Test selection based on finite state models

S. Fujiwara* , G.v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi
Departement d'informatique et de recherche operationnelle
Universite de Montreal
Montreal, Canada

Abstract

The selection of appropriate test cases is an important issue for conformance testing of protocol implementations as well as in software engineering. A number of methods are known for the selection of a test suite based on the specification of the implementation under test, assumed to be given in the form of a finite state machine. This paper presents a new method which provides a logical link between several of the known methods. Called "partial W method", it has general applicability, full fault detection power, and yields shorter test suites than the W method. The second part of the paper discusses various other issues which have an impact on the selection of a suitable test suite. This includes the consideration of interaction parameters, various test architectures for protocol testing, and the fact that many specifications do not satisfy the assumptions made by most test selection methods, such as complete definition, a correctly implemented reset function, a limited number of states in the implementation, and determinism.

1. Introduction

Methods for the development of test cases have received much attention recently in relation with conformance testing of communication protocols [Rayn 87, Sari 89]. The test cases are intended to determine whether a given protocol implementation satisfies all properties required by the protocol specification. The purpose of a test selection method is to come up with a set of test cases, usually called "test suite", which has the following (conflicting) properties:

(a) The test suite should be relatively short, that is, the number of test cases should be small, and each test case should be fast and easily executable in relation with the implementation under test (IUT).

(b) The test suite should cover, as much as possible, all faults which any implementation may contain.

---

* On leave from NTT Network Systems Development Center, Japan (1989-1990)

Existing test selection methods differ in the kind of compromise which is reached between these two conflicting objectives, and in the amount of formalism which is used to define the method. In the case that a formal specification of the protocol is available, the test selection and fault analysis can be based on this specification [Sari 89, Boch 89m]. It is important to note that many of the here discussed issues also arise in the more general context of software and hardware testing. Most of the aspects discussed in the paper apply in this general framework.

Many test selection methods have been developed for the case that the specification of the system to be tested is given in the form of a finite state machine (FSM). The best known methods are called Transition tour [Nait 81], W-method [Chow 78], Distinguishing Sequence method [Gone 70], and Unique-Input-Output (UIO) method [SaDa 88]. The W-method and the Distinguishing Sequence (DS) method were originally developed in the context of software and hardware, respectively.

The test suites derived by each of the above methods will detect any output error of the implementation, that is, if the implementation follows the FSM specification except for the output produced for certain state transitions, the error will be detected during the execution of the test suite. However, transfer errors, that is, errors in the next state reached by a transition, will not always be found. Nevertheless, the W-method and Distinguishing Sequence method will find all such errors provided that the number of states of the implementation remains within a certain bound.The discussion of the fault coverage of the test methods is therefore based on the fault model of FSM "output" and "transfer" faults.

The DS method uses a two-phase approach, where the tests of the first phase check that each state defined by the specification also exists in the implementation, and the tests of the second phase check all remaining transitions defined by the specification for correct output and transfer in the implementation. This two phase approach is also used by Vuong's UIOv method [Vuon 89], the SW method ("Single transition checking method using W set") [Sato 89a] and by the partial W method (Wp) described in Section 3.

One aspect of checking a transition is to verify that it reaches the specified next state. It is therefore necessary to identify the next state reached by a given transition. In the case of the UIO and DS methods, the reached state is identified by the output obtained in response to a single sequence of inputs, that is, a single sequence allows to discriminate the expected next state from all other states of the specification. This simplifies the testing procedure. In the case of the other methods, certain states require the separate application of several input sequences. This necessitates a return to the tested state after the application of each sequence, except the last one. For this purpose, the W-method assumes that a reset operation has been correctly implemented which allows a safe return to the initial state of the implementation. This approach is also used for the partial W method described in Section 3.

The purpose of this paper is two-fold. First, the so-called Partial W method is introduced in Section 3. It is

a variation of the W method and provides shorter test sequences than the latter. As explained in Section 4, this method is also a binding element which allows the formal comparison of various test selection methods. In fact, the DS and UIOv methods can be considered special cases of the partial W method, while its relation with the traditional W-method, the SW method, the Transition tour, and the UIO method is also straightforward.

The second purpose of the paper is to provide a discussion of various other issues which have a strong impact on usability, effectiveness and fault coverage of test suites. This includes empirical considerations of test coverage, the handling of interaction parameters, architectural considerations for protocol conformance testing, and the standardization of testing methods and test suites [Rayn 87, OSI C]. These issues must be considered in relation with the selection of a testing method. Whereas most of these issues have not been addressed in many papers on FSM-based test case selection methods, Section 5 of this paper tries to put these methods into an overall perspective.

2. Definitions

The purpose of this section is to introduce some notations and concepts related to finite state machines (FSM) which are used in the following sections of this paper.

A deterministic FSM M can be represented by a quintuple (X, E, Y, T, O) where :
X : Set of inputs, written x in the following,
E : Set of states Mi, including a special state Mo called the initial state,
Y : Set of outputs, written y in the following, including the null output (-),
T : Transition function , X x E --> E,
O : Output function, X x E --> Y.

We use the notation "Mi -x/y-> Mj" to indicate that the FSM M in state Mi responds with an output y and makes the transition to the state Mj when the input x is applied. An input (or output) sequence p (or o) is a suite of inputs x (outputs y), which may be the empty sequence (ε). We use "Mi -p-> Mj" to indicate that the FSM M is originally in state Mi and goes to state Mj when an input sequence p is applied. In this notation, only the reached state Mj is relevant, the output sequence is ignored. However, the notation Milp is used to represent the output sequence given as response by M in state Mi, when the input sequence p is applied.

M is completely specified, if from each state of M there exists a transition for each input symbol in X. The machine M is strongly connected, if for each pair of states (Mi, Mj), there exists an input sequence which takes M from Mi to Mj.
The concatenation of two sets V_1 and V_2 of input sequences is a set of input sequences defined as follows:

$V1.V2 =_{def} \{v_1.v_2 \mid v_1 \in V1, v_2 \in V2\}$, where $v_1.v_2$ stands also for the concatenation of the two sequences v_1 and v_2. Let V^n denote n-times concatenation of V ($V^n = V.V^{n-1}$). The notation X[k] is used to define the set $\{\varepsilon\} \cup X \cup X^2 \cup \ldots \cup X^k$, where k denotes a non-negative integer.

Let S and I be two FSMs. (Note: In the following sections S usually represents a protocol specification and I an implementation). We write Si and So for the state i and the initial state of S, respectively. Similarly, Ik and Io represent the state k and the initial state of I, respectively. The following notations and definitions used for the definition and the proof of the W-method [Chow 78], will also be used in the definition of the partial W (Wp) method.

Definition 1:
Given a set V of input sequences, two states Si and Ik are V-equivalent (written as "$Si \approx_V Ik$"), if S in Si and I in Ik respond with identical output sequences to each input sequence in V.

Definition 2:
Two states Si and Ik are equivalent (written as "$Si \approx Ik$"), if they are V-equivalent for any set V.

Definition 3:
Two FSMs S and I are equivalent if their initial states So and Io are equivalent.

An FSM M is minimal if the number of states in M is less than or equal to the number of states for any machine M' which is equivalent to M.

