Fault Coverage of Tests Based on Finite State Models

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

Département d'informatique et de recherche opérationnelle
Université de Montréal, CP. 6128, Succ. Centre-Ville
Montréal (Québec), Canada H3C 3J7

The finite state models have been used extensively in protocol conformance testing, as well as in software and hardware testing. It is well known that exhaustive testing is often impractical since it may require the execution of a huge number of test cases. In practice, testing is a trade-off between increased confidence in the correctness of the implementation under test and constraints on the amount of time and effort that can be spent in testing. Therefore, the fault coverage, or adequacy of the test suite, becomes a very important issue. In this paper, we analyze various techniques used to evaluate fault coverage based on finite state models in the form of finite state machines (FSM) or labeled transition systems (LTS). We also point out certain issues which need further study.

Key Word Codes: C.2.2: D.2.5
Key Words: Network Protocols; Testing and Debugging

1. INTRODUCTION

Testing is a critical phase in the development life cycle of a system. Testing consists of a number of execution scenarios of a system implementation against a selected set of test cases called a test suite. A faulty implementation is said to be detected if its execution against a test case distinguishes its behavior (or output) from what is expected.

Testing is used to ascertain the correctness of a system implementation with respect to its requirement specification. The essential idea of this correctness-proving viewpoint is that an execution scenario in which no fault is detected ensures that the implementation is free of certain kinds of faults. Therefore, exhaustive testing, which requires that all the possible execution scenarios of the implementation be carried out, is able to prove the correctness of the implementation. In general, exhaustive testing is often impractical since it may involve a huge number of execution scenarios to be performed. In the more practical and so-called fault-based testing approaches [More90], [Boch91], a fault model is first selected. It specifies a set of faults of which an implementation should be free. However, it still can be too expensive to prove by testing that an implementation is free of all the specified faults, as it still may require the execution of a very large number of test cases. In practice, therefore, a test suite which consists of only a relatively small number of test cases will be actually employed to test the implementation. As such, testing is often a trade-off between increased

confidence in the correctness of the implementation under examination and constraints on the amount of time and effort that can be spent in testing the implementation. Whenever such a trade-off is made, it is desired to have a measure of the effectiveness of testing in terms of the percentage of the specified faults that can be detected.

The finite state models have been extensively used in hardware and software testing, such as conformance testing of communication protocols [PBD93], [BoPe94]. They are also being used in the testing of object-oriented systems, see for example [HoSt93], [TuRo92]. Testing based on the finite state models is always treated within certain restricted frameworks. With such a restricted framework, the concept of full or complete fault coverage can be addressed, and fault coverage analysis of a given test suite with respect to a finite state specification can be performed. In this paper, we are going to give a review of fault coverage analysis methods based on the finite state models. Our discussion will be mainly based on the finite state machine (FSM) model. However, we will also address relevant issues for the labeled transition systems (LTS).

The rest of the paper is organized as follows: In Section 2, a framework of fault coverage analysis based on the traditional model of FSMs is presented. In Section 3, we discuss certain parameters of the specification machines and properties of test suites which play an important role in achieving full fault coverage. The existing approaches for evaluating the fault coverage of tests with respect to deterministic FSMs are considered in Section 4. Section 5 highlights some fault coverage analysis problems which mostly remain open in the cases of nontrivial test architectures and nondeterministic specifications, FSMs and LTSs. We conclude in Section 6 by discussing possible applications of techniques for fault coverage analysis.

2. BASIC FRAMEWORK FOR FAULT COVERAGE ANALYSIS

We start with an overview of a fault coverage framework, based on the traditional model of completely specified deterministic FSMs.

FSM-based testing is usually formalized as the problem of testing an FSM implementation: given an FSM representation (specification) of a system (denoted henceforth as M_S) and an implementation of the system (denoted henceforth as M_I), we are required to determine if the implementation machine M_I *conforms to* (i.e., is *correct* with respect to) the specification machine M_S by testing M_I as a black-box. This implies that we should generate from M_S a set of input sequences, called a test suite, and the corresponding set of expected output sequences such that M_I conforms to M_S if and only if, when the input sequences in the test suite are applied to M_I, the observed output sequences from M_I are the same as the corresponding expected output sequences. As already pointed out in the literature, this problem is not solvable unless it is dealt within a restricted framework. Therefore, some assumptions should be made about the specification machine M_S and the implementation machine M_I. It is important at this point to summarize them, since any further results in fault coverage should in one way or another relax at least one of these assumptions.

Typical assumptions about the specification machine are that M_S is not only complete and deterministic, but also reduced (and therefore minimal), and initially or strongly connected (later in this paper, we also consider other classes of machines which do not satisfy these

assumptions).

In case of black-box testing based on the specification, the test suite is usually developed based on the abstract interfaces defined within the specification and directly accessed by a tester. Here we refer to the local single-layer test method defined by the international standard [IS9646]. It must be assumed that the (abstract) properties of these interfaces are satisfied by the concrete realization of these interfaces in the implementation under test (IUT); we call this the *correct interface assumption* [BoPe94]. In the context of FSM-based testing, it is assumed that the interface is event-driven, and all events on the interface are alternately controlled by the environment and the IUT. The correct interface assumption implies that any IUT can also be modeled as an FSM. In particular, no IUT can "refuse" to execute any input in any of its states or produce an output without any input. Any IUT is thus a completely specified FSM M_I, even if M_S is partial.

Moreover, a test suite (TS) is typically constructed based on the input alphabet of the specification FSM. Therefore, it is assumed that the input alphabet of M_I at least covers that of M_S, or in other words, an implementation to be tested was constructed from this specification and not from any other. Yet another assumption about the deterministic behavior of M_I is typically made when M_S is deterministic.

A test suite TS can be one of two types: (1) it consists of a single sequence, or (2) it contains several sequences, which are supposed to be applied to the initial state of an implementation. In the latter case, it is assumed that the implementation has the reliable reset facility ensuring that each test sequence is applied to the same initial state. As shown later, the reliable reset assumption usually facilitates fault coverage analysis, as certain properties of the TS become more evident. This assumption also makes it possible to deal with specification FSMs which are only initially connected but not strongly connected.

The formidable obstacle in constructing or analyzing a test suite is that it must verify whether a given conformance relation (in the case considered here, the equivalence relation between FSMs) defined on infinite sequences, holds between two machines. At the same time, the test suite has to be finite for practical reasons. The apparent contradiction between these two requirements is usually resolved by assuming that in testing, we are dealing with a finite number of implementations derived from the given specification machine M_S. A common way to limit their number is to assume a fault model. Once the set of possible implementations, denoted henceforth as **Impl**(M_S), is limited, we can verify with the help of suitable tests whether a machine from this set is conforming to the specification. Thus the validity of a conclusion drawn from the test campaign heavily depends on how this finite set of implementations has been chosen, whether it reflects all practical cases or not. The more we know about the real IUTs, the more precisely this set (the fault model) can be determined, yielding a shorter test suite. However, black-box testing usually implies that not much is known about the IUTs before testing.
Apart from explicitly enumerating all machines of **Impl**(M_S), which is feasible for a small number of FSMs, there are at least two other techniques to describe this set:(1) limiting the number of states; and (2) the use of fault functions [PeYe92].

In the first case, all possible implementations are modeled by the universe **Impl**(m, M_S) of all complete FSMs which have the same input and output alphabets as M_S and up to m states,

where m≥n and n is the number of states of M_S. This is the fault model most widely used in the context of black-box testing. It is based on the observation that all FSMs in **Impl**(m, M_S) can be treated as mutants of M_S. A mutant is an FSM obtained by applying to M_S (which might be a partial machine) each of the following four types of operations, in any order, a certain number of times (including zero times):
Type 1: change the tail state of a transition;
Type 2: change the output of a transition;
Type 3: add a transition; and
Type 4: add an extra state.
Such a mutation technique works well in the context of deterministic specifications. However, in the case of nondeterministic systems and the conformance relation based on trace inclusion, i.e. the reduction relation [PBD93], the correspondence between mutations and erroneous behavior is not so straightforward.

In the second case, it is assumed that mutants may deviate from the specification machine only in a fixed number of so-called suspicious transitions, i.e. the above operations are applied to certain transitions, while the remaining transitions are assumed to be correctly implemented. Such grouped faults are compactly represented by a fault function which is the behavior function of an appropriate nondeterministic FSM constructed for the specification FSM M_S in such a way that M_S is one of its deterministic submachines. As an example, output faults correspond to a special form of the fault function [PeYe92].

A TS is said to be *complete for M_S in the class* **Impl**(M_S) if for each machine from this set which is not conforming to M_S (in the sense of a given conformance relation), there is an input sequence in TS that can detect this mutant. A TS which is complete in **Impl**(m, M_S) is simply called *m-complete*. Complete test suites were first introduced as checking experiments for completely specified FSMs [Moor56].

Since completeness of a test suite is usually not easy to achieve in testing, its fault coverage, i.e., the ability of the test suite to detect faulty implementations, can be characterized by a real number between 0 (no fault covered) and 1 (complete in the given class). Let M_S be the specification machine, TS be a test suite, and **Impl**(m, M_S) be the class of implementations. To define the fault coverage, we need to use the following notations:
$\mathbf{N_t}$(m, M_S) - the total number of machines in **Impl**(m, M_S);
$\mathbf{N_c}$(m, M_S) - the number of machines in **Impl**(m, M_S) which conform to M_S;
$\mathbf{N_p}$(m, M_S, TS) - the number of machines in **Impl**(m, M_S) which can pass the given test suite TS.

