Improving Test Quality Using Robust Unique Input/Output Circuit Sequences (UIOCs)

[†]Qiang Guo, [†]Robert M. Hierons, [‡]Mark Harman and [†]Karnig Derderian
[†]School of Information Systems, Computing and Mathematics
Brunel University, Uxbridge, Middlesex UB8 3PH, UK
[‡]Department of Computer Science
King's College London, Strand, London WC2R 2LS, UK
[†]{*Qiang.Guo, Rob.Hierons, Karnig.Derderian*}*@brunel.ac.uk*
[‡]{*Mark@dcs.kcl.ac.uk*}

July 27, 2005

Abstract

In finite state machine (FSM) based testing, the problem of fault masking in the unique input/output (UIO) sequence may degrade the test performance of the UIO based methods. This paper investigated this problem and proposed the use of a new type of unique input/output circuit (UIOC) sequence for state verification, which may help to overcome the drawbacks that exist in the UIO based techniques. When constructing a UIOC, overlap and internal state observation schema are used to increase the robustness of a test sequence. Test quality was compared by using the forward UIO method (F-method), the backward UIO method (B-method) and the UIOC method (C-method) separately. Robustness of the UIOCs constructed by the algorithm given in this paper was also compared with those constructed by the algorithm given previously. Experimental results suggested that the C- method outperforms the F- and the B- methods and the UIOCs constructed by the algorithm given in this paper are more robust than those constructed by other proposed algorithms.

Keywords: FSMs, Conformance Test, F-UIOs, B-UIOs, UIOCs, Fault Masking, Test Quality.

1 Introduction

Testing is an expensive process. Finding effective strategies to automate the generation of efficient test sequences, which can help to reduce development costs and to improve the quality of (or at least confidence in) a system, is extremely important. An efficient test sequence should cover, as much as possible, all faults which any implementation may have and should be relatively short. Many approaches have been proposed for generating an efficient test. These methods are mostly based upon traditional finite automata theory and, in particular, they model the system as a finite state machine (FSM) [2, 3, 10, 11, 12, 17, 20, 21, 22, 23, 26].

In FSM–based testing, a standard test strategy consists of two parts, namely, transition test and tail state verification. The former part aims to determine whether a transition of an implementation under test (IUT) produces the expected output while the latter checks that the IUT arrives at the specified state when a transition test is finished. Three techniques are proposed for state verification: Unique Input/Output Sequence (UIO), Distinguishing Sequence (DS), and Characterizing Set (CS). Due to its practical characteristics, UIO based techniques are often used for test sequence generation [14]. Aho *et al.* [2] showed how an efficient test sequence may be produced using UIOs for state verification. Shen *et al.* [21] extended the method by using multiple UIOs for each state and showed that this leads to a shorter test sequence. Yang *et al.* [26], Miller [17] and Hierons [10, 11] showed that overlap can be used in conjunction with (multiple) UIOs to further reduce the test sequence length. All these works mainly concerned minimizing the length of a test sequence.

When testing from an FSM M, ideally we use a complete test sequence: a test sequence that aims to determine correctness if the number of states of the IUT does not exceed some predetermined bound. However, all approaches to generating a complete test sequence either rely on the existence of a distinguishing sequence for M [8, 9, 25], assume that the IUT has a reliable reset [4], or produce a test sequence whose size is exponential in terms of the number of states of M [19]. However, an FSM need not have a distinguishing sequence and often the IUT does not have a reliable reset and there has thus been much interest in alternative test techniques, often based on UIOs.

Sidhu *et al.* [20] concluded that the U-, D-, and W-methods produce identical fault coverage and ensure the detection of all faults. However, this conclusion was challenged by Chan *et al.*, arguing that the problems of fault masking in UIOs may degrade the performance of UIO based methods. In their paper [3], Chan *et al.* showed that U- and D-methods produce identical fault coverage only when UIOs generated from the specification FSM can identify the corresponding states in the IUT as well. A UIO may lose its property of uniqueness in some faulty implementations, which leads to the failure of corresponding state verification. To overcome these problems, Chan *et al.* proposed the UIOv method where all UIOs are checked first for their uniqueness in the IUT before being selected for test sequence generation. Although this operation helps to improve the test quality, it might significantly increase the test cost. Meanwhile, in a system without reset function, it might also make the procedure of testing discontinuous. Naik [18] further studied the problems and pointed out that a fault in a UIO can be masked either by some erroneous outputs or by an incorrect state transfer. In order to enhance the ability of UIOs to resist fault masking, he suggested that, when generating a test sequence those UIOs with maximal strength should be considered first. He also proposed an algorithm to construct UIOs with high strength. This method can effectively reduce the chance that faults are masked by error outputs, but might lack the ability to handle the situation that faults are masked by incorrect state transitions.

Shen *et al.* [22] showed that using a backward UIO (B-UIO) in a transition test helps to improve test quality. By applying a B-UIO, the initial state of the transition is also verified. All states in B-UIO method are verified twice. The test quality is therefore improved. However, the use of B-UIOs can also lead to some problems. These are discussed in section 3. In [23], Shen *et al.* extended the work by using unique input/output circuit sequences (UIOCs) for state verification. A UIOC was constructed by using a F-UIO and a B-UIO for a state. If the F-UIO and the B-UIO do not naturally form a circuit (the tail state of the F-UIO is not the initial state of the B-UIO), a transfer sequence will be added to complete it. This operation may give rise to some problems. If the gap between the tail state of the F-UIO and the initial state of the B-UIO is too long (needs a long transfer sequence), the operation may reduce the robustness of the UIOC for verification.

This paper investigated the problem of fault masking in UIOs and proposed the use of a new type of unique input/output circuit sequence (UIOC) for state verification to overcome the problem. UIOCs themselves are particular types of UIOs where the ending states are the same as their initial states. When constructing a UIOC, by further checking the tail state and by using overlap or internal state observation scheme, the fault types of UIOs discussed in section 3 can be avoided, which makes the UIO more robust. Based on rural Chinese postman algorithm and UIOCs for state verification, a new approach is proposed for generating a more robust test sequence. An approach is also suggested for the construction of B-UIOs. Test performance among F-UIO, B-UIO and F-UIO, and UIOC based methods is compared through a set of experiments. The relative robustness of the UIOCs constructed by the algorithm given in this paper and those constructed by the algorithm given in [23] are also experimentally compared.

The rest of this paper is organised as follows: section 2 presents the basic definitions and notations regarding finite state machines; section 3 discusses the problems associated with F-UIOs and B-UIOs; section 4 introduces the proposed methods that aim to overcome these problems; section 5 presents the experimental results. Finally, conclusions are drawn in section 6.