Definition 4:
Let Q be a set of input sequences. Q is a state cover set of S if for each state Si of S, there is an input sequence $pi \in Q$ such that So-pi->Si. For the initial state So, we have So -ε->So. The empty input sequence (ε) belongs to Q. (Note: In many cases, one uses a state cover set that is closed under the operation of "selecting a prefix").

Definition 5:
Let P be a set of input sequences. P is a transition cover set of S if for each transition Si-x/y->Sj, there are sequences p and p.x in P such that So-p->Si and So -p.x->Sj.
The empty sequence (ε) is a member of P. By definition, each transition cover set P contains a subset which is also a state cover set. The set of all partial paths in the testing tree of S, as defined in [Chow 78], is a transition cover set. A procedure for the construction of this set is also given there.

3. The partial W method

In the following we consider the problem of test case selection where the specification of the implementation under test (IUT) is given in the form of a FSM called S. It is also assumed that the IUT can be modelled by a FSM which is called I.

3.1. Review of the W method

For the understanding of the Wp-method, a brief review of the original W-method as described in [Chow 78], is necessary. The W-method involves the selection of two sets of input sequences: The W-set and the P-set. The latter represents a transition cover set of S, defined in the previous section. The former represents a characterization set of S. The set W consists of input sequences that can distinguish between the behaviors of every pair of states in S.

The method makes some assumptions about the specification S and the IUT I. The specification S should be minimal. This is a necessary (and sufficient) condition for the existence of a characterization set W. In order to guarantee the error detection power of the W-method, S and I are assumed to be be completely specified and deterministic. All states in S and I are assumed to be reachable from the initial one. The existence of a reset operation is assumed in S. This operation is also assumed to be correctly implemented in I. The same input set is also assumed for both machines. It is assumed that the number of states in I is bounded by an integer m, which may be larger than the number n of states in S.

The W-method provides a set of test sequences formed by the concatenation of P and the distinguishing set Z (i. e. P.Z), where $Z = (\{\varepsilon\} \cup X \cup X^2 \cup \ldots \cup X^{m-n}).W = X[m-n].W$. The use of the set Z instead of the characterization set W is due to the bound of the number of states in the IUT I, which may be larger than the number of states n in the specification. In the case that m=n, one obtains Z=W and the set of test sequences consists of the concatenation of the sets P and W (i.e. P.W). Each test sequence starts with the initial state, after the application of the reset operation. In this case, to identify a reached state Ik after a transition, all the sequences contained in W are applied to I, separately. The length of the test suite composed by the concatenation of these test sequences is "proportional" to the cardinal of W.

The set of test sequences P.Z detects any output or transfer error in the IUT, as long as the number of states in this implementation is not larger than m. The proof of the error detection power of the W-method is given in [Chow 78].

3.2. Definition of the partial W method (Wp-method)

The assumptions about S and I made for this method are similar to those made for the W-method. In the following we assume that, the number of states in I is bounded by an integer m, which is equal to the number n of states in S (m = n). The more general case of m > n is discussed in Section 3.4.

The main advantage of the Wp-method, over the W-method, is to reduce the length of the test suite. Instead of using the set W to check each reached state Si, only a subset of this set can be used in certain cases. This subset Wi depends on the reached state Si, and is called an identification set for Si.

Definition: A set of input sequences Wi, is an identification set of state Si if and only if for each state Sj in S (with $i \neq j$), there exists an input sequence p of Wi such that Silp $\neq$ Sjlp and no subset of Wi has this property.

The union of identification sets Wi for all states Si is a characterization set .

The Wp-method consists of two phases which have the following purposes:

Phase 1: This phase checks that all the states defined by the specification are identifiable in the implementation, and also checks, for each state Ik, that it can be identified by the smaller set Wk. At the same time, the transitions leading from the initial state to these states are checked for correct output and state transfer.

Phase 2: This phase checks the implementation for all the transitions defined by the specification, which were not checked during the first phase.

More precisely, the Wp-method proceeds as follows. A transition cover set P is determined which includes a state cover set Q. For each state Si of S, an identification set Wi is determined and W is defined as a set of input sequences including at least all sequences of all the Wi (i. e. it could be constructed as the union of the Wi). The set of sets Wi is called $\mathcal{W}$. It is to be noted that different test sequences are obtained depending on the choices made for the sets P, Q and Wi.

The test sequences of Phase 1 consist of the concatenation of Q with W (i.e. Q.W). Each state Si of the specification is checked in the implementation with the W set. If the test is successful, we have $S \approx_{Q.W} I$, and the number of states in the implementation I is equal to the number of states in the specification S. Since the Wi are subsets of W, this phase also verifies that a set Wi is suitable to identify the state Ii in the implementation.

The test sequences of Phases 2 consist of the sequences of P that are not contained in Q, concatenated with

the corresponding Wi, written $R \otimes \mathcal{W}$, where R = P - Q. More precisely,

$$R \otimes \mathcal{W} = \bigcup_{p \in R} \{p\}.Wj,$$

where Wj is the identification set of Sj in $\mathcal{W}$, and Sj is reached by p, i.e.So -p-> Sj.
During this phase, the remaining transitions are checked. Instead of using the W set for identifying the final state of the transitions, only the subset Wj is used.

If the implementation I passes the tests of both phases, it is equivalent to the specification S. A proof of this assertion is given in Appendix.

3.3. An example for the case m=n

Let S be the FSM shown in Figure 1, with X = {a, b, c} and Y = {e, f}. We assume in addition the existence of a reset transition with no output (r/-) leading to the initial state So for every state of S.

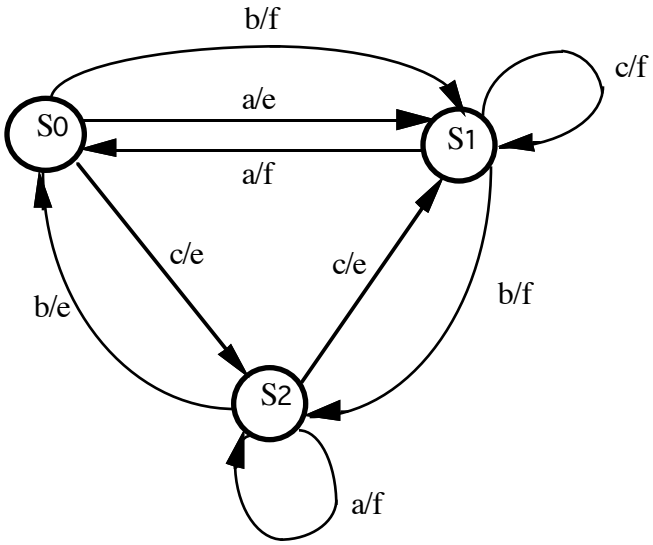

Figure 1: Specification S

The specification has the following characterization set : W = { a, b}. In fact we have:
For state S_o : a/e, b/f
For state S_1 : a/f, b/f
For state S_2 : a/f, b/e

From the above, we get the following identification sets:
W_o = {a}, distinguishes the state S_o from all other states,
W_1 = {a, b}, all the sequences in W are needed to identify the state S_1,
W_2 = {b}, distinguishes the state S_2 from all other states,
$\mathcal{W}$ ={{a}, {a, b}, {b}}.

Q = {ε, b, c } is a state cover set for S.

P = {ε, a, b, b.c, b.a, b.b, c, c.a, c.c, c.b } is a transition cover set for S, which includes Q.

R = P – Q ={ a, b.c, b.a, b.b, c.a, c.c, c.b }.