It is clear that,
$\mathbf{N_t}$(m, M_S) - $\mathbf{N_c}$(m, M_S) is the number of machines in **Impl**(m, M_S) which do not conform to M_S;
$\mathbf{N_t}$(m, M_S) - $\mathbf{N_p}$(m, M_S, TS) is the number of machines in **Impl**(m, M_S) which cannot pass the given test suite TS (and therefore do not conform to M_S).

The *fault coverage* of a test suite TS with respect to M_S, denoted as **FC**(m, M_S, TS), is

$$\mathbf{FC}(m, M_S, TS) = \frac{\mathbf{N_t}(m, M_S) - \mathbf{N_p}(m, M_S, TS)}{\mathbf{N_t}(m, M_S) - \mathbf{N_c}(m, M_S)}$$

This kind of fault coverage measure has been used in a number of papers, see, for instance, [Boch91], [DaSa88]. There are, however, several problems with this formula:

1) The fault coverage is determined, only if we make the additional assumption that the output alphabets of all the machines in **Impl**(m, M_S) are subsets of Y - the output alphabet of M_S (under this assumption, $\mathbf{N_t}(m, M_S) = (m|Y|)^{m|X|}$), where X is the input alphabet of M_S), otherwise the cardinality of **Impl**(m, M_S) would remain unknown. Thus, the "real" coverage is much higher, since a large number of implementations with a "foreign" output symbol are also detected by the test suite.

2) The value of **FC**(m, M_S, TS) is not equally distributed over [0,1]. A TS consisting of a single test event already has the coverage of more than (|Y|-1) / |Y|. If we are given an FSM with, for instance, just ten outputs, the fault coverage of a single test event is already over 90%. As the number of outputs in the M_S increases, the fault coverage of a single test event approaches 100%.

3) For real protocol machines, it often occurs that $\mathbf{N_t}(m, M_S) >> \mathbf{N_c}(m, M_S)$ and $\mathbf{N_t}(m, M_S) >> \mathbf{N}_p(m, M_S, TS)$ and therefore **FC**(m, M_S, TS) ♠ 100%. Thus calculations with a normal precision might not be sufficient to compare the test suites by their fault coverages.

We think it is important to keep in mind these drawbacks of the fault coverage formula when applying it in practice. Actually, the so-called “order coverage” [YPB94b] has been proposed to overcome the drawback (3) mentioned above. The “order coverage”, denoted as $\mathbf{FC_O}$(m, M_S, TS), can be defined by the following formula:

$$\mathbf{FC_O}(m, M_S, TS) = \frac{\mathbf{log}\,(\mathbf{N_t}(m, M_S)) - \mathbf{log}\,(\mathbf{N_p}(m, M_S, TS))}{\mathbf{log}\,(\mathbf{N_t}(m, M_S)) - \mathbf{log}\,(\mathbf{N_c}(m, M_S))}\ .$$

It is easy to prove that, for TS_1 and TS_2, $\mathbf{FC}(m, M_S, TS_1) \le \mathbf{FC}(m, M_S, TS_2)$ if and only if $\mathbf{FC_O}(m, M_S, TS_1) \le \mathbf{FC_O}(m, M_S, TS_2)$. It is clear that, even when $\mathbf{N_t}(m, M_S) >> \mathbf{N_c}(m, M_S)$ and $\mathbf{N_t}(m, M_S) >> \mathbf{N_p}(m, M_S, TS)$, $\mathbf{log(N_t}(m, M_S)\mathbf{)}$ can still be comparable with $\mathbf{log(N_c}(m, M_S)\mathbf{)}$ and $\mathbf{log(N_p}(m, M_S, TS)\mathbf{)}$. Therefore, the difference between $\mathbf{FC_O}(m, M_S, TS_1)$ and $\mathbf{FC_O}(m, M_S, TS_2)$ is often larger than the difference between $\mathbf{FC}(m, M_S, TS_1)$ and $\mathbf{FC}(m, M_S, TS_2)$. This implies that in most cases the test suites can be distinguished by the corresponding “order coverages”.

The fault coverage formulae considered above are not the only possible ways of characterizing test suites, for example, the other approach [AlVu93] was proposed in a setting quite different from the one adopted in this paper. The framework of this paper is common for test derivation and analysis. It makes the relationship between these two problems more evident.

3. CONDITIONS FOR COVERAGE

There exist several test derivation methods for FSMs which guarantee complete fault coverage. Based on these methods, we can formulate certain sufficient conditions as well as necessary ones for the completeness of a given test suite. Here we consider only those tests which rely on the reliable reset feature in the implementations under test. (The corresponding conditions for the case without reset are somewhat more complicated, see [YPB93]).

3.1. Tests for complete deterministic FSMs

Let $M_S = <S, X, Y, s_1, \delta, \lambda>$ be a complete deterministic FSM (CDFSM), where S is a set of n states $\{s_1, s_2, ..., s_n\}$ with s_1 as the initial state; X is the input alphabet; Y is the output alphabet; δ is the transfer function; λ is the output function. Let TS be a test suite for the given reduced CDFSM M_S. By X^a we denote the set of all input sequences (including the empty sequence e) which have the length of at most a. The prefix set **AP**(TS) of a test suite TS is the set which consists of all the prefixes of all the test cases in TS, i.e., **AP**(TS) = { p | p is a prefix of some test case in TS }.

The test suite TS is called an *a-exhaustive* test suite for M_S, if **AP**(TS) $\supseteq X^a$.

As is stated in [Gill62], any two distinguishabe CDFSMs with n and m states, respectively, can be distinguished by a sequence whose length is n+m-1 in the worst case. Thus, we have the following property.

Sufficient Condition 3.1.1. Let TS be an a-exhaustive test suite for the CDFSM M_S with n states. If $a \geq m+n-1$, then TS is m-complete for M_S.

This property relies on the worst case assumptions on the CDFSM M_S, since only one parameter, namely, the number of states, is taken into consideration. In fact, we have a refined property of a test suite if additional parameters of M_S, namely, the degrees of accessibility and distinguishability, are determined.

We say that state s of M_S is k-reachable from the initial state, if there is an input sequence of length k which transfers M_S from its initial state into s. The minimum r such that each state is k-reachable, $k \leq r$, is said to be the *degree of accessibility* of M_S. Any initially connected FSM has $r \leq n-1$. If the value of r is known then it is possible to construct a *state cover* V of M_S which is a set of n transfer sequences to every state, each of the length of at most r. This parameters characterizes the controllability of the given FSM.

Following [Gill62], we say that two states s and p of M_S are *d-distinguishable* if there exists an input sequence of the length d which causes different output sequences from these states. The minimum d such that any distinct states of M_S are d-distinguishable is said to be the *degree of distinguishability* of the given FSM M_S. It is known that for any complete reduced FSM with n states, $1 \leq d \leq n-1$. This parameter characterizes the observability of the given FSM. A set of input sequences which separates (distinguishes) all states is called a *characterization set* W of M_S. Such a set is used for state identification. It is always possible to construct a W set such that the length of its longest sequence does not exceed d. As shown in [TyBa75], the

total length of sequences of W has the least upper bound of n(n-1)/2. We note that simple sequences [Hsie71], commonly known as UIO-sequences [SiLe89], may well exceed this bound.

The size of complete test suites as well as fault coverage of an arbitrary test suite for an FSM heavily depend on the values of these parameters. Their upper bounds are given above. However, as shown in [TrBa73], in "almost all" cases, the degree of accessibility and distinguishability may actually be far smaller, being of the order of the logarithm and repeated logarithm, respectively, of the number of states. Several empirical studies of protocols confirm that existing protocols indeed tend to have rather short transfer and state identification sequences.

Sufficient Condition 3.1.2. Let TS be an a-exhaustive test suite for M_S with n states, the degrees of accessibility r and distinguishability d. If $a \geq m-n+r+d+1$, then TS is m-complete for M_S.

This statement can be proven based on results of [Vasi73], [Chow78]. There are certain subsets of a-exhaustive test suites whose completenesses are also relatively easy to verify. Even though in the general case, the upper bound on the length of sequences in X^a cannot be lowered, it is still possible that the test suite is complete and contains only a subset of X^a. We consider in the following several types of such subsets.

Let V be a state cover of the given FSM M_S. Since the machine may possess several state covers, the set V is not necessarily minimal, i.e. it may contain sequences of length greater than r. Consider the set VX^b which is the concatenation of the two sets V and X^b.

The test suite TS is called a *b-canonical* test suite for the FSM M_S, if **AP**(TS) $\subseteq VX^b$. We have the following property of such a test suite [YePe89].

Sufficient Condition 3.1.3. Let TS be a b-canonical test suite for M_S with n states, the state cover V, and the degree of distinguishability d. If $b \geq m-n+d+1$, then TS is m-complete for M_S.

This condition implies Condition 3.1.1. Certain special cases are worth considering at this point. Assume the FSM M_S has an input which distinguishes all states. In other words, assume a single input symbol is the diagnostic sequence and a common simple (UIO-) sequence at the same time. The degree of distinguishability is equal to 1, and any test suite which contains the set VX^2 is n-complete in the class. Another extreme case is where the degree of distinguishability of the given FSM reaches its maximum, i.e. d=n-1. A b-canonical test suite VX^b is b-complete for such a machine ($b \geq m$).

