Guaranteeing Full Fault Coverage for UIO-Based Methods

Ricardo Anido *Ana Cavalli*

Relatório Técnico DCC–95-06

Junho de 1995

Guaranteeing Full Fault Coverage for UIO-based Testing Methods

Ricardo Anido[1]
Ana Cavalli
{ranido, anna}@hugo.int-evry.fr
Institut National des Télécommunications
9, rue Charles Fourier, 91011 Evry Cedex, France

Abstract. This paper presents an analysis of the fault coverage provided by the UIO-based methods for testing communications protocols. Formal analysis of the fault coverage for the non-optimized method and for some of its optimized versions are presented. A test is said to provide *full coverage* if no erroneous implementation can pass the test. In the case of optimizations based on the Rural Chinese Postman Tour [1] it is shown that unless certain conditions are met the method does not guarantee full fault coverage, even when, as suggested in [3], the uniqueness of UIO sequences (or Partial UIO sequences) are verified in the implementation. The result of the analysis suggests how the existing methods for generating test sequences should be changed in order to guarantee full fault coverage.

Keywords: communication protocols, conformance testing, verification, finite state machines, test generation, test coverage.

1.0 Introduction

The use of a precise set of communication rules, called a *protocol*, is essential for the design and implementation of distributed systems and communication networks. A protocol defines all possible interactions among the components of the system. After the system has been implemented, the protocol implementation must be verified to conform to its specification, to ensure that the system will operate correctly. This procedure is known as conformance testing, and can be accomplished by applying a sequence of inputs to the implementation, by means of an external tester, and verifying if the sequence of outputs is the one specified.

1.On leave from Departamento de Ciência da Computação, Universidade Estadual de Campinas, Brazil. Work supported by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), grant 94/3667-3.

If a test sequence is capable of detecting all erroneous implementations, it is said to provide *full* fault coverage. There are many methods for generating automatically a test sequence to verify a given implementation against a specification. Several of these methods are based on the Unique Input/Output (UIO) technique. A number of papers have studied the fault coverage of the UIO methods and its optimizations, mainly by *evaluating* the fault coverage through simulation of some test cases. This paper presents a formal analysis of the fault coverage of the UIO methods, showing that, under certain conditions, this evaluation is not needed, since the UIO method and its optimizations can be shown to guarantee full fault coverage. The result of the analysis can be used to improve existing test generation methods so that they guarantee full fault coverage.

The paper is organized as follows. In the rest of this section the basic concepts of test generation are reviewed. The UIO method and its optimizations are well known, but since they are the main subject of this paper they are described in some detail before their fault coverage is analysed in Sections 2 and 3, respectively. In Section 4.0 the work presented in this paper is related to other approaches in the literature. Finally, Section 5.0 summarizes the main results.

1.1 Basic concepts

A protocol specification is typically composed by a control portion and a data portion. This paper deals with the control portion only; other approaches are oriented to the analysis of control and data dependencies [14].The control portion of a protocol, which will be referred as protocol specification, can be modelled as a deterministic finite-state machine (FSM) with a finite set of states $S = \{s_1, s_2,..., s_n\}$, a finite set of inputs $I = \{a_1, a_2,..., a_k\}$, and a finite set of outputs $O = \{x_1, x_2,..., x_m\}$. The next state ($\sigma$) and output ($\varphi$) functions are given by a set of mappings $\sigma: S \times I \rightarrow S$ and $\varphi: S \times I \rightarrow O$.

The FSM is usually also represented by a direct graph $G = (V, E)$, where the set $V = \{v_1, v_2,..., v_n\}$ of vertices represents the set of states S, and a directed edge represents a transition from one state to another in the FSM. Each edge in G is labelled by an input a_r and a corresponding output x_q. An edge in E from v_i to v_j which has label a_r/x_q means that the FSM, in state s_i, upon receiving input a_r produces output x_q and moves to state s_j. A triplet $(s_i, s_j, a_r/x_q)$ is used to denote a transition in the text.

The graph representation is useful for describing and reasoning about test generation algorithms. Within this context, some basic definitions from graph theory are briefly reviewed. A graph is said to be strongly connected if for any pair of distinct vertices v_i and v_j there is a walk which starts on v_i and ends on v_j. A *walk W* over a graph is a finite, non-null, sequence of consecutive edges. *Head(W)* and *Tail(W)* denote respectively the vertex where the walk *W* starts and the vertex where it ends. A *path* is a walk in which each edge of *G* appears exactly once.

An input/output sequence $U= (a_{r1}/x_{q1}, a_{r2}/x_{q2}, ... a_{rn}/x_{qn})$ is said to be *specified* for state s_i in an specification FSM if there exists a walk W with origin s_i in the graph representation of the FSM such that $W = \{(s_i, s_{j1}, a_{r1}/x_{q1}), (s_{j1}, s_{j2}, a_{r2}/x_{q2}), ... , (s_{jn-1}, s_{jn}, a_{rn}/x_{qn})\}$. A FSM is said to be *fully specified* if from each state it has a transition for every input symbol; otherwise the FSM is said to be *partially specified*. If a FSM is partially specified and a non specified transition is applied, under the *Completeness Assumption* the FSM will either stay in the same state without any output or signal an error. In this paper we consider FSMs under the completeness assumption. The *initial state* of a FSM is the state the FSM enters immediately after power-up. A FSM is said to have the *reset* capability if it can move from any state directly into the initial state with a single transition, denoted "*ri/null*" or simply "*ri*". State s_i is said to be weakly equivalent to state s_j if any specified input/output sequence for s_i is also specified for s_j. If two states are weakly equivalent to each other they are said to be strongly equivalent. It is assumed here that the FSMs are deterministic; that is, for some state $s_i \in S$, with two associated transitions (s_i, s_j, a_r, x_q) and (s_i, s_k, a_w, x_p), $a_r \neq a_w$.

A graph representation of a FSM is depicted in Fig. 1. For the FSM represented, *I={a,b,ri}* and *O={0,1,null}*. Reset edges are not shown in the figure, but are assumed to be labelled "*ri/null*" and directed towards the initial state v_1, the designated initial state.

