A Structural Analysis Approach to the Evaluation of Fault Coverage for Protocol Conformance Testing*

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

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

Abstract

In this paper, we propose a structural analysis approach to the evaluation of fault coverage of protocol conformance testing based on the finite state machine model. The attractiveness of this approach is its low computational complexity. It allows us to calculate the fault coverage of a given test suite by directly analyzing the test suite against the specification machine. Therefore, it avoids the generation and execution of mutants. The approach has been implemented and a number of experiments has been carried out. Some of the experimental results are summarized in this paper to show the accuracy of this approach compared with the mutation analysis technique.

Keywords: I.3, I.6, III,1, IV.3, IV.4.

1 Introduction

The finite state machine (FSM) model has been widely used in the development of hardware and software systems. Especially in the recent years, it has been extensively used in the conformance testing of communication protocols. Quite a number of methods have been proposed in the literature for generating test suites from protocol specifications given in the form of FSMs. An important issue related to the test suite generation is the *effectiveness* of a test suite which depends primarily on its fault coverage, i.e., its capability of detecting faults in a potential implementation. Approaches based on Monte-Carlo simulation have been proposed in the literature to evaluate the fault coverage of a test suite generated from a given FSM specification [DaSa 88, DDB 91, SiLe 89 and MCS 93]. These approaches require a (large) number of mutants of the given specification machine to be enumerated and executed against the test suite. In our recent work [YPB 94], a different procedure has been developed which, without the need of explicitly generating and then executing a certain (and often large) number of mutant machines, can decide if the given test suite provides full fault coverage (i.e., if it can detect all bad implementation machines). However, this approach does not provide a numeric measure for a test suite which does not have full fault coverage. Consequently, it is impossible to use this approach to compare the fault coverage of two test suites, if none of them provides full fault coverage. In this paper, we will propose a structural analysis approach to evaluate the fault coverage of a given test suite. The basic idea of this approach is, by analyzing the given test suite against the specification machine, to make an estimation on the number of mutants

* This research was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at University of Montreal.

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

representing potential bad implementations that can be detected by the test suite without the need to generate and execute the mutants.

The rest of the paper is organized as follows. In Section 2, the FSM model is introduced and a framework of software testing based on this model is presented. The structural analysis approach to the evaluation of fault coverage of a test suite is developed in Section 3. The "Order Coverage", which is based on the structural analysis approach and proposed to deal with large complex specification machines, is then presented in Section 4. Finally, in Section 5, our approach is compared with other related work.

2 A Framework of Testing Based on the FSM Model

We will first introduce the finite state machine model and then present a software testing framework based on this model.

2.1 The FSM Model

A *finite state machine* (FSM), often simply called a *machine* throughout this paper, is essentially an *initialized Mealy machine* which can be formally defined as follows.

Definition 2.1 (finite state machine)
A finite state machine is a 7-tuple <S, X, Y, S_1, δ, λ, D>, where
S is a set of n states {S_1, S_2, ..., S_n} with S_1 as the initial state;
X is a finite set of input symbols;
Y is a finite set of output symbols;
D is a specification domain which is a subset of S x X;
δ is a transfer function δ: D --> S;
λ is an output function λ: D --> Y. ■

An FSM is said to be *completely specified* (*defined*), iff D = S x X. Otherwise it is said to be *partially* or *incompletely specified* (*defined*). Since δ and λ are required to be functions, this FSM model is *deterministic*. That is, for each (S_i, x) [D, there should be exactly one state S_j [S and exactly one output symbol y [Y such that $\delta(S_i, x) = S_j$ and $\lambda(S_i, x) = y$. In this case, we say there is a transition leading from state S_i to S_j with input x and output y. Such a transition is usually written as S_i -x/y-> S_j, or as a triplet < S_i; x/y; S_j >. S_i is said to be the *head* or *starting* state of the transition, while S_j is said to be the *tail* or *ending* state of the transition. An FSM can be given in a graph form, with the states and transitions of the FSM represented by the vertices and arcs of the graph, respectively. As an example, Figure 1 gives a FSM which is partially specified since, at state S_3, no transition is specified for input symbol 1.

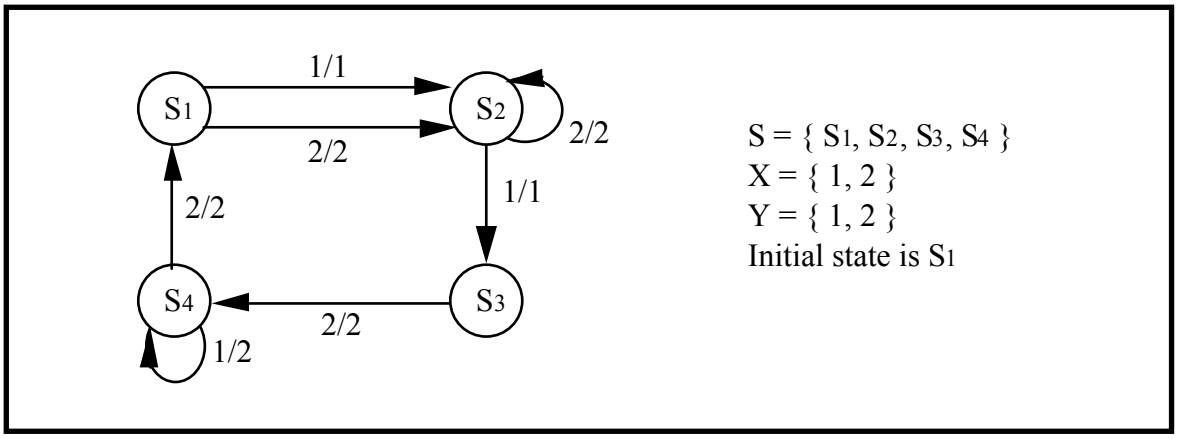


Figure 1: An example FSM

The following notations will be used throughout the paper. For a given symbol set Z, Z^* is used to represent the set of words constructed on Z and "ε" to represent the *empty* word, i.e., the word consisting of no symbols. Also, the dot "·" is used to represent the concatenation operation of two words. However, this dot symbol is often omitted when no ambiguity arises. Furthermore, |Z| is used to represent the cardinality of Z.

Definition 2.2 (defined input sequence)

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

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

in the finite state machine. ■

We use $\psi(S_i)$ to denote the set of all the defined input sequences for state S_i. A sequence of transitions such as (2-1) can be abbreviated as $S_i \text{ -p/q-> } S_{ik}$, which, when we do not care about the output sequence q, can be further simplified as $S_i \text{ -p-> } S_{ik}$, with the meaning that the FSM, when in state S_i and given an input sequence p, will enter state S_{ik}. The definitions of the transfer function δ and output function λ can be naturally extended to apply not only to single inputs, but also to sequences of inputs.