Another type of regular test suite can be defined if a characterization set W of the given FSM M_S is introduced. Note that the set X^d which is an interior part of a complete b-canonical test suite ($b \geq d$), already contains a characterization set of M_S. Consider a test suite where not all sequences of length d are applied to the states of M_S. However, the applied sequences may still embody a characterization set W. We note that like state covers, the machine can possess several (not necessarily minimal) characterization sets, i.e. the length of their sequences may exceed the value of parameter d.

Similarly to a-exhaustive and b-canonical test suites, we define a *complete c-characterization* test suite which contains the set VX^cW, where V is a state cover, and W is a characterization set of the FSM M_S. Based on the ideas of the W-method [Vasi73], [Chow78], the following property can be stated.

Sufficient Condition 3.1.4. Let TS be a complete c-characterization test suite for M_S with n states. If $c \geq m-n+1$, then it is m-complete for M_S.

Fault coverage analysis in this case involves a more difficult task of checking if a characterization set is properly embodied in the given test suite, i.e. if it does contain the set VX^cW. In the case where m is assumed to be equal to n, a complete 1-characterization test suite covers all transitions of the given FSM and there exists a characterization set whose sequences are applied to the initial and next states of every transition at least once.

Next we define the so-called partial c-characterization test suites. Let W_i be a state s_i identifier which is a subset of W (in [Fuji91], it is called an identification set of state s_i). Let I be the set of n state identifiers, and R be a set of input sequences. We denote by R□I the set $\{ \alpha W_i \mid \alpha \square R \ \& \ \delta(s_1,\alpha)=s_i \}$. The test suite TS is called a *partial c-characterization* test suite for the FSM M_S, if **AP**(TS) □$VX^{c-1}W \approx VX^c$□I. This expression reflects two phases of testing used by the Wp-method. It is obvious that VX^cW □$VX^{c-1}W \approx VX^c$□I. Based on the results of [Fuji91], we state the following sufficient condition for the completeness of the test suite.

Sufficient Condition 3.1.5. Let TS be a partial c-characterization test suite for M_S with n states. If $c \geq m-n+1$, then it is m-complete for M_S,

The Wp-method does not restrict the choice of a characterization set and state identifiers. In particular, if every state of the FSM M_S possesses a simple (UIO) sequence, then the set W includes all UIO-sequences (more precisely, their input parts), and the expression $VX^{c-1}W \approx VX^c$□I gives a version of the UIOv-method [VCI89] generalized to cover the case where m>n.

A slightly different sufficient condition can be obtained if, instead of the two sets W and I, the so-called "harmonized" state identifiers are considered [Petr91]. The set H of *harmonized* state identifiers $\{W_1, ..., W_n\}$ is defined as follows: $\forall s_i, s_j \square S \ \exists \alpha \square$ **AP**$(W_i) \leftrightarrow$ **AP**(W_j) $(\lambda(s_i, \alpha) \square \lambda(s_j, \alpha))$. The approach based on harmonized state identifiers [Petr91], [PeYe92], [YePe90], [LPB94b] does not require the existence of the two phases of testing, in contrast to the Wp- and UIOv-methods. In the special case, when the FSM M_S possesses a diagnostic (distinguishing) sequence DS, all the methods, W-, Wp-, UIOv-, HSI-, DS-methods (with reset) come to the same, i.e. they can produce the same test suite.

The test suite TS is called a *harmonized c-characterization* test suite for the FSM M_S, if **AP**(TS) □VX^c□H. We have the following sufficient condition [Petr91].

Sufficient Condition 3.1.6. Let TS be a harmonized c-characterization test suite for M_S with n states. If $c \geq m-n+1$, then it is m-complete for M_S.

Sufficient conditions presented so far are all derived from either the known upper bounds on

tests or the existing methods for test derivation with the proven complete fault coverage. There exists yet another sufficient condition (the weakest among those considered here) which was formulated for the case where m=n in [YPB94a] based on the results of [Petr91], [YePe90]. It is, in fact, a slightly relaxed version of Condition 3.1.6.

Sufficient Condition 3.1.7. TS is an n-complete test suite with respect to the reduced machine M_S if

(1) **AP**(TS) contains a state cover V and a transition cover TC; and

(2) for each pair of sequences $\alpha, \beta \in V$ such that $\delta(s_1, \alpha) \neq \delta(s_1, \beta)$, there should be two sequences $\alpha\gamma, \beta\gamma \in$ **AP**(TS) such that $\lambda(\delta(s_1, \alpha), \gamma) \neq \lambda(\delta(s_1, \beta), \gamma)$; and

(3) for each $\alpha \in (TC \setminus V)$ and each $\beta \in V$ such that $\delta(s_1, \alpha) \neq \delta(s_1, \beta)$, there should be two sequences $\alpha\gamma, \beta\gamma \in$ **AP**(TS) such that $\lambda(\delta(s_1, \alpha), \gamma) \neq \lambda(\delta(s_1, \beta), \gamma)$.

It is interesting to note that any test suite generated by the DS- method (based on reset) [Gone70], the UIOv-method [Vuon89], the W-method [Chow78], [Vasi73], the Wp-method [Fuji91] or the methods based on harmonized state identifiers [Petr91], [YePe90], [LPB94b] satisfies this sufficient condition and therefore is n-complete.

Sufficient conditions can be used for fault coverage analysis. In fact, we may conclude that a given test suite is m-complete if it contains a subset which satisfies at least one of these conditions. However, even if none of these conditions is satisfied, the given test suite may still be complete for a certain m. The condition 3.1.7. is the weakest sufficient condition. However, we could not yet formulate the necessary and sufficient ones in a similar way. At the same time, it is still possible to present certain necessary conditions. Further research is needed to see if the gap between these conditions can be further filled.

Necessary Condition 3.1.8. If, for some m, $m \geq n$, TS is m-complete for M_S with n states and the degree of accessibility r, then the length of its longest sequence will not be less than r+m-n.

In particular, if, for instance, m=n and the length of each test sequence is less than r, then there is at least one state of M_S (reachable with a sequence of length r) that is not traversed by the given test suite. We have, in fact, an even stronger condition which explicitly requires that all states of M_S must be covered by the given test suite.

Necessary Condition 3.1.9. If, for some m, $m \geq n$, TS is m-complete for M_S, then there exists a state cover V such that $V \prod$ **AP**(TS).

It is implied by the results of [Vasi73] and noted in [Yann91] that the set X^{m-n} must also be incorporated in any m-complete test suite in such a way that the traversal of all the m possible states in the implementation is ensured. This requirement is intuitively clear, but it still remains to be formalized as a necessary condition in a proper way.

Based on the requirement that all transitions between states have to be traversed by a test suite, we can formulate the following conditions.

Necessary Condition 3.1.10. If, for some m, $m \geq n$, TS is m-complete for M_S with n states and the degree of accessibility r, then the length of its longest sequence will not be less than r+m-

n+1.

Necessary Condition 3.1.11. If, for some m, m≥n, TS is m-complete for M_S, then there should exist a transition cover TC such that TC∏**AP**(TS).

Here it is required that at least all transitions in the specification machine are covered. An even stronger necessary condition could be stated if we require that the next state of every transition must be eventually distinguished from any other state. We believe that further research will expand the list of conditions for (in-) completeness of a test suite with respect to the various types of FSMs, eventually making fault coverage analysis much more practical. These conditions also help to identify which obligatory part of the test suite is missing and should be added in order to increase its fault coverage, if required.

3.2. Tests for partial deterministic FSMs

A specific feature of partial machines is that they have "undefined" transitions. There are several different conventions used for "undefined" transitions [PBD93]. Under the completeness assumption, the given partial FSM is substituted by a quasi-equivalent complete FSM, and the problems of test derivation and fault coverage analysis are then reduced to those associated with complete machines. The other conventions for undefined transitions, namely, "undefined by default" (also called “don’t care”) and "forbidden", require a test suite constructed from acceptable (defined in M_S) input sequences. If some test sequence covers an undefined transition, then, under the "undefined by default" convention, its verdict must be inconclusive; under the "forbidden", this test sequence is not executable and should be excluded from the test suite. Several testing situations requiring the model of partial FSMs with these conventions have been already identified, see [PYL93], [PYD94]. This demonstrates that the completeness assumption is not as universal as it was once considered earlier in protocol engineering, and partial machines deserve a special treatment.

Test derivation, and therefore fault coverage analysis for reduced partial (deterministic) machines, can still be performed in a manner similar to that used for complete machines, because all states are distinguishable and can be identified. Problems arise when some states are not distinguishable, i.e. the machine becomes unreduced. In this case, the number of states of the given machine does not characterize the properties of the machine essential for fault coverage. As shown in [Yevt89], a more subtle parameter, the so-called fuzziness degree, better characterizes such a machine.

Consider the partition of the set S of states of the given FSM M_S into subsets $\{S_1, ..., S_f\}$ such that every subset includes only pairwise distinguishable states. The number f of subsets in the minimal partition is called the *fuzziness degree* of the PFSM M_S [YePe89], [LPB94b]. If M_S happens to be complete and minimal, then f=1. The larger the number of pairs of quasi-equivalent states, the higher the fuzziness degree usually is. For a "fully unreduced" machine, f = n.

In spite of these distinctive features of partial machines, it is still possible to extend the notion of regular test suites (introduced for completely specified machines) to these machines.

The test suite TS is called an *a-exhaustive* test suite for an FSM M_S, if **AP**(TS) □$X^a \leftrightarrow X^*_A$, where X^*_A is the set of acceptable input sequences of M_S.