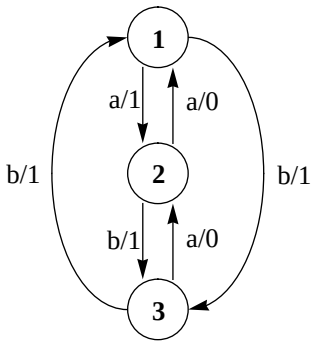


Fig. 1: A graph representation of a FSM

1.2 Test generation techniques

The purpose of test generation is to produce a sequence of inputs, called a test sequence, which can be applied to an implementation to verify that it correctly implements the specification. There is a number of necessary assumptions that must be made in order to make the experiment possible: (1) the specification FSM is strongly connected, so that all states can be visited; (2) the specification FSM does not have strongly equivalent states (it is minimized); (3) there is an upper bound on the number of states in the implementation FSM (otherwise one could always construct a machine which would pass a given test sequence by using as many states as there are transitions in the sequence). In relation to assumption (3) it is usual in the literature to consider that the implementation has no more

states than the specification. Note that if the implementation is correct, by this assumption it will have the same number of states than the specification. If the implementation is not correct, however, it may have fewer states than the specification and still pass a test sequence which does not provide full fault coverage. We consider also that the specification FSM is deterministic, and completely specified.

One of the simplest methods for generating test sequences is the *transition tour* method [11]: a test sequence is generated by simply applying random inputs, constructing a random walk over the graph representing the specification FSM until all transitions have been traversed. Obviously, the sequence generated may contain redundant inputs which in turn generate *loops* in the walk; these redundant inputs may be removed using a reduction procedure (but it is interesting to note that some redundancy may be important to enhance the fault coverage of the test sequence, as will be discussed in Section 4.0). As an example, the following test sequence is generated using the transition tour method for the automata of Fig. 1:

ri/null a/1 b/1 b/1 a/1 b/1 a/0 a/0 a/1 a/0 b/1 *[SEQ 1]*

In general, the fault coverage for tests generated by transition tours is worse than that obtained by other methods considered in this paper. Intuitively, this derives from the fact that verifying whether a transition produces the correct output is not enough to guarantee that the transition is correct: one should also verify that the new state of the implementation FSM after the transition is the one expected. That is what was done in the first methods for generating testing sequences, developed in the 60's.

Kohavi's book gives a good exposition of these earlier results on testing FSMs, motivated mainly by testing of switching circuits [7]. If the specification FSM has a *distinguishing sequence* (a sequence which produces a different output for each different state), the test procedure presented in [7] for testing an implementation FSM is divided into two parts. In the first part, called *state identification*, the implementation is forced to display the response of each state to the distinguishing sequence, while in the second part, called *transition identification*, each transition is verified.

The rationale for the state identification part is that the distinguishing sequence method includes a "hidden" assumption, namely that the distinguishing sequence is valid not only for the specification, but also for the implementation. The transition identification part is carried out by applying an input which causes the desired transition to be exercised and identifying the new state by means of the distinguishing sequence.

The problem with the method presented above is that not all FSMs have distinguishing sequences, and these can be in general very long. In [12] it was first presented the idea of using a Unique Input-Output (UIO) sequence as a means of solving these shortcomings. A UIO for state s_i, denoted *UIO(i)*, is an input/output sequence with origin s_i such that there is no $s_j \neq s_i$ for which *UIO(i)* is an specified sequence for starting state s_j. Most FSMs have

UIO sequences for every state, and UIOs are never longer than distinguishing sequences, being in practice usually much shorter. An extension to the UIO method can be used when some states have no UIOs, as seen in Section 3.6.

1.3 Conformance

Since the implementation is tested as a black box, the strongest conformance relation that can be tested is *trace equivalence*: two FSMs are trace equivalent if the two cannot be distinguished by any sequence of inputs. That is, both implementation and specification will generate the same outputs ("trace") for all specified input sequences. To prove trace equivalence it suffices to show that (a) there is a set of implementation states $\{p_1, p_2, ...p_n\}$ respectively isomorphic to specification states $\{s_1, s_2, ...s_n\}$, and (b) every transition in the specification has a corresponding isomorphic transition in the implementation.

2.0 Fault coverage of the non-optimized UIO method

The original paper introducing the UIO method proposed to use only the transition identification part as a testing sequence, apparently assuming it would suffice to provide good fault coverage. In [3] it was shown, by examples, that if the state identification part is not performed as well, some errors in the implementation may remain undetected. The argument is the same presented above for distinguishing sequences: the validity of the method is based on the fact that the UIO is unique both in the specification and in the implementation.

In this section a formal analysis of the fault coverage for the basic UIO method with the modification proposed in [3] is presented. Although the full fault coverage has already been argued in [3] (in a less formal manner), the results from this section will be used when analysing the fault coverage of the optimizations of the UIO method.

2.1 State identification

The state identification part of the test for the UIO method is slightly different from the one presented above for distinguishing sequences. To verify that each UIO is unique to a state in the implementation, it must be verified that the UIO for one state is accepted by that state and is *rejected* by all other states.

The procedure for verifying the rejection of *UIO(j)* for state s_i is the following:

1. The implementation is put into state p_i, presumably isomorphic to s_i, by applying a *reset* followed by some path *Preamb(i)* from the initial state s_1 to s_i. For efficiency reasons *Preamb(i)* should be some shortest path from s_1 to s_i but that is not relevant. However, once chosen, *Preamb(i)* must be fixed for the duration of the test.

2. *UIO(j)* is applied to the implementation and the output is checked to verify that the resulting output is not what it would be expected if the *UIO(j)* were applied to s_j.

The procedure for verifying the acceptance of *UIO(i)* for state s_i is similar, with the obvious difference that in step 2 it must verified that the output is indeed the one expected. However, UIO acceptance does not need to be tested in the state identification part, since if an implementation state does not accept its UIO the transition identification part will detect the error. The state identification part for the UIO method therefore consists of verifying, for all pairs *i, j* $(i \neq j)$, the rejection of *UIO(j)* by state p_i. An implementation state p_i, which rejects *UIO(j)* for all $(i \neq j)$ will be said *UIO-isomorphic* to specification state s_i. Note that the reset feature plays an important role in this part of the test.