2 Preliminaries

2.1 Finite State Machines

A deterministic FSM M is defined as a quintuple $(I, O, S, \delta, \lambda)$ where $I, O,$ and S are finite and nonempty sets of input symbols, output symbols, and states, respectively; $\delta : S \times I \longrightarrow S$ is the state transition function; and $\lambda : S \times I \longrightarrow O$ is the output function. If the machine receives an input

$a \in I$ when in state $s \in S$, it moves to the state $\delta(s,a)$ and produces output $\lambda(s,a)$. Functions δ and λ can be extended to take input sequences in the usual way.

Two states s_i and s_j are said to be *equivalent* if and only if for every input sequence $\alpha \in I^*$ the machine produces the same output sequence, i.e., $\lambda(s_i,\alpha) = \lambda(s_j,\alpha)$. Machines M_1 and M_2 are *equivalent* if and only if for every state in M_1 there is a corresponding state in M_2, and vice versa. A machine is *minimal* (*reduced*) if and only if no two states are equivalent. It will be assumed that any FSM being considered is minimal since any (deterministic) FSM can be converted into an equivalent (deterministic) minimal FSM [13]. An FSM is *completely specified* if and only if for each state s_i and input a, there is a specified next state $s_{i+1} = \delta(s_i,a)$, and a specified output $o_i = \lambda(s_i,a)$. Otherwise, the machine is *partially specified*. An FSM is *strongly connected* if, given any ordered pair of states (s_i,s_j), there is a sequence of transition that moves the FSM from s_i to s_j.

It will be assumed throughout this article that an FSM is deterministic, minimal, completely specified, and strongly connected.

2.2 Conformance Testing

Given a specification FSM M, for which we have its complete transition diagram, and an implementation M', for which we can only observe its I/O behaviour ("black box"), we want to test to determine whether the I/O behaviour of M' conforms to that of M. This is called *conformance testing*. A test sequence that solves this problem is called a *checking sequence*. I/O behavioural difference between specification and implementation can be caused by either an incorrect output (an output fault) or an earlier incorrect state transfer (a state transfer fault). The latter can be detected by adding the final state check after a transition test. Three techniques are proposed for state verification: Distinguishing Sequence (DS), Unique Input/Output (UIO) sequence and Characterizing Set (CS). Due to its practical characteristics, UIO is often used for test sequence generation [14]. Two kinds of UIO sequences can be used for state verification. A forward UIO (F-UIO) sequence provides evidence that the FSM starts from a known state while a backward UIO (B-UIO) sequence shows that the FSM arrives at a known state.

Definition 1 *A forward UIO (F-UIO) sequence of state s_i is an input/output sequence x/y, that may be observed from s_i, such that the output sequence produced by the machine in response to x from any other state is different from y, i.e. $\lambda(s_i,x) = y$ and $\lambda(s_i,x) \neq \lambda(s_j,x)$ for any $i \neq j$.*

Definition 2 *A Backward UIO (B-UIO) sequence of state s_i is an input/output sequence x/y, that can be observed only if the ending state of transitions is s_i, i.e. $\forall s_j$ $(s_j \in S)$, $\lambda(s_j,x) = y \implies \delta(s_j,x) = s_i$.*

When only F-UIOs are considered for test sequence generation, the method is called the F-UIO method, or simply F-method. A standard test strategy is:

1. Homing: Move M' to a transition's initial state s;
2. Output Check: Apply an input α and compare the outputs generated by M and M' separately;
3. Tail State Verification: Using state verification techniques to check the final state. It also provides evidence that we are then in the correct state but this is explained below.

The first step is known as homing a machine to a desired initial state. This can be accomplished by using a homing sequence.

Definition 3 *A homing sequence H is an input sequence such that the output sequence on H uniquely determines the state reached after applying H.*

The second step checks whether M' produces the desired output sequence. The last step checks whether M' is in the expected state $s' = \delta(s,\alpha)$. When both F-UIOs and B-UIOs are considered, the method is called the B-UIO method, or simply B-method. The test strategy is then:

1. Homing: Move M' to the initial state of the B-UIO;
2. Initial State Verification: Apply the B-UIO sequence to move M' to the initial state of a transition;
3. Output Check: Apply an input α and compare the outputs generated by M and M' separately;
4. Tail State Verification: Apply F-UIO to check the final state.

The F-UIO is used to provide confidence that the tested transition arrives at a correct state while the B-UIO is used to increase the confidence that an assigned transition has been tested.

2.3 Construction of UIOs

The problem of computing UIOs (F-) is NP-hard [14]. Lee *et al.* [14] noted that F-UIOs can be constructed from some state splitting trees (SSTs). A state splitting tree is a rooted tree that is used to construct adaptive distinguishing sequences or F-UIOs from an FSM. Each node is associated with a set of states. Each node in the tree has a predecessor (parent) and successors (children). A tree starts from a root node and terminates at discrete partitions: sets that contain one state only. The predecessor of the root node, which contains the set of all states, is null. The nodes corresponding to a single state have empty successors. These nodes are also known as terminals. A child node is connected to its parent node through an edge labelled with characters. The edge implies that the set of states in the child node is partitioned from that in the parent node upon receiving the labelled characters. The splitting tree is complete if the partition is a discrete partition. Once a discrete partition is established, input/output sequence labelled on the path from the discrete partition node to the root node constitutes a F-UIO for the state related to this node. Figure 1 illustrates an example of SST where partitions are generated when each state of an FSM responds to an input sequence *aba* respectively.

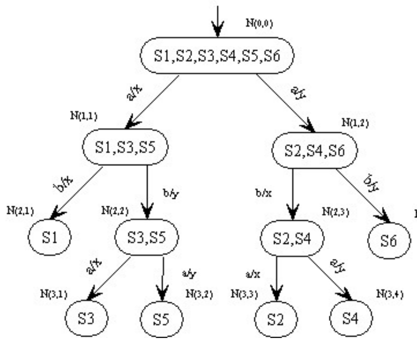

Figure 1: A state splitting tree from an FSM

However, finding data to construct SSTs is also NP hard. Using metaheurisitic optimisation techniques (MOTs) such as genetic algorithms (GAs) and simulated annealing (SA) for the construction of UIOs (F-) is therefore suggested in ref. [6] and [7].