Based on these sets, the Wp-method yields the following test sequences:

Phase 1: The test sequences for this phase are: Q.W = {a, b, b.a, b.b, c.a, c.b} Phase 2 : The test sequences for this phase are: R ⊗W = {a.a, a.b, b.c.a, b.c.b, b.a.a, b.b.b, c.a.b, c.c.a, c.c.b, c.b.a }

Table 1: Test sequences generated with W ={{a}, {a, b}, {b}}

We note that, for the same sets P and W, the W-method generates the following additional test sequences (with respect to those in Table 1) : b.a.b, b.b.a, c.a.a, c.b.b.

Let us consider the faulty implementation I shown in Figure 2. It contains a transfer fault I_2 -a/f-> I_1 (fat arrow) instead of I_2 -a/f-> I_2 as defined in the specification. The application of the test sequences of the first phase (note that each sequence is prefixed by the reset operation "r"), leads to the following output sequences: e, f, f.f, f.f, e.f, e.e.

These sequences of output are the expected ones (according to the specification S). No faults have been detected during this phase.

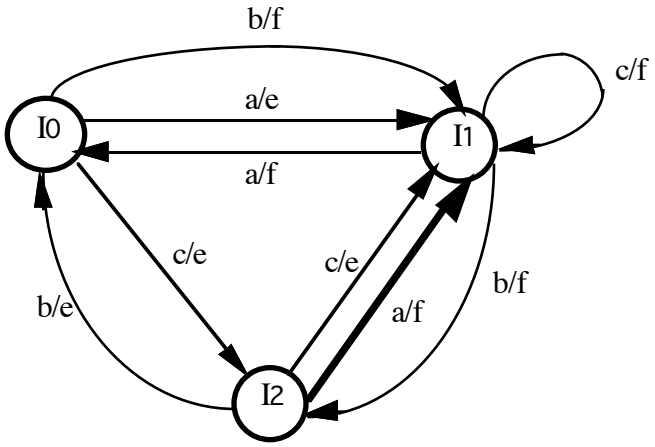

Figure 2: An implementation I

For the second phase, the application of the test sequences leads to the following sequences of outputs : e.f, e.f, f.f.f, f.f.f, f.f.e, f.f.e, e.f.**f**, e.e.f, e.e.f, e.e.e.

The output printed in bold is different from the one expected according to the specification. Therefore, the fault in the implementation I is detected by this test sequence.

We note that another identification set W_1'= {c} can be chosen for state S_1 since the following holds:
For state S_0: c/e.
For state S_1: c/f.
For state S_2: c/e.
Using this identification set instead of the one above, we get $\mathcal{W}$' ={{a}, {c}, {b}}, and may choose as new W set the value W' = {a, b, c}. With the sets P and Q remaining the same, these sets result in the following test sequences:

Phase 1: The test sequences for this phase are: Q.W' = {a, b, c, b.a, b.b, b.c, c.a, c.b, c.c}
Phase 2 : The test sequences for this phase are: R $\otimes\mathcal{W}$' = {a.c, b.c.c, b.a.a, b.b.b, c.a.b, c.c.c, c.b.a }

Table 2: Test sequences generated with $\mathcal{W}$' ={{a}, {c}, {b}}

We note that the number of test sequences for Phase 1 is larger than that for Phase 1 in Table 1, while the number of test sequences for Phase 2 is smaller than that for Phase 2 in Table 1. The issue of minimizing the total test sequence by selecting appropriate sets P, Q and Wi is not addressed in this paper. We also note that the obtained test suites can be further optimized. In the case of Table 1 and Table 2, for instance, the tests of Phase 1 are included in the tests of Phase 2 (e.g. a is included in a.c). Therefore, only the tests of Phase 2 need to be executed.

3.4. Wp-method for implementations with additional states

The generalization of the Wp-method is described in this section for the case where the bound m, for the number of states of the implementation, may be larger than the number n of states of the specification (m>n). The W method handles this general case by using the set Z = X[m-n].W instead of W. The set Z includes W as a subset. The test suite becomes P.X[m-n].W instead of P.W. The key idea in this extension is that Z can distinguish every pair of states in the implementation I [Chow 78]. The set Z is used for checking the reached state after each transition to be tested.

In the Wp-method, we adopt the two-phase approach described in Section 3.2. For the general case of m>n, Phase 1 uses the set Z instead of W (like the W method). In Phase 2, a subset of Z is used depending on the state reached by the transition to be tested. If the transition reaches state Si in the specification, we apply the set Zi (instead of Wi) for checking the reached state in the implementation, where $Z_i = \{p_1.p_2 \mid p_1 \in X[m-n], p_2 \in W_j, S_i\text{-}p_1\text{->}S_j \}$.
Since $Z=\{p_1.p_2 \mid p_1 \in X[m-n], p_2 \in W\}$ and $W_j \subseteq W$, it is clear that Zi is a subset of Z.

More precisely, the Wp-method described in Section 3.2 can be extended to the general case of m>n as follows.

The test sequences of Phase 1 consist of the concatenation of Q with Z (i.e. Q.Z=Q.X[m-n].W). Each state Si of the specification is checked in the implementation with the set Z. If the implementation merges two of states of the specification, this error will be detected by the simple set Q.W for n = m, which is included in the general case. If such merging does not occur then the input sequences in Q will take the implementation I to n different states. Since the number of states in I is bounded by m, the number of states that are not visited by applying Q is (m-n) at most. Then the test sequences Q.X[m-n] will visit all states in I. Therefore the test sequences Q.X[m-n] followed by W means that it will visit all states in I and check if the reached state is W-equivalent to the corresponding state in S. In other words the success of Phase 1 verifies that W-equivalence partitions I into exactly n classes and that Wi is suitable to identify the class with respect to W in the implementation.

The test sequences of Phase 2 consist of the concatenation of R with a subset Zi depending on the reached state. If we follow the definition of partial concatenation ⊗ introduced in Section 3.2, these sequences can be written as R.X[m-n] ⊗ $\mathcal{W}$. More precisely,

$$R.X[m-n] \otimes \mathcal{W} = \bigcup_{p1 \in R} \{p1\} . \left(\bigcup_{p2 \in X[m-n]} \{p2\}.Wj \right)$$

where Wj is the identification set of Sj in $\mathcal{W}$, Sj is reached by p2 from Si (i.e. Si -p2->Sj), and Si is reached by p1 from So (i.e. So -p1-> Si).

(Note that $\bigcup_{p2 \in X[m-n]} \{p2\}.Wj = Zi$).

During this phase the remaining transitions are checked. Instead of using the complete Z set, only the subset Zi is used to check the state Si reached by a sequence of R. In other words, the corresponding Wj is used to check the state Sj which is reached by a sequence of R.X[m-n]. It is important to note that , in Phase 1, each Wi is checked to identify the class in the implementation with respect to W-equivalence. Therefore, Wi-equivalence is sufficient for checking W-equivalence as long as the corresponding Wi is used.

If the implementation I passes the tests of both phases, it is equivalent to the specification S. The proof is given in the Appendix. As described above, the Wp-method for the general case of m>n is extended in a natural way and it can detect any output and transfer faults as long as the number of states remains within the bound m.