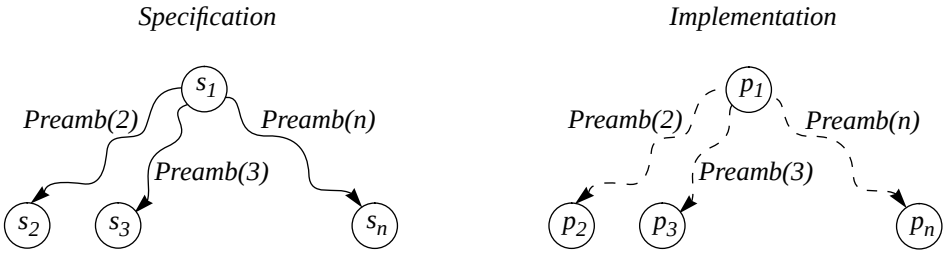


Fig. 2: State UIO-isomorphism after the state identification part

The state identification part determines that the implementation has at least *n* states p_1, p_2,..., p_n which are accessible using the same preamble as their respective UIO-isomorphic states s_1, s_2,..., s_n in the specification. In other words, after the state identification part the following two properties hold, for all *i*:

Property 1: There exists one, and only one, implementation state p_i which is UIO-isomorphic to specification state s_i.

Property 2: Implementation state p_i is reachable from initial implementation state p_1 by using *Preamb(i)*.

Note that although several transitions have already been used in *Preamb(i)* and in *UIO(i)*, the state identification part does not guarantee that these transitions are correct in the implementation. The state identification only asserts what the two properties state. Due to multiple faults, *Preamb(i)* may traverse faulty transitions and still take the implementation to the desired state p_i, and UIO may traverse faulty transitions and still give the correct output. This situation is depicted in Fig. 2, where dotted lines are used to emphasise the fact that *Preamb(i)* may include faulty transitions.

2.2 Transition identification

The procedure for testing transition $t = (s_i, s_j, a_r/x_q)$, a transition from state s_i to s_j with input/output a_r/x_q, is the following:

1. The implementation is put into state p_i, known to be UIO-isomorphic to s_i, by applying a reset followed by *Preamb(i)*;
2. Input a_r is applied and the output is checked, to verify that it is x_q as expected;
3. The new state of the implementation is tested to verify it is state p_j as expected, by applying *UIO(j)* and checking that the resulting output is the one expected.

In the transition identification part, all transitions in the specification are tested using the procedure above. A graphical representation of a transition test can be seen in Fig. 3.

2.3 Fault coverage

In this section the UIO method as described is shown to provide full fault coverage, i.e., there are no faulty implementations with at most the same number of states as the specification which can pass a test generated by the method.

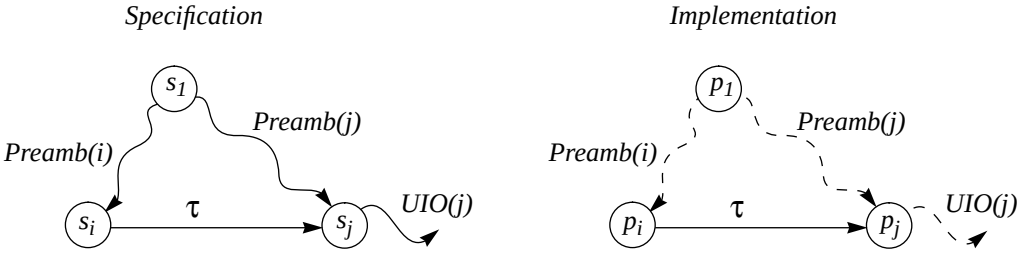


Fig. 3: Transition identification part

Consider a specification FSM with *n* states. If the implementation has fewer than *n* states, it will fail the state identification part, since some specification states will not have a corresponding UIO-isomorphic implementation state. If the implementation has *n* states, the transition identification part will test each transition $\tau = (s_i, s_j, a_r/x_q)$, by the sub-sequence (the symbol @ means concatenation, the transition under test is shown in bold)

ri @ Preamb(i) @ $\boldsymbol{a_r/x_q}$ *@ UIO(j)*

To show the correctness of the test method the two properties defined above are used. Suppose that there is a transition $\tau = (s_i, s_j, a_r/x_q)$ which is wrongly implemented but the error remains undetected after the test. Since all transitions are tested individually, the only way a faulty transition can escape detection is if the transition is not tested at all in the implementation. That is possible only if (a) the implementation is not in state p_i, UIO-isomor-

phic to s_i, when the transition is tested; (b) p_k, the state the implementation is in when the transition is tested, is UIO-isomorphic to state s_k and there exists a transition $\tau_a = (s_k, s_j, a_r/x_q)$ in the specification. However, (a) cannot happen because by property (2) the implementation is indeed in the corresponding UIO-isomorphic state p_i before the transition τ is applied, i.e., the intended transition will be tested in the implementation. If τ produces the correct output, there is obviously no output fault. And if *Tail*(τ) passes *UIO(j)*, by property (1) *Tail*(τ) must indeed be p_j, UIO-isomorphic to s_j. Therefore, there is no transition which is faulty and is not detected by the test. In other words, the UIO method without any optimization provides full fault coverage.

3.0 Fault coverage of optimizations to the UIO method

Several optimizations have been proposed to the UIO method over the years ([1], [4], [8], [9], [13]). Interestingly, all optimizations focused on the transition identification part, disregarding completely the state identification part. The general (hidden) assumption seems to be the same one made by [12], i.e., the transition identification part would suffice to provide full fault coverage.

All these optimizations, if used as proposed, without the corresponding state identification part, will not provide full fault coverage, since they are all basically the UIO method. The safety of some optimizations proposed is now analysed. The main result is to show that optimizations based on a "global optimization" technique will only provide full fault coverage under certain conditions.