3 Problems of F- and B-Methods

3.1 Problems of F-Method

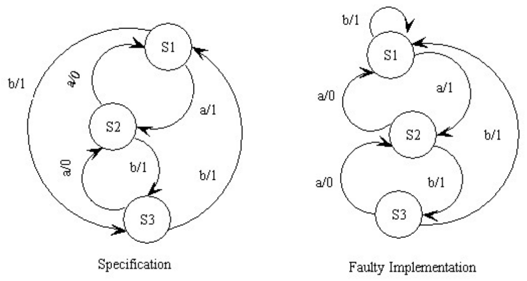

Figure 2: An FSM and its faulty implementation copied from [3]

UIOs uniquely identify states in the specification FSM. The F-method is based on the assumption that UIOs generated from a specification FSM can uniquely identify the corresponding states in the IUT as well. This assumption is however not always true. A faulty example is copied from [3] shown in Figure 2. In the specification FSM, sequence $(b/1)(a/1)$ is a UIO for s_3. However, in the faulty implementation, s_1 and s_3 produce the same output (11) when responding to *ba*. The UIO loses its property of uniqueness in the IUT and fails to identify s_3. The problem is called *fault masking in UIOs*. The capability of a UIO to resist this problem is called its *strength* [18]. In UIO based test methods, the use of UIOs with low strength may lead to a test sequence that is not robust.

3.2 Problems of B-Method

A B-UIO provides evidence that an FSM is currently in a known state, but does not show from which state it initially comes. A B-UIO may have several valid initial states that satisfy the definition of this B-UIO. An example is shown in Figure 3 where the FSM (Table 1) is defined in section 5. Sequence *dccd* is a B-UIO for s_0. It can be seen that the B-UIO sequence has 4 initial states (s_1, s_4, s_6, s_9) that satisfy the definition. If a B-UIO has more than one valid initial state and is chosen for a transition test, it is possible that a fault that occurred in the previous transition test

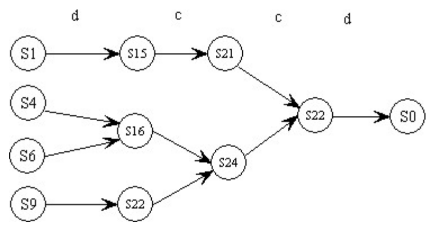

Figure 3: "dccd/yyyy": BUIO of s_0 in the FSM defined in table 1

is masked by this B-UIO. An example is illustrated

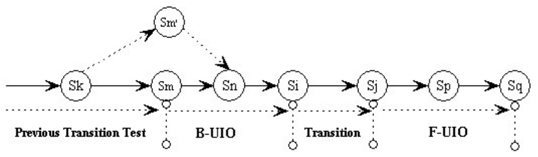

Figure 4: Problem of B-UIOs

in Figure 4 where the test segment for transition $s_i \rightarrow s_j$ is formed by concatenating the input part of the B-UIO for s_i with input of this transition and the input part of the F-UIO for s_j. Suppose the B-UIO sequence chosen for s_i has more than one valid initial state while s_m and $s_m^{'}$ are both valid ones and, according to the specification FSM, the previous transition test should end up with s_m. If, in a faulty implementation, the tail state of the previous transition test happens to be in $s_m^{'}$, the selected B-UIO for s_i will automatically mask this fault. This makes the test sequence less likely to detect this faulty implementation.

It can be noted that the fault masking problems described in the F-method may happen in the B-method as well since in the B-method, a transition test consists of a part that uses the F-UIO for the tail state verification. However, since the B-method not only verifies the tail state of a transition, but also checks the initial state, the robustness of a test sequence can be enhanced.

4 Improving Fault Coverage Using UIOCs

Lombardi *et al.* [16] formalises the faulty implementations of UIOs into two basic types. In this section, we proposed solutions to overcome the associated problems.

4.1 Basic Fault Types

Two basic fault types are formalised in Figure 5. In type 1, the tail state of the UIO in the implementation is different from that in the specification while in type 2 the UIO and its faulty implementation have an identical ending state. The following explains how the faults are masked.

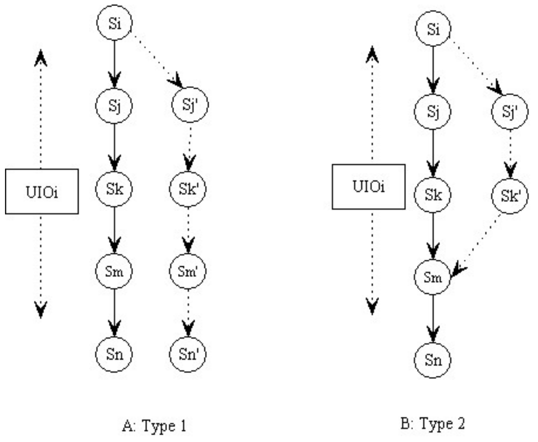

Figure 5: Types of faulty UIO

Suppose $i_1i_2i_3i_4$ is the input sequence from a UIO for s_i. When the FSM is in s_i and receives $i_1i_2i_3i_4$, it produces $o_1o_2o_3o_4$, visiting s_j, s_k, s_m and s_n correspondingly. In type 1, a fault is caused by an erroneous state transfer $s_i- > s_j^{'}$ that has the same I/O as $s_i- > s_j$. Instead of being in s_j, the FSM arrives at $s_j^{'}$. If the following outputs are $o_2o_3o_4$, these outputs are then masking the state transfer fault. Suppose the following transition test is for a transition $T : s_n- > s_x$. If there exists another transition $T^{'} : s_n^{'}- > s_x$ that has the same I/O behaviour as T, and, rather than T, the transition $T^{'}$ is tested, the test sequence will therefore be unable to detect the fault in T.

In type 2, an erroneous transition $s_i- > s_j^{'}$ that has the same I/O as $s_i- > s_j$ occurs. This fault is masked first by an output o_2 and then by another erroneous transition $s_k^{'}- > s_m$ that has the same I/O behaviour as $s_k- > s_m$.

It can also be noted that the faulty implementations described in F-UIOs can be extended to that in B-UIOs by considering the tail states as initial states.

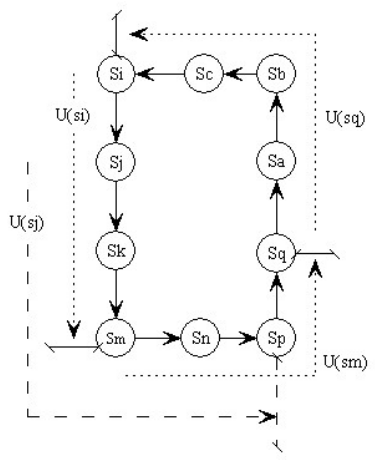

Figure 6: Construction of a UIOC sequence

4.2 Overcoming Type 1

In type 1, the ending state of a faulty UIO is different from that of its specification, and so one way to detect this error is to further verify it. This is illustrated in Figure 6. Suppose, according to its specification, an FSM should be in s_m after applying a UIO sequence U_{s_i} for s_i. To check whether the FSM is in s_m, a UIO sequence U_{s_m} for s_m is then applied, moving the FSM to s_q. If the input/output behaviour is identical to that described in the specification, it then provides evidence that the ending state of U_{s_i} (s_m) is correct. A question then arises: how can we be sure that the FSM is in s_q? A UIO sequence for s_q can be used to further check it. The procedure of repeating the verification for the ending state of a UIO gives evidence that all previous UIO sequences make the FSM arrive at correct ending states. However, the procedure of verifying the ending states of UIO sequences should terminate. UIOs should construct a unique input/output circuit (UIOC) to terminate the verification. The following will give a detailed explanation of the control scheme (Figure 6).