Sufficient Condition 3.2.1. Let TS be an a-exhaustive test suite for the FSM M_S with n states. If $a \geq mn$, then TS is m-complete for M_S.

The bound mn can be reached in very special, pathological FSMs, deliberately created (an example can be found in [YePe89]). Luckily, the existing protocol machines do not resemble these FSMs.

Similarly, the notion of a b-canonical test suite is generalized to partial machines.

Sufficient Condition 3.2.2. Let TS be a b-canonical test suite for the FSM M_S with n states, the state cover V, the degrees of distinguishability d, and fuzziness f. If $b \geq fm-n+d+1$, then TS is m-complete for M_S.

In the special case, where f=1, this condition reduces to the condition 3.1.3.

It was shown in [Petr91], [YePe90], [LPB94a] that c-characterization test suites (and therefore sufficient conditions for the completeness of test suites such as 3.1.4 - 3.1.6) can be generalized to partial, even unreduced machines. The basic idea behind these methods which generalizes the traditional state identification approach, lies in counting distinct states traversed by a test sequence to ensure that all possible states and transitions in an IUT are covered and tested. Even though these methods produce test suites whose structure is like those for complete machines, the complete test suites for partial machines may be longer than those for complete machines with comparable parameters. In this case, the degree of distinguishability has a tight bound of n(n-1)/2, instead of n-1 for complete machines. Moreover, certain states might not be distinguishable at all.

4. DIFFERENT APPROACHES TO FAULT COVERAGE ANALYSIS

Fault coverage analysis is to calculate **FC**(m, M_S, TS), for the given TS, m and M_S. As is clear from its definition, in order to calculate the fault coverage, we need to find the number of machines in **Impl**(m, M_S) which cannot pass the given test suite TS, or we need to find the number of machines in **Impl**(m, M_S) which can pass the given test suite TS. However, their exact values are in general too difficult to find. In the following subsections, we will review the approaches that are proposed to tackle this problem.

4.1. Exhaustive mutation analysis

Exhaustive mutation analysis can be used to find the exact value of $\mathbf{N_p}$(m, M_S, TS). The idea is to enumerate all the implementation machines in **Impl**(m, M_S) and to execute (simulate) all of them, one by one, against the given test suite TS. By counting the number of implementation machines that can pass the test suite TS we obtain the exact value of $\mathbf{N_p}$(m, M_S, TS). In this way, the precise value of fault coverage **FC**(m, M_S, TS) can be calculated. The complexity of this approach is high since the total number of machines $\mathbf{N_t}$(m, M_S) is equal to $(m|Y|)^{m|X|}$, where X and Y are the input and output alphabets of M_S. The exhaustive approach is only applicable for rather small m, X and Y.

4.2. Monte-Carlo simulations

For large m, X and Y, the exhaustive mutation analysis approach is not feasible due to the huge number of implementation machines that should be executed. Therefore, the partial mutation analysis approach based on Monte-Carlo simulation has been proposed in [SaSp90], [DDB91], [SiLe89] and [MCS93]. The idea is to first randomly generate k implementation machines in **Impl**(m, M_S), where k is a relatively small integer. Each randomly generated machine is then executed to see if it can or cannot pass the test suite. A randomly generated machine capable of passing the test suite is then further checked to see if it conforms to the specification machine M_S. In this way, we can find k', the number of randomly generated machines which cannot pass the test suite, and k'', the number of randomly generated machines which conform to M_S. Then k'/(k - k'') gives an approximate value of the fault coverage **FC**(m, M_S, TS). The set of implementation machines **Impl**(m, M_S) is often partitioned into several classes, such as the class of machines containing one transfer fault (only Type 1 change is applied once), the class of machines containing two transfer faults (only Type 2 change is applied twice) and so on [DDB91], [SiLe89], [MCS93]. For each class of implementation machines, the Monte-Carlo simulation is then applied. Therefore, we have the estimated fault coverage for each class of implementation machines.

4.3. Structural analysis

Both the exhaustive mutation analysis and Monte-Carlo simulation approaches require (part of) the implementation machines in **Impl**(m, M_S) to be generated and executed against the given test suite. When m = n, i.e. the upper bound m on the number of states of an implementation is equal to the number of states n of M_S, the structural analysis approach proposed in [YPB94b] can be used to avoid the generation and execution of implementation machines. The idea of this approach is to obtain an estimated value of $\mathbf{N_p}(n, M_S, TS)$ by directly analyzing the structure of specification machine against the test suite. Let $\mathbf{N_p}(n, M_S, TS)$ denote the estimated value of $\mathbf{N_p}(n, M_S, TS)$. Then

$$\mathbf{FC}_e(n, M_S, TS) = \frac{\mathbf{N_t}(n, M_S) - \mathbf{N_p}(n, M_S, TS)}{\mathbf{N_t}(n, M_S) - \mathbf{N_c}(n, M_S)}$$

gives us the estimated fault coverage. The value of $\mathbf{N_t}(n, M_S)$ is given by $\mathbf{N_t}(m, M_S) = (m|Y|)^{m|X|}$ (see Section 2), which corresponds to the fact that for each transition with given state and input, there are m|Y| pairs of possible next state and output values. The value of $\mathbf{N_c}(n, M_S)$ is given by $\mathbf{N_c}(n, M_S) = (n-1)!$. The formula (n-1)! reflects the fact that any permutation among the names of the states of the FSM, except for the initial state, does not change its behavior, and there are (n-1)! such permutations.

The technique used to derive the estimated value $\mathbf{N_p}(n, M_S, TS)$ is based on the following concepts. The tail state s_j of a transition $< s_i; x/y; s_j >$ in M_S is said to be *distinguished* from another state s_k by TS if there are α, αx, $\alpha x\gamma$, β, $\beta\gamma$ [**AP**(TS) such that

$$\delta(s_1, \alpha) = s_i, \delta(s_1, \alpha x) = s_j, \ \delta(s_1, \beta) = s_k \text{ and } \lambda(\delta(s_1, \alpha x), \gamma) \neq \lambda(\delta(s_1, \beta), \gamma).$$

For a transition $< s_i; x/y; s_j >$ in M_S, we use **Tail_Dis**($< s_i; x/y; s_j >$, TS) to denote the set of states from which the tail state s_j of transition $< s_i; x/y; s_j >$ is distinguished. Then, **Tail_NDis**($< s_i; x/y; s_j >$, TS) is the set of states from which the tail state s_j of transition $< s_i;$

$x/y; s_j >$ is not distinguished.

For a transition $< s_i; x/y; s_j >$ covered by TS, we can easily calculate **Tail_Dis**($< s_i; x/y; s_j >$, TS) and therefore **Tail_NDis**($< s_i; x/y; s_j >$, TS). Since a state in **Tail_NDis**($<s_i; x/y; s_j>$, TS) is not distinguished from the tail state s_j of $< s_i; x/y; s_j >$, changing the tail state s_j of transition $< s_i; x/y; s_j >$ to any state in **Tail_NDis**($< s_i; x/y; s_j >$, TS) will give us an implementation machine which can pass the test suite TS. Therefore, there are |**Tail_NDis**($< s_i; x/y; s_j >$, TS)| possible ways to make such a Type 1 change. However, we should note that, as transition $< s_i; x/y; s_j >$ is covered by TS, changing the output symbol "y" to any other output symbol (Type 2 change) will result in an implementation machine which is very likely to be detected by TS. Therefore, to guarantee generation of an implementation machine which can pass TS, we have only one choice: keeping the output "y" of the transition. Consequently, the given transition $< s_i; x/y; s_j >$ covered by TS gives us |**Tail_NDis**($< s_i; x/y; s_j >$, TS)| possible ways of generating an implementation machine which can pass the test suite TS. It can be noted that if the condition 3.1.7 is satisfied then **Tail_NDis**($< s_i; x/y; s_j >$, TS) = { s_j } and |**Tail_NDis**($< s_i; x/y; s_j >$, TS)| = 1.

For a transition $< s_i; x/y; s_j >$ not covered by the given test suite TS, its tail state s_j is not distinguished by TS from any state. Therefore, we have **Tail_NDis**($< s_i; x/y; s_j >$, TS) = { s_1, s_2, ..., s_n }. Furthermore, since the transition is not covered by TS, we can change the output symbol "y" to any symbol in Y and still get a mutant machine which can pass TS. Combining the possible ways of changing the tail state (Type 1 change) and the possible ways of changing the output (Type 2 change), we can immediately conclude that, for the transition $< s_i; x/y; s_j >$ which is not covered by TS, there are |**Tail_NDis**($< s_i; x/y; s_j >$, TS)| x |Y| = n|Y| possible ways to generate an implementation machine which can pass the test suite. As a result, the set of all the transitions not covered by TS gives us $(n|Y|)^u$ choices to generate an implementation machine which can pass TS (where u is the number of transitions not covered by TS).

As we have discussed in Section 2, an implementation machine is completely defined. Therefore, for the given specification machine M_S which is in general partially specified and has the specification domain D_S, we need to apply the Type 3 operation to add an extra transition for each (s_i, x) [(S x X) \ D_S. Since the tail state of such an extra transition can be any of the n states s_1, s_2, ..., s_n and the output symbol can be any one in Y, we know that there are a total of $(n|Y|)^{n|X| - |D_S|}$ possible ways to generate an implementation machine by adding $n|X| - |D_S|$ extra transitions.