3.1 Rural Chinese Postman Optimization

Let us call a *test segment* the subsequence *tr @ (UIO(Tail(tr))* used to test a single transition *tr*. The transition identification part therefore consists of as many test segments as there are transitions in the specification FSM. As seen, in the non-optimized UIO method each test segment is preceded by a *reset* followed by a *Preamble* sequence, with the purpose of bringing the implementation into state *Orig(tr*) so that *tr* can be applied and tested. In [1] it was first presented the idea of optimizing the cost of connecting the test segments, by using transfer sequences which used not only *reset* and *Preamble* sequences, but could include any specified transition.

The optimization is elegantly presented as a Rural Chinese Postman Tour Problem, which is NP-complete for the general case, but has a low-degree polynomial time solution for weakly connected graphs (they also showed that if a FSM has a reset feature or has a self loop for each state, its corresponding graph is weakly connected). The optimization problem is formulated as follows. The FSM is represented as a graph $G = (V, E)$. Consider the graph $G' = (V', E')$ such that $V' \equiv V$ and $E' = E \cup E_C$, where E_C is the set of all test segments. That is,

$$E_C = \{(s_i, s_k, tr \text{ @ } UIO(j)) \mid (s_i, s_j, tr) \in E \text{ and } Tail(UIO(j)) = s_k\}$$

In *G'*, traversing an edge in E_C corresponds to realizing a test segment; the cost of the traversal is usually taken to be the total length of the test segment. Notice that the edge-induced subgraph $G[E_C] = (V, E_C)$ is a spanning subgraph of *G'*. Therefore the optimization objective becomes traversing each edge in E_C at least once with a minimum cost tour of *G'*. Such a tour is a Rural Chinese Postman Tour. Similar to the Chinese Postman Problem, the problem is first reduced to that of finding a symmetric augmentation graph $G^* = (V^*, E^*)$, constructed such that $V^* = V'$ and E^* contains all edges in E_C, and possibly some edges in *E*. The basic idea is to minimize the number of edges chosen from *E* and at the same time make the graph G^* symmetric, that is, a graph for which every vertex has an in-degree equal to its out-degree. Finally, an Euler tour can be constructed in linear time from the symmetric, strongly connected graph G^*. The tour is then used as the transition identification part for the test sequence.

In Section 2.0, a central argument in the proof of full fault coverage of the basic UIO method is that, when testing transition $tr = (s_i, s_j, a_r/x_q)$, it can be guaranteed that the implementation is definitely into the state UIO-isomorphic to s_i before applying *tr*. That guarantee is given by the state identification part, which uses the same preamble sequence. In what follows it is shown that, due to limited controllability of the implementation, that guarantee may be lost when the RCP optimization is introduced.

Analysis of the fault coverage

Suppose there is one transition τ which is faulty in the implementation, and the fault is not detected by the state identification part. Since a test segment is executed to specifically test each transition, if the implementation passes the test sequence the only possibility is that when executing the test segment τ @ $UIO(Tail(\tau))$ some other transition is traversed instead of τ. And since the error remains undetected, there must be that the transition mistakenly traversed produces the same output (otherwise the error would be detected) and takes the implementation to the same state as τ should (otherwise $UIO(Tail(\tau))$ would fail). That is, if the erroneously implemented transition is specified as $\tau = (s_i, s_j, a_r/x_q)$, there must exist another transition $\tau' = (s_k, s_j, a_r/x_q)$ in the specification.

Therefore, for a faulty transition to remain undetected the only possibility is that the specification includes two edges going to the same state with the same input/output label, and one of these edges represents the faulty transition. Let us call state s_j *convergent* if there are edges going from states s_i and s_k into s_j with the same input/output label. Edges going into the same state with the same input/ouput label will be called *converging* edges, or transitions. The reasoning above leads to the following lemma:

Lemma 1*: All implementation errors in transitions which are not converging are detected by the UIO method with the Rural Chinese Postman optimization.*

Proof: The state identification part guarantees that each state in the specification has a corresponding UIO-isomorphic state in the implementation, i.e., each implementation state p_j will reject the *UIO(i)* for all $i \neq j$. When a test segment is executed and no error is detected, the transition under test must have produced the expected output and must have ended in the correct state. If a faulty transition τ is not converging, its test segment will fail, since either τ will not produce the expected output or *UIO(Tail(*τ*))* will fail. ♦

A first result can then be presented:

Proposition 1*: For a FSM which does not include a convergent state the UIO method with the Rural Chinese Postman optimization provides full fault coverage.*

Proof: It follows directly from *Lemma 1*.♦

Let us assume the specification FSM includes a convergent state s_j, with converging transitions $\tau = (s_i, s_j, a_r/x_q)$ and $\tau' = (s_k, s_j, a_r/x_q)$. If the test segment for τ succeeds and τ is admittedly faulty, it must be that the test segment was executed when the implementation was in state p_k instead of p_i. That is, when preparing to execute the test segment for τ, a transfer sequence was applied which supposedly should take the implementation into p_i but took it into p_k instead. That is only possible if there is a transition which is faulty and was traversed when executing the transfer to p_i. Therefore, if the error in τ is to remain undetected there must exist a faulty transition which is traversed when preparing to verify τ. The analysis is now divided in three separate cases. The first case is when no UIO used in the test sequence traverses a converging edge; the second case is when UIOs are allowed to use a converging edge; and the third case is when neither the UIO sequences nor transfer paths between test segments include a converging edge.

In the first case, by *Lemma 1*, if the test sequence succeeds all transitions used in all UIOs are correctly implemented. Therefore, when any UIO is executed, it leaves the implementation in the state it is supposed to. In particular, the UIO which is executed immediately before the test segment for the erroneously implemented transition $\tau = (s_i, s_j, a_r/x_q)$ leaves

the implementation in the correct state; let p_y denote that state, as depicted in Fig. 4.