Definition 2.3 (extensions of transfer and output functions to input sequences)

Let $p = x_1x_2...x_k \in \psi(S_i)$ and ε be the empty word. Then,

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

$\lambda(S_i, \varepsilon) = \varepsilon, \quad \lambda(S_i, p) = \lambda(S_i, p').\lambda(\delta(S_i, p'), x_k)$

where $p' = x_1x_2...x_{k-1}$. ■

Definition 2.4 (compatible states and distinct states)

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

According to the above definition, if $\psi(S_i) \cap \psi(S_j) = \phi$, then S_i is compatible with S_j. If the FSM happens to be completely specified, then the definition of compatible states given above reduces to the definition of *equivalent states* as found in the literature (see for example, [Gill 62, Koha 78]).

Definition 2.5 (reduced machine)
A FSM is said to be *reduced* if and only if no two states are compatible. ■

It is easy to verify that the FSM given in Figure 1 is reduced.

Definition 2.6 (reachable state and strongly connected FSM)
A state S_i is said to be *reachable* (from the initial state S_1) if there exists an input sequence p [$\psi(S_i)$ such that S_1 -p->S_i. A machine is said to be *initially connected* if all the states are reachable. ■

Apparently, all the states of the FSM in Figure 1 can be reached from the initial state and therefore this example FSM is initially connected.

Definition 2.7 (mutant machine)
Let M_1 and M_2 be two given FSMs. M_2 is said to be a *mutant* machine of M_1 if M_2 is obtained by applying to M_1 each of the following four types of operations, in any order, for a certain number of times (including zero times):
Type 1: change the tail state of a transition;
Type 2: change the output of a transition;
Type 3: add a transition; and
Type 4: add an extra state. ■

The following corollary follows directly from the above definition.

Corollary 2.8
A machine is a mutant machine of itself. ■

2.2 A Testing Framework Based on FSM Model

The FSM model was widely used in traditional hardware testing. In recent years, this model has also received much attention in the testing of certain software systems such as communication protocols [PBD 93] and object-oriented programs [HoSt 93, TuRo 92]. Testing based on the FSM model can be formalized as the problem of *testing a FSM implementation*[Ural 91]: given a FSM representation (specification) of a system (denoted henceforth as M_S) and an implementation of the system (denoted henceforth as M_I), we are required to determine if the implementation machine M_I *conforms to* (i.e., is *correct* with respect to) the specification machine M_S by testing M_I as a black-box. This implies that we should generate from M_S a set of input sequences, called a test suite, and the corresponding set of expected output sequences such that M_I conforms to M_S if and only if, when the input sequences in the test suite are applied to M_I, the observed output sequences from M_I are the same as the corresponding expected output sequences. As already pointed out in the literature [Moor 56, Gill 62, YPB 93a, YPB 93b], 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. Firstly, the restrictions on the specification machine are summarized in the first assumption.

Assumption 1: (reduced and initially connected specification machine)
The given specification machine M_S is reduced and initially connected. ■

Secondly, testing based on the FSM model is essentially a mutation testing. Therefore, for the given specification machine M_S, an implementation machine M_I is actually a mutant machine (of M_S) obtained from M_S by applying each of the four types of operations listed in Definition 2.7 for a number of times (including zero times). These four types of operations represent the basic types of changes that can be made during the implementation of M_S. However, it should be noted that, in practice, the implementation machine M_I is normally completely defined even though the given specification machine M_S is often only partially specified. Therefore, the following assumption is made throughout this paper.

Assumption 2: (completeness of an implementation machine)
For the given specification machine M_S, an implementation machine M_I is a completely defined mutant machine of M_S. ■

Thirdly, if the number of changes of Type 4 applied to the given specification machine M_S is not limited, the number of mutants of M_S will be infinite and the problem of testing will become intractable. Therefore, in practice, the number of changes of Type 4 is always limited to an upper bound. Throughout this paper, we simply do not allow any change of Type 4 as stated in the next assumption.

Assumption 3: (limited number of states in an implementation machine)
For the given specification machine M_S, any operation of Type 4 is not allowed and therefore the number of states in an implementation machine M_I will not exceed that of M_S. ■

We also note here that additional types of changes, such as changes of inputs, changes of head states and missing states, may be introduced [MiPa 92]. However, these types of changes are not necessary for our discussion as the FSM model is deterministic (Definition 2.1) and implementation machines are assumed to be completely defined (Assumption 2). The following example explains that the same consequences of a missing state can be achieved by changing the tail states of certain transitions (Type 1). Figure 2 (a) shows that, when implementing the FSM given in Figure 1, state S_4 is not implemented (i.e., missing in the implementation) and the transition $< S_3; 2/2; S_4 >$ is changed to $< S_3; 2/2; S_1 >$. However, we can still think that state S_4 is present in the implementation as shown in Figure 2 (b). The reason is that S_4 is no longer reachable and therefore, during the black-box testing, whether S_4 is present or missing in the implementation makes no difference.

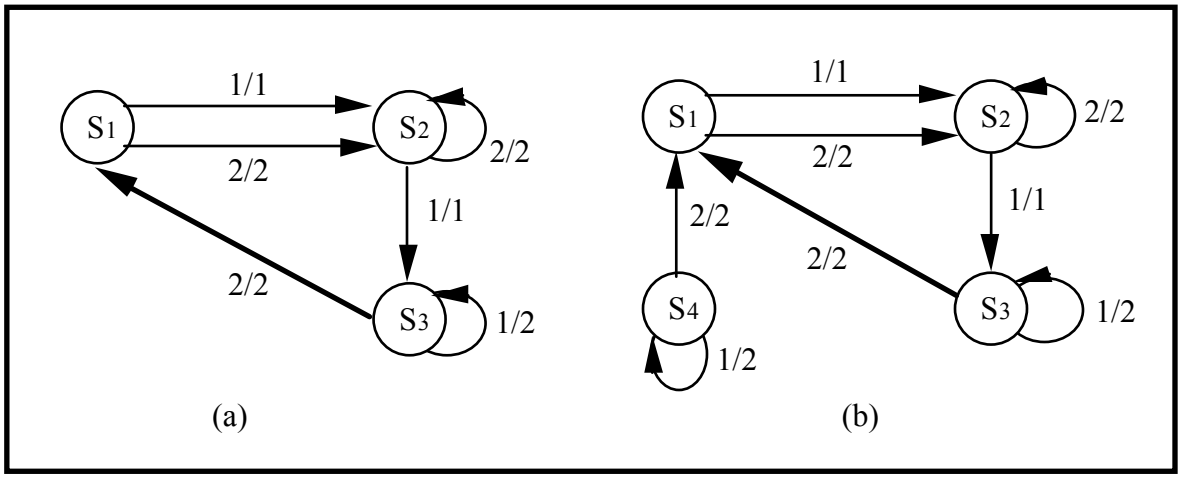


Figure 2: missing state

Therefore, as a matter of fact, all the n states S_1, S_2, ..., S_n of the specification machine M_S are assumed to be present in an implementation machine M_I. However, some of these states may become unreachable in M_I due to the changes of Type 1 introduced during the implementation. Confusion may arise because the same state names S_1, S_2, ..., S_n are used for both M_S and M_I. It is therefore often helpful, although not necessary, to make things clear by renaming the states S_1, S_2, ..., S_n in M_I to I_1, I_2, ..., I_n, respectively. Then without loosing generality, let

$M_S = < \{S_1, S_2, ..., S_n\}, X, Y, S_1, \delta_S, \lambda_S, D_S >$, and

$M_I = < \{I_1, I_2, ..., I_n\}, X, Y, I_1, \delta_I, \lambda_I, D_I >$.

