On Fault Coverage of Tests for Finite State Specifications

A. Petrenko and G. v. Bochmann

Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CP. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7, Canada

Abstract

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 coverage, or adequacy of the test suite, becomes a very important issue. In this paper, we analyze basic ideas underlying the techniques for fault coverage analysis and assurance mainly developed in the context of protocol conformance testing based on finite state models. Special attention is paid to parameters which determine the testability of a given specification and influence the length of a test suite which guarantees complete fault coverage. We also point out certain issues which need further study.

1. INTRODUCTION

Testing is a critical phase in the development life cycle for any hardware and software system, and in particular for communication systems for which it ensures the conformance to the relevant standards and compatibility or interoperability with other systems. 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 external behavior (or output) from what is expected.

Testing is used to ascertain the correctness of a system implementation with respect to its requirement specification, given as a relevant normative document. The essential idea of this correctness-proving viewpoint is that an execution scenario in which no error is detected should ensure that the implementation is free 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. Unfortunately, exhaustive testing is impractical since it may involve a huge or infinite number of execution scenarios to be performed. Testing is thus 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 desirable to have a measure of the effectiveness of testing by assessing the test suite executed.

Quality or adequacy of tests for a given specification is commonly evaluated by checking the extent to which tests satisfy a particular test coverage criterion. A coverage criterion guides a testing strategy and sets requirements for a test suite based upon coverage of certain characteristics associated with the given specification. The list of characteristics considered for defining the test coverage criteria includes the following:
- conformance requirements;
- specification structure;
- input domains.

We note that there are no clear demarcation lines between these different notions of coverage, which we will try to briefly describe below. First, however, let us mention the notion of a test purpose which is is widely used by human experts for designing protocol tests and eventually in documents standardizing tests for conformance testing [Rayn87]. A test purpose is usually an informal statement which denotes an "important" part of a specification that should be tested [GHN93]. Simple test purposes may refer to single transitions defined in state tables found in annexes to normative documents [PhGr91]. A more complex test purpose, such as the capability of an implementation under test (IUT) to support several connections, involves the complete functionality. Any attempt to quantitatively access the coverage of test purposes in a formal way requires that first all test purposes are formalized and then the notion of "coverage" is assigned a formal meaning. Even in the case where test purposes are based on a single transition, the question of whether or not the tail state of the transition under test has to be verified should be answered. The question also remains of how many test purposes should be combined within a test suite in order to obtain sufficient coverage.

The system requirements are often stated in the form of a list of properties that should be satisfied by the implementation. This list may include examples of dynamic behavior, sometimes called use cases, static properties, such as number of users supported, or invariant properties that should hold throughout the operations of the system. A testing strategy may aim at covering all properties specified in the list of requirements.

A structural coverage of a specification is usually defined in a way similar to that of white-box software testing, where the control and data flow structure of the program is the reference for test coverage. In the case of black-box testing, however, the specification is the reference, and the coverage criteria relate to the structure of the specification. A test exercises, or covers, structural components of the specification such as statements, branches, arbitrary control paths, or data flow relations, such as *def-use* pairs or I/O-associations [UrWi93]. Tests covering only branches or statements of a protocol specification are normally considered insufficient for conformance testing, though a coverage in the 70% range is considered excellent in software testing. In the case of an FSM specification, a branch cover constitutes a transition tour which leaves many control errors untested. At the same time, all paths cannot be covered since protocols exhibit a cycling behavior. Data-flow test criteria may be used as an intermediate measure between simple branch coverage and full path coverage. These coverage criteria become more relevant when protocol specifications in the formal description techniques (FDT) such as SDL, LOTOS and ESTELLE are directly treated for testing.

The idea of domain coverage also stems from general software testing where functional test methods attempt to cover the domain of each input parameter by executing tests with different input values covering typical and extremal values of the allowed input domain. The essential difference for protocol testing is the fact that protocols are reactive systems; instead of having a single set of input parameters, the input to a reactive system is an infinite sequence of input events, each including a certain number of input parameters. A recent research undertaken in [VuCu91], [AlVu93] attempts to assign a formal notion to the coverage of an infinite input space.

The coverage criteria described above may be used with two different testing paradigms: (1) exhibiting correct behavior concerning the criteria in question, or (2) discovering any implementation faults in relation to the criteria in question. Most structural software testing methods take the paradigm of exhibiting correct behavior, covering certain branches, paths, data flow relations or other structural aspects of the specification or implementation. The transition tour testing method for FSM specifications also belongs to this category.

In the case of fault based testing , the purpose of testing is to detect any fault, with respect to the specification, which the implementation may contain [More90]. One usually considers a certain fault model which defines all the faults that are to be found. Each fault corresponds to a mutant of the specification, and coverage is usually measured in terms of number of faults that could be detected by the test suite, compared to the total number of faults included in the fault model; this measure is called fault coverage. Then a fault coverage of a given test can be measured as the percentage of the specified faults that can be detected. The fault-model based approaches can be considered the most practical as they directly capture the intention of the test engineer to detect and locate any implementation error. Fault coverage is often defined based on some knowledge of the implementation process. For example, in the area of hardware testing, there exist widely accepted types of faults, such as stack-at faults. However, in protocol conformance testing, there does not yet exist any such "standardized" fault model. Much work has been done on fault-based testing for finite state models, see for example, [SiLe89], [BDD91], [PeYe92], [Ural92], [PBD93] and extended finite state machines (EFSMs) [GrPe90], [FaPe90], [PrGu91], [MiPa92], [ChZh93], [WaLi93]. In software testing, the relation between input domains and execution paths has been extensively studied and fault models for the domain boundaries have been used for test suite development [JeWe94].

The finite state models have been extensively used in conformance testing of communication protocols [PBD93], [BoPe94] as well as in hardware and software testing. 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 focus our attention mainly on fault coverage analysis problems related to the finite state models of protocols. Other notions of test coverage deserve a separate study. Our discussion will be primarily based on the finite state machine (FSM) model; however, we will also address relevant issues for labeled transition systems (LTS).

The rest of the paper is organized as follows. In Section 2, a framework for 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 problems in the cases of nondeterministic specifications, FSMs and LTSs. We demonstrate that an LTS specification in a given semantics can be translated into an FSM specification, allowing the FSM-based techniques to be applied to LTSs as well. Fault coverage problems in the case of nontrivial test architectures are addressed in Section 6, where we also consider the model of communicating FSMs, which more adequately represent testing through the environment, and show how an embedded component can be tested. We conclude in Section 7 by discussing possible applications of techniques for fault coverage analysis.

2. BASIC FRAMEWORK FOR FAULT COVERAGE ANALYSIS

2.1. General notions, definitions, and assumptions

We start with a fault coverage framework based on the traditional model of deterministic FSMs.

Let $M_S = <S, X, Y, s_1, \delta, \lambda, D_S>$ be a deterministic FSM, 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; $\delta: D_S \rightarrow S$ is the transfer function; $\lambda: D_S \rightarrow Y$ is the output function; D_S is a specification domain of M_S, i.e. a subset of $S \times X$. If $D_S = S \times X$ then M_S is completely specified or *complete*, otherwise it is *partial*.

FSM-based testing is usually formalized as the problem of testing an FSM implementation: given an FSM representation (specification) of a system 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 (later in this paper, we also consider other classes of machines which do not satisfy these assumptions), and initially or strongly connected.

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. This situation is called the local single-layer test method in the context of OSI conformance testing

[Rayn87]. 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 specified only partially.

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 a reliable reset facility ensuring that each test sequence is applied to the same initial state. 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 FSMs which are only initially connected but not strongly connected.