3.5. An example for the case m>n

We consider again the specification S of Figure 1. Figure 3 shows a faulty implementation I which contains an extra state I_3. It is obvious that I_3 is W-equivalent to Io for W={a,b}. But I_3 is not equivalent to Io. In fact, a**.**W distinguishes I_3 from Io. The extra state I_3 has the same outputs as Io for all inputs, but differs in the next states.

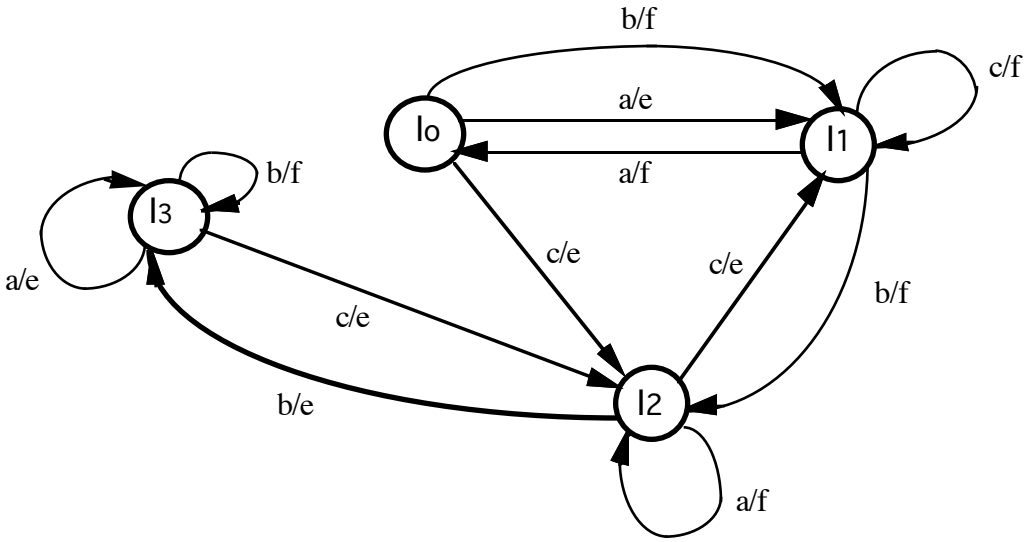

Figure 3 : An implementation I

If we take m=3 and apply the Wp-method, we will get the test sequences shown in Table 1 or 2. It is easy to see that the test suite will not detect any fault in the implementation; the faulty implementation passes the test. The W method with W={a,b} does not detect the fault either. Since I_3 is W-equivalent to Io, the characterization set W can not distinguish I_3 from Io.

Now let us take m=4 and the values for P, Q, Wi, and W as Table 1. Then the Wp-method yields test sequences shown in Table 3. The application of the test sequences of the first phase, leads to the expected outputs according to the specification S. No faults are detected during this phase. It is noted that the application of the sequences contained in {c.b}.W visits the extra state I_3 in the implementation and checks the outputs for W.

For the second phase, the application of the test sequences R $\otimes$ W leads to the expected outputs according to the specification S. No faults are detected up to this point. The application of the test sequences R.X$\otimes$W, however, yields outputs different from the expected ones. For instance, the implementation produces the output sequence e.e.e.**e** in response to the inputs r.c.b.a.a. The output printed in bold is different from the one expected according to the specification. Therefore, the fault is detected by this test sequence. As shown in this example, the Wp-method can detect faults including extra states if the bound m is chosen properly.

Phase 1: The test sequences for this phase are:

$Q.W = W \cup \{b\}.W \cup \{c\}.W$

$= \{a, b, b.a, b.b, c.a, c.b\}$ (same as in Table 1)

$Q.X.W = \{a\}.W \cup \{b\}.W \cup \{c\}.W \cup \{b.a\}.W \cup \{b.b\}.W \cup \{b.c\}.W \cup \{c.a\}.W \cup \{c.b\}.W \cup \{c.c\}.W$

$= \{a.a, a.b, b.a, b.b, c.a, c.b, b.a.a, b.a.b, b.b.a, b.b.b, b.c.a, b.c.b, c.a.a, c.a.b, c.b.a, c.b.b, c.c.a, c.c.b\}$

Phase 2 : The test sequences for this phase are:

$R \otimes W = \{a\}.W_1 \cup \{b.c\}.W_1 \cup \{b.a\}.W_0 \cup \{b.b\}.W_2 \cup \{c.a\}.W_2 \cup \{c.c\}.W_1 \cup \{c.b\}.W_0$

$= \{a.a, a.b, b.c.a, b.c.b, b.a.a, b.b.b, c.a.b, c.c.a, c.c.b, c.b.a\}$ (same as in Table 1)

$R.X \otimes W = \{a.a\}.W_0 \cup \{a.b\}.W_2 \cup \{a.c\}.W_1 \cup \{b.c.a\}.W_0 \cup \{b.c.b\}.W_2 \cup \{b.c.c\}.W_1 \cup \{b.a.a\}.W_1 \cup \{b.a.b\}.W_1 \cup \{b.a.c\}.W_2 \cup \{b.b.a\}.W_2 \cup \{b.b.b\}.W_0 \cup \{b.b.c\}.W_1 \cup \{c.a.a\}.W_2 \cup \{c.a.b\}.W_0 \cup \{c.a.c\}.W_1 \cup \{c.c.a\}.W_0 \cup \{c.c.b\}.W_2 \cup \{c.c.c\}.W_1 \cup \{c.b.a\}.W_1 \cup \{c.b.b\}.W_1 \cup \{c.b.c\}.W_2$

$= \{a.a.a, a.b.b, a.c.a, a.c.b, b.c.a.a, b.c.b.b, b.c.c.a, b.c.c.b, b.a.a.a, b.a.a.b, b.a.b.a, b.a.b.b, b.a.c.b, b.b.a.b, b.b.b.a, b.b.c.a, b.b.c.b, c.a.a.b, c.a.b.a, c.a.c.a, c.a.c.b, c.c.a.a, c.c.b.b, c.c.c.a, c.c.c.b, c.b.a.a, c.b.a.b, c.b.b.a, c.b.b.b, c.b.c.b\}$

Table 3: Test sequences generated with $W = \{\{a\}, \{a, b\}, \{b\}\}$ and m=4

We note that the obtained test suites can be optimized by removing the test sequences which are included in other test sequences. In the case of Table 3, for example, the tests of Phase 1 are included in the tests of Phase 2. Furthermore, the test sequences $R \otimes W$ is included in the test sequences $R.X \otimes W$. Therefore, only the tests $R.X \otimes W$ of Phase 2 need to be executed. The length of test suite varies depending on the choice of the set P, Q, Wi, and W. The issue of minimizing the total test sequences, however, is not addressed in this paper.

4. Comparison of various test selection methods

4.1. Classification of methods by generality

As discussed in Section 3, the W method described by Chow [Chow 78] and the Wp-method defined in Section 3.2 are applicable to any FSM specification satisfying the usual assumptions of minimality, complete specification, and reachability of all states from the initial one. All output and transfer faults of a tested implementation will be detected by the derived test suite, provided that the number of states of the implementation is smaller than a given bound m, which may be larger than the number n of states of the specification. Also the correct implementation of a reset function is assumed. However, in contrast to the testing methods not using this function, it is not necessary for the specification to be strongly connected.