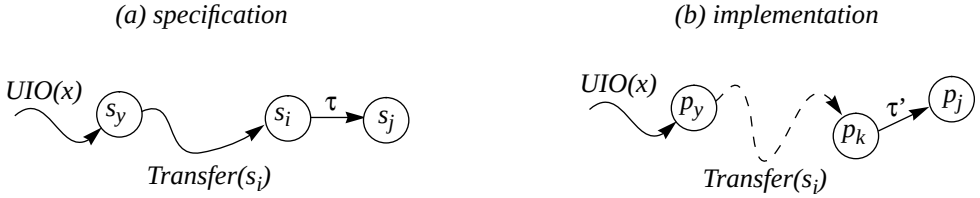


Fig. 4: *Transfer(si)* does not take the implementation into p_i

Since it was established that there must exist a faulty transition which is traversed when preparing to verify τ, as the transfer path starts from the correct state there must exist another transition τ_1 which is also faulty in the transfer path from s_y to s_i. Note that τ_1 could not be the same transition τ, otherwise the transfer path from s_y to s_i would not include τ_1.

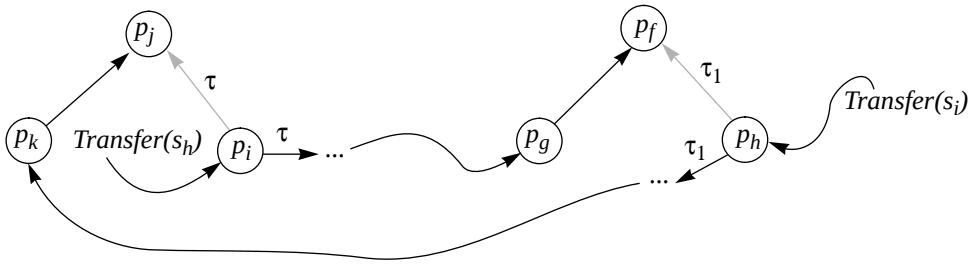


Fig. 5: Interdependence of errors

Using for τ_1 the same arguments used for τ, one comes to the conclusion that (*i*) τ_1 must be a converging edge; and (*ii*) if the error in τ_1 is to remain undetected there must be a faulty transition which is traversed when preparing to verify τ_1. Unfortunately, as depicted in Fig. 5, the sequence of faulty transitions does not have to continue unlimitedly, i. e., this scenario does not depend on the existence of transitions τ_2, τ_3, ... (in which case it would have been proved that faulty transition τ could not exist). There may exist an interdependence of faults which causes transition τ_1 to be traversed when preparing to verify τ, and transition τ to be traversed when preparing to verity τ_1, such that both errors remain undetected. Note that only some of the transitions and states are shown in Fig. 5. In this example, transitions $\tau = (s_i, s_j, a_r/x_q)$ and $\tau_1 = (s_h, s_f, a_w/x_t)$ are erroneously implemented, but τ_1 is traversed when preparing to test τ (resulting that the implementation is erroneously put into state p_k, and τ is not really tested) and τ is traversed when preparing to test τ_1 (and as a result τ_1 is not correctly tested). Therefore the following result can be presented:

***Proposition 2**: The UIO method with the Rural Chinese Postman optimization provides*

full fault coverage if none of the UIOs used traverses a converging transition and the specification FSM includes at most one pair of converging transitions.

Proof: The state identification part guarantees that each state in the specification has a corresponding UIO-isomorphic state in the implementation. As discussed in the previous paragraph, if the specification includes only one pair of converging transitions, any transition error would be detected in the transition identification part. ♦

In the second case, some UIOs may include converging edges. Therefore, in this case the state the implementation is after the UIO is not guaranteed to be the one expected. Fig. 6 shows a possible scenario of the situation (again, only some of the specified transitions and states are shown). In this example the faulty transition $\tau = (s_i, s_j, a/0)$ is the last transition of some *UIO(x)*, and is chosen by the optimization to be verified immediately after *UIO(x)* is executed, using the transfer path *b/1* (note that as τ is the last transition in *UIO(x)*, $Tail(UIO(x)) = s_j$). That is, the test sequence includes (the transition under test is shown in bold)

... UIO(x) @ b/1 @ **a/0** @ UIO(j) ...

Fig. 6a shows a specification and Fig. 6b shows a possible (wrong) implementation; only part of the specified states and transitions are shown. In this situation, the test segment for τ will succeed despite the erroneous implementation, so that the fault in τ remains undetected.

Fig. 6: Another case of interdependence of errors

Therefore, when UIOs are allowed to use converging edges, even if the specification contains one only pair of converging edges the RCP optimization does not guarantee full fault coverage.

In the third case, when neither UIOs nor transfer paths traverse a converging edge, it is guaranteed that before executing each test segment the implementation is in the state it is supposed to be. That leads to another result:

Proposition 3*: The UIO method with the Rural Chinese Postman optimization provides full fault coverage if neither UIOs nor transfer paths traverse a converging transition.*

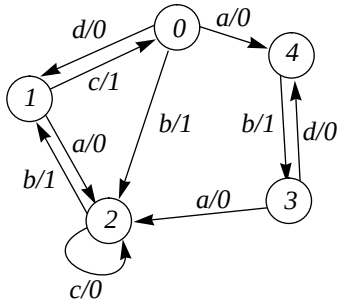
Proof: The state identification part guarantees that each state in the specification has a corresponding UIO-isomorphic state in the implementation. If neither UIOs nor transfer paths use converging edges, by Lemma 1 it is guaranteed that when a test segment is applied, it is applied with the implementation in the correct UIO-isomorphic state. Since the correct transition is tested by each test segment, any error will be detected. ♦

If the specification FSM includes one or more convergent states, Proposition 3 shows how the Rural Chinese Postman optimization should be modified in order to guarantee full fault coverage. The choice of UIOs should be restricted to those which do not traverse converging edges, and converging edges should not be used during the symmetric augmentation of graph $G[E_C]$. The first restriction is not normally difficult to satisfy, since in most FSMs states present a choice of UIOs; the second restriction is also not difficult in general to satisfy, since instead of using a converging edge from s_i to s_j a *virtual edge* (any path from s_i to s_j not including a converging edge) in graph $G[E_C]$ can be used.

If these two restrictions cannot be satisfied, to guarantee full fault coverage all converging transitions should be tested using the non-optimized reset-preamble method, before applying the RCP optimization to test the remaining transitions.