2.2. Conformance relations

As mentioned above, the conformance relation determines whether the behavior of a given implementation conforms to the corresponding specification. In the context of finite state machines with input and output interactions, the behavior of an implementation is usually characterized by the set of possible execution sequences of input and output interactions, also called traces. In the simplest situation, the conformance relation is the equivalence relation between FSMs, that is, an IUT conforms to a specification M_S if the IUT produces the same outputs as the machine M_S for all possible sequences of applied inputs. It is important to note that conformance does not mean equality. For any FSM specification M_S, there exist many different equivalent implementation FSMs, which have usually more states than M_S if M_S is minimal.

While the FSM equivalence is an appealing conformance relation for FSM testing, it is not satisfactory in several situations. In fact, a large number of different conformance relations have been proposed, which are useful in various situations. In particular, in the case that the specification is given in the form of a partially defined FSM, quasi-equivalence is often used as conformance relation. According to this relation, an implementation M_I conforms to a specification M_S if for each input sequence for which M_S is defined, M_I produces the same output sequence as defined by M_S; for all other input sequences any output may be produced (see Section 3.2). In the context of nondeterministic specifications, allowing for several different output sequences for each input sequence applied, a relation called "reduction of nondeterminism" is often considered, which allows that an implementation realizes only a certain subset of the specified output sequences. In particular, a deterministic implementation producing only one of the specified output sequences would satisfy this conformance relation (see Section 5).

In the context of nondeterministic specifications, the semantics solely based on the possible traces of input/output interactions is not powerful enough to describe the blocking behavior of a system. Therefore, the concept of refusals has been introduced in the framework of specifications given in the form of labeled transition systems, and various conformance relations have been proposed to take into account the blocking behavior defined by a specification. Some of these issues are further discussed in Section 5.

For the discussion of test coverage, it is important to note that the adopted conformance relation determines which of the possible implementations conform to the specification, and which are to be considered as faulty. In the following subsection, we discuss the concept of a fault model which serves to enumerate all possible implementations (at least those that are of interest), independently of whether they are faulty or not.

2.3. Fault models

The formidable obstacle in constructing or analyzing a test suite is that it must verify whether a given conformance relation defined on an *infinite* set of input 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. Such a model can be formulated in terms of elements of the FSM model, since the correct interface assumption implies that an implementation is an FSM. Once the set of possible implementations, denoted henceforth as $\boldsymbol{Impl}(M_S)$, is limited, we can verify with the help of suitable tests whether a machine from this set conforms 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, and 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 $\boldsymbol{Impl}(M_S)$, 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 $\boldsymbol{Impl}(m, M_S)$ of all complete FSMs which have the same input alphabet X as M_S and up to m states, where $m \geq n$ and n is the number of states of M_S. $M_S \in \boldsymbol{Impl}(m, M_S)$. 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 $\boldsymbol{Impl}(m, M_S)$ 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:	alter the tail state of a transition (a *transfer* fault);
Type 2:	alter the output of a transition (an *output* fault);
Type 3:	add a transition; and
Type 4:	add an extra state.

These operations model possible alterations of the specification machine made during the implementation process. The third type is used in the context of partial and nondeterministic machines. The number of mutants in the set $\boldsymbol{Impl}(m, M_S)$ is $(m|Y|)^{m|X|}$, where Y is the output alphabet of the implementation machines.

Such a mutation technique works well in the context of minimal deterministic specifications. It is clear that a mutant of a specification machine M_S with equivalent states might be equivalent to M_S. Moreover, in the case of nondeterministic systems and the conformance relation based on the reduction relation [PBD93], the correspondence between mutations and erroneous behavior is not so straightforward either.

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, 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 [PeYe92]. As an example, consider the FSM M_S with two states {1,2}, two inputs {a,b} and two outputs {y,z} shown in Figure 1. Assume that the transition 1-a/y->2 is correctly implemented, and the remaining are suspicious transitions; in particular, transition 1-b/z->1 can have only output faults, 2-b/y->1 can have only transfer faults, and transition 2-a/y->1 can have any fault. The corresponding fault function is represented by Table 1. Here, the entries shown in bold represent the state table of the specification FSM in Figure 1.

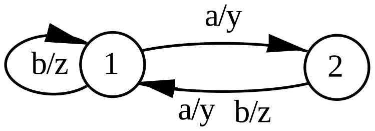


Figure 1: An FSM

Table 1: A fault function

	a	*b*
1	**2/*y***	**1/*z*,** 1/*y*
2	**1/*y*,** 1/*z*, 2/*y*, 2/*z*	**1/*y***, 2/*y*

Table 1 can be interpreted as a state table of a nondeterministic FSM. It represents $1 \times 2 \times 2 \times 4 = 16$ mutants that are deterministic submachines of this nondeterministic machine, in other words, $|\boldsymbol{Impl}(M_S)| = 16$. Note that there exist 128 different FSM with two states, two inputs and two outputs, i.e. $|\boldsymbol{Impl}(2, M_S)| = 128$. Fault functions are a flexible tool to express various fault assumptions, for example, a suspicious transition may be viewed as a particular test purpose to be "covered" by a test suite. Moreover, the first technique based on the universal set $\boldsymbol{Impl}(m, M_S)$ turns to be a special case of the technique based on fault functions. However, both techniques rely on the given bound on the number of states in implementation machines. As shown in [PYD94], the fault-function based technique for specifying restricted faults lends itself well to component testing and grey-box testing (see Section 6).

2.3. Fault coverage

Given a finite class of implementations $\boldsymbol{Impl}(M_S)$ of the specification machine M_S, we can define a so-called complete test suite for M_S or a test suite with guaranteed fault coverage. A *TS* is said to be *complete for M_S in the class* $\boldsymbol{Impl}(M_S)$ if for each machine *M* 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 distinguishes *M* from M_S by their output reactions. A *TS* which is complete in $\boldsymbol{Impl}(m, M_S)$ is simply called *m-complete*. Complete test suites were first introduced as checking experiments for completely specified FSMs [Moor56], [Henn64] and the equivalence relation between FSMs as a conformance relation.

Since completeness of a test suite is not easy to achieve, 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 $\boldsymbol{Impl}(m, M_S)$ be the class of implementations. To define the fault coverage, we need to use the following notations:

$\boldsymbol{N_t}(m, M_S)$ - the total number of machines in $\boldsymbol{Impl}(m, M_S)$;

$\boldsymbol{N_c}(m, M_S)$ - the number of machines in $\boldsymbol{Impl}(m, M_S)$ which conform to M_S;

$\boldsymbol{N_p}(m, M_S, TS)$ - the number of machines in $\boldsymbol{Impl}(m, M_S)$ which can pass the given test suite *TS*.

It is clear that,

$\boldsymbol{N_t}(m, M_S)$ - $\boldsymbol{N_c}(m, M_S)$ is the number of machines in $\boldsymbol{Impl}(m, M_S)$ which do not conform to M_S;

$\boldsymbol{N_t}(m, M_S)$ - $\boldsymbol{N_p}(m, M_S, TS)$ is the number of machines in $\boldsymbol{Impl}(m, M_S)$ 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 $\boldsymbol{FC}(m, M_S, TS)$, is

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

Similarly, the fault coverage can be defined for an arbitrary set $\boldsymbol{Impl}(M_S)$ of implementations representing restricted types of faults. This kind of fault coverage measure has been used in a number of papers, see, for instance, [BDD91]. 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 $\boldsymbol{Impl}(m, M_S)$ are subsets of *Y* - the output alphabet of M_S (under this assumption, $\boldsymbol{N_t}(m, M_S) = (m|Y|)^{m|X|}$, where *X* is the input alphabet of M_S), otherwise the cardinality of $\boldsymbol{Impl}(m, M_S)$ 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 $\boldsymbol{FC}(m, M_S, TS)$ is not equally distributed over [0,1]. A TS consisting of a single test event already has the coverage of more than $(|Y|\text{-}1) / |Y|$. 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 $\boldsymbol{N_t}(m, M_S) >> \boldsymbol{N_c}(m, M_S)$ and $\boldsymbol{N_t}(m, M_S) >> \boldsymbol{N_p}(m, M_S, TS)$ and therefore $\boldsymbol{FC}(m, M_S, TS) \approx 100\%$. Thus calculations with a normal precision might not be sufficient to compare the test suites by their fault coverages.