Following from the above discussions, we have

$$\mathbf{N_P}(n, M_S, TS) = p\,(n|Y|)^{n|X| - |D_S| + u \prod_{\substack{< s_i; x/y; s_j > \\ \text{covered}}} |\text{Tail\_NDis}(<s_i; x/y; s_j >, TS)|}$$

implementation machines in **Impl**(n, M_S) which are estimated to be able to pass the given test suite. Here u is the number of transitions not covered by TS, and p is a factor which represents the estimated number of permutations that are possible among those states that are

identified by the test suite. (Note that the factor p was not included in our previous version of this formula published in [YPB94b]).

An estimation for p may be obtained by the following reasoning. In the case of a completely specified FSM and a nearly complete test suite which covers all states and transitions (u = 0), the above formula becomes

$$(n-1)! \prod_{\substack{< s_i;\, x/y;\, s_j > \\ \text{covered}}} |\text{Tail\_NDis}(< s_i;\, x/y;\, s_j >, TS)|$$

where the factor (n-1)! reflects the fact that any permutation of the names of the states of the FSM, except the initial one, does not change its behavior. For every implementation machine counted by the product expression of the formula, there are (n-1)! isomorphic machines. The value of p is (n-1)! in this case.

At the other extreme, a test suite of length one for a completely specified FSM, covering a single transition, does not identify any state. Such a test suite allows for n different next states for the covered transition and for (n|Y|) possibilities for each of the other transitions. This gives a total of $n(n|Y|)^{n|X|-1}$ different implementations. The factor n corresponds to the fact that **Tail_NDis** is equal to n for the transition covered. Comparing this with the above formula for $\mathbf{N_p}(n, M_S, TS)$, we find p = 1; this corresponds to the fact that only the initial state is identified and no other permutations exist. This confirms that $1 \leq p \leq (n-1)!$.

While in the above extreme cases, it is possible to find the exact number of permutations of identified states, in all the other cases, we can only try to estimate the value of p from the list of state pairs distinguished by the TS. Assuming that the number of state permutations allowed by the TS is proportional to the number d of state pairs distinguished by the TS, we obtain the following estimation for p:

$$p = ((n-1)! - 1)\frac{d}{n(n-1)/2} + 1.$$

The structural analysis approach has a very low computational complexity $O(L^2)$, where L is the size of the test suite in terms of the total number of inputs in the test suite. It has been implemented and a number of experimental results have been reported in [YPB94b].

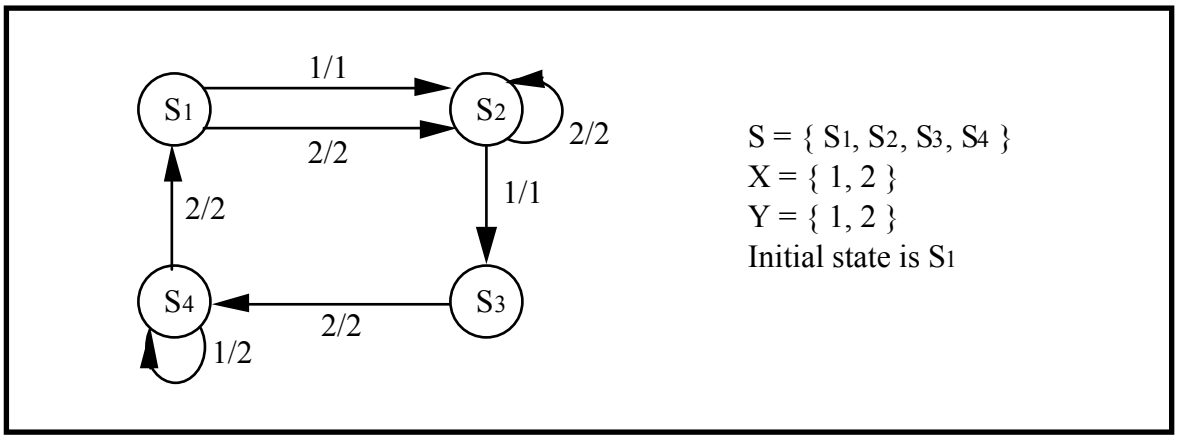


Figure 1: An example FSM

Here we present the results for the following eleven test suites derived from the FSM M_S presented in Figure 1.

$TS_1 = \phi$
TS_2 = { r.1 }
TS_3 = { r.1.1.2, r.2.1.2.1 }
TS_4 = { r.1.1.2, r.2.1.2.1, r.2.2.1 }
TS_5 = { r.1.1.2, r.2.1.2.1, r.2.2.1.2 }
TS_6 = { r.1.1.2, r.2.1.2.1.1, r.2.2.1.2 }
TS_7 = { r.1.1.2, r.2.1.2.1, r.2.2.1.2.2 }
TS_8 = { r.1.1.2, r.2.1.2.1.1, r.2.2.1.2.2 }
TS_9 = { r.1.1.2, r.2.1.2.1.1, r.2.2.1.2.2.1 }
TS_{10} = { r.1.1.2, r.2.1.2.1.1, r.2.2.1.2.2.1.2.1 }
TS_{11} = { r.1.1.2.1.1.1, r.1.1.2.1.2.1, r.1.2.1.2.1, r.1.1.2.2.1.1.2.1, r.1.1.2.2.1.2.1,
r.1.1.2.2.2.1, r.1.2.2.1, r.2.1.2.1.1, r.2.2.1.2.2 }

Applying our structural analysis approach to these test suites yields the estimated fault coverage listed in the third column of Table 1. The numbers of state pairs distinguished by test suites are given in the second column. To assess the accuracy of these estimated fault coverage values, we compare them with the precise fault coverage values for these test suites. Fortunately, for the small specification machine given in Figure 1, we have been able to make an exhaustive mutation analysis. We have written a program which generates and executes one by one all the $(4 \times 2)^{(4 \times 2)} = 16777216$ possible implementation machines against each of the above eleven test suites. Therefore, we have been able to calculate the precise fault coverage for these test suites and the results are listed in the forth column of Table 1. The differences between the estimated and precise fault coverage are listed in the fifth column of Table 1. The last column gives the relative error for each test suite, i.e. the absolute deviation value devided by 1 - $\mathbf{FC_p}(n, M_S, TS)$.

TS_i	d	$\mathbf{FC_e}(n,M_S,TS)$	$\mathbf{FC_p}(n,M_S,TS)$	Deviation	Relat. error
1	0	0.00000%	0.00000%	0.00000%	0.00000%
2	0	50.00014%	50.00014%	0.00000%	0.00000%
3	3	97.69313%	99.25871%	-0.43442%	58.55348%
4	4	99.52420%	99.66497%	-0.14077%	42.01713%
5	4	99.52420%	99.66898%	-0.14478%	43.73754%
6	4	99.76223%	99.77655%	-0.01432%	6.40859%
7	4	99.76223%	99.88084%	-0.11861%	99.53844%
8	4	99.88132%	99.92032%	-0.03900%	48.94578%
9	5	99.97884%	99.95232%	0.02652%	55.62081%
10	6	99.99341%	99.97926%	0.01415%	68.22565%
11	6	100.0000%	100.0000%	0.00000%	0.00000%

Table 1: Fault coverage for the FSM in Figure 1

The above considerations apply also to the order coverage. In this case, we have, in fact, a very simple way to estimate the order coverage of the given test suite based solely on the number u of transitions uncovered. If we use logarithms of the numbers of machines as we did in the formula for the order coverage then the fault coverage can be approximated as follows:

$$\mathbf{FC_o}(n, M_S, TS) \approx \frac{D_S - u}{D_S}.$$

The order coverage of the given test suite is approximately proportional to the number of transitions covered by it. This formula reflects a simplified intuitive understanding of the fault coverage as a percentage of transitions covered by the test suite.

4.4. Deciding the m-completeness of a test suite

The other type of fault coverage analysis is used to decide if a given test suite is m-complete with respect to a specification machine. In certain cases, m-completeness can be verified based on the sufficient conditions given in Section 3, however, an arbitrary test suite requires more complicated methods.

In the case of deterministic FSMs, this problem is actually very close to an automaton identification problem considered in automata theory [Gill66], [Kell71], [Sier93]. Given a set of input/output sequences TS, list all the machines within the given bound m on the number of states, which can produce it. The methods solving this problem can be classified into two categories: the enumerative and constructive methods. The classical approach to this problem is based on general state minimization techniques for partial FSMs, as any deterministic TS can be interpreted as a partially specified deterministic machine. By its nature, the problem of state minimization involves enumeration of solutions. However, due to special structure of this partial FSM it is possible to find improved state merging methods. In particular, an efficient method was elaborated in [Kell71] for the case where TS is a single sequence, which was shown to be faster and more efficient than the state minimization algorithm for arbitrary partial FSMs. Based on this result, we have suggested [YPB94a] a more general state minimization procedure to treat test suites with multiple sequences (the reset assumption). As often happens, a long standing problem is later tried with a new approach: in [VuKo90], [ZhCh94], the constraint satisfaction techniques of artificial intelligence were applied to generate machines that can pass the given test suite. Automated tools for fault coverage have been implemented independently for the two recent techniques [YPB94a] and [ZhCh94].

Once we construct the machines which have no more than m states and can pass the given test suite, the remaining problem becomes simple. If all these machines are (quasi-) equivalent to the given specification machine M_S then the TS is m-complete; otherwise, it is not m-complete. If all non-conforming machines constructed from TS are retained, then its fault coverage can also be characterized numerically. Techniques for comparing two FSMs with respect to the quasi-equivalence are explained, for instance, in [Gill62] and in [DaSa88], where this relation was called the containment.