Since M_I is supposed to be completely defined, we know that $D_I = \{I_1, I_2, ..., I_n\} \times X$ and therefore $\psi(I_i) = X^*$ and $\psi(S_j) \{ \psi(I_i)$, for any I_i and S_j. Now, we need to introduce some important concepts. The first concept required is the so-called *conformance relation* which essentially defines when M_I is a correct implementation of M_S. This concept is defined through the following two definitions.

Definition 2.9 (equivalence of states in respect to a set of input sequences)

Let I_i be a state of M_I and S_j a state of M_S. V is a set of input sequences such that $V \{ \psi(S_j)$. Then

$I_i \sim_V S_j$ if $\lambda_I (I_i, p) = \lambda_S(S_j, p)$, for : $p [V$. ■

Definition 2.10 (conformance relation)

M_I conforms to M_S, written M_I **CONF** M_S, if and only if $I_1 \sim_{\psi(S1)} S_1$, where I_1 and S_1 are the initial states of M_I and M_S, respectively. ■

The above defined conformance relation corresponds to the notion of weak conformance [SaDa 88, SiLe 89 and MiPa 92]. The relationship between the above defined conformance relation and the types of operations listed in Definition 2.7 is established by the following lemma of which the proof is similar to that of Lemma A.1 given in the appendix of [YPB 93a].

Lemma 2.11

For the specification machine M_S and implementation. Then M_I conforms to M_S if and only if there exists a mapping **f**: $\{S_1, S_2, ..., S_n\}$ -> $\{I_1, I_2, ..., I_n\}$, such that
(1) **f** is one-to-one; and
(2) If S_i - x/y -> S_j is in M_S, then I_k - x/y -> $I_¬$ is in M_I, where I_k = **f**(S_i) and $I_¬$ = **f**(S_j). ■

Since the implementation machine M_I is treated as a black-box, test cases should be generated from the specification machine M_S. The following two definitions formally defines the concepts of test case and test suite.

Definition 2.12 (test case)
A test case is a sequence of inputs which should be of finite length and in $\psi(S_1)$. ■
As is clear from the above definition, a test case always starts from the initial state S_1 of the specification machine M_S. Accordingly, each test case should be applied to the implementation machine M_I when it is in its initial state I_1. Therefore, an important assumption in the testing based on the FSM model is about the availability of the so-called *reliable reset* function and is summarized as our fourth (and final) assumption.

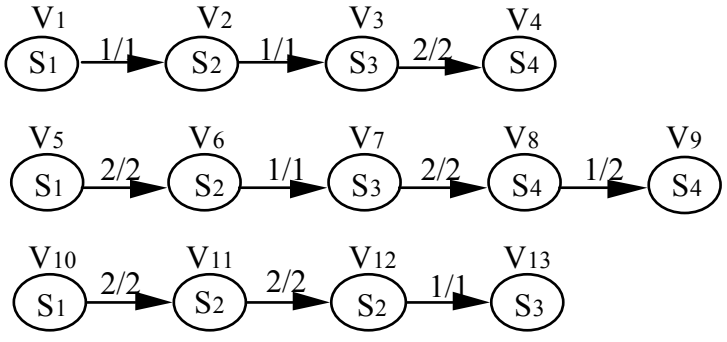
Assumption 4: (availability of reliable reset)
The *reliable reset* is an operation that, when activated, will bring the implementation from any other state back into its initial state. It is assumed to be available in an implementation under test. ■

A special input symbol "r" representing the invocation of the reset operation is added to the beginning of each test case.

Definition 2.13 (test suite)
A test suite is a finite set of test cases. ■

TS = { r.1.1.2, r.2.1.2.1, r.2.2.1 }

(a)



(b)

Figure 3: A test suite generated from the example FSM

Figure 3 (a) lists the test cases of a test suite generated from the machine shown in Figure 1. Each of the test cases is prefixed by the reset symbol "r". Applying the three test cases to the initial state S_1 results in the three sequences of transitions shown in Figure 3 (b) that will be executed.

Definition 2.14 (to pass a test suite)
Let TS be a test suite and p [TS be a test case. We say that a given implementation M_I passes the test case p, written M_I **pass** p, iff $\lambda_I (I_1, p) = \lambda_S(S_1, p)$. Further, we say that M_I passes the test suite TS, written M_I **pass** TS, iff M_I **pass** p, for : p [TS. ■

An implementation machine which cannot pass a given test suite is said to *fail* the test suite or to be *detected* by the test suite.

Let **Impl**(M_S) represent the set of all the implementation machines of M_S, i.e., all the completely defined mutant machines with same number of states as M_S. Then the following lemma follows directly from the above definitions.

Lemma 2.15
Let TS be a given test suite for M_S and M_I [**Impl**(M_S). M_I does not pass TS implies that M_I does not conform to M_S.

■

3 A Structural Analysis Approach to Evaluating Fault Coverage

In this section, we are going to present a structural analysis approach to the evaluation of fault coverage of a test suite in respect to a given specification machine. This approach avoids the necessity of explicit generation and execution of mutant machines representing possible implementations of the given specification machine M_S. First of all, let us introduce the following notations:

N_1(M_S) - the total number of machines in **Impl**(M_S);
N_2(M_S) - the number of machines in **Impl**(M_S) which conform to M_S;
N_3(M_S) - the number of machines in **Impl**(M_S) which do not conform to M_S;
N_4(M_S, TS) - the number of machines in **Impl**(M_S) which can pass the given test suite TS;
N_5(M_S, TS) - the number of machines in **Impl**(M_S) which do not conform to M_S and cannot pass the given test suite TS;

We have the following lemma whose validity is obvious (see [Gill 62, SiLe 89]).