It is important to keep in mind these drawbacks of the fault coverage formula when applying it in practice [BPY94]. Actually, the so-called "order coverage" [YPB94b] has been proposed to overcome the drawback (3) mentioned above. The "order coverage", denoted as $\boldsymbol{FC_O}(m, M_S, TS)$, can be defined by the following formula:

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

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

Fault coverage estimation remains effective even in the cases where the actual number of states in implementations under test is unknown. By comparing any two test suites by their fault coverages with respect to a chosen bound m we normally conclude that the one with a higher coverage would have a better fault coverage with respect to any other bound $q>m$.

3. CONDITIONS FOR FAULT COVERAGE

There exist several test derivation methods for FSMs which guarantee complete fault coverage. Based on these methods and on the existing bounds of automata theory, 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 a reliable reset feature in the implementations under test. The corresponding conditions for the case without reset are somewhat more complicated [Henn64], [Hsie71], [YPB93]. Other methods for test derivation [SiLe88], [SiLe89], [BoUy91], [Ural92], [MiPa93], [UrZh93], [MCS93] do not guarantee fault coverage in all cases.

3.1. Tests for complete deterministic FSMs

Let $M_S = <S, X, Y, s_1, \delta, \lambda>$ be a complete reduced deterministic FSM (CDFSM). Let TS be a test suite for the 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* $\boldsymbol{AP}(Z)$ of a set Z of sequences is the set which consists of all the prefixes of all the sequences in Z, i.e., $\boldsymbol{AP}(Z) = \{ p \mid p$ is a prefix of some sequence in $Z \}$.

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

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 with $k \leq r$, is said to be the *degree of accessibility* of M_S. Any initially connected FSM has $1 \leq r \leq n-1$. 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, one per state, each of length of at most r. One may say that this parameter characterizes the controllability of the given FSM [PDK93], [BoUy91].

Following [Gill62], we say that two states s and p of M_S are *d-distinguishable* if there exists an input sequence of 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, $1 \leq d \leq n-1$. One may say that this parameter characterizes the observability of the given FSM [PDK93], [BoUy91]. 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. For more discussion on construction of distinguishing sequences the reader is referred to [LeYa94].

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 the parameters r and d. Their least 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 completeness 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, called regular test suites.

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 $\boldsymbol{AP}(TS) \supseteq VX^b$. 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 which has n states, a 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 distinguishing sequence and therefore, it is a common simple (UIO-) sequence at the same time. The degree of distinguishability is equal to one, and any test suite which contains the set VX^2 is n-complete. Another extreme case is where the degree of distinguishability of the given FSM reaches its maximum, i.e. $d=n-1$. A m-canonical test suite VX^b is m-complete for such a machine.

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 [FBK91], it is called an identification set of state s_i). Let I be a set of n state identifiers (one for each state of the specification), and R be a set of input sequences. We denote by $R \otimes I$ the set $\{ \alpha W_i \mid \alpha \in R \ \& \ \delta(s_1,\alpha)=s_i \}$. The test suite TS is called a *partial c-characterization* test suite for the FSM M_S, if $\boldsymbol{AP}(TS) \supseteq VX^{c-1}W \cup VX^c \otimes I$. This expression reflects two phases of testing used by the Wp-method. It is obvious that $VX^cW \supseteq VX^{c-1}W \cup VX^c \otimes I$. Based on the results of [FBK91], 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 $VX^{c-1}W \cup VX^c \otimes I$ 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]. A set H of *harmonized* state identifiers $\{W_1, ..., W_n\}$ is defined as follows: $\forall s_i, s_j \in S, \ \exists \alpha \in \boldsymbol{AP}(W_i) \cap \boldsymbol{AP}(W_j) \ \ (\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha))$. The approach based on harmonized state identifiers [Petr91], [PeYe92], [YePe90], [LPB94a] 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 distinguishing sequence DS, all the methods, W-, Wp-, UIOv-, HSI-, DS-methods (with reset) come to the same, i.e. they may produce the same test suite.

The test suite TS is called a *harmonized c-characterization* test suite for the FSM M_S, if $\boldsymbol{AP}(TS) \supseteq VX^c \otimes H$. 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.

The 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) $\boldsymbol{AP}(TS)$ contains a state cover V and a transition cover TC; and

(2) for each pair of sequences α, β [V such that $\delta(s_1, \alpha) \neq \delta(s_1, \beta)$, there should be two sequences $\alpha\gamma, \beta\gamma$ [$\boldsymbol{AP}(TS)$ such that $\lambda(\delta(s_1, \alpha), \gamma) \neq \lambda(\delta(s_1, \beta), \gamma)$; and

(3) for each α [$TC \backslash V$ and each β [V such that $\delta(s_1, \alpha) \neq \delta(s_1, \beta)$, there should be two sequences $\alpha\gamma, \beta\gamma$ [$\boldsymbol{AP}(TS)$ 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 [VCI89], the W-method [Chow78], [Vasi73], the Wp-method [FBK91] or the methods based on harmonized state identifiers [Petr91], [YePe90], [LPB94a] 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 complete if it contains a subset of test sequences 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. Condition 3.1.7 is the weakest known 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 reduced.

Necessary Condition 3.1.8. If, for some m, $m \geq n$, TS is m-complete for M_S which has 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 $V \subseteq \boldsymbol{AP}(TS)$.

It is implied by the results of [Vasi73] and noted in [YaLe91] 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 which has 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, $m \geq n$, TS is m-complete for M_S, then there should exist a transition cover TC such that $TC \subseteq \boldsymbol{AP}(TS)$.

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.

3.2. Tests for partial deterministic FSMs

A specific feature of partial machines is that they have "undefined" transitions. Let $M_S = \langle S, X, Y, s_1, \delta, \lambda, D_S \rangle$ be a partial deterministic FSM. The set $(S \times X) \backslash D_S$ represents all undefined transitions. A complete machine accepts every possible input sequence, whereas a partial machine does not. An input sequence $x_1 \ldots x_k$ is called an *acceptable* sequence for state s if a sequence of transitions $s\text{-}x_1\text{->}s_1\text{-}x_2\text{->}s_2 \ldots s_{k-1}\text{-}x_k\text{->}s_k$ is defined in M_S. Similarly to the case of complete machines, states s and s' of a partial machine are *distinguishable* if there exists an input sequence acceptable for both states that causes different output sequences from these states; otherwise they are *compatible* states [Gill62]. Given an implementation machine M_I which is a complete FSM with the initial state i_1, M_I is said to be *quasi-equivalent* to M_S if i_1 and s_1 produce the same output sequence when any input sequence acceptable for s_1 is applied to both states. The quasi-equivalence relation was called containment in [DaSa88]. If the two machines are complete then the quasi-equivalence and equivalence relations coincide.

For partial machines, there are several different conventions used for "undefined" transitions: "implicitly defined", "undefined by default", and "forbidden" transitions [PBD93].

In the first case, one makes a *completeness* assumption; the given partial FSM is substituted by a quasi-equivalent complete FSM. A particular form of the completeness assumption is to require that the machine should remain in the present state without producing any output (the *null* output) for any unspecified input (as adopted for the SDL language [BHS91]). Another form of completeness can be obtained by introducing an "error" state. Based on such completeness assumptions the problems of test derivation and fault coverage analysis are reduced to those associated with complete machines, because the equivalence with the completed specification is taken as conformance relation. As an example, the partially specified specification machine given in Figure 2(a) will have to be converted into the complete machine shown in Figure 2(b). Any other machine such as the one shown in Figure 2(c) with a different behavior would be treated as non-conforming (non equivalent) to the completed machine, even if it is quasi-equivalent to the original machine.