Suppose U_{s_i} is a F-UIO for s_i and its ending state is s_m. By applying a F-UIO U_{s_m} for s_m, evidence is given, indicating that the FSM is previously in s_m. Continuing to apply F-UIO U_{s_p} for s_p, the I/O behaviour therefore provides evidence that the FSM arrives at s_q. Suppose that there exist a F-UIO U_{s_q} for s_q with tail state s_i, the application of U_{s_q} provides evidence for the correct arrival of s_q. The structure of the UIOC shows that, if the input sequence is executed more than one times, the I/O behaviour of U_{s_i} will repeat, which provides evidence for the correct arrival of s_i. Thus, by constructing UIOC, each UIO provides evidence of the correct arrival of its previous UIO's tail state.

The control scheme shown in Figure 6 is an ideal situation. In some applications, the complete UIOC may not be constructed. For example, instead of terminating at s_i, the last UIO sequence in the UIOC may make the FSM arrive at s_c, forming a gap between the tail state and s_i. When dealing with this situation, a shortest transfer sequence can be considered since the extended sequence of a UIO for a state is still a UIO. Meanwhile, when constructing UIOC, UIOs that result in a minimal gap between the tail state of the last UIO and s_i need to be considered. For instance, if there are two sets of UIOs where the first set moves the FSM to s_c while the other moves the FSM to s_b, the first set of UIOs is better than the other if the gap between s_c and s_i is shorter than that between s_b and s_i.

The UIOC can be constructed by using B-UIOs as well. Suppose, in Figure 6, U_{s_i}, U_{s_m} and U_{s_q} are B-UIOs for s_m, s_q and s_i correspondingly, then U_{s_q} provides evidence that the FSM is in s_i (U_{s_i} starts from s_i), U_{s_m} provides evidence that the FSM is in s_q (U_{s_q} starts from s_q) and U_{s_i} provides evidence that the FSM is in s_m (U_{s_m} starts from s_m). Therefore, in a UIOC constructed by B-UIOs, each B-UIO provides evidence for the next B-UIO's initial state.

The advantages of using B-UIOs are that: 1. In a deterministic FSM, each state has at least one B-UIO; 2. A minimal FSM with n states has a homing sequence of length $O(n^2)$ that can be constructed in time $O(n^3)$[13], and B-UIOs can be derived from homing sequences by concatenating the input part of the homing sequence with a transfer sequence that moves the FSM from the tail state of a homing sequence to the target state.

Proposition 1 *Given a deterministic, reduced and strongly connected finite state machine M, there exists at least one B-UIO sequence for each state of M.*

Proof: In a minimal deterministic FSM, there exists at least one homing sequence H [13]. Suppose, when responding to H in some state s, the FSM ends up at state s_i and produces $H_{(o)}$, $H/H_{(o)}$ is a B-UIO for s_i. Given a state s_j, $s_i \neq s_j$, there exists an I/O sequence $L/L_{(o)}$ that moves the FSM from s_i to s_j since the FSM is strongly connected. I/O sequence $HL/H_{(o)}L_{(o)}$ is a B-UIO for s_j. □

Proposition 2 *In a deterministic finite state machine, if an I/O sequence L_i/L_o is a F-UIO for s_i such that $s_j = \delta(s_i, L_i)$ and $L_o = \lambda(s_i, L_i)$, then L_i/L_o is also a B-UIO for s_j with one valid initial state.*

Proof: Suppose L_i/L_o is a F-UIO for s_i and, when responding to L_i, the FSM arrives at s_j. Since the FSM is deterministic, s_j is the only state reachable by L_i/L_o from s_i. Since L_i/L_o is the F-UIO for s_i, L_i/L_o is a B-UIO for s_j with one valid initial state. □

However, the UIOCs constructed by a complete set of B-UIOs may not be UIOs (F-). Therefore, before choosing the UIOCs for state verification, the uniqueness of the I/O sequences need to be checked. Only input/output sequences that form F-UIOs are used.

A UIOC can be constructed by using both F-UIOs and B-UIOs. It can be seen that the sequence formed by concatenating a F-UIO (head state is s_F) with a B-UIO (tail state is s_B) is a F-UIO for s_F and a B-UIO for s_B. In [23] a UIOC was also used for state verification. A UIOC was constructed by using a F-UIO and B-UIO for a state. If the F-UIO and B-UIO do not naturally form a circuit (the tail state of the F-UIO is not the initial state of the B-UIO), a short transfer sequence is applied to complete it. It can be noted that, if the F-UIO and B-UIO form a circuit, the UIOC in [23] is identical to that in this work. The construction of UIOCs in [23] can be viewed as a special case of this work. However, if there exists a gap between the tail state of the F-UIO and the initial state of the B-UIO, a short transfer sequence has to be applied to complete the circuit. This may degrade the robustness of the UIOC.

This paper proposes a new method for generating UIOCs where the F-UIO and the B-UIO do not form a circuit. In section 5, we report the result of an experiment devised to compare the relative robustness of UIOCs that were constructed by the methods given in this paper and in [23]. The experimental results show that the UIOCs constructed according to the algorithm given in this paper are more robust than those constructed by the algorithm given in [23].

The use of UIOCs for state verification will increase the length of a test sequence. When constructing a UIOC, if there exists more than one set of F-UIOs or B-UIOs that can form UIOCs, the set with the least number of elements should be used to get a UIOC with the minimal length. Meanwhile, overlap among UIOs needs to be considered as well to further reduce the length.

4.3 Overcoming Type 2

4.3.1 Overlap Scheme

In type 2, a transfer fault may be masked by another transfer error. When constructing a UIOC, the consideration of overlaps among UIOs for internal states can help to overcome this problem. For example, in Figure 6, U_{s_i}, U_{s_m} and U_{s_p} form a UIOC for s_i. If there exists U_{s_j} in the circuit that is a UIO for s_j, the chance that the UIOC fails to find type 2 fault can be reduced. If F-UIOs for all internal state's UIOs, say s_j, s_k and s_m, are included in the UIOC, then the chance to fail to detect type 2 will be reduced.

