figure 1: An example FSM

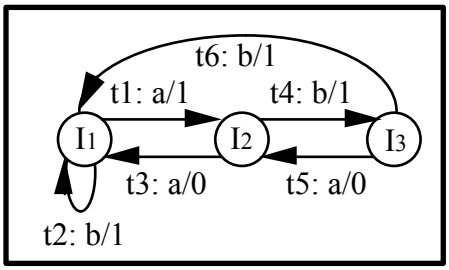

Figure 2: A faulty implementation

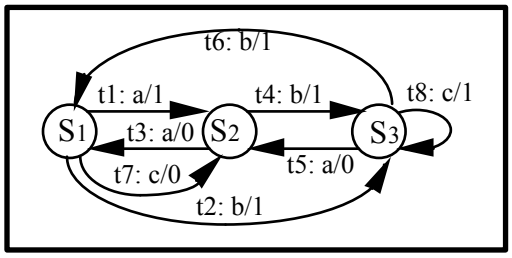

Figure 3: An extension of the FSM in Figure 1

States	UIO's	UIS's
S_1:	a/1	a
S_2:	a/0.a/1	aa
S_3:	b/1.a/1	ba

Table 1: UIOs and UISs

n_{t1}: [1] a [2] a [1] a [2] n_{t2}: [1] b [3] b [1] a [2] n_{t3}: [2] a [1] a [2]
n_{t4}: [2] b [3] b [1] a [2] n_{t5}: [3] a [2] a [1] a [2] n_{t6}: [3] b [1] a [2]

TS = [1] a [2] a [1] a [2] a [1] a [2] b [3] b [1] a [2] a [1] b [3] b [1] a
[2] b [3] a [2] a [1] a [2] b [3] b [1] a [2]

Table 2: A test case generated by an optimization technique [Aho88][+]

[+] The integer in a pair of square brackets in an input sequence indicates a state number. For instance, [1] a [2] a [1] means the input sequence "aa" starts from state S_1, passes state S_2 and ends at state S_1.

Sets of input sequence	Partitions	Sizes of the partitions
$W_1 = \{a\}$	$P_1 = \{S_1, S_2S_3\}$	$m_1 = 2$
$W_2 = \{a, aa\}$	$P_2 = \{S_1, S_2, S_3\}$	$m_2 = 3$
$W_3 = \{a, aa, ba\}$	$P_3 = \{S_1, S_2, S_3\}$	$m_3 = 3$

Simple looping sequences

$\beta_{11} = [1]\,aa\,[1]$

$\beta_{21} = [2]\,aa\,[2]$ $\beta_{22} = [2]\,aa\,[2]$

$\beta_{31} = [3]\,aa\,[3]$ $\beta_{32} = [3]\,aab\,[3]$ $\beta_{33} = [3]\,bab\,[3]$

Distinguishing words

$\alpha_{11} = [1]\,a\,[2]$

$\alpha_{21} = [1]\,a\,[2]$ $\alpha_{22} = [2]\,aa\,[2]$

$\alpha_{31} = [1]\,ba\,[2]$ $\alpha_{32} = [2]\,ba\,[2]$ $\alpha_{33} = [3]\,ba\,[2]$

Table 3a

State verification subsequences

$\gamma_1 = [1]\,a\,[2]$

$\gamma_2 = [2]\,a\,[1]\,a\,[2]\,a\,[1]\,a\,[2]\,a\,[1]\,a\,[2]\,a\,[1]\,a\,[2]$

$\gamma_3 = [3]\,a\,[2]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]\,a\,[1]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]$
$[2]\,a\,[1]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]\,b\,[3]\,a\,[2]\,b\,[3]\,b\,[1]\,a\,[2]$

UIS verification subsequences

$\mu_{11} = [1]\,a\,[2]\,a\,[1]\,a\,[2]$ $\mu_{12} = [1]\,a\,[2]\,a\,[1]\,a\,[2]$ $\mu_{13} = [1]\,a\,[2]\,a\,[1]\,b\,[3]\,a\,[2]$

$\mu_{22} = [2]\,a\,[1]\,a\,[2]\,a\,[1]\,a\,[2]$ $\mu_{23} = [2]\,a\,[1]\,a\,[2]\,b\,[3]\,a\,[2]$

$\mu_{33} = [3]\,b\,[1]\,a\,[2]\,b\,[3]\,b\,[1]\,a\,[2]$

Transition checking subsequences

$\nu_{t1} = [1]\,a\,[2]\,a\,[1]\,a\,[2]\,a\,[1]\,a\,[2]$ $\nu_{t4} = [2]\,a\,[1]\,a\,[2]\,b\,[3]\,b\,[1]\,a\,[2]$