It is clear that the Wp method yields a smaller test suite than the W method. During the first phase, n transitions will be checked using the same approach as the W method, however, in the second phase, the remaining transitions will be checked using the smaller sets Wi or Zi instead of the set W or Z. Clearly, the smaller the sets Wi or Zi, and the shorter the length of the sequences in the Wi or Zi, the shorter will be the resulting total test suite. In the following we consider Wp-method particularly for the case m=n to make comparison with other methods.

If the set Wi for a given state Si contains only a single sequence, this sequence is called a unique input/output (UIO) sequence. For those specifications where a UIO sequence exists for each state, the situation becomes particularly simple. In this situation, the test selection methods called UIO [SaDa 88] and UIOv [Vuon 89] apply and the Wp method becomes equivalent to the UIOv method. The Wp method would use the union of all UIO sequences as the W set. In Phase 1 therefore, the method checks each state with all UIO sequences. This corresponds to Vuong's (Uv) and (~Uv) phases. Phase 2 of the Wp method corresponds directly to the Transition testing (Tt) phase of the UIOv method. It is noted that the UIOv method can be also extended to the case m >n in the same way as the Wp-method.

For certain specifications, the situation can be simplified even further. If the same sequence can serve as UIO sequence for all states of the specification, it is called a distinguishing sequence (DS). The so-called DS test selection method [Gone 70] can be used in this situation. The Wp method becomes particularly simple as well. W and all Wi consist of a single sequence, the DS. This sequence is appended to all sequences in the transition cover set P (thus covering Phase 1 and Phase 2). We note that the Wp method uses resets, while the original DS method [Gone 70] does not require such a function.

We can therefore conclude that the Wp test selection method is a generalization of the UIOv method, and that in the case where a distinguishing sequence exists, the Wp and UIOv methods reduce to a method which is closely related to the DS method, although the latter does not require the reset function. Among

these methods, the W and Wp methods are the only ones of general applicability, in the sense that a characterization set W and identification sets Wi always exist for any minimal FSM specification. However, the UIO and DS sequences do not always exist, even for the FSM specifications satisfying the assumptions of minimality, complete specification, and strong connection.

4.2. Partial target state identification

The test methods discussed above systematically identify the target state of each transition which allows the detection of all transfer faults. Certain test selection methods do not systematically identify the target states of the tested transitions. This leads to shorter test suites, but also has the effect that no guarantee can be given that transfer faults of the implementation will be detected. Any output error of the tested transitions will, however, be detected (except for the ST method mentioned below).

The extreme example is the transition tour (TT) method [Nait 81] which executes all transitions of the specification at least once, but does not make any effort to identify the target states.

The UIO method [SaDa 88] applies the UIO sequence to the target state of each transition, however, there is no guarantee that the UIO sequences have the identification power in the case of a faulty implementation [Vuon 89]. This is the reason why the Wp method uses the complete W set during Phase 1, and not only the corresponding Wi, as in Phase 2.

A test selection method with even lower fault detection power than the TT method is a "modified T method" [Sato 89b], in the following called state tour (ST) method. This method covers all states (but not necessarily all transitions) and does not identify the states reached.

4.3. Use of the reset function

Up to this point, we have mainly discussed test selection methods which assume the presence of a correctly implemented reset function, which brings the implementation, as well as the specification, back into the initial state. This reset is performed at the beginning of each test sequence included in the test suite.

Certain test selection methods make no use of a reset function, but simply concatenate the different test sequences, sometimes by inserting so-called transfer sequences if the final state of the preceding test sequence is different from the initial state of the subsequent one. (Note that, in principle, it is not necessary to start all tests in the initial state). For instance, the transition tour simply tries to concatenate all transitions into a single sequence (tour). The UIO method has also been used without resets which allows certain optimizations [Aho 88, Shen 89].

The W and Wp methods require the reset because during Phase 1 each tested state must be reached several times for applying the different sequences in W. The reset function and the determinicy of the implementation insure that each time the same input sequence is applied, the same state of the implementation is reached. Only in the case that the W set contains only a single element, the reset function is not really required. However, if the W set contains a single sequence, this sequence is a DS. These considerations make the existence of the DS method [Gone 70] without reset plausible. In fact, Phase 1 of the Wp method corresponds to first phase of the DS methods, called "state identification" phase, and Phase 2 corresponds to the second "transition checking" phase of the DS method.

The SW method [Sato 89a] also consists of these two phases. Since this method uses a W set to identify the states, it is necessary, in general, to return several times to the same state for identification. Instead of using resets, the SW method uses transfer sequences for this purpose. No proof is given that all transfer faults will be detected by this method. If the SW method actually detects all errors, it is clear that the same would be true for the Wp method used without resets. The first phase could be identical to the first phase of the SW method, while the second phase of the Wp method without reset would be shorter since the Wi would be used instead of the complete W set.

5. Issues related to test selection

5.1. Test suite length and coverage

For the practical application of the test selection methods the following questions are of prime importance:
(a) What is the length of the resulting test suite? - Or more precisely, what is the cost of executing the test suite? where the cost may involve more factors than simply the number of inputs in the test suite.

(b) What is the fault coverage, that is, what is the confidence that any possible fault of the implementation is detected by the execution of the test suite?

It is clear that, in general, the elimination of a test from a test suite reduces its coverage, and, inversely, the addition of a test will increase the coverage if a suitable test was selected.

The nature of the different test methods, as discussed in Section 4, implies certain relations between the length of the resulting test suite, as shown in Figure 4. The figure also shows the relation for the theoretical fault coverage, based on a model of output and transfer faults and the assumption of a limited number of states in the implementation, as discussed above. The coverage is complete for the W, SW (as far as we know, the proof of this claim is not given), Wp, UIOv, and DS methods, any output fault is also detected by the UIO and TT methods, while the ST method does not even check all transitions. The W and Wp method guarantees complete coverage even for the case where the number of states of implementation

may be larger than that of the specification by fixed bound. Formulas for the upper bound for the length of test suites are given in [Sari 84].

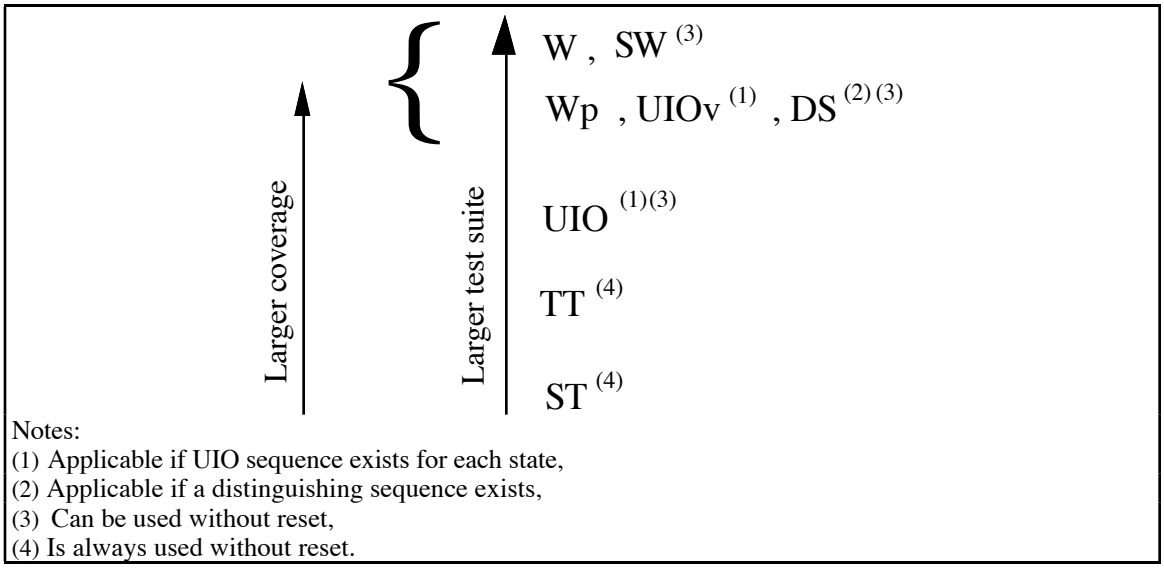

Figure 4: Test suite length and coverage relation between various methods

Since the above theoretical coverage measures are based on certain assumptions, as discussed below, it is useful to obtain some empirical results about the practical fault coverage of these different test selection methods. The experiments described in [Sidh 89] confirm the coverage relations of Figure 4 also for partially defined specifications.