Lemma 3.1
The total number of implementation machines, that is the cardinality of **Impl**(M_S), is

$$\mathbf{N_1}(M_S) = \left|\mathbf{Impl}\left(M_S\right)\right| = (n|Y|)^{n|X|} \qquad \textbf{(3-1)}$$

Among these implementation machines, there are

$$\mathbf{N}_2(M_S) = (n - 1)!\,(n|Y|)^{n|X| - |D_S|} \quad \textbf{(3-2)}$$

implementation machines which conform to M_S, where n, X, Y and D_S are the number of states, the input set, the output set and the specification domain of M_S, respectively. ■

It is also easy to see that $\mathbf{N_3}(M_S) = \mathbf{N_1}(M_S) - \mathbf{N_2}(M_S)$ and $\mathbf{N_5}(M_S, TS) = \mathbf{N_1}(M_S) - \mathbf{N_4}(M_S, TS)$. Therefore, we can give the following so-called *precise fault coverage* of a given test suite TS in respect to a given specification machine M_S.

Definition 3.2 (precise fault coverage)
The precise fault coverage of a test suite TS in respect to M_S, denoted as $\mathbf{FC}_p(M_S, TS)$, is

$$\mathbf{FC}_p(M_S, TS) = \frac{\mathbf{N}_5(M_S, TS)}{\mathbf{N}_3(M_S)} = \frac{\mathbf{N}_1(M_S) - \mathbf{N}_4(M_S, TS)}{\mathbf{N}_1(M_S) - \mathbf{N}_2(M_S, TS)}$$ ■

As is clear from the above definition, in order to calculate the fault coverage, we still need to find $\mathbf{N_4}(M_S, TS)$, the number of machines in $\mathbf{Impl}(M_S)$ which can pass the given test suite TS. The exact value of $\mathbf{N_4}(M_S, TS)$ is in general too difficult to find without using the exhaustive mutation analysis technique. However, as we have already mentioned, the exhaustive analysis technique is often not feasible in practice due to the high cost. Therefore, in our approach, we will use an estimated value, denoted as $\mathbf{N_4}(M_S, TS)$, of $\mathbf{N_4}(M_S, TS)$. Substituting $\mathbf{N_4}(M_S, TS)$ for $\mathbf{N_4}(M_S, TS)$ in the calculation of the fault coverage as defined in Definition 3.2 results in the following estimated fault coverage.

Definition 3.3 (estimated fault coverage)
The estimated fault coverage of a test suite TS in respect to M_S, denoted as $\mathbf{FC}_e(M_S, TS)$, is

$$\mathbf{FC}_e(M_S, TS) = \frac{\mathbf{N}_1(M_S) - \mathbf{N}_4(M_S, TS)}{\mathbf{N}_1(M_S) - \mathbf{N}_2(M_S, TS)}$$ ■

Definition 3.4 (prefix set of a test suite)
The prefix set **AP**(TS) of a test suite TS is the set which consists of all the prefixes of all the test cases in TS, i.e.,
AP(TS) = { p | p is a prefix of some test case in TS }. ■

Definition 3.5 (transition covered by TS)
A transition $< S_i; x/y; S_j >$ in M_S is said to be covered by TS, if there are α, αx ∈ **AP**(TS) such that $\delta_S(S_1, \alpha) = S_i$ and $\delta_S(S_1, \alpha x) = S_j$. ■

Definition 3.6 (tail state S_j of a transition distinguished from S_k by TS)
The tail state S_j of a transition $< S_i; x/y; S_j >$ in M_S is said to be distinguished from another state S_k by TS if there are α, αx, $\alpha x\gamma$, β, $\beta\gamma$ ∈ **AP**(TS) such that
$\delta_S(S_1, \alpha) = S_i$, $\delta_S(S_1, \alpha x) = S_j$, $\delta_S(S_1, \beta) = S_k$ and $\lambda_S(\delta_S(S_1, \alpha x), \gamma) \neq \lambda_S(\delta_S(S_1, \beta), \gamma)$. ■

We will proceed in two steps to find the value of $\mathbf{N_4}(M_S, TS)$. In the first step, we will make an estimation, denoted as $\mathbf{N_6}(M_S, TS)$, on the number of implementation machines which can pass the given test suite TS, by analyzing the structures of the given specification machine and test suite. In the second step, $\mathbf{N_6}(M_S, TS)$ will be adjusted to $\mathbf{N_4}(M_S, TS)$.

To find the value of $\mathbf{N_6}(M_S, TS)$, we need to classify the transitions of M_S into two classes: the first class includes the transitions covered by TS, while the second class consists of the transitions not covered by TS. Taking the specification machine given in Figure 1 and the test suite TS shown in Figure 3 (a) as an example, we can easily check that, among the seven specified transitions in Figure 1, the six transitions listed in Table 1 (a) are covered by the test suite TS, while the other transition given in Table 1 (b) is not covered.

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$(< S_i; x/y; S_j >, TS) = \{ S_1, S_2, ..., S_n \}$ - **Tail_Dis**$(< S_i; x/y; S_j >, TS)$
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**$(< S_i; x/y; S_j >, TS)$. As an example, let us consider one of the covered transition $< S_1; 2/2; S_2 >$ given in Table 1 (a). As is clear from Figure 3 (b), this transition is covered twice by the test suite. The tail state S_2 at point V_6 is distinguished from state S_4 at point V_8. The same tail state S_2 at point V_{11} is distinguished from state S_3 at point V_7. Therefore, **Tail_Dis**$(< S_1; 2/2; S_2 >, TS) = \{ S_3, S_4 \}$ and **Tail_NDis** $(< S_1; 2/2; S_2 >, TS) = \{ S_1, S_2 \}$.

$< S_1; 1/1; S_2 >$ $< S_1; 2/2; S_2 >$ $< S_2; 1/1; S_3 >$
$< S_2; 2/2; S_2 >$ $< S_3; 2/2; S_4 >$ $< S_4; 1/2; S_4 >$

(a) transitions covered by TS

$< S_4; 2/2; S_1 >$

(b) transition not covered by TS

Tail_Dis ($< S_1; 1/1; S_2 >$, TS) = { S_4 }
Tail_Dis ($< S_1; 2/2; S_2 >$, TS) = { S_3, S_4 }
Tail_Dis ($< S_2; 1/1; S_3 >$, TS) = { S_1, S_2 }
Tail_Dis ($< S_2; 2/2; S_2 >$, TS) = { S_4 }
Tail_Dis ($< S_3; 2/2; S_4 >$, TS) = { S_1, S_2 }
Tail_Dis ($< S_4; 1/2; S_4 >$, TS) = ϕ

(c)

Tail_NDis ($< S_1; 1/1; S_2 >$, TS) = { S_1, S_2, S_3 }
Tail_NDis ($< S_1; 2/2; S_2 >$, TS) = { S_1, S_2 }
Tail_NDis ($< S_2; 1/1; S_3 >$, TS) = { S_3, S_4 }
Tail_NDis ($< S_2; 2/2; S_2 >$, TS) = { S_1, S_2, S_3 }
Tail_NDis ($< S_3; 2/2; S_4 >$, TS) = { S_3, S_4 }
Tail_NDis ($< S_4; 1/2; S_4 >$, TS) = { S_1, S_2, S_3, S_4 }

(d)

Table 1: Intermediate calculation results for the example FSM and TS

The related results for the other five covered transitions can be found in Table 1 (c) and (d). 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 (as defined in Definition 2.7). 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 to generate an implementation machine which can pass TS, we have only one choice of 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, with certain test suite generation methods such as the W method [Chow 78], all the transitions in the specification machine will be covered

and the tail state of each transition will be distinguished from all other states. Therefore, **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 $(n|Y|)^m$ choices to generate an implementation machine which can pass TS (where m is the number of transitions not covered by TS).