It was often argued (see, for example, [DaSa88], [SiLe89], [Kars94]) that even when the completeness assumption is used, it is still possible to check only the quasi-equivalence, called also the *weak* conformance or containment, instead of the *strong* conformance, i.e. the equivalence. However, such a possibility heavily depends on particular distinguishing sequences included in a test suite. Once input *a* of the FSM (b) is chosen for distinguishing states, it is no longer possible to solely test the weak conformance. Consider a fragment of a test sequence *baaba* which attempts to execute the three specified transitions and to verify the tail state of every transition by applying input *a*. It distinguishes the machines (b) and (c), though (c) is weak conformant (quasi-equivalent) to (a). Under the completeness assumption, such distinguishing sequences would always be chosen for quasi-equivalent states making it *impossible* to separate weak and strong conformance testing. An approach has been developed in [YePe90], [Petr91], [LBP94] to derive test suites exclusively from acceptable sequences which verify the quasi-equivalence relation without the completeness assumption.

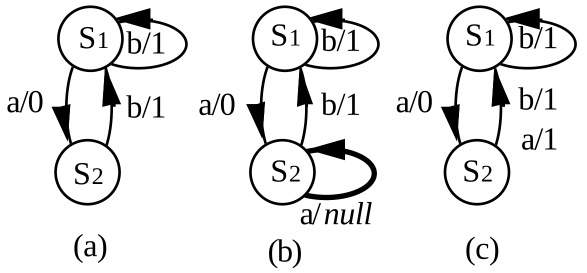


Figure 2: Difference between the equivalence and quasi-equivalence

The other conventions for undefined transitions, namely, "undefined by default" (also called “don’t care”) and "forbidden", require a test suite constructed from acceptable input sequences. If some test sequence covers an undefined transition, then, under the "undefined by default" convention, its verdict must be inconclusive. A partial machine with *q* "don't care" transitions can be interpreted as a set of $(n|Y|)^q$ complete machines. A test sequence covering a "forbidden" transition is not executable and should be excluded from the test suite. Several testing situations requiring the model of partial FSMs with these conventions have already been identified, in particular, if a testing architecture prevents a tester from submitting an input in a certain situation then the "forbidden" convention should be applied (see Section 5 and [PYL93], [PYD94]). This demonstrates that the completeness assumption is not as universal as it was 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 compatible and thus 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 [YePe89], 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], [LPB94a]. If M_S happens to be complete and minimal or partial and reduced, then f=1. The larger the number of pairs of compatible states, the higher the fuzziness degree usually is. For a "fully unreduced" machine such as the one in Figure 2, $f = n$.

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 $\boldsymbol{AP}(TS) \supseteq X^a \cap X_S$, where X_S is the set of acceptable input sequences of M_S. Based on the results of [YePe89], [PYD93], we have the following proposition.

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 which has n states, a state cover V, the degrees of distinguishability d and fuzziness f. If $b \geq fm\text{-}n+d+1$, then TS is m-complete for M_S.

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

It was shown in [Petr91], [YePe90] 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. The reason is that the degree of distinguishability has a tight bound of $n(n\text{-}1)/2$, instead of n-1 for complete machines. The total length of sequences of a characterization set W has the least upper bound of $[(n^2\text{-}n+1)^2\text{-}1]/8$ for reduced partial FSMs. Moreover, certain states might not be distinguishable at all.

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.

4. FAULT COVERAGE ANALYSIS

The purpose of fault coverage analysis is to calculate the fault coverage for a given test suite TS, a specification M_S and a fault model. In the following, we limit ourselves to the fault model $\boldsymbol{Impl}(m, M_S)$ of all complete FSM specifications with a number of states less or equal to m. As is clear from its definition, in order to calculate the fault coverage, we need to find the number of machines in $\boldsymbol{Impl}(m, M_S)$ which cannot pass the given test suite TS, or we need to find the number of machines in $\boldsymbol{Impl}(m, M_S)$ which can pass the given test suite TS. However, their exact values are in most cases 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 $\boldsymbol{N_p}(m, M_S, TS)$. The idea is to enumerate all the implementation machines in $\boldsymbol{Impl}(m, M_S)$ 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 $\boldsymbol{N_p}(m, M_S, TS)$. In this way, the precise value of fault coverage $\boldsymbol{FC}(m, M_S, TS)$ can be calculated. The complexity of this approach is high since the total number of machines $\boldsymbol{N_t}(m, M_S)$ 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 [SiLe89], [SaSp90], [DDB91], and [MCS93]. The idea is to first randomly generate k implementation machines in $\boldsymbol{Impl}(m, M_S)$, 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 $\mathrm{M_S}$. Then $k'/(k - k'')$ gives an approximate value of the fault coverage $\boldsymbol{FC}(m, M_S, TS)$.

The set of implementation machines $\boldsymbol{Impl}(m, M_S)$ 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 $\boldsymbol{Impl}(m, M_S)$ 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 $\boldsymbol{N_p}(n, M_S, TS)$ by directly analyzing the structure of the specification machine against the test suite. Let $\boldsymbol{N_p}(n, M_S, TS)$ denote the estimated value of $\boldsymbol{N_p}(n, M_S, TS)$. Then

$$FC_e(n, M_S, TS) = \frac{N_t(n, M_S) - \mathsf{N}_p(n, M_S, TS)}{N_t(n, M_S) - N_c(n, M_S)}$$

gives us the estimated fault coverage. The value of $\boldsymbol{N_t}(n, M_S)$ is given by $(n|Y|)^{n|X|}$ (see Section 2), which corresponds to the fact that for each transition with given state and input, there are $n|Y|$ pairs of possible next state and output values. The value of $\boldsymbol{N_c}(n, M_S)$ is given by $\boldsymbol{N_c}(n, M_S) = (n\text{-}1)!$. The formula $(n\text{-}1)!$ 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 $(n\text{-}1)!$ such permutations.

The technique used to derive the estimated value $\boldsymbol{N_p}(n, M_S, TS)$ 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 $\alpha, \alpha x, \alpha x\gamma, \beta, \beta\gamma$ [***AP***(*TS*) such that

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

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) [$(S \times X) \setminus D_S$. 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 $(n\cdot|Y|)^{n|X| - |D_S|}$ possible ways to generate an implementation machine by adding $n|X| - |D_S|$ extra transitions.

Following from the above discussions, we have

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

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 *t* is a factor which represents the estimated number of permutations that are possible among those states that are identified by the test suite.

An estimation for *t* 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

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

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 t 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 $\boldsymbol{N_p}(n, M_S, TS)$, we find t = 1; this corresponds to the fact that only the initial state is identified and no other permutations exist. This confirms that $1 \leq t \leq (n-1)!$.

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 t 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 t:

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

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].

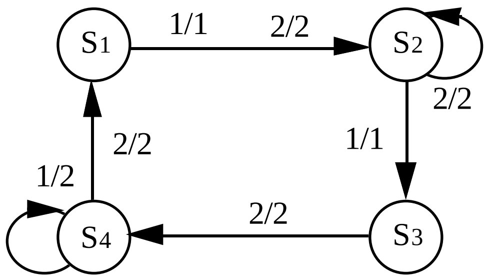


Figure 3: An example FSM

Here we present the results for the following eleven test suites for the FSM shown in Figure 3.