With our approach [YPB94a], for example, the checking of the m-completeness of a test suite proceeds in three steps:
Step 1: Convert the given test suite TS into a tree machine M which is essentially the same as the so-called test tree in some literature.

Step 2: Call the state minimization procedure to minimize the tree machine M.
Step 3: If a minimal (or reduced form) of the tree machine is found in Step 2 which does not conform to M_S and has no more than m states, the test suite is not m-complete; Otherwise, it is m-complete.

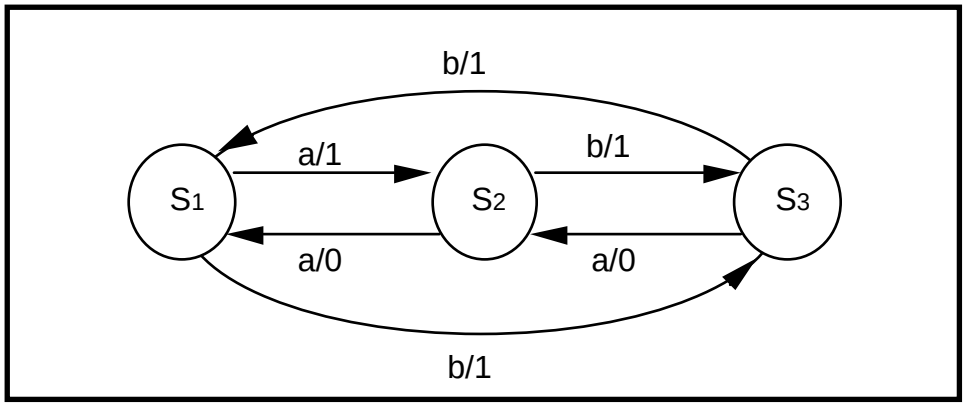
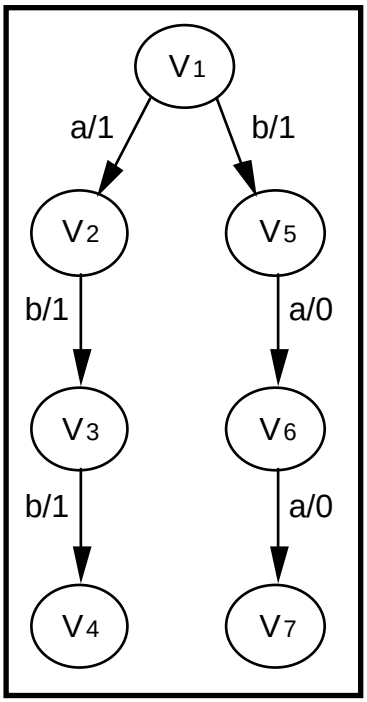


Figure 2: An example specification FSM



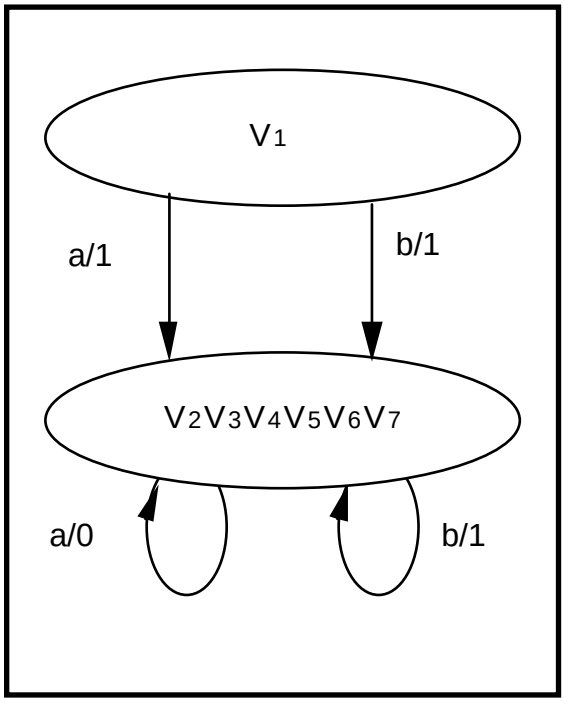


Figure 3: The tree machine **Figure 4: The minimal form found**

As a concrete example, let us consider the test suite TS = {r.a.b.b, r.b.a.a} derived from the specification machine shown in Figure 2. We want to check if this test suite is 3-complete. The tree machine M representing this test suite is shown in Figure 3. By using the state minimization procedure, a minimal form of the tree machine can be found which has two states as shown in Figure 4, and does not conform to the specification machine of Figure 2. The two states of the minimal form are actually obtained by merging the states of the tree machine in Figure 3. It can therefore be concluded that the given test suite is not 3-complete.

The approach presented in [YPB94a] can be applied to specification machines which are partially specified, initially connected and possibly non reduced. The corresponding tool provides three options when performing fault coverage analysis: (1) searching for a mutant which does not conform to the specification machine M_S; (2) searching for a mutant which has the smallest number of states and does not conform to the specification machine M_S; and (3) searching for all the mutants which do not conform to the specification machine M_S. The upper bound on the complexity of this approach is $O(m^L)$, where L is the number of states of the tree machine representing the test suite. However, a number of experiments have shown that the actual complexity in practice is far less than this upper bound.

Another approach for deciding the m-completeness of a test suite can be found in [ZhCh94]. The idea behind this approach comes from the CSP method for test suite generation [VuKo90]. The so-called variables in this approach correspond to the states of the tree machine (representing the test suite) in the state minimization based approach [YPB94a]. The preprocessing and backtracking algorithms presented in [ZhCh94] correspond to the two techniques used in [YPB94a], namely the generation of compatible partitions (coverings) and the verification of the closure property, respectively. Therefore, for a given completely specified specification machine, these two approaches give the same computational complexity. In fact, the construction of a minimal FSM from the given set of input/output sequences was shown to be computationally hard [Gold78].

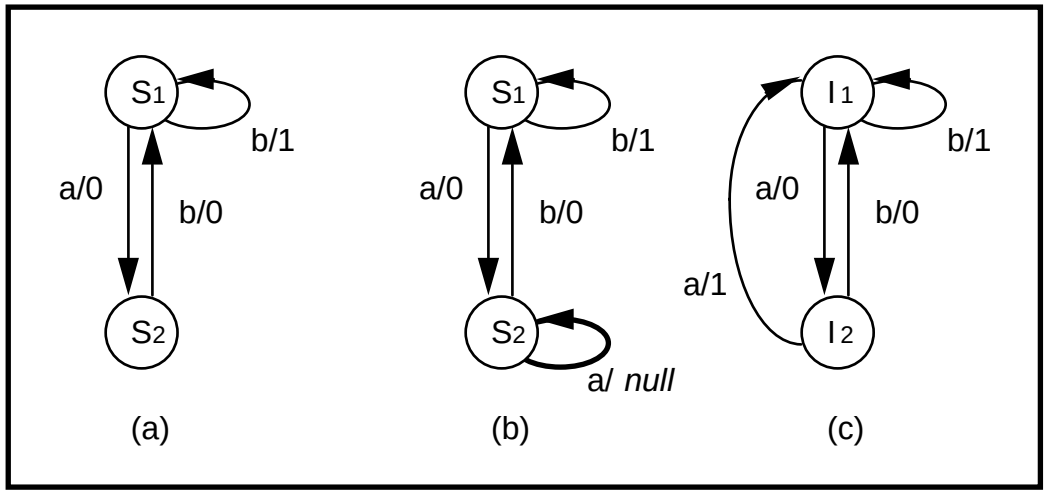


Figure 5: Difference between strong and weak conformance

A crucial difference between these two approaches can be observed in the case of a partially specified specification machine. Zhu and Chanson's approach [ZhCh94] is based on the equivalence relation (called *strong conformance* [SiLe89]) and therefore can only be applied to completely specified machines. In the case that a given specification machine is only partially specified, the specific completeness assumption is made that the machine will remain in the present state without producing any output (or *null* output) for any unspecified input. As an example, the partially specified specification machine given in Figure 5(a) will have to be converted into the machine shown in Figure 5(b). On the other hand, the state minimization based approach [YPB94a] uses the quasi-equivalence relation (called *weak conformance* in [SiLe89]) and accordingly can be directly applied to partially specified machines, such as the one given in Figure 5(a). The non-specified part is treated as "don't

care" and can be implemented in an arbitrary way. For instance, the implementation machine shown in Figure 5(c) can pass the test suite { r.a.b, r.b.a } and is taken as a conforming implementation. However, with Zhu and Chanson's approach, this implementation will have to be listed as a non-conforming mutant as it is not equivalent to the specification machine given in Figure 5(b).

5. FURTHER PROBLEMS RELATED TO FAULT COVERAGE ANALYSIS

There are several directions for future work in fault coverage analysis of protocol tests. In what follows, we briefly address some of them.