As we have assumed in Section 2, an implementation machine should be completely defined. Therefore, for the given specification machine M_S which is in general partially specified, we need to apply the Type 3 operation to add an extra transition for each (S_i, x) [S x X - D_S. Since the tail state of such an extra transition can be any of the n states S_1, S_2, ..., S_n and the output symbol can be any one in Y, we know that there are a total of $(n|Y|)^{n|X| - |D_S|}$ possible ways to generate an implementation machine by adding n|X| - |D_S| extra transitions.

Following from the above discussions, we have

$$\mathbf{N}_6(M_S, TS) = (n|Y|)^{n|X| - |D_S| + m} \qquad |\mathbf{Tail\_NDis}\ (< S_i; x/y; S_j >, TS)| \qquad \textbf{(3-3)}$$

$$< S_i; x/y; S_j > \text{ covered}$$

implementation machines in **Impl**(M_S) which are estimated to be able to pass the given test suite, where m is the number of transitions not covered by TS.

However, we should note that, among all the implementation machines in **Impl**(M_S), there are $\mathbf{N}_2(M_S) = (n - 1)!\,(n|Y|)^{n|X| - |D_S|}$ implementation machines which conform to the given specification machine M_S and therefore can pass any test suite. As such, we need to adjust the estimated number of implementation machines that can pass the test suite TS and use $\mathbf{N_4}(M_S, TS) = \mathbf{max}(\mathbf{N_2}(M_S), \mathbf{N_6}(M_S, TS))$ (i.e., the maximum of the two)

$$\mathbf{FC}_e(M_S, TS) = \frac{\mathbf{N}_1(M_S) - \mathbf{max}(\mathbf{N}_2(M_S, TS),\ \mathbf{N}_6(M_S, TS))}{\mathbf{N}_1(M_S) - \mathbf{N}_2(M_S, TS)} \qquad \textbf{(3-4)}$$

Let us continue our example with the specification machine given in Figure 1 and the test suite shown in Table 1 (a). For this particular example, |D_S| = 7, m = 1, n = 4, |X| = |Y| = 2. Therefore, we have $\mathbf{N_1}(M_S) = 16777216$, $\mathbf{N_2}(M_S) = 48$, $\mathbf{N_6}(M_S, TS) = 18432$ and finally the estimated fault coverage $\mathbf{FC}_e(M_S, TS) = 99.89042\%$.

Several properties of $\mathbf{FC}_e(M_S, TS)$ given in (3-6) are summarized in the following theorem.

Theorem 3.7
(1) $0 \le \mathbf{FC}_e(M_S, TS) \le 1$;
(2) $\mathbf{FC}_e(M_S, TS) = 1$ if $\mathbf{FC}_p(M_S, TS) = 1$;
(3) $\mathbf{FC}_e(M_S, TS) = 0$ if $\mathbf{FC}_p(M_S, TS) = 0$; and
(4) $\mathbf{FC}_e(M_S, TS) \le \mathbf{FC}_e(M_S, TS')$ if $\mathbf{AP}(TS) \{ \mathbf{AP}(TS')$. ■

It is quite straightforward to prove these properties. However, we feel that the meaning of the forth property needs some explanation. We note that **AP**(TS) is the prefix set of TS and that **AP**(TS) { **AP**(TS') implies that TS' has more or longer test cases than TS. The forth property essentially tells us that the estimated fault coverage for TS and TS' coincide with the intuition that TS' provides better fault coverage than TS.

The calculation of the estimated fault coverage given in (3-4) can be simplified since a common factor $(n|Y|)^{n|X| - |D_S|}$ exists in $\mathbf{N_1}(M_S)$, $\mathbf{N_2}(M_S)$ and $\mathbf{N_6}(M_S, TS)$ as shown in (3-1), (3-2) and (3-3), respectively. After eliminating this common factor from all the items in (3-4), we have the following simplified formula:

$$\mathbf{FC}_e(M_S, TS) = \frac{\mathbf{K}_1(M_S) - \mathbf{max}(\mathbf{K}_2(M_S, TS),\ \mathbf{K}_3(M_S, TS))}{\mathbf{K}_1(M_S) - \mathbf{K}_2(M_S, TS)} \quad , \qquad \textbf{(3-5)}$$

where
$\mathbf{K}_1(M_S) = (n|Y|)^{|D_S|}$,
$\mathbf{K}_2(M_S, TS) = (n - 1)!$,
$\mathbf{K}_3(M_S, TS) = (n|Y|)^m \cdot \left| \mathbf{Tail\_NDis}\ (< S_i; x/y; S_j >, TS) \right|$

$< S_i; x/y; S_j >$ covered

From this simplified formula, we can see that only the specified transitions in a given specification machine will contribute to the calculation of the estimated fault coverage and those non specified can be excluded from consideration.

The above structural analysis approach to the evaluation of fault coverage has been implemented under SUN/UNIX and a number of experiments have been done to show how accurately the estimated fault coverage can match the precise fault coverage. Taking the FSM given in Figure 1 as a specification machine, we generated the following nine test suites:

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

Applying our structural analysis approach to these test suites yields the estimated fault coverage listed in the second column of Table 2. To assess the accuracy of these estimated fault coverage values, we need to compare them with the precise fault coverage values for these test suites. Fortunately, for the small specification machine given in Figure 1, we have been able to make an exhaustive mutation analysis. We have written a program which generates and executes one by one all the $(4 \times 2)^{(4 \times 2)} = 16777216$ possible implementation machines against each of the above nine test suites. Therefore, we have been able to calculate the precise fault coverage for these test suites and the results are listed in the third column of Table 2. The differences between the estimated and precise fault coverage are listed in the forth column of Table 2.

TS_i	$FC_e(MS, TS_i)$	$FC_p(MS, TS_i)$	**Deviation**
TS_1	0.00000%	0.00000%	0.00000%
TS_2	50.00014%	50.00014%	0.00000%
TS_3	99.34110%	99.25871%	0.08239%
TS_4	99.89042%	99.66497%	0.22545%
TS_5	99.89042%	99.66898%	0.22144%
TS_6	99.94535%	99.77655%	0.16880%
TS_7	99.94535%	99.88084%	0.06451%
TS_8	99.97282%	99.92032%	0.05250%
TS_9	100.00000%	100.00000%	0.00000%

Table 2: Fault Coverage for the FSM in Figure 1

We have also applied this structural analysis approach to some real protocol machines. For instance, the FSM shown in Figure 4 actually represents the control portion of the NBS transport protocol (class 4) [SiLe 89]. It has 15 states, 27 inputs, 46 outputs and 62 specified transitions. For this FSM, the following two test suites are generated in [SiLe 89] with the transition tour method and the UIO method, respectively.