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

Applying our structural analysis approach to these test suites yields the estimated fault coverage $\boldsymbol{FC_e}$ listed in the third column of Table 2. 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 3, we have been able to make an exhaustive mutation analysis. A program was written 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 ***FC*$_p$** for these test suites and the results are listed in the forth column of Table 2. The differences between the estimated and precise fault coverage are listed in the fifth column of Table 2. The last column gives the relative error for each test suite, i.e. the absolute deviation value divided by 1 - ***FC*$_p$**(n, M_S, TS).

Table 2: Fault coverage for the FSM in Figure 3

TSi	*d*	$\boldsymbol{FC_e}(n,MS,TS)$	$\boldsymbol{FC_p}(n,MS,TS)$	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%

The above considerations apply also to 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:

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

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 order fault coverage as a percentage of transitions covered by the test suite.

4.4. Deciding the *m*-completeness of a test suite

Another 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 the 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 the *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, a more general state minimization procedure was proposed [YPB94a] 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 this type of analysis 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 the *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].

With the 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 the minimal (or reduced) form of the tree machine found in Step 2 does not conform to M_S and has no more than *m* states, then the test suite is not *m*-complete; otherwise, it is *m*-complete.

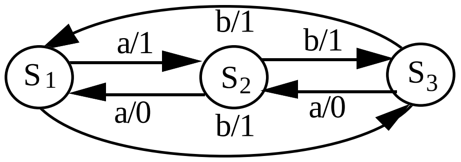
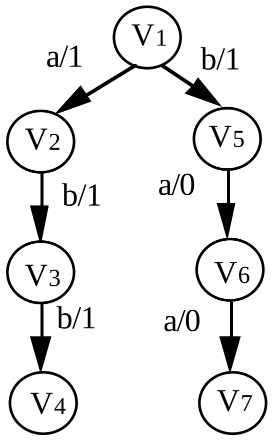


Figure 4: An example specification FSM



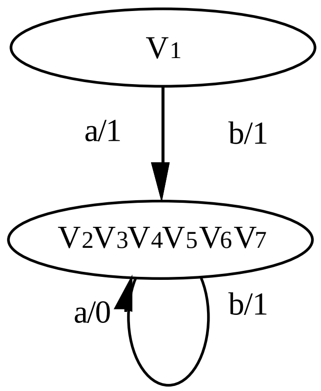


Figure 5: The tree machine **Figure 6: The minimal form found**

A reduced form of the tree machine still conforms to the given test suite, however, it accepts more input sequences than the tree machine for which the conformance to the specification machine may not hold.

As a concrete example, let us consider the test suite *TS* = {*a.b.b, b.a.a*} derived from the specification machine shown in Figure 4. We want to check if this test suite is 3-complete. The tree machine *M* representing this test suite is shown in Figure 5. By using the state minimization procedure, a minimal form of the tree machine can be found which has two states as shown in Figure 6, and does not conform to the specification machine of Figure 4. The two states of the minimal form are actually obtained by merging the states of the tree machine in Figure 5. 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 nonreduced. The corresponding tool provides three options when performing fault coverage analysis:
(1) searching for a mutant which does not conform to the specification machine *MS*;
(2) searching for a mutant which has the smallest number of states and does not conform to the specification machine *MS*; and
(3) searching for all the mutants which do not conform to the specification machine *MS*.

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].

A crucial difference between these two approaches can be observed in the case of partial specification machines. The approach of [ZhCh94] is based on the equivalence relation (strong conformance) and therefore can only be applied to completely specified machines. On the other hand, the state minimization based approach [YPB94a] uses the quasi-equivalence relation (called weak conformance) and accordingly can be directly applied to partially specified machines.

5. FAULT COVERAGE FOR NONDETERMINISTIC FSMs AND LTSs

By their nature, nondeterministic systems are more difficult to analyze than 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 (NFSM) or LTS specifications have been reported so far. However, several methods have already been developed for deriving tests from these models with guaranteed fault coverage, though this is a relatively new research subject. Nondeterminism causes several problems such as the following:

- a test sequence can cause various observations in a nondeterministic implementation, therefore 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.

First, we consider nondeterministic FSMs. A *nondeterministic* FSM M_S is a 6-tuple (S, X, Y, h, s_1, D_S), where S is a set of states with s_1 as the initial state; X - a finite set of input symbols; Y - a finite set of output symbols; D_S - a specification domain which is a subset of $S \times X$; h - a behavior function $h: D_S \rightarrow P(S \times Y)$, and $P(S \times Y)$ is the powerset of $S \times Y$ [PBD93]. An NFSM can be partial or complete; in the case of partial machines, the notion of acceptable input sequences is also used. This is a rather general model of finite state machines; all types of machines considered in this paper are special cases of the above model. Note that unlike the models of accepting automata and LTSs, it does not include spontaneous transitions. Then to implicitly model situations involving spontaneous transitions, for example timeouts [PBD93], the null event should be additionally introduced in the alphabets since according to the semantics of the input/output behavior, all transitions are labeled by a pair of input and output.

The equivalence and quasi-equivalence relations for NFSMs are directly extended from those for deterministic machines [Star72], [PBD93], [LBP94]. Given a specification NFSM M_S and complete implementation NFSM M_I, the machine M_I is said to be a *reduction* of M_S if output sequences of the initial state of M_I are included in those of the initial state of M_S for any input sequence acceptable for the initial state of M_S [PBD93]. The equivalence, quasi-equivalence, and reduction relations between NFSMs can be used as the conformance relations between implementations and their specifications in test derivation based on this model.

Test derivation from an NFSM is facilitated when the machine is given in its observable form. An NFSM is said to be *observable* if, starting from any given state, the machine can reach only one state in response to any input/output sequence accepted for the given state. It is clear that all deterministic machines are observable; however, an observable machine can still be nondeterministic. The equivalent transformation of nonobservable machines into observable forms is possible [Star72], [LBP94].

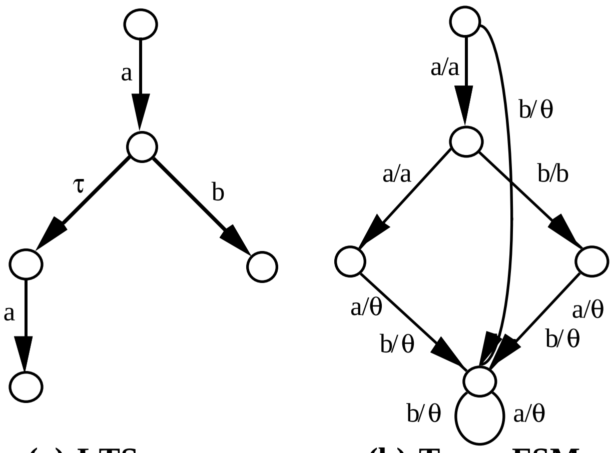
Several heuristic approaches to test derivation from NFSMs can already be found in the literature; however, they are of little help in tackling the problems related to fault coverage analysis for nondeterministic machines. There are also some methods which guarantee completeness of test suites w.r.t. the various relations [YLP91], [PYL93], [LBP94], [LPB94a], [PYB94]. For example, in [LPB94a], it is proven that a harmonized *c*-characterization test suite is complete for an arbitrary observable (possibly nonreduced, 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. This can also reduce the number of test runs needed to observe all possible output sequences. 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, the choice of the appropriate conformance relation for test derivation and analysis is limited by a small number of alternatives, namely trace equivalence, quasi-equivalence and reduction, as explained above. Usually, the choice between these relations can be made in an obvious way. If, however, the specification is an LTS, the choice of a conformance relation becomes less evident. This is important since test derivation and test analysis rely on the notion of a faulty implementation defined by the chosen conformance relation; it is also crucial in analysing the fault coverage of a test suite. This relation might be either the equivalence or a preorder in a particular semantics [Glab90]. In the context of LTSs, the trace semantics is usually considered too coarse since nondeterministic aspects of the system behavior cannot be explicitely modelled. The failure semantics [Hoar85] can be used to describe the nondeterministic possibilities for blocking, characterized by a set of refusal, where each refusal consists of a finite trace and a refusal set which indicates a set of actions for which the implementation may exhibit blocking behavior after the execution of the given trace.