$\nu_{t2} = [1]\,a\,[2]\,a\,[1]\,b\,[3]\,b\,[1]\,a\,[2]$ $\nu_{t5} = [3]\,b\,[1]\,a\,[2]\,b\,[3]\,a\,[2]\,a\,[1]\,a\,[2]$

$\nu_{t3} = [2]\,a\,[1]\,a\,[2]\,a\,[1]\,a\,[2]$ $\nu_{t6} = [3]\,b\,[1]\,a\,[2]\,b\,[3]\,b\,[1]\,a\,[2]$

Table 3b: State verification, UIS verification and Transition Checking subsequences

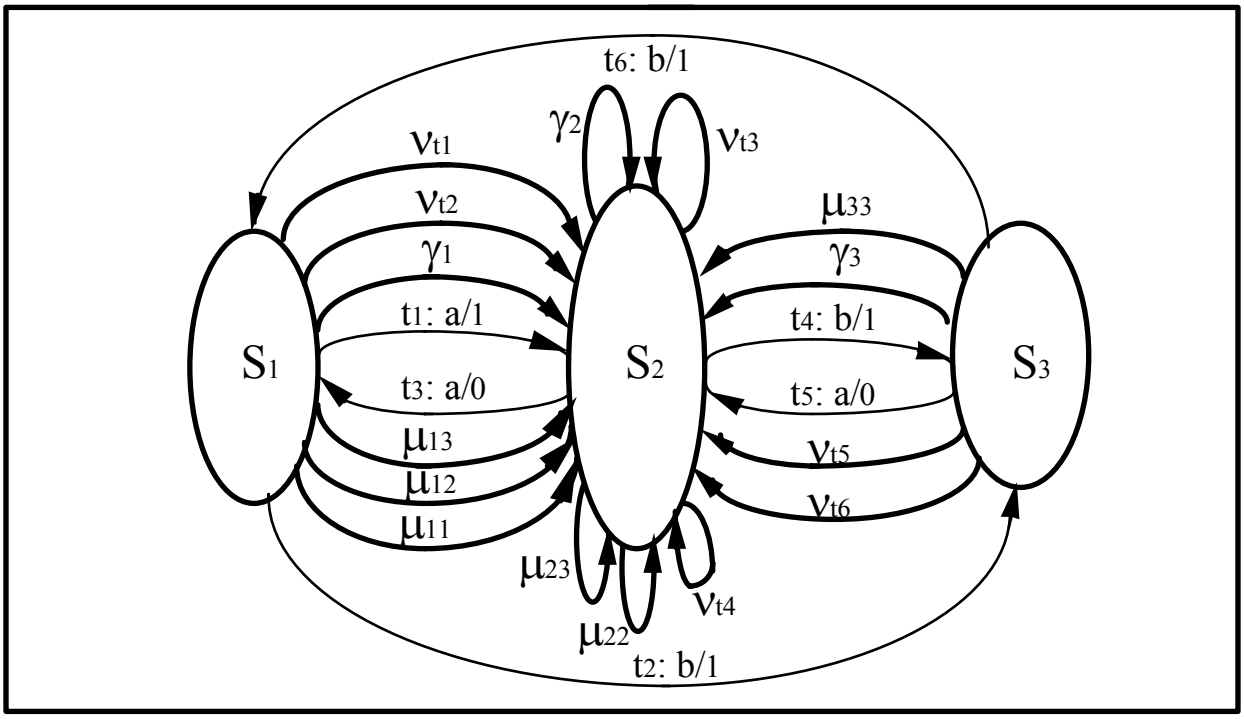

Figure 4: The augmented graph for test sequence generation

TS = [1] a [2] a [1] b [3] a [2] b [3] a [2] b [3] a [2] b [3] a [2] a [1] b [3] a [2] b [3] a [2] b [3] a [2] b [3] a [2] a [1] b [3] a [2] b [3] a [2] b [3] a [2] b [3] a [2] a [1] a [2] a [1] a [2] a [1] a [2] a [1] a [2] b [3] b [1] a [2] a [1] b [3] b [1] a [2] a [1] a [2] b [3] a [2]

Table 3c: A test sequence of length 53 (by our basic UIOG Approach)

State verification subsequences

γ_1 = [1] a [2]