TS_T = { r.x1.x14.x5.x5.x19.x24.x12.x3.x8.x9.x8.x11.x16.x8.x7.x11.x24.x2.x17.x1.x15.x5.
x24.x13.x19.x1.x14.x8.x10.x11.x16.x8.x7.x8.x9.x24.x12.x5.x24.x17.x19.x13.x18.
x21.x23.x24.x1.x15.x6.x24.x12.x22.x4.x19.x24.x17.x2.x27.x1.x14.x23.x24.x12.x3.
x9.x16.x11.x7.x24.x17.x1.x15.x23.x24.x1.x14.x25.x26.x16.x5.x24.x1.x15.x20.x6.
x24.x13.x18.x3.x16.x9.x8.x7.x24.x12.x3.x16.x10.x11.x7.x10.x11.x9.x24.x1.x14.
x16.x8.x9.x11.x7.x24.x1.x14.x16.x23.x24.x1.x14.x16.x.25.x26 }

TS_{UIO} = { r.x1.x14.x5.x5, r.x1.x14.x8.x9, r.x1.x14.x23.x5, r.x1.x14.x25.x8,
r.x1.x14.x26.x8, r.x1.x14.x16.x5.x5, r.x1.x14.x16.x8.x7, r.x1.x14.x16.x23.x5,
r.x1.x14.x16.x25.x8, r.x1.x14.x16.x26.x8, r.x1.x15.x5, r.x12.x3.x8.x9,

r.x12.x3.x9.x8.x16.x7, r.x12.x3.x9.x11.x16.x7, r.x12.x3.x9.x16.x8.x7,
r.x12.x3.x9.x16.x7.x5, r.x12.x3.x9.x16.x11.x7, r.x12.x3.x10.x9,
r.x12.x3.x11.x9, r.x12.x3.x16.x8.x7, r.x12.x3.x16.x7.x8.x9,
r.x12.x3.x16.x7.x9.x5, r.x12.x3.x16.x7.x10.x9, r.x12.x3.x16.x7.x11.x9
r.x12.x3.x16.x9.x8.x7, r.x12.x3.x16.x9.x7.x5, r.x12.x3.x16.x9.x11.x7
r.x12.x3.x16.x10.x7, r.x12.x3.x16.x11.x7, r.x12.x5.x5.x5, r.x12.x5.x19.x5
r.x12.x5.x24.x12, r.x12.x5.x11.x5, r.x12.x21.x3, r.x12.x23.x5, r.x12.x22.x3
r.x12.x4.x5.x5, r.x12.x4.x6.x5, r.x12.x4.x19.x5, r.x12.x4.x23.x5
r.x12.x4.x20.x5, r.x2.x12, r.x17.x1.x14, r.x17.x19.x12, r.x17.x2.x27
r.x17.x27.x12, r.x13.x19.x12, r.x13.x18.x3 }

For both these two test suites, our structural analysis approach yields the estimated fault coverage to be 100%. For such a large machine, we are apparently unable to use the exhaustive mutation analysis technique to simulate all the 690^{405} possible implementation machines. Therefore, we

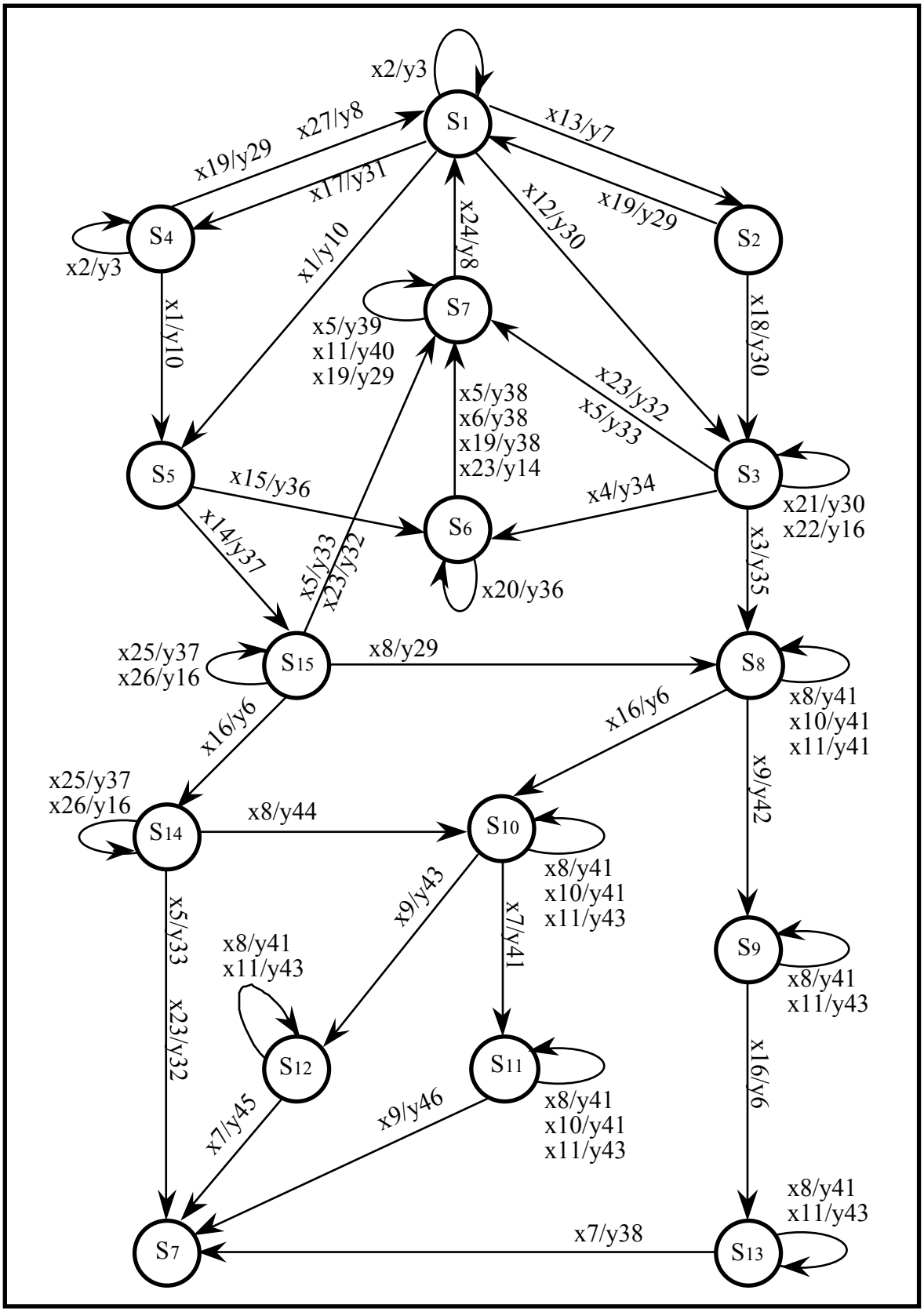


Figure 4: The NBS Transport Protocol (Class 4)