Example

As an example of a FSM which may cause fault masking when using the RCP optimization consider the FSM of Fig. 7.



i	*UIO(i)*	*Preamb(i)*
0	*d/0 a/0*	-
1	*c/1*	*d/0*
2	*c/0*	*b/1*
3	*d/0 b/1*	*a/0 b/1*
4	*b/1 d/0*	*a/0*

Fig. 7: An example FSM and its UIO and Preamble sequences

Note that state *2* is convergent, with converging transitions *(1, 2, a/0)* and *(3, 2, a/0)*, and that the first of these transitions was chosen to be used in *UIO(0)* (this represents therefore an example for the second case in the analysis). The state identification part consists of verifying the rejection of subsequences *ri @ Preamb(i) @ UIO(j)* for all pairs *i, j* ($i \neq j$). A RCP optimization produces the following sequence for the transition identification part, where transitions used in transfers are printed in bold:

ri/null b/1 @ UIO(2) @ ***b/1*** *c/1 @ UIO(0) @* ***b/1*** *a/0 @ UIO(2) @ c/0 @ UIO(2) @* ***b/1 c/1*** *d/0 @ UIO(1) @*
a/0 UIO(4) @ ***b/1*** *d/0 @ UIO(4) @ b/1 @ UIO(3) @ a/0 @ UIO(2) @ b/1 @ UIO(1)*

Expanded, this sequence produces:

ri/null b/1 c/0 b/1 c/1 d/0 a/0 b/1 a/0 c/0 c/0 c/0 b/1 c/1 d/0 c/1 a/0 b/1 d/0 b/1 d/0 b/1 d/0 b/1 d/0 b/1 a/0 c/0 b/1 c/1

Consider now the implementation FSM depicted in Fig. 8, in which transition *(1, 2, a/0)* is erroneously implemented as *(1, 4, a/0)*. Note that the state identification and the transition identification parts are successfully executed for this implementation. Therefore, the error remains undetected.

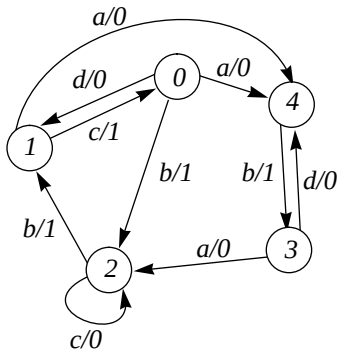


Fig. 8: An erroneous implementation which passes the test

The error would be detected if *UIO(0)* is chosen to be *b/1 c/0*, and care is taken so that no convergent edges are used in transfers. A RCP optimization produces in this case the following sequence for the transition identification part (again, transitions used in transfers are shown in bold):

ri/null b/1 @ UIO(2) @ ***b/1*** *c/1 @ UIO(0) @* ***b/1*** *a/0 @ UIO(2) @ c/0 @ UIO(2) @* ***b/1 c/1*** *d/0 @ UIO(1) @ a/0 @ UIO(4) @* ***b/1*** *d/0 @ UIO(4) @ b/1 @ UIO(3) @ a/0 @ UIO(2) @ b/1 @ UIO(1)*

Expanded, it results in:

ri/null b/1 c/0 b/1 c/1 b/1 c/0 b/1 a/0 c/0 c/0 c/0 b/1 c/1 d/0 c/1 a/0 b/1 d/0 b/1 d/0 b/1 d/0 b/1 d/0 b/1 a/0 c/0 b/1 c/1

which correctly detects the implementation error.

3.2 RCP with multiple UIOs

It was shown in [13] that using different UIOs for identifying a state in different test segments can reduce the total length of the transition part. The basic idea is to obtain a graph $G[E_C]$ which is closer to symmetry, so that fewer edges need to be added to make it symmetric. However, as already noted in [15], the fact that the uniqueness of any UIO-

sequence used must be verified, when using multiple UIOs any gain in the transition identification part may be lost by an increase in the state identification part.

In relation to fault-coverage, the fact that multiple UIOs are used does not change any of our previous results, assuming of course their uniqueness is verified in the state identification part.

3.3 RCP with overlaps

To further minimize the transition identification part, overlapping of test segments can be used. If the last part of a test segment T_i coincides with the first part of another test segment T_j, they can be merged so that the overlapping edges would serve to both T_i and T_j. If T_i is completely contained in T_j, T_i does not have to be executed at all, and can be removed from the test sequence. The "full overlap" optimization was in fact proposed in the original UIO paper, [12]; the general overlap was mentioned as a possible extension in [1] but the first solution appeared in [4].

In [4], *overlap links*, with negative cost, are introduced into the graph $G[E_C]$ to capture the concept of overlaps into the optimization. The optimization problem is then solved as a minimum cost — maximum cardinality matching problem in a bipartite graph.

Rather than presenting the proposed method in more detail, let us certify ourselves that the overlapping of test segments does not introduce any possibility of fault masking. Suppose two test segments $T_i = t_i$ @ $UIO(Tail(t_i))$ and $T_j = t_j$ @ $UIO(Tail(t_j))$ are overlapped, with T_i being executed first. Accordingly to our previous results, it is assumed also that neither UIOs nor transfer paths traverse converging edges. Therefore when test segment T_i starts, it is guaranteed that the implementation is indeed in the UIO-isomorphic state it should be, such that the correct transition t_i is exercised. If the correct transition is exercised, any output or transfer error would be detected, either by the transition failing to produce the correct output or by the $UIO(Tail(t_i))$ failing to produce the correct output. Consider now transition t_j. As the two test segments are made to overlap, t_j must be a component of $UIO(Tail(t_i))$, which means t_j itself is not a a converging edge. However, as it has been seen, if a transition error is to remain undetected, the transition must be a converging edge. Therefore, by *Lemma 1*, any error in t_j will be detected.

3.4 RCP with multiple UIOs and overlaps

In [9] it is shown how multiple UIOs and overlaps can be combined to obtain a further reduction on the transition identification part. It is interesting to note that they proposed different algorithms depending whether the specification FSM includes a convergent state or not. That is, they noticed convergent states are a possible cause of trouble, although they did not pursue the issue. As explained in the paper, their approach can be seen as first

finding the test sequence (for the identification part only) and then justifying that all needed test segments are included in the sequence found.