γ_2 = [2] a [1] a [2]

γ_3 = [3] a [2] a [1] b [3] a [2] a [1] b [3] b [1] a [2]

UIS verification subsequences

μ_{11} = [1] a [2] a [1] a [2]

μ_{13} = [1] a [2] a [1] b [3] a [2]

μ_{22} = [2] a [1] a [2] a [1] a [2]

μ_{23} = [2] a [1] a [2] b [3] a [2]

μ_{33} = [3] b [1] a [2] b [3] b [1] a [2]

Table 4a: The new state verification and UIS verification subsequences (by our improved approach)

TS = [1] a [2] a [1] a [2] a [1] a [2] b [3] b [1] a [2] b [3] a [2] a [1] a [2] b [3] a [2] a [1] b [3] a
[2] a [1] b [3] b [1] a [2] b [3] b [1] a [2] a [1] b [3] b [1] a [2] a [1] b [3] a [2]

Table 4b: A test sequence of length 31 (by our improved UIOG approach)

ν_{t7} = [1] a [2] a [1] c [2] a [1] a [2] ν_{t8} = [3] b [1] a [2] b [3] c [3] b [1] a [2]

TS = [1] a [2] a [1] a [2] a [1] a [2] b [3] b [1] a [2] b [3] a [2] a [1] a [2] b [3] a [2] a [1] b [3] a
[2] a [1] b [3] b [1] a [2] b [3] b [1] a [2] a [1] b [3] b [1] a [2] a [1] b [3] a [2] a [1] a [2] a
[1] c [2] a [1] a [2] b [3] b [1] a [2] b [3] c [3] b [1] a [2]

Table 5: An example for the incremental test generation

r. a.a.a r.a.b.a r.a.b.b.a r.b.a.a.a r.b.b.a
where r represents "reset"

Table 6: A test suite generated by UIOv method [Vuon89] for the FSM in Figure 1

Appendix: Proofs of the theorems

This appendix gives the proofs for the theorems in this paper. It is important to keep in mind that these theorem are proved within the testing framework presented in section 3, that is the assumptions on the specification machine FSM_S and implementation machine FSM_I are supposed to hold throughout the proofs.

Lemma A.1

FSM_I **CONF** FSM_S iff there exists a mapping **f**:

$\{S_1, S_2, ..., S_n\}$ ---> $\{I_1, I_2, ..., I_U\}$, such that

(1) **f** is one-to-one (which implies $u = n$); and

(2) If S_i - x/y -> S_j is in FSM_S, then I_k -x/y ->$I_\neg$ is in FSM_I, where I_k =**f**(S_i), $I_\neg$ =**f**(S_j).

[*Proof*]:

[<= part]: If there exists such a mapping **f** , then FSM_I **CONF** FSM_S.
The proof for this part is omitted, since it is obvious.

[=> part]: If FSM_I **CONF** FSM_S , then there exists such a mapping **f**.
According to the definition of "**CONF**", FSM_I **CONF** FSM_S implies $I_1 \sim_{\psi(S1)} S_1$. So we can construct a mapping **f** from the states of FSM_S to the states of FSM_I in the following way:

$$\mathbf{f}(S_i) = I_j$$

iff there exists an input sequence $p \in \psi(S_1)$, such that

$$S_1 \text{ -}p\text{-> } S_i \quad \text{and} \quad I_1 \text{ -}p\text{-> } I_j.$$

Since FSM_S is assumed to be strongly connected and $I_1 \sim_{\psi(S1)} S_1$, we can conclude that each state S_i in FSM_S should be mapped to at least one state I_j in FSM_I by the above defined mapping **f**.

Now we prove that the mapping **f** satisfies the two required conditions.

(1) **f** is a one-to-one mapping: This can be proved by contradiction.
If not, then there should be a state $I_k = \mathbf{f}(S_i) = \mathbf{f}(S_j)$, for some states S_i, S_j, $i \neq j$. Since FSM_S is deterministic, there should be two different input sequences p_1 and p_2 in $\psi(S_1)$, such that:

$$S_1 \text{ -}p_1\text{-> } S_i \quad \text{and} \quad I_1 \text{ -}p_1\text{-> } I_k$$

$$S_1 \text{ -}p_2\text{-> } S_j \quad \text{and} \quad I_1 \text{ -}p_2\text{-> } I_k.$$