cannot compare our estimated fault coverage values with the corresponding precise fault coverage values here. However, we have also used the Monte-Carlo simulation approach to estimate the fault coverage for these two test suites. 10^6 randomly generated mutants, each of which does not conform to the specification machine given in Figure 4 and contains a random number of transfer and/or output faults, have been simulated. It turns out that both test suites can detect all these 10^6 non conforming mutants. Therefore, with the Monte-Carlo simulation approach, we have found that the fault coverage of these two test suites to be also 100%. This coincides with the result obtained with our structural analysis approach. It should be noted that our Monte-Carlo simulation differs from the one used in [SiLe 89] where 10 classes of mutants were defined and only mutants within those classes were simulated.

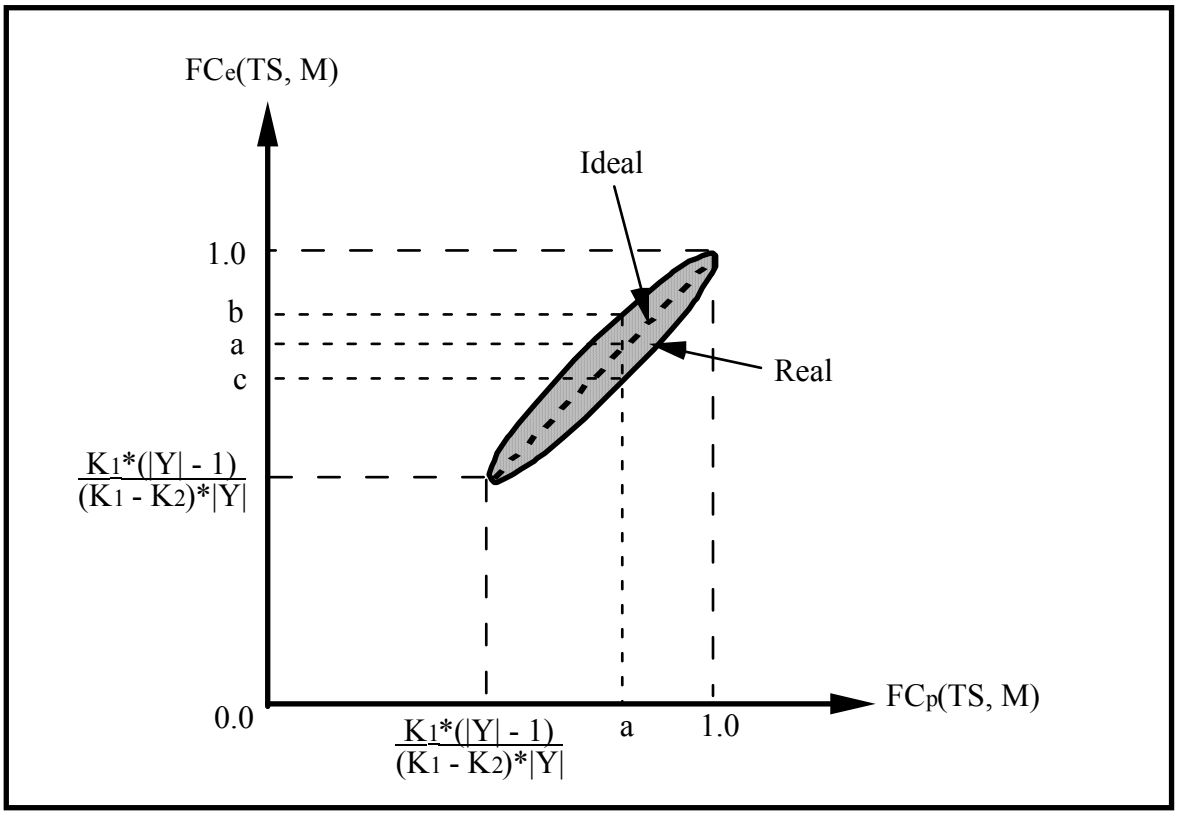


Figure 5: $\mathbf{FC}_e(M_S, TS)$ vs. $\mathbf{FC}_p(M_S, TS)$ for a given M_S

As is clear from the above two examples, the estimated fault coverage approaches the precise fault coverage when the latter is either very high (almost 100%) or relatively low. The general relationship between the estimated fault coverage and the precise fault coverage is shown in Figure 5. We have already pointed out in Theorem 3.7 that the estimated fault coverage $\mathbf{FC}_e(M_S, TS)$ takes the same value as the precise fault coverage $\mathbf{FC}_p(M_S, TS)$ whenever the latter is equal to 0 or 1. It is also shown in Figure 5 that, for a test suite TS which has only one input symbol (excluding the reset), both the estimated fault coverage $\mathbf{FC}_e(M_S, TS)$ and the precise fault coverage $\mathbf{FC}_p(M_S, TS)$ will be equal to $(K_1*(|Y| - 1))/((K_1 - K_2)*|Y|)$. For a test suite TS which has more than one input, the precise fault coverage $\mathbf{FC}_p(M_S, TS)$ will take a value "a", where $(K_1*(|Y| - 1))/((K_1 - K_2)*|Y|) \leq a \leq 1$. Ideally, we would like the estimated fault coverage

$\mathbf{FC}_e(M_S, TS)$ to take the same value "a". In reality, $\mathbf{FC}_e(M_S, TS)$ approximates $\mathbf{FC}_p(M_S, TS)$ by taking a value in the section [c, b], where $c \le a \le b$.