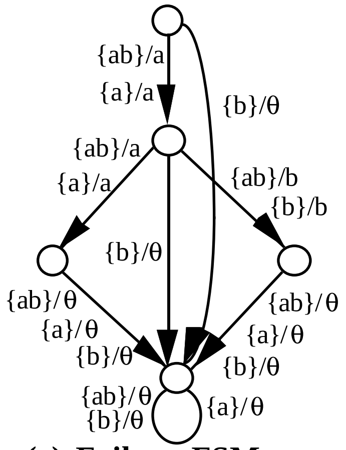
Recent results in test derivation for LTSs demonstrate a strong tendency toward reusing the ideas developed in the context of FSM-based testing for LTS-based testing. In particular, several papers extend the state identification approach to specifications given in form of an LTS [FuBo91], [CKM92], [Arkk93]. It is also suggested [PBD93] that tests for a given LTS could be obtained from tests directly generated by the existing FSM-based methods applied to a proper FSM constructed from the LTS in the chosen semantics. We briefly explain the main idea of translating an LTS into an FSM for various semantics.

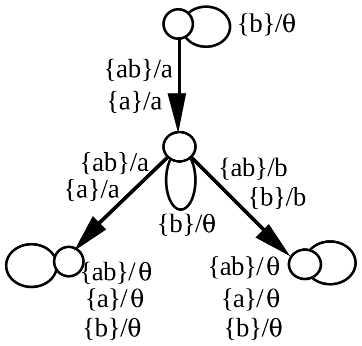
An LTS implementation under test is viewed as a black box which interacts with its environment through rendezvous interactions. During each interaction, the LTS may choose one of the actions offered by the environment or, if none of these offers corresponds to a transition of the LTS, exhibits a blocking behavior. In the case of blocking, the possibility of further evolution is determined by the particular semantics. According to the trace and failure semantics, the system must restart after a deadlock. After blocking, a further interaction can only be executed when a semantics such as the failure-trace semantics is considered. The behavior of a given LTS in the semantics mentioned above can be modeled by an FSM, in which for each interaction with the LTS, the set of offered actions is viewed as an input; the action chosen by the implementation is considered the output, and blocking is modelled by a special output, called *null* output θ. The null output of FSMs and deadlock of LTSs are equivalent notions in the following sense. Regardless of the model, the tester performing an experiment on a physical system should measure the time elapsed after it has offered an event. The tester will conclude that nothing happened in the system if no evolution is observed during a time interval which typically exceeds the internal delays of the system.

Subsets of actions serve as input symbols of a corresponding FSM, whereas single actions along with the null output θ become outputs of the FSM. In trace semantics, deadlock properties are not important, therefore, it is sufficient to consider only single actions as inputs of a *trace* FSM. Transitions labeled with the null output are looping transitions in a *failure-trace* FSM; however, in trace and *failure* FSMs they lead to a sink or terminal state which only produces the null output. Given a nondeterministic LTS, a corresponding trace FSM is deterministic, as shown in [TPB95]; however, failure and failure-trace machines are usually not. Figure 7 exemplifies the transformation of an LTS (a) into the trace FSM (b), failure FSM (c) and failure-trace FSM (d). All these machines produce as output all of the traces of the LTS; in addition, the failure and failure-trace FSMs preserve information of the refusal sets; and only the failure-trace machine keeps track of the interleaving of accepted events and refusal sets given by the LTS. For more details the reader is referred to [PBD93] and [TPB95].



(a) LTS **(b) Trace FSM**





(c) Failure FSM **(d) Failure-Trace FSM**

Figure 7: Representing an LTS by the FSMs

It now becomes clear that the FSM-based results on fault coverage analysis are also applicable to the LTS formalism. 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 in this case. However, this is only one possible scenario of future research in fault coverage analysis in terms of the LTS model.

6. TEST ARCHITECTURES AND COMMUNICATING FSMs

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 it is 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 remote test method [Rayn87]. If no control can be exercised over the upper interface of an IUT then the behavior of upper layers should be adequately modeled only by a nondeterministic machine. Then, even if an IUT is perfectly deterministic the joint behavior as observed by a tester through a perfect media is nondeterministic. Test derivation has to be based on a properly computed NFSM. An example of such a situation is given in [LBP94].

Next, consider the distributed test method [Rayn87]. 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, a complete test suite may not be synchronizable. The situation is in some sense similar to the case of testing partially specified FSMs: test sequences should be chosen from the acceptable ones. It is noted [LDB93] that the distributed test architecture weakens fault coverage, since 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 characterizing of such faults still remains open. An external synchronization of testers usually solves the problem [CTU93].

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 [Rayn87]. In this case, the given test suite might be initially composed of events on interfaces which are not directly accessible by tester(s). Then it must first be analyzed if all inputs can actually be excited by, and outputs uniquely delivered to, the tester(s). Even if this is the case, depending on properties of the environment of the IUT, certain faults may be masked and may not be easily activated [PYD94]. Similar observations are reported for LTS specifications [DAS93]. As follows from these results, the problem of fault coverage analysis for embedded testing becomes more cumbersome.

We conclude that a black box representation of the system under test becomes inadequate for non-trivial test architectures, more research is required in this direction. One possible approach is based on the model of communicating FSMs. This model is usually defined as a structured system of FSMs (CFSMs) communicating with each other over FIFO channels; it is also used for protocol validation. In fact, CFSMs is an adequate model of testing an IUT with a particular test method discussed above. In particular, one machine could model the behavior of the lower service provider, others - an IUT, as well as adjacent protocol machines.

The model of CFSMs, in its most general form, includes unbounded channels and is as powerful as Turing machines for which testing an equivalence relation is undecidable. Note that this is also the case for extended FSM models. Therefore, based on such a model, some simplifying assumptions have to be made in order to ensure finite testing with a predictable fault coverage.

The simplest way of treating CFSMs for testing is to assume channels to be bounded or even absent if FSMs communicate synchronously (via rendezvous) [LSK93], [LBP94], [PYD94]. A bounded channel can be represented as an FSM, therefore, it is always possible to compute a composed FSM which represents the joint behavior of CFSMs provided that the given system has no live-lock [LBP94]. Well-designed protocols normally do not exhibit such a non-progressive behavior. Once such an equivalent FSMs is computed, the methods based on a single FSM become applicable. However, this approach suffers from several drawbacks. Firstly, the approach faces the well-known state explosion effect. Secondly, it completely ignores the structure of CFSMs which in many cases is retained in IUTs, as for example in the embedded test architecture. A resulting test suite can easily be regarded redundant, especially if one takes into account that certain component FSMs might be reasonably assumed error-free.

An attempt to avoid a composed FSM computation was made in [LSK93], where authors assumed that global states as well as every communication between machines can be observed. Based on these assumptions, a guided random walk is proposed to cover individual transitions of component FSMs. However, this approach cannot be applied for non-trivial test architectures, because it requires additional points of observation and control within the system under test. Heuristic techniques for deriving tests from CFSMs without any guarantee of fault coverage were also considered in [DKK91], [MiLu91].