As FSM_S is also minimal, there should be an input sequence $r \in \psi(S_i) - \psi(S_j)$, such that

$$\lambda_s(S_i, r) \neq \lambda_s(S_j, r).$$

Because FSM_I is also deterministic, we can conclude that at least one of the following two inequalities should hold:

$$\lambda_I(I_k, r) \neq \lambda_s(S_i, r)$$

$$\lambda_I(I_k, r) \neq \lambda_s(S_j, r).$$

If the first one holds, then the input sequence $p_1.r$ (which is in $\psi(S_1)$) will produces different output sequences when it is applied to S_1 and I_1, respectively, that is

$$\lambda_s(S_1, p_1.r) \neq \lambda_I(I_1, p_1.r)$$

which contradicts to $I_1 \sim_{\psi(S1)} S_1$, that is FSM_I **CONF** FSM_S .

Similar arguments apply if the second inequality holds. So we can conclude that **f** should be a one-to-one mapping.

(2) If S_i - x/y -> S_j is in FSM_S, then for every $I_k = \mathbf{f}(S_i)$, there exists some $I_\neg = \mathbf{f}(S_j)$, such that I_k - x/y -> $I_\neg$ is in FSM_I. The reason follows.

Since $I_k = \mathbf{f}(S_i)$ and $I_1 - \psi(S_1)\ S_1$, there should be two input sequences p and $p.x$ in $\psi(S_1)$ with two corresponding output sequences q and $q.y$, such that:

$$S_1\text{-}p/q\text{->}S_i\text{-}x/y\text{->}S_j \text{ and } I_1\text{-}p/q\text{->}I_k\text{-}x/y\text{->}I_\neg.$$

According to the construction of **f**, we know that $I_\neg = \mathbf{f}(S_j)$. □

Theorem 5.1

Let TS_S be an input sequence which begins with γ_1, that is, the state verification subsequence for the initial state S_1, and then traverses *in any order* all the other $n - 1$ state verification subsequences in **(5-2)** at least once. If an implementation FSM_I passes TS_S, then

(1) for each state S_i, we can find in the implementation a state denoted (without loosing generality) as I_i, such that $\alpha_1, \alpha_2, \cdots, \alpha_i$ are applied to I_i and its responses to these input sequences are the same as those of S_i, that is

$$S_i - W_i\ I_i \qquad \text{for } i = 1, 2, ..., n$$

where $W_i = \{ \alpha_1, \alpha_2, \cdots, \alpha_i \}$ as given in **(5-1)**;

(2) $I_1, I_2, ..., I_n$ are distinct (that is pair wisely non-compatible) and therefore the mapping

$$\Phi: \{ S_1, S_2, ..., S_n \} \text{ --> } \{ I_1, I_2, ..., I_n \}$$

defined by

$$\Phi(S_i) = I_i \qquad \text{for } i = 1, 2, ..., n$$

is one-to-one.

[*Proof*]:

The proof for the first conclusion is based on the following reasonings. Each reasoning step consists of a claim which is followed by the reasons in square brackets to show why that claim is true. We use $\Delta_{11}, \Delta_{21}, \Delta_{22}, ...$ to indicate certain places of points in the state verification subsequences:

$$\gamma_1 = \Delta_{11}\alpha_1$$
$$\gamma_2 = \Delta_{21}\ \beta_{21}\ \Delta_{22}\ \beta_{21}\ \Delta_{23}\ \cdots\ \Delta_{2\mathbf{r}}\ \beta_{21}\Delta_{2,\mathbf{r}+1}\alpha_2$$
$$\vdots$$
$$\vdots$$

(R1) FSM_I passes TS_S which traverses each state verification subsequences at least once. [given condition]

(R2) Consider the state at the place Δ_{11}:

(R2a) In FSM_S, the state at Δ_{11} is S_1. [property of γ_1]

(R2b) Denote the state in FSM_I at Δ_{11} as I_1. Then $S_1 - W_1\ I_1$. [by (R1)]

(R3) Let $r = n - m_1 + 2$. Consider the states at the places $\Delta_{21}, \Delta_{22}, ..., \Delta_{2r}, \Delta_{2,r+1}$.