As shown in the example of Section 3.3, the initially obtained test suite can often be optimized [Sidh 89], since certain tests are included in others and need not be executed. In the case of the Wp method, as mentioned in Section 3, the length of the test suite depends on the sets P, Q, and Wi on which the test selection is based. The example of Section 3.3 is interesting in this respect. It shows that a smaller Wi may give rise to a larger W, thus increasing the tests of Phase 1, but decreasing those of Phase 2.

For those methods that do not use the reset function, further optimization can be performed by concatenating the individual tests of the suite in such an order as to minimize the required transfer sequences (see for instance [Aho 88, Shen 89]). Empirical comparisons of test suite lengths are also given in [Sari 82, Sari 84, Sidh 89].

5.2. Justification of theoretical assumptions

The proof of error detection of the test selection methods is based on certain assumptions, which are not necessarily satisfied in practice. The most important assumptions are the following.

5.2.1. Limited number of states in the implementation

For the detection of transfer errors, it is usually assumed that the number of states of the implementation is not larger than the number n of states of the specification. In the case of the W-method and Wp-method, this bound can be increased by a small integer, however, introducing at the same time an exponential growth of the length of the test sequence. Therefore the limit remains for most practical applications equal to n. In the case of black box testing, there is no guarantee that the effective number of states of the implementation is smaller or equal to n. In this case, it can be argued that the theoretical error detection power of the test methods is of very limited value.

5.2.2. Resets

The test methods using resets assume that a reset is performed correctly by the IUT after (or before) each test case. No prescription for testing the correct implementation of the reset function is provided. Although a computer system usually has a reset function in the form of a cold start, this procedure is rarely used between individual test cases. Often, the protocol entity under test (within the system under test) is not even reset, but in order to facilitate the testing procedure, the communication connection used during the last test case is simply eliminated by the execution of the "disconnect" function. This function, however, is usually part of the protocol to be tested, and its correct implementation should not be assumed.

An argument against the use of the reset function is as follows: Given that many errors may only be exhibited in relation with additional states of the implementation which may only be reached after relatively long input sequences (as discussed in 5.2.1 above), the use of resets will prevent the encounter of many of these errors, since the length of the input sequences (between consecutive resets) is not long enough to reach the erroneous implementation states. It can therefore be argued that the UIO method (without resets) is better than, say, the UIOv method, although the former does not provide the error detection guarantee provided by the latter.

5.2.3. Incompleteness and special input/output interactions

The UIO, UIOv and DS methods require the existence of the UIO or D sequences, respectively. They are therefore not applicable in all cases. The (partial) W method is always applicable, however, the size of the W set has a strong impact on the length of the test suite. These problems increase in the case of a specification with an incomplete definition of the behavior. It is quite common to encounter FSM specifications which include "don't care" entries for certain inputs in certain states. In the case of communication protocols, for instance, the behavior of the protocol entity in response to invalid inputs from the user is usually not defined. In the case of such specifications, even the existence of a W set is not guaranteed any more [Sari 82]. Some authors propose to complete an incompletely defined specification,

for instance, by introducing error transitions.

All these problems can be avoided if the specification includes a so-called "read-state" interaction which provides as output an identification of the state of the tested system. This single interaction represents a DS (and can therefore also be used as universal UIO sequence).

The test suite can be further shortened if the specification contains so-called "set-state" interactions (one for each state of the specification) which can be applied in any state and have the effect of transferring the tested system into the state indicated by the interaction. A single interaction of this kind can be used as transfer sequence into a new state.

5.3. Architectural issues

For the above sections of this paper, it was implicitly assumed that all interactions of the IUT are directly visible to the tester. However, this is not always true in the case of protocol testing, as shown in Figure 5 which shows the distributed test architecture [Rayn 87, OSI C] for OSI conformance testing. In this architecture, the upper and lower testers see only the interactions taking place on the upper and lower interface of the protocol entity, respectively.

The synchronization between the upper and lower testers is in general a problem. If certain conditions are satisfied by the test cases [Sari 84], the synchronization can be maintained simply through the interactions with the protocol entity. Such test cases are called self-synchronizing. If a separate communication channel is available between the upper and lower testers, the synchronization problem can be partly solved by a test coordination protocol [Zeng 86]. The problem disappears if the testing is limited to the observation of the lower interaction point, which is the case for the so-called remote testing architecture [OSI C].

As explained in [Sari 84], special precautions can be taken during the test case development, using any one of the methods discussed in Section 4, in order to obtain self-synchronizing test cases. However, a specification may contain a transition for which no self-synchronizing test exists. A similar problem is the testing of situations where two inputs from the upper and lower interface collide in the protocol entity. The delays in the implemented interfaces, and in particular the communication delay for accessing the lower interface in the distributed architecture of Figure 5, make it difficult to synchronize the two inputs.

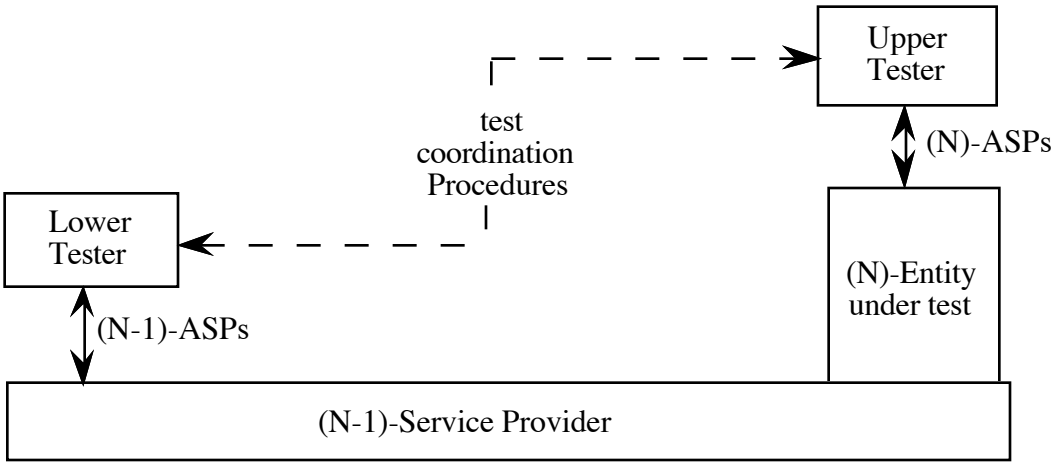

Figure 5: The distributed test architecture

In the area of OSI conformance testing, there is a tendency of specifying so-called generic test cases which are formulated independently of the testing architecture. They must later be adapted to a particular architecture. The issues discussed above are of capital importance in this context.