5.1. Extended FSMs

An extended FSM model in its most general form makes it impossible to directly apply the FSM-based techniques for fault coverage evaluation. The problem is that it is unrealistic to execute during testing all the possible combinations of protocol parameters (as implied by the methods considered in Section 4), unless the domains of variables are reduced in such a way that the EFSM can be unfolded into an FSM of reasonable size. Next issue is the executability of test sequences with respect to the EFSM specification. Since it is not obvious that any given input/output sequence can be produced by the EFSM, the test suite should be validated [NaSa93] before being analyzed for its coverage. Last but not least, there is the problem of finding an appropriate model of faults in terms of EFSMs. The notion of fault coverage for EFSMs should be in general based on a more restricted type of faults than those modeled by pure FSMs while at the same time allowing one to deal with both control and data flows of protocols. However, regardless of some initial work in this direction [WaLi93], [GuPr90], [FaPe90], it is not yet well understood which fault models are most appropriate for the combined testing of control and data aspects of protocols. The approach based on user-defined fault models [PeYe92], [WaLi93] may be helpful in dealing with the EFSM specifications.

5.2. Nondeterministic FSMs and LTSs

By their nature, nondeterministic systems are more difficult to analyze than the deterministic ones, and fault coverage analysis is no exception. To the best of our knowledge, no direct results on fault coverage analysis with respect to nondeterministic FSM or LTS specifications have been reported so far. Nondeterminism causes several problems such as the following:

- an additional assumption about IUTs must be made, namely, the complete testing assumption, viz. the IUT should exhibit during testing all its nondeterministic choices;
- not just one, but several relations may be used as a conformance relation, therefore a test suite complete with respect to one relation might not be complete with respect to another (finer) relation;
- the notion of a fault must be defined in the context of a particular conformance relation. A simple mutation technique employed in the deterministic case may fail to explain faults in nondeterministic implementations.

These problems are also addressed in test derivation for nondeterministic systems (FSMs and LTSs). As work in this direction continues, we may expect that these results will offer solutions to fault coverage analysis. Note that test derivation for NFSMs, for instance, is also a relatively new research subject. Several heuristic approaches can already be found in the

literature; however, they are of little help in tackling the problems related to fault coverage analysis for nondeterministic machines. There are also some methods which guarantee m-completeness of test suites w.r.t. the quasi-equivalence and the reduction relations [YLP91], [PYL93], [LPB94a], [LPB94b]. As an example, in [LPB94b], it is proven that a harmonized c-characterization test suite, that is, a TS of the form $(VX^{fm-n+1} \leftrightarrow X^{*}_{A}) \square H$, is m-complete for an arbitrary observable (possibly unreduced, partial and nondeterministic) FSM with respect to the quasi-equivalence relation. In fact, nondeterminism does not seem to create any insurmountable difficulties for test derivation and analysis with respect to this conformance relation. We conjecture that several results and approaches discussed in this paper for deterministic machines can be extended to nondeterministic ones in a similar way. The new essential feature is that a test suite includes not only input events but also output events, a complete test suite must foresee all the specified reference reactions to its input sequences to ensure the (quasi-) equivalence between an IUT and the specification machine. There is a need, therefore, for a kind of validation of a given test suite to further decide its completeness. Such a validation is quite straightforward in the case of the quasi-equivalence relation. However, in the case of the reduction relation, when a conforming implementation is allowed to produce a subset of specified output sequences, it is often possible that only deterministic implementations are submitted for testing. Then, it becomes less obvious whether the input/output sequences included in the given test suite can be produced by a deterministic reduction of the specification NFSM [PYL93]. The study of the reduction relation between NFSMs initiated in [PYL93] and [PYB94] should help to better understand the issues related to fault coverage analysis for nondeterministic machines.

In the case of FSM specifications of protocols, the choice of the appropriate conformance relation for test derivation and analysis is limited by a small number of alternatives and usually can be made in an obvious way. If, however, the protocol is abstracted into an LTS model, then this choice becomes less evident. Both test derivation and test analysis rely on the notion of a faulty implementation defined by the chosen conformance relation. This relation might be either the equivalence or a preorder in a particular semantics. At the same time, it is not even clear which LTS semantics better reflects the conformance requirements for protocol implementations. The semantics which determines the conformance with respect to the given LTS specification affects the way deadlocks are treated in conformance tests and is crucial in analyzing their fault coverage.

Recent results in test derivation for LTSs demonstrate a strong tendency toward reusing the ideas developed in the context of FSM-based testing for LTS-based testing. In particular, several papers extend the state identification approach to specifications given in form of an LTS [FuBo91], [CKM92], [Arkk93]. It is also suggested [PBD93] that tests for a given LTS could be obtained from tests directly generated by the existing FSM-based methods from a proper FSM constructed from the LTS in the chosen semantics. As the work in this direction continues we may well expect that FSM-based results on fault coverage analysis will also be adopted in the context of the LTS formalism. Some conditions for completeness of test suites discussed above can be reformulated for LTSs and a relevant equivalence relation. The idea of checking the completeness of a test suite through state minimization seems to work for LTSs as well. The basic ideas behind the structural analysis explained in Section 4.3 can also be adopted for this case. However, this is only one possible scenario of future research in fault coverage analysis in terms of the LTS model.

5.3. Test architectures

So far we have considered strategies for fault coverage analysis in the case where any given test suite can be executed by an appropriate (abstract) tester, since test events were assumed to be events on the interfaces of an IUT. As well known, more complex test architectures hinder the achievement of full fault coverage. Therefore a particular test architecture for which the given test suite is designated has to be taken into consideration.

As an example, consider the distributed test method. The abstract tester is implemented in two testers, and in certain testing situations, it is required that the test suite should also perform a synchronization between the testers. This requirement imposes an additional restriction on the selection of test sequences [SaBo84], [BoUr91], [LDB93]. As a result, all possible m-complete test suites may not be synchronizable. It is noted [LDB93] that the distributed test architecture weakens fault coverage in the sense that certain faults can never be detected by synchronizable tests. In this context, fault coverage analysis of the given synchronizable tests has to be adjusted to consider only detectable faults. However, even the problem of characterization of such faults still remains open.

A similar deterioration of testability arises when a test suite to be analyzed for its completeness must be executed through other components, for instance, in the embedded test architecture. In this case, the given test suite is initially composed of events on interfaces which are not directly accessible by tester(s). It must first be analyzed if all inputs can actually be excited by, and outputs uniquely delivered to, tester(s). Even if it is the case, depending on properties of the environment of the IUT, certain faults can be masked or cannot be activated, as discussed in the accompanying paper [PYD94]. Similar observations are reported for LTS specifications [DAS93]. As follows from these results, the problem of fault coverage analysis for the embedded testing becomes more cumbersome. At the same time the idea of determining a testable representation of the IUT [PYD94] could provide a possible solution.

We conclude that future research should address the influence of the test architecture on the fault coverage of a test suite.

6. CONCLUSIONS

Fault coverage of tests for the given finite-state specification of a protocol has been recognized as an essential and challenging problem by the protocol engineering community. Test analysis and test derivation are closely related topics. Moreover, results achieved in one direction help to make further progress in the other, as we have tried to demonstrate in this paper. Our discussion on fault coverage methods would not be complete unless additional potential areas of their application are mentioned.

Besides their immediate predestination, the techniques for fault coverage analysis can be used for incremental test suite development. When a given test suite is found to be not m-complete, then a machine should have been generated which is not quasi-equivalent to the specification machine. An additional test case can now be derived which distinguishes this generated machine from the specification machine. The test suite, which includes the newly generated additional test case, is then checked again for m-completeness. This process is repeated until the test suite has achieved m-complete fault coverage. However, it is not yet clear how much

one could benefit from this approach compared to the conventional test derivation methods which guarantee complete fault coverage.

Another possible application of the techniques for deciding the m-completeness of tests is to the diagnostics of FSM implementations [GhBo92]. If an FSM implementation based on an FSM specification fails to pass a given test suite, then a tree machine is constructed with the input sequences in the test suite and the corresponding output sequences observed during the testing. This tree machine is then minimized. Any reduced machine obtained definitely does not conform to the given specification machine, and therefore represents a possible faulty implementation machine. By comparing this reduced machine with the specification machine, we are able to tell what implementation faults are in the implementation machine. However, such a diagnosis may become less informative if several such reduced machines are obtained in the cases of multiple faults and tests with a poor fault coverage.

Fault coverage analysis is also useful for elaborating the heuristics which could guide test derivation in the absence of an explicit fault model. The heuristic methods attempt to produce a test suite of a reasonable size at the price of a possibly weakened fault detection power, compared to the methods which guarantee complete fault coverage. In particular, there is an ever growing number of methods where a test sequence is constructed from the UIO-sequences in one way or another without having an explicit fault model beforehand. As recently shown in [ZhCh94], further reducing the length of a test sequence by overlapping test segments may lead to a loss of fault coverage. In this context, the fault coverage analysis of UIO-based tests undertaken in [MCS93], [LoSh92], [MiPa92], [ZhCh94] could lead to refined test selection criteria with a better compromise between the fault coverage and the size of a test suite.

Acknowledgments. This research was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at the Université de Montréal. The authors would like to thank S. A. Ezust for comments.

REFERENCES

[AlVu93] J. Alilovic-Curgus and S. T. Vuong, "A Metric Based Theory of Test Selection and Coverage", IFIP Transactions, the Proceedings of the IFIP 13th Symposium on Protocol Specification, Testing and Verification, Liege, Belgium, 1993, pp. 289-304.

[Arkk93] J. Arkko, "On the Existence and Production of State Identification Machines for Labelled Transition Systems", FORTE'93.

[Boch91] G. v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo, "Fault Models in Testing", IFIP Transactions, Protocol Test Systems, IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland.

[BoPe94] G. v. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing", ISSTA'94, ACM International Symposium on Software Testing and Analysis, Seattle, U.S.A., 1994.