Another approach to CFSMs testing or grey-box FSM testing has been initiated in [PYL93] and [PYD94]. In particular, the following two testing problems have been addressed:
(a) Certain faults can be masked and cannot be easily activated. This raises the question of how to formally characterize detectable and undetectable faults in a given component machine. One of the possible solutions lies in computing a fault function for the composed machine and applying the FF-method [PeYe92] to derive tests tuned to detectable faults.
(b) Given a component embedded within a system of CFSMs, the ability to control its inputs and observe its outputs is decreased. This raises the question of transforming the component FSM into an FSM such that its acceptable input sequences can always be delivered to this component, and its output sequences represent those which result in the same external outputs. This is the so-called testable representation (an embedded equivalent) of the component under test [PYD94].

An embedded equivalent of even a deterministic and complete FSM is typically a partial nondeterministic FSM; acceptable input sequences represent executable test sequences, and nondeterministic transitions characterize undetectable faults in the IUT in the sense that they result in the same external behavior observed by testers. To exemplify this, consider the case of an embedded IUT where the environment does not deliver any message to the IUT until the environment reaches a data transfer phase when it is transparent to correct messages; on the other hand, the environment ignoring various erroneous interface messages from the IUT does not deliver them to testers either. The implementations are deterministic machines; therefore, the reduction relation can serve as a conformance relation in this case.

To further clarify the idea of embedded equivalents, we reproduce here an example of two cascaded FSMs from [PYD94]. Figure 8 (a) shows a system *C* of the two FSMs *H* and *T* shown in Figure 8 (b) and (c), respectively. The resulting external behavior is specified by the composed machine (d).

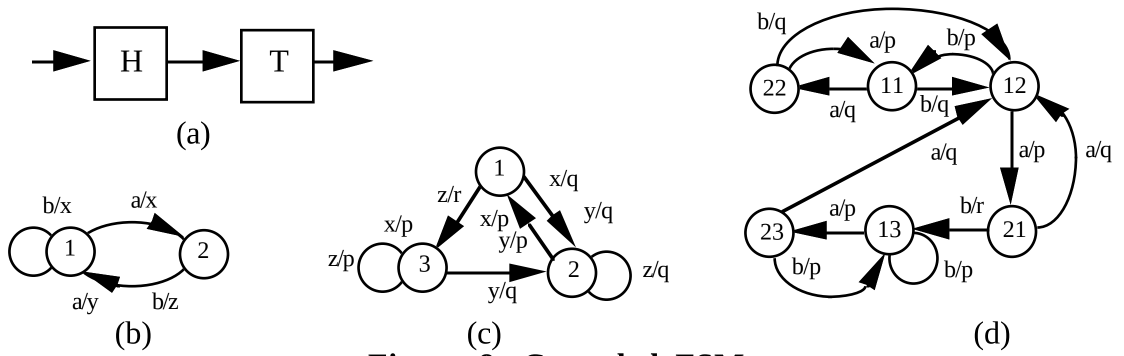


Figure 8: Cascaded FSMs

The head *H* component is controllable, but not directly observable. As an example, consider sequences described by the regular expression (x+y)(x+y), all result in the same output sequence *qp* of the composed machine starting from the initial state 11. All such sequences are represented by an embedded equivalent of the head machine *H* that is the nondeterministic FSM shown in Figure 9 (a). The FSM *H* is a deterministic reduction of this NFSM. To test only the head component *H* we should derive a test suite for the NFSM with respect to the reduction relation; an appropriate method can be found in [PYB94].

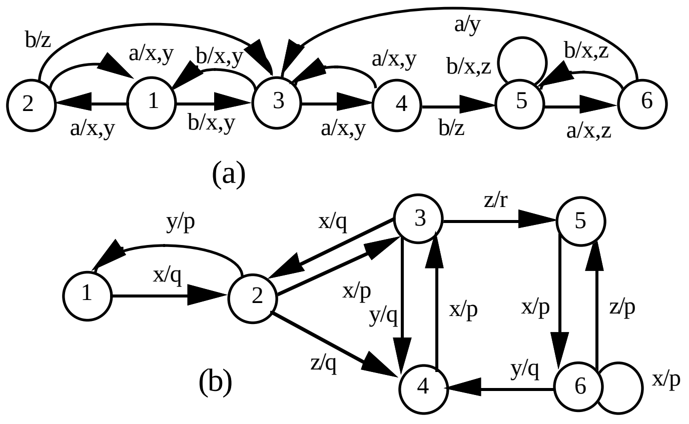


Figure 9: Embedded equivalents

The tail machine *T* is observable, but not directly controllable. As an example, consider an input *z* which cannot be delivered to *T* by *H*, both starting from initial states. An embedded equivalent of *T* is the partial FSM shown in Figure 9 (b). Every sequence acceptable by this machine can be delivered to *T*. The FSM *T* is quasi-equivalent to this partial machine. To test only the tail component we should derive a test suite for the FSM (Figure 9b) with respect to the quasi-equivalence relation, assuming the "forbidden transition" convention. Appropriate methods exist [Petr91], [YePe90], [LPB94b]. It is clear that in the case of feedback compositions, an embedded equivalent becomes partially specified and nondeterministic at the same time.

Note that the relevance of nondeterministic models for protocol testing has often been questioned by the observation that protocol implementations tested in practice typically exhibit a deterministic behavior**.** The above results clearly show that nondeterministic machines are a useful model in the grey-box setting. Further research is still required to develop methods for test derivation with the desirable fault coverage which could be applicable to the component testing in general, and to the standardized test architectures for communication protocols [Rayn87] in particular.

7. 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. Theoretical results discussed in this paper might not be all directly applicable to most practical testing situations, however, they help to better understand the limitations of conformance testing and to characterize various testing strategies in terms of their fault coverage. Moreover, these results facilitate related work on design for testability (DFT) of protocols in general and on testability evaluation and enhancement in particular [PDK93], [VLC93], [YPD95].

The techniques of fault coverage analysis can be used for incremental test suite development. When a given test suite is found to be not 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 completeness. This process is repeated until the test suite has achieved complete (or at least acceptable) 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 completeness of tests is in the area of diagnostics of FSM implementations [GhBo92], [LeSa93]. If an FSM implementation 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 (see Section 4.4). 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 poor fault coverage.

Fault coverage analysis is also useful for elaborating the heuristics or, speaking more generally, the testing strategies which 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 testing strategies with a better compromise between the fault coverage and the size of a test suite.

Further research is needed to elaborate testing strategies combining the fault coverage criterion with various coverage criteria for specifications in the form of extended finite state machines and existing formal description techniques.

Acknowledgments

This research was supported by the Hewlett-Packard-NSERC-CITI Industrial Research Chair on Communication Protocols at the Université de Montréal and an NSERC strategic research grant. We owe many thanks to R. Dssouli, G. Luo, Q. Tan, M. Yao, N. Yevtushenko, and others for the fruitful discussions on this topic.

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.

[BDD91] 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.

[BHS91] F. Belina, D. Hogrefe, and A. Sarma, SDL with Applications from Protocol Specification, Prentice Hall International, 1991.

[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, pp. 109-124.

[BPY94] G. v. Bochmann, A. Petrenko, and M. Yao, "Fault Coverage of Tests Based on Finite State Models", the Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, 1994, Japan.

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

[BoUy91] B. S. Bosik and M. U. Uyar, "Finite State Machine Based Formal Methods in Protocol Conformance Testing: from Theory to Implementation", Computer Networks and ISDN Systems, 22, 1991, pp.7-33.

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

[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, pp. 212-220.

[ChZh93] S. T. Chanson and J. Zhu, "A Unified Approach to Protocol Test Generation", in Proc. of the IEEE INFOCOM'93, pp.106-114.

[CTU93] W. H. Chang, C. Y. Tang, and H. Ural, "Minimum-cost Synchronizable Test Sequence Generation via the Duplex U-digraph", Proc. of the IEEE INFOCOM, 1993, pp. 128-135.

[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, “An Experience in Estimating Fault Coverage of a Protocol Test”, Proc. of the IEEE INFOCOM, 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, pp.195-206.