4 Fault Coverage at the Order Level

For a large specification machine M_S (i.e., a machine that has large numbers of states, outputs and specified transitions) and a non trivial test suite TS that covers most or all of the transitions, it can be observed from (3-5) that $\mathbf{K_1}(M_S) >> \mathbf{K_2}(M_S)$ and $\mathbf{K_1}(M_S) >> \mathbf{K_3}(M_S, TS)$ and therefore $\mathbf{FC}_e(M_S, TS) \spadesuit 1$. For example, we have found with formula (3-5) that the estimated fault coverage for the two test suites TS_T and TS_{UIO} of the NBS Transport Protocol shown in Figure 4 to be 100%. However, it is well known in the literature that a test suite generated with the UIO method will in general provide better fault coverage than a test suite generated with the transition tour method. It is therefore necessary to provide some kind of mechanism which can make a distinction between the two. Actually, the following formula, which we call "Order Coverage", can be used for this purpose:

$$\mathbf{FC}_o(M_S, TS) = \frac{\mathbf{Log}_{10}(\mathbf{K}_1(M_S)) - \mathbf{Log}_{10}(\mathbf{max}(\mathbf{K}_2(M_S, TS), \mathbf{K}_3(M_S, TS)))}{\mathbf{Log}_{10}(\mathbf{K}_1(M_S)) - \mathbf{Log}_{10}(\mathbf{K}_2(M_S, TS))} \qquad \textbf{(3-6)}$$

The intuition behind this formula is as follows. For a large specification machine and a non trivial test suite such that $\mathbf{K_1}(M_S) >> \mathbf{K_2}(M_S)$ and $\mathbf{K_1}(M_S) >> \mathbf{K_3}(M_S, TS)$, $\mathbf{K_2}(M_S)$ and $\mathbf{K_3}(M_S, TS)$ can be neglected in the calculation with formula (3-5). However, the orders (i.e., the logarithms) of $\mathbf{K_2}(M_S)$ and $\mathbf{K_3}(M_S, TS)$ are still comparable with that of the $\mathbf{K_1}(M_S)$ and therefore will not be neglected in the calculation with formula (3-6). For instance, although $\mathbf{FC}_e(M_S, TS_T) = \mathbf{FC}_e(M_S, TS_{UIO}) = 100\%$, we can still see a difference between the two test suites since $\mathbf{FC}_o(M_S, TS_T) = 63.82301\%$ and $\mathbf{FC}_o(M_S, TS_T) = 66.05067\%$.

The following theorem summarizes the properties of the "order coverage".

Theorem 4.1
(1) $0 \le \mathbf{FC}_o(M_S, TS) \le 1$;
(2) $\mathbf{FC}_o(M_S, TS) = 1$ if $\mathbf{FC}_e(M_S, TS) = 1$;
(3) $\mathbf{FC}_o(M_S, TS) = 0$ if $\mathbf{FC}_e(M_S, TS) = 0$; and
(4) $\mathbf{FC}_o(M_S, TS) \le \mathbf{FC}_o(M_S, TS')$ if $\mathbf{FC}_e(M_S, TS) \le \mathbf{FC}_e(M_S, TS')$. ■

5 Conclusions

In this paper, we have presented a structural analysis approach to the evaluation of fault coverage of a test suite in respect to a system specification given in the form of a finite state machine. This approach differs from those methods proposed in [DaSa 88, SiLe 89, DDB 91 and MCS 93] as it avoids the necessity of generating and executing a (large) number of mutant machines. Instead, it evaluates the fault coverage of a given test suite by directly analyzing the test suite against the specification machine. It provides a numeric measure for a test suite no matter whether the test suite has full fault coverage (i.e., 100%) or not. This feature makes the metric approach different from one of our previous work [YPB 94] where a test suite is analyzed only to see if it provides full fault coverage or not. As we have seen in Section 3, applications of

our approach to a number of examples have shown that the estimated fault coverage of a test suite is quite close to the precise fault coverage, especially when the test suite approaches full fault coverage (100%). Furthermore, this approach has very low computational complexity. Actually, it is not difficult to prove that its complexity is $O(L^2)$, where L is the size of a test suite in terms of the total number of inputs in the test suite. We also proposed the "Order Coverage" based on the structural analysis approach for large complex specification machines. Other related work can be found in [MiPa 92, LoSh 92] where they aimed at generating test suites to achieve full fault coverage (or maximal fault coverage as they called) rather than the evaluation of fault coverage of a given test suite.

We also note that the structural analysis approach has been developed under certain assumptions (Assumptions 1-4 as introduced in Section 2) which are the most relaxed ones compared with other work based on the FSM model [SiLe 89, DaSa 88 etc.]. In particular, we have not assumed the specification machine to be completely specified. Therefore, our approach can be applied to partially specification machines. We believe that this is very important for its practical applications since the real-world systems, such as protocol machines, are normally partially specified.

References

[Chow 78] T.S. Chow, “Test Design Modeled by Finite-State Machines”, IEEE Trans. SE-4, 3, 1978, pp. 178-187.

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

[DDB 91] M. Dubuc, R. Dssouli and G.V. Bochmann, “TESTL: A Tool for Incremental Test Suite Design Based on Finite State Model”, 4th International Workshop on Protocol Test Systems, Holland, November 1991.

[Gill 62] A. Gill, “Introduction to the Theory of Finite-State Machines” McGraw-Hill Book Company Inc., 1962, pp. 207.

[Koha 78] Z. Kohavi, “Switching and Finite Automata Theory”, New York, McGraw-Hill, 1978, pp. 658.

[LoSh 92] F. Lombardi and Y.N. Shen, "Evaluation and Improvement of Fault Coverage of Conformance Testing by UIO Sequences", IEEE Trans. Commun., Vol. COM-40, 8, August, 1992, pp. 1288-1293.

[MCS 93] H. Motteler, A. Chung and D. Sidhu, "Fault Coverage of UIO-based Methods for Protocol Testing", Proc. IWPTS, Pau, France, 28-30 September, 1993, pp. 21-33.

[MiPa 92] R.E. Miller and S. Paul, "Structural Analysis of a Protocol Specification and Generation of a Maximal Fault Coverage Conformance Test Sequence", submitted for publication.

[Moor 56] E.F. Moore, “Gedanken-Experiments on Sequential Machines”, Automata Studies, Princeton University Press, Princeton, New Jersey, 1956.

[PBD 93] A. Petrenko, G.v. Bochmann and R. Dssouli, "Conformance Relations and Test Derivation", Proc. IWPTS, Pau, France, 28-30 September, 1993, pp. 157-178.

[Petr 91] A. Petrenko, “Checking Experiments with Protocol Machines”, Proc. of the 4th Int. Workshop on Protocol Test Systems, 1991.

[SiLe 89] D.P. Sidhu and T.K. Leung, “Formal Methods for Protocol Testing: A Detailed Study”, IEEE Trans. SE-15, 4, April 1989, pp. 413-425.

[Ural 91] H. Ural, “Formal Methods for Test Sequence Generation”, Computer Communications, Vol. 15, No. 5, June 1992, pp. 311-325.

[YPB 93a] M. Yao, A. Petrenko and G.v. Bochmann, "Conformance Testing of Protocol Machines without Reset", Department Publication #861, Département d'informatique et de recherche opérationnelle, Université de Montréal, February 1993, 27 p.

[YPB 93b] M. Yao, A. Petrenko and G.v. Bochmann, "Conformance Testing of Protocol Machines without Reset", Proc. of the 13th IFIP Symposium on Protocol Specification, Testing and Verification, Liege, Belgium, May 25-28, 1993, pp. 241-253.

[YPB 94] M. Yao, A. Petrenko and G.v. Bochmann, “Fault Coverage Analysis in Respect to an FSM Specification”, Accepted by IEEE INFOCOM’94 to be held in Toronto, Canada, June 12-16, 1994.