4.3.2 Internal State Sampling Scheme

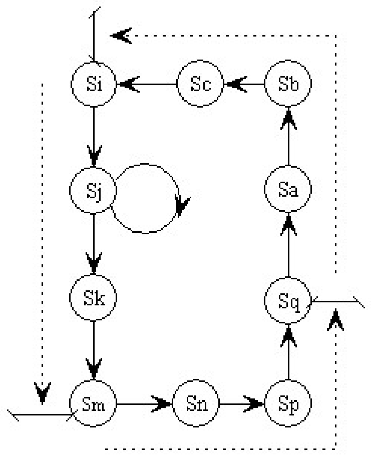

Figure 7: Sample an internal state

When constructing a UIOC for a state, internal states may not be verified. An alternative way to overcome type 2 is then to check internal states by adding additional observers that are self-loops. Figure 7 shows the scheme where s_j is checked. In a faulty implementation, an observer may either make the FSM produce an erroneous output or arrive at an erroneous state that may be detected by the following verification. This helps to increase confidence that UIOCs constructed from specification FSM remains UIO (F-) in the IUT. Ideally, the observers are F-UIOs that are naturally loops. But if the F-UIO is too long, a shortest loop could be substituted for the function.

Definition 4 *A loop sequence LS_i/LS_o for s_i is an I/O sequence such that $s_i = \delta(s_i, LS_i)$ and $LS_o = \lambda(s_i, LS_i)$.*

It would make the verification more robust if all internal states are checked by their observers. However, this will make the test sequence very long. Thus, instead of checking all, one state is selected for verification by the scheme shown in Figure 8. Suppose input sequence U_{s_i} is a F-UIO for state s_i. When responding to U_{s_i}, the FSM produces an output sequence O_{s_i} and gives a trace of s_j, s_k, s_l, $s_i^{'}$. Putting U_{s_i} to all other states, we get output sequences and traces correspondingly. Suppose, by comparing the output sequences, a common I/O area is found shown between two dotted lines. The area is said to be a highly dangerous area. State transfer error is likely to be masked in such an area. For example, $s_i \rightarrow s_j$ might be replaced by $s_i \rightarrow s_p$. This mistake might be masked later by $s_x \rightarrow s_i^{'}$ or by $s_q \rightarrow s_l$. A state between s_j and s_l needs to be further checked by its observer. The middle state, namely s_k, is considered.

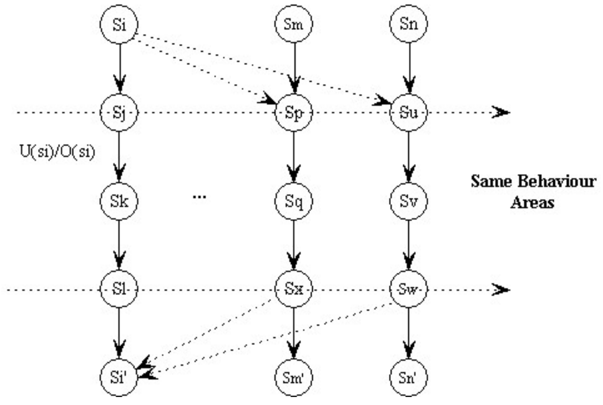

Figure 8: Rule on selection of an internal state

4.4 Construction of B-UIOs

B-UIOs might be considered for the construction of UIOCs. However, the reviewed literatures do not give a complete algorithm to construct B-UIOs. Although homing sequences can be applied as bases for the construction, the B-UIOs obtained might be long, which will increase the cost in the forthcoming test. Based on the studies of SST, we propose State Merging Tree (SMT) for the construction of B-UIOs.

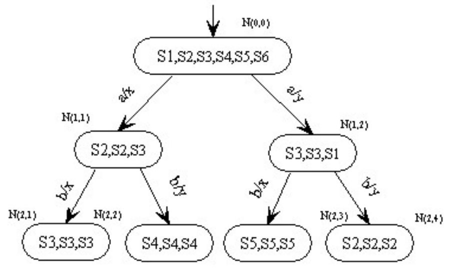

Figure 9: A state merging tree

Similar to a state splitting tree, a SMT is a rooted tree. Each node in a SMT contains a set of states where the root node contains the complete set of states and the discrete nodes (terminals) contain one state. A node is connected to its parent by an edge labelled with characters, indicating the situation of state merging. However, differences exist between SST and SMT where at each single input stage, the SST only cares about the initial state from which the current state came while the SMT takes not only the initial state, but also the current state into account. To give a further explanation, an example is shown in Figure 9 where the FSM has 6 states, the input set is $\{a, b\}$, and the output set is $\{x, y\}$. Suppose, when responding to a, $\{s_1, s_3, s_5\}$ produce x and arrives at $\{s_2, s_2, s_3\}$ [1], while $\{s_2, s_4, s_6\}$ produce y and arrive at $\{s_3, s_3, s_1\}$. a is said to merge s_1 and s_3 at s_2 by producing x and to merge s_2 and s_4 at s_3 by producing y. Two nodes are generated from the root node indicated by $N(1,1)$ and $N(1,2)$. Continuing to input the FSM with b, if states reached from $\{s_1, s_3, s_5\}$ by a arrive at $\{s_3, s_3, s_3\}$ producing x or arrive at $\{s_4, s_4, s_4\}$ producing y, ab is said to merge $\{s_1, s_3, s_5\}$ at s_3 by producing xx or $\{s_1, s_3, s_5\}$ at s_4 by producing xy. Two nodes rooted from $N(1,1)$ are then generated indicated by $N(2,1)$ and $N(2,2)$. Once a discrete node such as $N(2,1)$ occurs, the path from the root node to $N(2,1)$ forms a B-UIO for the corresponding state (in this case, s_3). By the nature of the tree, s_3 is the only state that can be reached by this input/output sequence. If all of the terminal nodes are discrete nodes and all the input sequences defined by paths from the root node to terminal nodes are the same input sequence x then x is a homing sequence.

It can be seen that one SMT may not contain B-UIOs for the complete set of states. It may be necessary to generate several SMTs to provide the

[1]Both s_1 and s_3 arrives at s_2. The set of ending states is actually $\{s_2, s_3\}$. However, to make the explanation clearer, all final states remain listed

B-UIOs for every state. Models proposed in [6, 7] for the construction of F-UIOs can be extended for the construction of B-UIOs by examining the patterns of SMTs.

4.5 Construction of UIOCs

The construction of UIOCs in this paper followed three schema: 1. A complete set of F-UIOs was used; 2. A mixed set of F-UIOs and B-UIOs was used. 3. A complete set of B-UIOs was used. When constructing a UIOC, the first scheme is considered. If a UIOC cannot be constructed by a complete set of F-UIOs, the second scheme will be considered. Only when the first and the second schema fail to construct a UIOC, will the third scheme be considered. All three schema should take the overlap or the internal state observation scheme into account. When the internal state observation scheme is used, a self-loop F-UIO (if the observed state has one) is first considered. However, if the F-UIO is too long, a loop sequence is substituted for the function. In this paper, if the length of a self-loop F-UIO of a state is greater than 4, a shorter loop sequence is then used for the observation of this state.