[BoUr91] S. Boyd and H. Ural, "The Synchronization Problem in Protocol Testing and its Complexity", Inf. Proc. Letters, Vol.40, No.8, 1991.

[CKM92] A. R. Cavalli, S. U. Kim, and P. Maigron, "Automated Protocol Conformance Test Generation Based on Formal Methods for LOTOS Specifications", Proceedings of

IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1992), Ed. by G. v. Bochmann, R. Dssouli, and A. Das, 1992, North-Holland.

[Chow78] T. S. Chow, “Test Design Modeled by Finite-State Machines”, IEEE Trans., SE-4, No.3, 1978, pp. 178-187.

[DAS93] K. Drira, P. Azema, B. Soulas, and A.M. Chemali, "Testability of a Communicating System through an Environment", Proc. of TAPSOFT'93.

[DaSa88] A. Dahbura and K. Sabnani, “Experience in Estimating Fault Coverage of a Protocol Test”, in Proc. IEEE INFOCOM’88, 1988, pp. 71-79.

[DDB91] M. Dubuc, R. Dssouli and G. v. Bochmann, “TESTL: A Tool for Incremental Test Suite Design Based on Finite State Model”, IFIP Transactions, Protocol Test Systems, IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland.

[FaPe90] A. Faro and A. Petrenko, "Sequence Generation from EFSMs for Protocol Testing", Participants' Proc. of COMNET'90.

[FuBo91] S. Fujiwara and G. v. Bochmann, "Testing Non-deterministic State Machines with Fault Coverage", IFIP Transactions, Protocol Test Systems, IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland.

[Fuji91] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models", IEEE Trans., SE-17, No.6, 1991.

[GhBo92] A. Ghedamsi and G. v. Bochmann, "Test Result Analysis and Diagnostics for Finite State Machines", Proc. of the 12-th Int. Conference on Distributed Systems, 1992.

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

[Gill66] A. Gill, “Realization of Input-Output Relations by Sequential Machines", J. of the ACM, Vol.13, No.1, 1966, pp. 33-42.

[Gold78] E M. Gold, "Complexity of Automaton Identification from Given Data", Information and Control", No.3. 1978, pp.302-320.

[GuPr90] F. Guo, R. Probert, "E-MPT Protocol Testing: Preliminary Experimental Results", IWPTS'90.

[HoSt93] D. Hoffman and P. Strooper, "A Case Study in Class Testing", in Proc. of CASCON'93, Toronto, Canada, 1993, pp. 472-482.

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

[IS9646] ISO 9646. OSI Conformance Testing Methodology and Framework.

[Kell71] J. Kella, “Sequential Machine Identification”, IEEE Trans., Vol. C-20, No.3, 1971, pp. 332-338.

[LDB93] G. Luo, R. Dssouli, G. v. Bochmann, P. Venkataram, and A. Ghedamsi, "Generating Synchronizable Test Sequences based on Finite State Machines with Distributed Ports", Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 139-153.

[LoSh92] F. Lombardi and Y. N. Shen, "Evaluation and Improvement of Fault Coverage of Conformance Testing by UIO Sequences", IEEE Trans. Commun., Vol. COM-40, No.8, 1992, pp. 1288-1293.

[LPB94] G. Luo, A. Petrenko, and G. v. Bochmann, "Generating Tests for Communication Software Modeled by Partially-Specified Finite State Machines", IEEE/ACM Transactions on Networking, (to appear in 1994).

[LPB94a] G. Luo, A. Petrenko, and G. v. Bochmann, "Test Selection based on Communicating Nondeterministic Finite State Machines using a Generalized Wp-Method", IEEE Trans., Vol. SE-20, No.2, 1994, pp.149-162.

[LPB94b] G. Luo, A. Petrenko, and G. v. Bochmann, "Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines", IWPTS'94.

[MCS93] H. Motteler, A. Chung, and D. Sidhu, "Fault Coverage of UIO-based Methods for Protocol Testing", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 21-33.

[MiPa92] R. E. Miller and S. Paul, "Structural Analysis of a Protocol Specification and Generation of a Maximal Fault Coverage Conformance Test Sequence", submitted for publication.

[Moor56] E. F. Moore, “Gedanken-Experiments on Sequential Machines”, Automata Studies, Princeton University Press, Princeton, N. J., 1956.

[More90] L. J. Morell, “A Theory of Fault-Based Testing”, IEEE Trans., SE-16, No.8, 1990.

[NaSa93] K. Naik and B. Sarikaya, "Test Case Verification by Model Checking", Formal Methods in Systems Design, 2, 1993, pp. 277-321.

[PBD93] A. Petrenko, G. v. Bochmann, and R. Dssouli, "Conformance Relations and Test Derivation", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 157-178.

[Petr91] A. Petrenko, "Checking Experiments with Protocol Machines", IFIP Transactions, Protocol Test Systems, IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp. 83-94.

[PeYe92] A. Petrenko and N. Yevtushenko, "Test Suite Generation for a FSM with a Given Type of Implementation Errors", IFIP Transactions, Protocol Specification, Testing, and Verification, XII (the Proceedings of IFIP TC6 12th International Symposium on Protocol Specification, Testing, and Verification, 1992), Ed. by R.J. Linn. Jr. and M.U. Uyar, 1992, North-Holland, pp. 229-243.

[PYB94] A. Petrenko, N. Yevtushenko, and G. v. Bochmann, "Experiments on Nondeterministic Systems for the Reduction Relation", Universite de Montreal, DIRO, Department Publication #932, 1994, p.23.

[PYD94] A. Petrenko, N. Yevtushenko, and R. Dssouli, "Testing Strategies for Communicating FSMs", IWPTS'94.

[PYL93] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das, "Nondeterministic State Machines in Protocol Conformance Testing", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 363-378.

[SaBo84] B. Sarikaya, G. v. Bochmann, "Synchronization and Specification Issues in Protocol Testing", IEEE Trans., vol. COM-32, No.4, 1984.

[SaSp90] M. Sahinoglu and H. Spafford, “Sequential Statistical Procedures for Approving Test Sets Using Mutation-Based Software Testing”, SERC-TR-79-P, Software Engineering Research Center, Purdue University, September, 1990.

[Sier93] I. Sierocki, "An Algebraic Transformation of the Minimum Automaton Identification Problem", EUROCAST'93, LNCS, No.763, pp. 220-230.

[SiLe89] D. P. Sidhu and T. K. Leung, “Formal Methods for Protocol Testing: A Detailed Study”, IEEE Trans. SE-15, No.4, April 1989, pp. 413-425.

[TrBa73] B. A. Trakhtenbrod and Ya. M. Barzdin, Finite Automata. Behavior and Synthesis,

(translated from Russian), North-Holland Publ. Comp., 1973, 321p.

[TuRo92] C. D. Turner and D. J. Robson, "The Testing of Object-Oriented Programs", Technical Report TR-13/92, University of Durham, 1992.

[TyBa75] T. Tylaska and J. D. Bargainer, "An Improved Bound for Checking Experiments that Use Simple Input-Output and Characterizing Sequences", IEEE Trans., Vol. C-24, No.6, 1975, pp. 670-673.

[Vasi73] M. P. Vasilevski, "Failure Diagnosis of Automata", Cybernetics, Plenum Publishing Corporation, New York, No.4, 1973, pp. 653-665.

[VCI89] S. T. Vuong, W. W. L. Chan, and M. R. Ito, "The UIOv-method for Protocol Test Sequence Generation", Proceedings of IFIP TC6 Second International Workshop on Protocol Test Systems, 1989, Ed. by J. de Meer, L. Machert and W. Effelsberg, North-Holland, pp. 161-175.

[VuKo90] S. T. Vuong and K. C. Ko, “A Novel Approach to Protocol Test Sequence Generation”, Proc. of GlobalCOM'90, 1990, pp. 1880-1884.

[WaLi93] C.-J. Wang and M. T. Liu, "Generating Test Cases for EFSM with Given Fault Models", INFOCOM'93, pp. 774-781.

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

[YePe89] N. Yevtushenko and A. Petrenko, "Fault-Detection Capability of Multiple Experiments", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.23, No.3, 1989, pp. 7-11.

[YePe90] N. Yevtushenko and A. Petrenko, "Method of Constructing a Test Experiment for an Arbitrary Deterministic Automaton", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.24, No.5, 1990, pp. 65-68.

[YLP91] N. Yevtushenko, A. Lebedev, and A. Petrenko, "On the Checking Experiments with Nondeterministic Automata", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.25, No.6, 1991, pp. 81-85.

[YPB93] M. Yao, A. Petrenko and G. v. Bochmann, "Conformance Testing of Protocol Machines without Reset", Proceedings of the IFIP 13th Symposium on Protocol Specification, Testing and Verification, Belgium, 1993, pp. 241 - 253.

[YPB94a] M. Yao, A. Petrenko and G. v. Bochmann, “Fault Coverage Analysis in Respect to an FSM Specification”, IEEE INFOCOM’94, Toronto, Canada, 1994.

[YPB94b] M. Yao, A. Petrenko and G. v. Bochmann, “A Structural Analysis Approach to the Evaluation of Fault Coverage for Protocol Conformance Testing”, FORTE'94.

[ZhCh94] J. Zhu and S. T. Chanson, “Fault Coverage Evaluation of Protocol Test Sequences”, Proc. of the 14th IFIP Symposium on Protocol Specification, Testing and Verification, Canada, 1994.