(R3a) In FSM_S, the state at these $r + 1$ places are the same, that is S_2. [property of β_{21} , γ_2]

(R3b) Denote the states in FSM_I at these $r + 1$ places as $J_1, J_2, ..., J_r, J_{r+1}$, respectively

(R3c) $S_2 - \{\beta_{21}\}\ J_i$ for $i = 1, 2, ..., r$. [by (R1)]

(R3d) $S_2 - \{\alpha_2\}\ J_{r+1}$. [by (R1)]

(R3e) $S_2 -\{\alpha_1\} J_i$ for $i = 1, 2, ..., r$. [(R3c) and α_1 is a prefix of β_{21}]

(R3f) α_1 is a prefix of β_{i1}, for $i = 1, 2, ..., n$ [property of β_{i1}'s]

and therefore α_1 is a prefix of γ_i, for $i = 1, 2, ..., n$ [property of γ_i's]

(R3g) In FSM_S, γ_i is applied to S_i by TS_S, for $i = 1, 2, ..., n$ [design of γ_i's]

(R3h) In FSM_S, α_1 is applied to S_i by TS_S, for $i = 1, 2, ..., n$ [by (R3f) and (R3g)]

(R3i) The n states $S_1, S_2, ..., S_n$ respond to α_1 in m_1 different ways. [$|\mathbf{P}_1| = m_1$]

(R3j) In FSM_I, there exist m_1 states which respond to α_1 in m_1 different ways. [By (R1)]

(R3k) The number of distinct states in FSM_I is no more than n. [Assumption on FSM_I]

(R3l) In FSM_I, there exist at most $r - 1 = n - m_1 + 1$ states which can respond to α_1 identically as S_2. [by (R3j) and (R3k)]

(R3m) There exist k, $\neg$, such that $1 \le k < \neg \le r$ and J_k and $J_\neg$ are the same, that is $J_k = J_\neg$ [by (R3e), (R3l) and *pigeonhole principle*]

(R3n) J_{r+1} and $J_{k+r+1-\neg}$ are the same, that is $J_{r+1} = J_{k+r+1-\neg}$, where $k+r+1-\neg \le r$ [by (R3m) and deterministic property of FSM_I]

(R3p) $S_2 -\{\alpha_1, \alpha_2\} J_{r+1}$. [by (R3d), (R3e) and (R3n)]

(R3q) By rewriting J_{r+1} as I_2, we have $S_2 -W_2 I_2$.

Following similar reasoning, we can prove that, by using each γ_i, there exist a state in FSM_I denoted as I_i, such that

$$S_i -W_i I_i \quad \text{for } i = 1, 2, ..., n.$$

Since $\gamma_1 = \alpha_1$ is required to be a prefix of TS_S, we know I_1 should be the initial state of the IUT. This completes the proof for our first conclusion.

For the second conclusion, we can prove it by contradiction. Suppose there exist i and j, such that $1 \le i < j \le n$ and $I_i = I_j$. Then $S_i -W_i S_j$, which implies that S_i and S_j will respond identically to each sequence in W_i, and in particular to α_i which is in W_i. However, this contradicts to the fact that α_i is a UIS for state S_i. So the mapping Φ should be one-to-one. □

Theorem 5.2

Let TS_u be an input sequence which begins with γ_1, that is, the state verification subsequence for the initial state S_1, and then traverses *in any order* all the other $n - 1$ state verification subsequences $\gamma_2, \gamma_3, ..., \gamma_n$ in **(5-2)** and all the UIS verification subsequences $\mu_{k\neg}$'s in **(5-3)** at least once. If an implementation FSM_I passes TS_u, then

(1) the LUISs $\beta_{11}, \beta_{22}, ..., \beta_{nn}$ are still LUISs in the implementation FSM_I; and

(2) the UISs $\alpha_1, \alpha_2, \cdots, \alpha_n$ are still UISs in the implementation FSM_I. and

[*Proof*]:

As TS_u traverses each state verification subsequence in **(5-2)** at least once, we know from the proof of Theorem 5.1 that, if an implementation FSM_I passes TS_u, then FSM_I has exactly n distinct states $I_1, I_2, ..., I_n$ which can be mapped to $S_1, S_2, ..., S_n$ by a one-to-one mapping Φ such that (without loosing generality)

$$\Phi(S_i) = I_i \quad \text{iff } S_i -W_i I_i \quad , \text{ for } i = 1, 2, ..., n.$$

Let us consider I_1 in particular. We have

(1) As α_1 is a UIS for S_1, we can conclude that only I_1 in the implementation can respond to α_1 as S_1 does, that is α_1 is a UIS for I_1;

(2) TS_u traverses the particular UIS verification subsequence $\mu_{11} = \beta_{11}.\alpha_{11}$ which, when written explicitly, is

$$S_1 - \alpha_1 -> Q_1 - T(Q_1, S_1) -> S_1 - \alpha_1 -> Q_1.$$

If FSM_I passes TS_u, FSM_I also passes this particular UIS verification subsequence. With the conclusion in (1), we can conclude that if a state in the implementation responds to $\beta_{11} = \alpha_1 \cdot T(Q_1, S_1)$ identically as S_1, then that state should be I_1 and the ending state should also be I_1; and