In all their cases, they show there must be a sequence of possibly overlapping test segments embedded into the sequence found, such that all test segments are executed. As we have already examined, multiple UIOs and overlaps do not interfere with the fault coverage. Therefore, in relation to fault coverage our results apply also in this case.

Although interesting, this approach has the same drawback as any method based on multiple UIOs: any gain in the length of the transition identification part due to the use of multiple UIOs incurs an increase in the length on the state identification part.

3.5 Greedy overlap

Another optimization method, presented in [4], differs from all previous methods described in that it does not use a global optimization to minimize the transfer sequences between test segments. Rather, it uses a greedy algorithm to construct step by step the test sequence for the transition identification part. As in this method the sequence produced is basically a different concatenation of overlapped test segments (possibly using multiple UIOs), all our results are also valid in this case.

3.6 Partial UIOs

Some FSMs do not possess UIO sequences for every state. Fig. 9 shows a FSM which does not have a UIO for state *1*: if *UIO(1)* starts with input *a* it cannot distinguish state *1* from state *3*; and if the *UIO(1)* starts with *b* it cannot distinguish state *1* from state *2*.

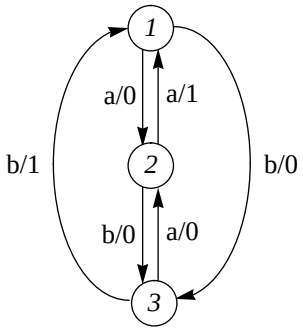


Fig. 9: A FSM which does not have UIO for all states

The approach proposed in [12] to verify a state s_j that does not have a UIO is to use a *signature*, a sequence which distinguishes s_j from each of the remaining states one at a time. A signature for s_j uses *(n-1)* subsequences *IO(j,m)*, each of which distinguishes s_j from one other state s_m. Before applying each IO subsequence the implementation must be put back into state s_j. Suppose that after the subsequence *IO(j,m)* is applied the specification is

in state s_k. The authors propose to use a transfer subsequence *Tr(k,j)*, which is some shortest path from s_k to s_j, in order to bring the implementation back to state s_j. Therefore, the signature for state s_j will be formed by concatenating the subsequences *IO(j,m) @ Tr(Tail(IO(j,m)),j)* for all $m \neq j$. That is, the test segment for a transition $t = (s_i, s_j, a_r/x_q)$, where s_j is a state which does not have a UIO, is composed by the sequence

a_r/x_q*@IO(j,1)@Tr(Tail(IO(j,1)),j)@IO(j,2)@Tr(Tail(IO(j,2)),j)@...@Tr(Tail(IO(j,n-1)),j)@IO(j,n)[SEQ 2]*

In [3] and [5] it was shown this signature method does not work in the general case, and a variation was proposed. The first improvement suggested in [3] is noting that a single IO sequence may distinguish state s_j from not only one other state, but from a group of states. An *IO set* for a state is composed by a certain number of IO sequences. Each sequence *IO(i, E_k)* in an IO set distinguishes state s_i from a subset of states $E_k \subset S$. E_k is called the exclusion set for that IO sequence in relation to state s_i. Therefore IO sets for different states may have different sizes (if the IO set has only one element the IO sequence is in fact a UIO), so that by selecting appropriate IO sequences for an IO set the size of the signature can be reduced. IO sets can be seen as a generalization of the UIO concept: an UIO is fact an IO set with one only element.

The second improvement proposed in [3] is not using a transfer function to concatenate IO sequences, since transfer sequences are a possible cause of fault masking when using signatures. In the transition identification part, instead of traversing the transition only once and verifying its final state as in [SEQ 2], the transition is traversed a number of times equals to the size of the IO set for that state, and each time a different IO sequence is used to verify the final state. At each time the reset-preamble sequence is used to bring the implementation to the correct state prior to traversing the transition. That is, transition *tr* = $(s_i, s_j, a_r/x_q)$ is verified by the sequence

ri/null @ Preamb(i) @ a_r/x_q *@ IO(j,*E_1*) @*
ri/null @ Preamb(i) @ a_r/x_q *@ IO(j,*E_2*) @*
...
ri/null @ Preamb(i) @ a_r/x_q *@ IO(j,*E_m*)*

In [5], this style of signature is called Partial UIOs, and an algorithm for generating them is presented. Provided that, as proposed in [3], the uniqueness of IO sets is also verified in the state identification part, their use for states which do not possess UIOs does not affect the validity of our results. Note however in this case the reset-preamble technique is fundamental not only when verifying the uniqueness of the IO sets, but also when using these IO sets in the transition identification part. Any tentative of "optimizing" the transition identification part involving the use of a transfer path to join the IO sequences (for example by using all test segments a_r/x_q *@ IO(j,*E_k*) @ Tr(Tail(IO(j,*E_k*)),i)* as edges of the graph *G[*E_c*]* when doing the RCP optimization) introduces the possibility of fault masking. The proof is similar to the ones presented earlier in the paper: to consider there is a fault transi-

tion and to show a scenario where the fault is not detected.

4.0 Related work

Several researchers have addressed the fault-coverage evaluation (as opposed to analysis) of testing methods, and in particular UIO-based methods. Some, however, considered only the transition identification part in their analysis ([8], [10], [17]), while others assumed the optimization itself would not interfere with the fault coverage ([15]). Most of the work on fault coverage evaluation uses the mutation technique: from the specification FSM a certain number of *mutant* (faulty) FSMs are randomly generated and verified with a given test sequence. The number of mutant FSMs which pass the test sequence is used as a measure of the fault coverage.

Another technique to evaluate the fault coverage is to estimate the number of FSM which could pass a given test sequence. This can be done by considering the test sequence as a form of FSM specification, and applying a minimization technique to this 'specification' [16]. The number of minimized machines not isomorphic to the real specification FSM gives a measure of the test sequence fault coverage. A more or less similar approach was proposed in [17], where a technique is used to reduce the number of possibilities when reconstructing, from the test sequence, all possible FSMs which would pass the given test sequence. The problem of these "exhaustive" approaches is that even with the reduction techniques the task of enumerating all viable solutions may be still too hard to be feasible in the general case.