5.4. Considerations of interaction parameters

The methods discussed above are based on a model of pure FSM's. Often this specification model is extended by the consideration of parameters of the input and output interactions. Each parameter of an interaction is of a particular data type, and may assume values consistent with this data type. Since not all parameter values can be tested, a fault model based on data flow has been proposed [Sari 87, Ural 87] similar to what is used in software testing.

An example is shown in Figure 6, which shows the specification of Figure 1 enhanced with a local variable Y and parameters x for the input interactions b and c, and the output interactions e and f. The transitions t1 and t5 are associated with the assignment of the input parameter to the variable; they are called defining transitions. The transitions t2 and t7 use the variable by assigning its value to the output parameter; they are called usage transitions. For the other transitions, it is assumed that the input and output parameters have no significance.

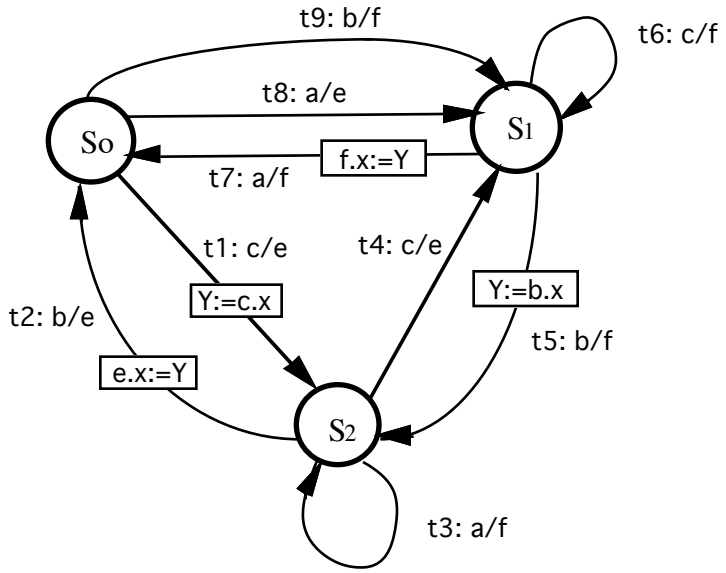

Figure 6: Specification S including data flow

A simple fault model assumes a fault in one of the defining or usage transitions. In order to detect such a fault, it is necessary to execute first a defining transition and then a usage transition, possibly connected through some transfer sequence. The whole sequence is called a data flow path (DFP) [Ural 87]. In the case of the example above, we identify the following four DFP's:

defining transition	usage transition	DFP	input sequence
t1	t2	t1.t2	c.b
t1	t7	t1.t4.t7	c.c.a
t5	t2	t5.t2	b.b
t5	t7	t5.t4.t7	b.c.a

Table 4: Data flow paths corresponding to Figure 6

In order to test each defining and each usage transition, it is sufficient to check either the first and last DFP of Table 4, or the second and third. A more powerful test method would check all DFP's. To check a given DFP, this DFP is usually executed several times with different values for the relevant input parameter(s) (see also [Rayn 87]).

In order to execute the DFP's, these sequences must be embedded into one or several test sequences, possibly separated by resets. In the case of our example, for instance, the first and second DFP can be found directly in Table 1 (input sequences c.b and c.c.a, respectively). The third and fourth DFP require a so-called preambule, since they do not start in the initial state. The input sequence b.b.b (with the preambule b) corresponds to the third DFP, however, no test sequence in Table 1 is suitable for the fourth DFP. An additional test sequence should be added, for instance b.b.c.a.

The above discussion shows that the optimization of a test suite including parameter tests is a complex task. The total length of the test suite is not the only criteria. Other criteria are the functional decomposition of the test suite [Sari 87], and the separation of FSM and parameter tests. It is often assumed that the FSM test sequences provide a good basis for testing the data flow aspects. However, the above example shows that additional test sequence may have to be added. It is also shown in [Amal 89] that the FSM subtours generated for FSM testing (see e.g. [Sari 87]) are not necessarily optimal for the testing of data flow aspects.

5.5. The case of non-deterministic implementations and/or specifications

So far we have assumed that the specification on which the selection of the test suite is based, and the implementation under test are deterministic machines. In this subsection, we briefly discuss the implications of non-determinicy.

In the case that the implementation is non-deterministic, it is impossible to have any guarantee for error detection. Consider, for example, that the implementation of Figure 2 contains an additional transition from state I_1 to state I_0 under input c. The state I_1 of the implementation would have a non-deterministic behavior for input c, since two transitions would be possible. For any given test suite, there is no guarantee that this additional transition will be detected, since the implementation may choose not to execute it during the test.

In the case that the specification is non-deterministic, the tests are not necessarily repeatable, that is, for a given sequence of input, there may be different resulting output sequences depending on the internal choices of the specification/implementation.

In this case, it is also not possible to assure a complete coverage of all parts of the specification/implementation. For example, Figure 7 shows a specification which is non-deterministic in state S_1 under input b. Depending whether the state S_2 or S_3 is reached, two different behaviors are possible. Although the test output will indicate which behavior was tested, it is not possible to force the testing of state S_3 after the behavior of state S_2 was already tested; the implementation may always choose to enter state S_2. In OSI conformance testing [OSI C], the verdict "inconclusive" is used to indicate that the implementation under test has chosen some allowed behavior which, however, does not correspond to what is to be checked by the test case in question.

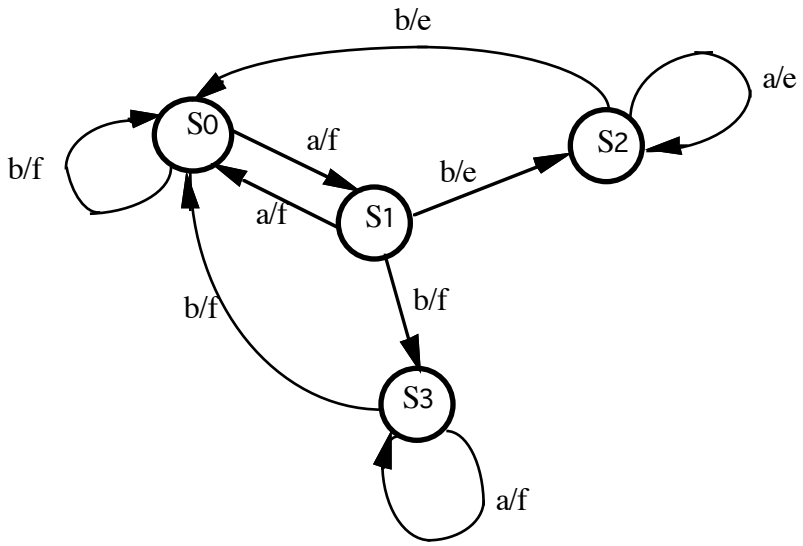

Figure 7: Non-deterministic specification

Systematic test case selection methods for non-deterministic specifications are not yet well developed. Some interesting approaches have been developed in relation with the LOTOS specification language [Weze 89].