(3) TS_u traverses the particular UIS verification subsequence $\mu_{12} = \beta_{11}.\alpha_{21}$ which, when written explicitly, is

$$S_1 - \alpha_1 -> Q_1 - T(Q_1, S_1) -> S_1 - \alpha_{21} -> Q_1.$$

If FSM_I passes TS_u, FSM_I also passes this particular UIS verification subsequence. With conclusion in (2), we can conclude that α_{21} is also applied to I_1 which gives the expected output sequence $\lambda_s(S_1, \alpha_{21})$.

Similarly, we can conclude that α_{31}, ..., α_{n1} are also applied to I_1 which gives the expected outputs.

By similar reasonings to (1) to (3), we can have the following results:

First, for each β_{ii}, if a state in FSM_I responds to β_{ii} in the same way as S_i does, then both that state and the ending state after β_{ii} should be I_i. That is, the LUISs β_{11}, β_{22}, ..., β_{nn} are still LUISs in the implementation FSM_I.

Second,

α_{32}, ..., α_{n2} are applied to I_2;

α_{43}, ..., α_{n3} are applied to I_3;

...

$\alpha_{n-1,n-1}$ are applied to I_{n-1}

and in all these cases, the expected outputs are observed. Combining the conclusion in Theorem 5.1, we can conclude that α_1, α_2, ..., α_n are UISs for I_1, I_2, ..., I_n, respectively. □

Theorem 5.3

Let TS be an input sequence which begins with γ_1, that is, the state verification subsequence for the initial state S_1, and then traverses *in any order* all the other $n - 1$ state verification subsequences γ_2, γ_3, ..., γ_n in **(5-2)**, all the UIS verification subsequences μ_k's in **(5-3)** and all the transition checking subsequences in **(5-4)** at least once (note that overlapping is allowed). Then TS is a complete test sequence for the given specification FSM_S with respect to the conformance relation **CONF**.

[*Proof*]:

According to Definition 3.6, to prove that TS is a complete test case, we need to show that, for any implementation FSM_I, FSM_I **CONF** FSM_S if and only if FSM_I passes TS.

[=> part]: If FSM_I **CONF** FSM_S then $I_1 - \psi(S_1)\ S_1$ and therefore FSM_I passes TS.

[<= part]: As TS traverses each of the state verification subsequences and each of the UIS verification subsequences, according to Theorem 5.1 and Theorem 5.2, if FSM_I passes TS, then

(1) FSM_I has exactly distinct states I_1, I_2, ..., I_n which can be mapped to S_1, S_2, ..., S_n by a one-to-one mapping Φ such that (without loosing generality)

$$\Phi(S_i) = I_i \quad \text{iff} \quad S_i \ -W_i \ I_i \ , \quad \text{for } i = 1, 2, ..., n;$$

(2) for each α_i, α_i is not only a UIS for state S_i in FSM_S, but also a UIS for the corresponding state I_i in FSM_I; and

(3) For each β_{ii}, if a state in FSM_I responds to β_{ii} in the same way as S_i does, then both that state and the ending state after β_{ii} should be I_i.

Now let us consider a transition t: $S_i -x/y-> S_j$ in FSM_S. Its transition checking subsequence is

$$\nu_t = \beta_{ii}.x.\alpha_j$$

which, if written explicitly, will be

$$S_i -\beta_{ii}-> S_i -x-> S_j-\alpha_j->.$$

Since TS traverses this transition checking subsequence and FSM_I passes TS, we know that FSM_I also passes this transition checking subsequence. Therefore there is the following sequence of transitions in FSM_I :

$$I_h -\beta_{ii}-> I_k -x-> I_\neg-\alpha_j->$$

such that S_i and I_h respond identically to β_{ii}, S_i and I_k respond identically to x and, S_j and $I_\neg$ respond identically to α_j. Then according to (2) and (3), we have

$$I_h = I_k = \Phi(S_i) = I_i \quad \text{and} \quad I_\neg = \Phi(S_j) = I_j$$

which implies that for transition t in FSM_S, there is a corresponding transition

$$I_i -x/y-> I_j$$

in FSM_I. So by Lemma A.1, we can conclude FSM_I **CONF** FSM_S. □