Our paper shows that for UIO-based methods, under certain conditions, these evaluation techniques are not required, since it can be guaranteed that by applying only safe optimizations the fault coverage of the generated sequence is total. The previous work which more relates to our approach is [8], where the authors also derive a set of rules to guarantee a better fault coverage for a given test sequence. However, their analysis is complicated by the fact that they considered only the transition identification part, and that they divided the possible faults in three different types.

Since the main interest of this paper was to investigated the fault coverage of the UIO-based *methods*, its results always apply to the worst case. But it must be noted that besides the fault coverage provided explicitly by the test generation method, any test sequence carries an *intrinsic* fault coverage. For example, the simple sequence

ri/null a/1 b/1 b/1 a/1 b/1 a/0 a/0 a/1 a/0 b/1 b/1 b/1 a/0 *[SEQ 3]*

which is an extension of [SEQ 1], the "random walk" sequence generated by the transition tour method, possesses an unexpected coverage power. In fact, [SEQ 3] provides full fault coverage, with no need for state identification or reset-preamble sequences, and is much shorter than test sequences generated by UIO based methods. The fault coverage can be verified in this small example by case analysis, or with a tool similar to [16] or [17].

This intrinsic fault coverage can be understood by realizing that some information may gained by the simple fact that one specific transition is concatenated after some other transition in the test sequence. For example, if a test sequence includes the subsequence *"a/0 a/2 a/1"*, in order to successfully execute it any deterministic implementation must possess at least three different states. Each transition added to the end of a test sequence will increase or diminish the test intrinsic fault coverage depending on its relation to all previous transitions.

Unfortunately, it seems to be difficult to devise a method which can exploit this intrinsic fault coverage in the general case.

5.0 Conclusions

This paper analysed the fault coverage of the basic UIO method and some of its optimizations. It presented the conditions under which these methods are guaranteed to provide full fault coverage. The main result is to show that optimizations to UIO-based methods are not safe in the general case. If the specification FSM of a protocol is fully specified (or the completeness assumption can be invoked), propositions 1-3 presented in this paper offer the conditions for the full coverage of the test sequence. If a FSM does not include any converging state, the UIO method optimizations analysed were shown to provide full fault coverage. This paper also showed that the optimizations to UIO-based methods offer full fault coverage when no converging edge is used in UIOs or transfer paths. If these restrictions cannot be adhered to, the only way to guarantee full fault coverage is to use the reset-preamble technique to test the converging transitions before applying the optimizations to test the other transitions.

6.0 References

[1] A. V. Aho, A. T. Dahbura, D. Lee, M. U. Uyar, An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours, *IEEE Transactions on Communications*, vol. 39, no. 11, November 1991.

[2] Bochmann, A. Petrenko, M. Yao, Fault coverage of tests based on finite state models, *Proc. 7th IFIP International Workshop on Protocol Test Systems*, pp. 55-74, Japan, November 1994.

[3] W. Y. Chan, S. T. Vuong, M. R. Ito, An improved protocol test generation procedure based on UIOs, *Proc. SIGCOM89*, pp. 283-294, 1989.

[4] M. S. Chen, Y. Choi, A. Kershenbaum, Approaches utilizing segment overlap to minimize test sequences, *Proc. 10th International IFIP Symposium on Protocol Specification, Testing and Verification*, pp. 67-84, Canada, June 1990.

[5] W. Chun, P. D. Amer, Improvements on UIO sequence generation and partial UIO sequences, *Proc. 12th International IFIP Symposium on Protocol Specification, Testing and Verification*, pp. 234-249, Florida, USA, June 1992.

[6] P. Kars, Test coverage estimation by explicit generation of faulty FSMs, *Proc. 7th IFIP International Workshop on Protocol Test Systems*, Japan, November 1994.

[7] Z. Kohavi, Switching and finite automata theory, McGraw-Hill, 1978.

[8] F. Lombardi, Y. N. Shen, Evaluation and improvement of fault coverage of conformance testing by UIO sequences, *IEEE Transactions on Communications*, vol. 40, no. 8, pp. 1288-1293, August 1992.

[9] R. E. Miller, S. Paul, Generating minimal length test sequences for conformance testing of communications protocols, *Proc. IEEE INFOCOM 91*, pp. 970-979, 1991.

[10] H. Motteler, A. Chung, D. Sidhu, Fault coverage of UIO-based methods for protocol testing, *Proc. 6th IFIP International Workshop on Protocol Test Systems*, pp. 21-33, France, September 1993.

[11] S. Naito, M. Tsunoyama, Fault detection for sequential machines by transitions tours, *Proc. IEEE Fault Tolerant Computer Systems*, 1981.

[12] K. K. Sabnani, A. T. Dahbura, A protocol test generation procedure, *Computer Networks and ISDN Systems*, vol. 15, no. 4, pp. 285-297, 1988.

[13] Y. Shen, F. Lombardi, A. T. Dahbura, Protocol conformance testing using multiple UIO sequences, *IEEE Transactions on Communications*, vol. 40, no. 8, pp. 1282-1287, August 1992.

[14] H. Show, H. Ural, Data flow oriented test selection for LOTOS, Technical Report TR 93-12, Department of Computer Science, University of Otawa, Canada, 1993.

[15] M. Yao, A. Petrenko, G. Bochmann, Conformance testing of protocol machines without reset, *Proc. 13th Symp. Protocol Specification, Testing and Verification*, Belgium, May 1993.

[16] M. Yao, A. Petrenko, G. Bochmann, A structural analysis approach to the evaluation of fault coverage for protocol conformance testing, *Proc. IFIP Conference on Formal Description Techniques - FORTE 94*, pp. 389-404, Switzerland, October 1994.

[17] J. Zhu, A. T. Chanson, Fault coverage evaluation of protocol test sequences, *Proc. 14th IFIP Symp. Protocol Specification, Testing and Verification*, Canada, 1994.