When the B-UIOs are used for the construction of a UIOC, those B-UIOs with fewer number of valid initial states should be used to avoid the fault masking problem caused by B-UIOs. When a UIOC is constructed by a complete set of B-UIOs, the UIOC may not be a UIO (F-). Therefore, before the UIOC is selected for the state verification, its uniqueness needs to be checked to make sure that it is a F-UIO.

5 Simulation

A set of experiments was devised to compare the test performance among F-, B-, and C-Method. In all our experiments, we are intended to use FSMs where the size of the state set and the input set are much higher than that of the output set. The structure of the FSMs could make the test harder. We believe that, in these kinds of FSMs, the UIOs for each state tend to be long and the problems of fault masking are likely to happen.

A randomly generated FSM is first defined in [2]table 1. The system has 25 states while the input set is $\{a, b, c, d\}$ and the output set is $\{x, y\}$. The FSM is reduced, deterministic and completely specified. There are $4 \times 25 = 100$ transitions. A mutant (faulty implementation) is generated by modifying a transition. The selected transition is changed either on its output or the ending state. There are $100 \times 24 + 100 = 2,500$ mutants. Test sequences are generated with the F-, the B- and the (new) C-method separately and then used to check all these mutants. To make the explanation clear, the test sequences generated with the F-, the B- and the C-method are called F-, B- and C- sequence correspondingly. By the end of experiment, we found that 199 mutants passed the F-sequence, 3 passed the B-sequence and none of them passed the C-sequence. This result suggests that the B-method is better than the F-method, which is consistent with the work of [22].

	a	b	c	d
s_0	s_3/y	s_{12}/y	s_{10}/x	s_{24}/x
s_1	s_4/x	s_{11}/x	s_{23}/y	s_{15}/y
s_2	s_{17}/x	s_9/x	s_{14}/y	s_0/y
s_3	s_5/x	s_0/y	s_{13}/x	s_{23}/y
s_4	s_{20}/x	s_{18}/x	s_{15}/y	s_{16}/y
s_5	s_3/y	s_1/x	s_{12}/x	s_{20}/y
s_6	s_1/y	s_7/x	s_{19}/x	s_{16}/y
s_7	s_{21}/x	s_{24}/y	s_9/y	s_6/x
s_8	s_{14}/y	s_{12}/y	s_{18}/x	s_5/x
s_9	s_{15}/x	s_2/y	s_6/y	s_{22}/x
s_{10}	s_{11}/x	s_{19}/y	s_{23}/y	s_8/x
s_{11}	s_{17}/x	s_{12}/x	s_0/y	s_6/y
s_{12}	s_9/x	s_{13}/y	s_{20}/x	s_1/y
s_{13}	s_7/x	s_4/y	s_{10}/x	s_{22}/y
s_{14}	s_8/x	s_3/y	s_{19}/y	s_{11}/x
s_{15}	s_6/x	s_{17}/x	s_{21}/y	s_2/y
s_{16}	s_7/y	s_{20}/y	s_{24}/x	s_4/x
s_{17}	s_{15}/y	s_{13}/x	s_2/x	s_8/y
s_{18}	s_{16}/x	s_5/y	s_{20}/x	s_{10}/y
s_{19}	s_{23}/x	s_{11}/y	s_9/y	s_{18}/x
s_{20}	s_{14}/y	s_{21}/x	s_{17}/y	s_7/x
s_{21}	s_4/x	s_{16}/x	s_{22}/y	s_1/y
s_{22}	s_{10}/x	s_2/x	s_{24}/y	s_0/y
s_{23}	s_{18}/y	s_{21}/x	s_{13}/x	s_3/y
s_{24}	s_{14}/y	s_5/y	s_{22}/x	s_8/x

Table 1: Specification FSM

However, there are still 3 mutants that passed the B-sequence but were found by the C-sequence. Six examples of faulty implementations where 3 passed the F-sequence and 3 passed B-sequence

[2]Contents in the first row are inputs. s_i: s_j/y means that, when the FSM is in s_i and receives an input shown in the first row, it moves to s_j and produces y

Methods	Specification	Mutants
F-	$s_4 - (a/x) \rightarrow s_{20}$	$s_4 - (a/x) \rightarrow s_7$
	$s_5 - (a/y) \rightarrow s_3$	$s_5 - (a/y) \rightarrow s_{23}$
	$s_{24} - (d/x) \rightarrow s_8$	$s_{24} - (d/x) \rightarrow s_0$
B-	$s_{18} - (c/x) \rightarrow s_{20}$	$s_{18} - (c/x) \rightarrow s_8$
	$s_{18} - (c/x) \rightarrow s_{20}$	$s_{18} - (c/x) \rightarrow s_{24}$
	$s_{23} - (d/y) \rightarrow s_3$	$s_{23} - (d/y) \rightarrow s_{23}$

Table 2: Examples of faulty implementations that F- and B-methods fail to detect

are shown in Table 2. This result suggests that the test sequence generated with the C-method is more robust than that with the B-method. We also compared the lengths of the test sequences. They are 506 (F-sequence), 915 (B-sequence) and 1015 (C-sequence). Comparing to the F-method, the B-method increases the length roughly by 45% while the C-sequence is approximately 10% longer than the B-sequence. We then increased the length of F-sequence and B-sequence to 1015 by adding input characters that were randomly selected from the input set. The experiment was repeated 10 times and the best result was selected. The final result showed that 89 mutants passed the extended F-sequence, and both the extend F- and B- sequences failed to find the three mutants that passed the test in the previous experiment. This experiment suggested that, although extending the F-sequence to a certain length may help to improve its ability to find more errors, it is still less robust than the B-sequence and the C-sequence.

FSM	F-Num	C-Num
$s_{18} - (c/x) - > s_{20}/s_8$	3	0
$s_{18} - (c/x) - > s_{20}/s_{24}$	3	0
$s_{24} - (d/x) - > s_8/s_0$	5	0
$s_5 - (a/y) - > s_3/s_{23}$	5	0
$s_5 - (c/x) - > s_{12}/s_{18}$	5	0

Table 3: Numbers of F-UIOs and UIOCs that lost the property of uniqueness in the faulty implementations

The numbers of F-UIOs and UIOCs that lost the property of UIOs in the faulty implementations was also studied. Five mutants were chosen for the experiment. Test results are shown in Table 3 where $s_{18} - (c/x) - > s_{20}/s_8$ indicates that a mutant was generated by changing the ending state s_{20} to s_8 while F-Num and C-Num show the numbers of F-UIOs and UIOCs that are no longer UIOs in the faulty implementations. From the table it can be seen that no UIOCs lost the property of UIOs but some F-UIOs did. The experiment suggested that the UIOCs provided by the algorithm given in this paper are more robust than F-UIOs.