[FaPe90] A. Faro and A. Petrenko, "Sequence Generation from EFSMs for Protocol Testing", Participants' Proc. of COMNET'90, Budapest, 1990.

[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, pp. 267-280.

[FBK91] 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, pp. 591-603.

[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.

[GHN93] J. Grabowski, D. Hogrefe, and R. Nahm, "Test Case Generation with Test Purpose Specification by MSGs", Proceedings of the 6th IFIP WG 6.1 International Conference on Formal Description Techniques, FORTE'93, pp.253-265.

[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.
[Glab90] R. J. v. Glabbeek, "The Linear Time-Branching Time Spectrum", LNCS, 458, 1990.
[Gold78] E. M. Gold, "Complexity of Automaton Identification from Given Data", Information and Control, No.3. 1978, pp. 302-320.
[Gone70] G. Gonenc, "A Method for Design of Fault Detection Experiments", IEEE Trans., Vol C-19, June, 1970, pp.551-558.
[Henn64] F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits", Proc. of the IEEE 5th Ann. Symp. on Switching Circuits Theory and Logical Design, 1964, pp. 95-110.
[Hoar85] C. A. R. Hoare. Communicating Sequential Processes, Prentice Hall, 1985.
[Hsie71] E. P. Hsieh, "Checking Experiments for Sequential Machines", IEEE Trans., Vol. C-20, No.10, 1971, pp. 1152-1166.
[JeWe94] B. Jeng and E. J. Weyuker, "A Simplified Domain-Testing Strategy", ACM Transaction on Software Engineering and Methodology, Vol.3, No.3, July 1994, pp. 254-270.
[Kars94] P. Kars, "Test Coverage Estimation by Explicit Generation of Faulty FSMs", the Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, 1994, Japan.
[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", IFIP Transactions, Protocol Test Systems, VI, (Proceedings of IFIP TC6 Sixth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 139-153.
[LeSa93] D. Lee and K. Sabnani, "Reverse-Engineering of Communication Protocols", Proc. of the IEEE International Conference on Network Protocols, California, 1993, pp.208-216.
[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.
[LeYa94] D. Lee and M. Yannakakis, "Testing Finite-State Machines: State Identification and Verification", IEEE Trans. on Computers, Vol. 43, No. 3, 1994, pp. 306-320.
[LBP94] G. Luo, G. v. Bochmann, and A. Petrenko, "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.
[LPB94a] G. Luo, A. Petrenko, and G. v. Bochmann, "Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines", the Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, 1994, Japan.
[LPB94b] G. Luo, A. Petrenko, and G. v. Bochmann, "Generating Tests for Communication Software Modeled by Partially-Specified Finite State Machines", to appear in IEEE/ACM Transactions on Networking.
[LSK93] D. Lee, K. Sabnani, D. Kristol, S. Paul, and M. Uyar, "Conformance Testing of Protocols Specified as Comminicating FSMs", in Proc. of the IEEE INFOCOM'93, pp.116-124.
[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 Sixth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 21-33.
[MiLu91] R. E. Miller and G. M. Lundy, "Testing Protocol Implementations Based on a Formal Specification", Protocol Test Systems, III, (Proceedings of IFIP TC6 Third International Workshop on Protocol Test Systems, 1990), Ed. by I. Davidson and D. W. Litwack, 1991, North-Holland, pp. 289-304.
[MiPa92] R. E. Miller and S. Paul, "Generating Conformance Test Sequences for Combined Control and Data of Communicaion Protocols", 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. 1-15.

[MiPa93] R. E. Miller and S. Paul, "Generating Maximal Fault Coverage Conformance Test Sequences of Reduced Length for Communicaion Protocols", Proceedings of the IEEE International Conference on Network Protocols, California, 1993, pp.217-224.

[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 Sixth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 157-178.

[PDK93] A. Petrenko, R. Dssouli, and H. Konig, "On Evaluation of Testability of Protocol Structures", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Sixth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 111-123.

[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.

[PhGr91] M. Phalippou and R. Groz, "Evaluation of an Empirical Approach for Computer-Aided Test Case Generation", Protocol Test Systems, III, (Proceedings of IFIP TC6 Third International Workshop on Protocol Test Systems, 1990), Ed. by I. Davidson and D. W. Litwack, 1991, North-Holland, pp. 131-147.

[PrGu91] R. Probert and F. Guo, "Mutation Testing of Protocols: Principles and Preliminary Results", Protocol Test Systems, III, (Proceedings of IFIP TC6 Third International Workshop on Protocol Test Systems, 1990), Ed. by I. Davidson and D. W. Litwack, 1991, North-Holland, pp. 57-76.

[PYB94] A. Petrenko, N. Yevtushenko, and G. v. Bochmann, "Experiments on Nondeterministic Systems for the Reduction Relation", Universite de Montreal, DIRO, Technical Report #932, 1994, p.23 (submitted for publication).

[PYD94] A. Petrenko, N. Yevtushenko, and R. Dssouli, "Testing Strategies for Communicating FSMs", the Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, 1994, Japan.

[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 Sixth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp. 363-378.

[Rayn87] D. Rayner, "OSI Conformance Testing", Computer Network and ISDN Systems, 14, No.1, 1987, pp.79-98.

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

[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.

[SiLe88] D. P. Sidhu and T. K. Leung, “Fault Coverage of Protocol Test Methods", Proc. of IEEE INFOCOM'88.

[Star72] P. H. Starke, Abstract Automata, North-Holland/American Elsevier, 1972, 419p.

[TPB95] Q. M. Tan, A. Petrenko, and G. v. Bochmann, "Modelling Basic LOTOS by FSMs for

Conformance Testing", to appear in the ISPSTV'95, Poland.
[TrBa73] B. A. Trakhtenbrod and Ya. M. Barzdin, Finite Automata. Behavior and Synthesis, (translated from Russian), North-Holland Publ. Comp., 1973, 321p.
[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.
[Ural92] H. Ural, "Formal Methods for Test Sequence Generation", Computer Comm., Vol. 15, No. 5, 1992, pp. 311-325.
[UrZh93] H. Ural and K. Zhu, "Optimal Test Sequence Generation Using Distingushing Sequences", IEEE Trans. on Networking, Vol.1, 3, 1993, pp.358-371.
[UrWi93] H. Ural and A. Williams, "Test Generation by Exposing Control and Data Dependencies within System Specification in SDL", Proc. of FORTE'93.
[Vasi73] M. P. Vasilevski, "Failure Diagnosis of Automata", Cybernetics, Plenum Publishing Corporation, New York, No.4, 1973, pp. 653-665.
[VLC93] S. T. Vuong, A. A. F. Loureiro, and S. T. Chanson, "A Framework for the Design for Testabilitiy of Communication Protocols", 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.89-108.
[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.
[VuCu91] S. T. Vuong and J. Curgus, "Test Coverage Metrics for Communication Protocols", 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.
[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", Proc. of the IEEE INFOCOM'93, pp. 774-781.
[YaLe91] M. Yannakakis and D. Lee, "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, "A 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, pp.768-775.
[YPB94b] M. Yao, A. Petrenko, and G. v. Bochmann, “A Structural Analysis Approach to the Evaluation of Fault Coverage for Protocol Conformance Testing”, Proceedings of the 7th IFIP WG 6.1 International Conference on Formal Description Techniques, FORTE'94, Ed. by D. Hogrefe and S. Leue, 1995, pp.399-414.
[YPD95] N. Yevtushenko, A. Petrenko, R. Dssouli, K. Karoui, and S. Prokopenko, "On the Design for Testability of Communication Protocols", Universite de Montreal, DIRO, Technical Report #971, 1994, p.23 (accepted to IWPTS'95, France).
[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.