An experiment was designed to compare the UIOC (constructed by the algorithm in this paper) with the F-UIO that are constructed by two shortest F-UIOs where one is used to verify the state under the test while the other to verify the tail state of the previous F-UIO. Experimental result showed that 47 mutants passed the sequence generated with the latter scheme. Comparing to the F-sequence and the randomly extended F-sequence, the test sequence generated by using two F-UIOs finds more faults, but is still less robust than the test sequence generated with C-method.

Next, the robustness of UIOCs that were constructed with different schema was compared. Four sets of UIOCs were used. One set was constructed by F-UIOs or B-UIOs, taking the overlap scheme or the internal state observation scheme into account. When only B-UIOs were used to construct a UIOC, the uniqueness of the UIOC was checked to make sure that it is a UIO (F-); one set was constructed by a complete set of F-UIOs, without using the overlap or the internal state observation scheme; one set was constructed by using F-UIOs and B-UIOs that can naturally form circuits; the last was constructed by using F-UIOs and B-UIOs that cannot form circuits. A set of transfer sequences was added to complete the circuits. The experimental showed that 4 mutants passed the UIOC sequence generated using F-UIOs, B-UIOs and transfer sequences; 1 passed the UIOC sequences generated using a complete set of F-UIOs without using overlap or internal observation scheme; 1 passed the UIOC using F-UIOs and B-UIOs without using overlap or internal state observation scheme. The latter two sequences failed to find the same mutant. It can be seen that test sequence generated using F-UIOs, B-UIOs and a set of transfer sequences showed even worse test performance than those generated with B-method. The result suggested that UIOCs constructed by the algorithm given in [23] (using transfer sequences to complete the circuits) are less robust than those that were constructed by the algorithm given in this paper. Comparing to the test results of corresponding test sequences, it can be suggested that the use of the overlap or the internal state observation scheme is likely to make the UIOCs more robust.

We also investigated the test performance of the F-UIO, B-UIO and UIOC methods when applied to FSMs with different numbers of states. All FSMs are randomly generated. They are completely specified, deterministic and strongly connected. The input set and the output set for all FSMs are $\{a, b, c, d\}$ and $\{x, y\}$. All UIOCs constructed by a complete set of B-UIOs are verified to be F-UIOs. The experiment result is shown in Table 4. The table shows no significant relationship between the number of states and the number of mutants that passed the test, which indicates that the quality of testing is not only determined by the test method, but also determined by the structure of systems. But it can still be seen that, for all FSMs tested, test sequences generated with the C-method is better than or equal to others. In the experiments, there are $20 \times 4 + 20 = 100$, 400, 900, 1600, 2500, 3600, 4900 and 6400 mutants in the FSMs with 5, 10, 15, 20, 25, 30, 35 and 40 states respectively. Therefore, the total number of mutants in the experiments is 30390. Three mutants passed the C-sequences, which implies that C-method achieves 99.99% fault coverage in the experiments.

States	F-method	B-method	C-method
5	0	0	0
10	1	1	1
15	1	1	0
20	17	8	0
25	199	3	0
30	0	0	0
35	156	8	1
40	9	2	1

Table 4: Mutants that pass the tests

The mutants that passed the C-sequences from different systems were also checked with their specification FSMs and found that they were not equivalent to the specification FSMs. The mutant that passed the C-sequence in the FSM with 10 states was studied. In the implementation, transition tr_{14}=$s_3-(c/y)->s_4$ is mutated by changing s_4 to s_3. All UIOC sequences generated from the specification FSM were then checked for the uniqueness in the IUT. When applying the input part of the UIOC sequence for s_4, we found bother s_3 and s_4 produced the same output. Thus, UIOC sequence for s_4 loses the uniqueness in the IUT and fails to identify s_4. Figure 10 shows the sequences of transitions traversed by this UIOC sequence in the specification FSM and the faulty implementation

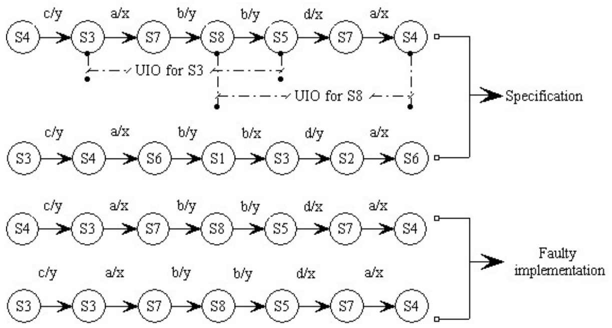

Figure 10: UIOC sequence for s_4 in the FSM with 10 states and the faulty implementation that causes the fault masking in the UIOC sequence

respectively. By examining the structures of all UIOC sequences, we found that none of the UIOC sequence traverses transition tr_{14}. This determines that the faulty implementation of tr_{14} is less likely to be detected in the stages of internal state verification, which reduces the chance on finding this error. From figure 10 it can also be noted that faulty implementation of tr_{14} ends up at the state that is exactly the first state, s_3, that the UIOC sequence traverses when it is applied to verify s_4. If the faulty implementation of tr_{14} ends up at another internal state of the UIOC sequence, the fault may be detected by the internal state verification. However, the fault occurs before the process of internal state verification starts. Since the tr_{14} holds the same I/O behaviour as that of transition $s_4-(c/y)->s_3$, the fault is likely to be masked in the C-sequence. This work did not provide solutions to overcome the problem. Future work will address it.

States	F-method	B-method	C-method
5	88	160	98
10	198	338	351
15	290	484	502
20	398	687	670
25	506	915	1015
30	684	1137	1123
35	740	1315	1339
40	857	1521	1568

Table 5: Lengths of the test sequences

The lengths of test sequences generated with F-, B- and C-methods were also compared. Table 5 shows the lengths of test sequences for different systems. It can be seen that the sequences generated with the F-method are always shorter than that with the B- and the C-method. However, test sequences generated with the C-method are not always longer than that with the B-method. In the majority of studies, the C-sequences were slightly longer than the B-sequence while in some cases such as the FSM with 20 states the C-sequence was shorter than the B-sequence.

6 Conclusion

This paper investigated the problem of fault masking in UIOs. Based on the work of [16], two basic types of fault masking involving UIOs were formalised. A new type of UIOC was proposed to overcome the two fault types. When constructing a UIOC sequence, by further checking the tail state of a UIO, type 1 in the faulty implementation may be avoided while by introducing the overlap scheme and internal state observation scheme, type 2 may be avoided. The procedure of verifying the ending states of UIOs was terminated by the construction of circuits where every UIO provides evidence for the correctness of the previous UIO's tail state.

A set of experiments was designed to study the test performance. Experimental results showed that many more faulty implementations passed the F-sequence than the B-sequence. This suggested that the B-method was more robust than the F-method, which is consistent with the work in [22]; meanwhile, in the experiment, no faulty implementation passed the C-sequence, which suggested that the C-method is more robust than the F-method and the B-method.

Performance of UIOCs constructed by the algorithm given in this paper and in [23] was also compared. Experimental results showed that UIOCs constructed by the algorithm given in [23](using transfer sequence to complete a circuit) were less robust than those constructed by the algorithm given in this paper. Experimental results also suggested that the use of the overlap or internal state observation scheme is likely to make the UIOCs more robust. In this work, internal state observation scheme considered the sampling of one internal state of a UIO sequence. If a UIO sequence is comparatively long, in order to increase the test confidence, more than one state might be considered for observation. In future work, more studies will be proceeded to investigate the effectiveness of internal sampling scheme.

A set of FSMs was devised to compare the test performance among different methods. Experimental results showed that the (new) C-method was consistently better than or equal to the F-method and the B-method. In the devised experiments, the (new) C-method achieved more than 99.99% fault coverage. It also showed that the C-sequences were not always longer than the B-sequences. In the majority of studies, the C-sequences were slightly longer than the B-sequence while in some cases, the C-sequences were shorter than the B-sequences.

However, it has also be noted that a few of faulty implementations passed the C-sequences in the experiments. More work needs to be carried on to study the facts that cause the failure of C-sequences.

References

[1] I. Ahmad, F.M. Ali, and A.S.Das, "LANG - algorithm for constructing unique input/output sequences in finite-state machines", *IEE Proceedings - Computers and Digital Techniques*, vol. 151, pp.131-140, 2004.

[2] A.V.Aho, A.T.Dahbura, D.Lee and M.U.Uyar, "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours", *IEEE Transaction on Communications*, vol.39, No.3, pp.1604-1615, 1991.

[3] W.Y.L.Chan, S.T.Vuong, and M.R.Ito, "An improved protocol test generation procedure based on UIOs", *ACM SIGCOMM Computer Communication* , Vol.19, No.4, pp.283-294, 1989.

[4] T.S.Chow, "Testing Software Design Modelled by Finite State Machines", *IEEE Transactions On Software Engineering*, Vol. 4, No.3, pp. 178-187, 1978.

[5] R.Dssouli, K.Saleh, E.Aboulhamid, A.En-Nouaary and C.Bourhfir, "Test development for communication protocols: towards automation", *Computer Networks*, Vol.31, No.17, pp.1835-1872, 1999.

[6] Q.Guo, R.M.Hierons, M.Harman and K.Derderian, "Computing unique input/output sequences using genetic algorithms", *3rd International Workshop on Formal Approaches to Testing of Software (FATES2003), in LNCS*, Vol. 2931, pp. 164-177, 2004.

[7] Q.Guo, R.M.Hierons, M.Harman and K.Derderian, "Constructing multiple unique input/output sequences using metaheuristic optimisation techniques", *IEE Proceedings - Software*, Vol. 152, No. 3, pp. 127-140, June 2005.

[8] G.Gonenc, "A method for the design of fault detection experiments", *IEEE Transaction on Computer*, Vol. C-19, pp.551-558, June 1970

[9] F.C.Hennie, "Fault Detecting Experiments for Sequential Circuits", *Proceedings of the Fifth Annual Switching Theory and Logical Design Symposium*, Princeton, New Jersey, pp.95-110, 1964

[10] R.M.Hierons, "Extending Test Sequence Overlap by Invertibility", *The Computer Journal*, Vol.39, No.4, pp.325-330, 1996.

[11] R.M.Hierons, "Testing From a Finite State Machine: Extending Invertibility to Sequences", *The Computer Journal*, Vol.40, No.4, pp.220-230, 1997.

[12] R.M.Hierons and H.Ural, "Reduced Length Checking Sequences", *IEEE Transactions on Computers*, Vol.51, No.9, pp.1111-1117, September 2002.

[13] Z.Kohavi, "Switching and Finite Automata Theory", *New York: McGraw-Hill*, 1978.

[14] D.Lee and M.Yannakakis, "Testing Finite State Machines: State Identification and Verification" *IEEE Transactions on Computers*, vol.43, No.3, pp.306-320, 1994.

[15] J.J.Li and W.E.Wong, "Automatic test generation from communicating extended finite state machine (CEFSM)-based models" *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp.181-185, 2002.

[16] F.Lombardi and Y.-N.Shen, "Evaluation and Improvement of Fault Coverage of Comformance Testing by UIO Sequences" *IEEE Transactions on Communications*, vol.40, No.8, pp.1288-1293, 1992.

[17] R.E.Miller and S.Paul, "On the Generation of Minimal-Length Conformance Tests for Communication Protocols" *IEEE/ACM Transactions on Networking*, Vol.1, No.1, Feb, 1993.

[18] K.Naik, "Fault-tolerant UIO Sequences in Finite State Machines", *Proceedings of the IFIP WG6.1 TC6 Eight International Workshop on Protocol Test Systems*, pp.201-214, 1995.

[19] A.Rezaki and H.Ural, "Construction of Checking Sequences Based On Characterization Sets", *Computer Communication*, Vol.18, pp.911-920, 1995

[20] D.P.Sidhu, and T.K.Leung, "Formal Methods for Protocol Testing: A Detailed Study" *IEEE Transactions on Software Engineering*, Vol.15, No.4, pp.413-426, April 1989.

[21] Y.N.Shen, F.Lombardi, and A.T.Dahbura, "Protocol Conformance Testing Using Multiple UIO Sequences", *IEEE Transactions on Communications*, Vol.40, No.8, pp.1282-1287, 1992.

[22] X.J.Shen, S.Scoggins, and A.Tang, "An Improved RCP-method for Protocol Test Generation Using Backward UIO sequences", Proceedings of ACM Symposium on Applied Computing (SAC 1991), pp.p284-293, 1991.

[23] X.J.Shen and G.G.Li, "A new protocol conformance test generation method and experimental results", Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's, pp.75-84, 1992.

[24] H.Sun, M.Gao and A.Liang, "Study on UIO sequence generation for sequential machine's functional test", *Proceedings of the 4th International Conference on ASIC*, pp.628 - 632, 23-25 Oct. 2001.

[25] H.Ural, X.L.Wu, and F.Zhang, "On Minimizing the Lengths of Checking Sequences", *IEEE Transactions on Computers*, Vol.46, No.1, pp.93-99. January 1997.

[26] B.Yang and H.Ural, "Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlapping" *ACM SIGCOMM 90: Communications, Architectures, and Protocols*, Twente, The Netherlands, Sep.24-27, pp.118-125. North-Holland, The Netherlands.