5.6. The practice of OSI protocol conformance testing

While the test selection methods discussed in this paper usually result in a more or less lengthy test sequence which covers all aspects of the FSM specification, the test suites developed for OSI conformance testing usually consist of a more or less large number of separate test cases. Each test case verifies a particular aspect of the protocol specification, called the test purpose. In many cases, the test purpose is as simple as the verification of a single FSM transition. Using the terminology of this paper, such an OSI test case would consist of a transfer sequence leading the IUT from the initial state into the starting state of the transition to be tested, followed by the input triggering the transition in question, and possibly terminated by the UIO sequence of the final state of the transition. This may be followed by a reset to the initial state. A test selection tool for the generation of such transition test cases is described in [Burg 89] .
A problem related to OSI conformance testing is the validation of the (often voluminous) specifications of standardized test cases for a given protocol. An automatic validation of these test cases and their verdicts for different responses from the IUT can be obtained by referring to a formal specification of the protocol [Boch 89j].

In addition, it would be useful to have a test selection tool which inputs a given set of (standardized) test cases, determines the fault coverage of the set, and possibly generates additional tests to cover those aspects of the specification which were not originally covered.

Many OSI protocols allow for a large number of implementation options. Therefore the tests executed during OSI conformance testing must be adapted to the options realized by the implementation. The (standardized) suite of OSI test cases for a given protocol usually contains separate test cases for each of the possible options. For the testing of each protocol implementation, the selection, from the test suite, of test cases to be executed is based on the so-called "protocol implementation conformance statement" (PICS) which states the supported options. For certain protocols this selection process, called "test selection" in the OSI context, is very complex and justifies its automation.

6. Conclusion

A unified view of various test selection methods for finite state machines is presented in this paper, based on a new method called "partial W" (Wp) method. As discussed in Section 4, this method provides a logical link between several FSM test methods described in the literature. It has the general applicability and fault detection power of the W method, but yields shorter test suites. In the case that the specification allows for unique input/output (UIO) sequences, it reduces to Vuong's UIOv method. Finally, in the case that a distinguishing sequence (DS) exists, it resembles the DS method, although the latter uses no resets.

These methods detect all transition output and transfer faults in an implementation, however, their applicability and fault detection power relies on a number of assumptions which are not always satisfied. Other methods, such as a UIO method without resets or the transition tour method, yield shorter test suites, but provide no guarantee for detecting all transfer errors.

It is in practice difficult to decide which of these methods is the most interesting to use. In fact, many other issues have an impact on the selection of a test suite. As discussed in Section 5, this includes the problem of synchronization between different points of observation in the case of protocol testing, the consideration of interaction parameters, and sometimes non-determinism. The use of different test methods during the different phases of the implementation development cycle can be envisioned [Sato 89b].

In the case of OSI conformance testing of protocol implementations, there are additional practical issues related to the adaptation of the test cases to the test architecture, the adaptation of the test suite to the implementation characteristics and implemented options, and the validation of the verdicts included in the lengthy descriptions of standardized test cases. In this context, various tools have been developed for developing test cases and test suites, for executing protocol conformance tests, and for analysing the results observed during the execution of test cases. It would also be useful to have tools which could analyse a test suite and determine its fault coverage. Such a tool should also allow the selection of additional test cases for checking particular aspects of the specified behavior. Finally, such a tool should also provide some diagnostic testing facility which would locate any detected error and pinpoint the fault in

the tested implementation. Further research is required in this area.

Acknowledgments

The authors would like to thank Q. Gao, C. Wu, and P. Mondain-Monval, for participating in the discussions that led to the preparation of this paper. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada and the Ministry of Education of Québec.

References

[Aho 88] A.V. Aho, A.T. Dahbura et al., "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours", Proc. IFIP Symposium on Protocol Specification, Testing and Verification, Atlantic City, 1988.

[Amal 89] M. Amalou, "Developpement de test pour le protocole de signalisation de RNIS basé sur une spécification formelle", M.Sc. Thesis, Université de Montréal, Dec. 1989.

[Boch 89j] G. v. Bochmann and O. B. Bellal, "Test Result Analysis in Respect to Formal Specification", in the 2-nd International Workshop on Protocol Test Systems, Berlin, Germany, Oct. 3-6, 1989.

[Boch 89m] G.v. Bochmann, R. Dssouli and J. R. Zhao, "Trace analysis for conformance and arbitration testing", IEEE Trans. on S. E., Nov. 1989.

[Burg 89] S. P. van de Burgt, J. Kroon, E. Kwast and H. J. Wilts, "The RNL Conformance Kit : Tool for automatic test suite generation", in the 2-nd International Workshop on Protocol Test Systems, Berlin, Germany, Oct. 3-6, 1989.

[Chow 78] T.S. Chow, "Testing Design Modelled by Finite-State Machines", IEEE Trans. S.E. 4, 3, 1978.

[Gone 70] G. Gonenc, “A method for the design of fault detection experiments“, IEEE Trans. Computer, Vol. C-19, pp. 551-558, June 1970.

[Nait 81] S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition-

Tours", Proc. of FTCS (Fault Tolerant Computing Systems), pp.238-243, 1981.

[OSI C] ISO TC97/SC16, DP 9646/1 and DP 9646/2: OSI Conformance methodology and framework, Part 1: General Concepts, Part 2: Abstract Test Suite Specification, 1987.

[Rayn 87] D. Rayner, "Standardizing Conformance Testing for OSI", Computer Networks and ISDN Systems, Vol. 14, No. 1, 1987.

[SaDa 88] K.K. Sabnani and A.T. Dahbura, "A protocol Testing Procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, 1988.

[Sari 82] B. Sarikaya and G. v. Bochmann, "Some experience with test sequence generation for protocols", Proc. 2-nd International Workshop on Protocol Specification, Testing and Verification, North-Holland, pp. 555-567, 1982.

[Sari 84] B. Sarikaya and G.v. Bochmann, "Synchronization and Specification Issues in Protocol Testing", IEEE Trans, on Comm., COM-32, No.4, April 1984, pp.389-395.

[Sari 87] B. Sarikaya, G.v. Bochmann and E. Cerny, "A Test Design Methodology for Protocol Testing", IEEE Trans. on SE, April 1987, pp.518-531.

[Sari 89] B. Sarikaya, "Conformance Testing: Architecture and Test Sequences", Computer Networks and ISDN Systems 17, pp. 111-126, 1989.

[Sato 89a] F. Sato, J. Munemori, T. Ideguchi and T. Mizuno, " Test sequence generation method based on Finite Automata - Single Transition Checking Method Using W Set", Trans. of EIC (in Japanese), Vol. J72-B-I, No. 3, pp. 183 - 192, 1989.

[Sato 89b] F.Sato, K.Katsuyama and T.Mizuno, "TENT: Test Sequence Generation Tool for Communication System", The 2-nd International Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols, FORTE'89, Dec. 5-8, 1989.

[Shen 89] Y. N. Shen, F. Lambardi and A. T. Dahbura, "Protocol Conformance Testing Using Multiples UIO Sequences", in E. Brinksma, G. Scollo, C. A. Vissers (eds), Protocol Specification, Testing and Verification, IX, (North-holland, Amsterdam, 1989).

[Sidh 89] D. P. Sidhu and T.K. Leung, "Formals Methods for Protocols Testing: A Detailed Stud", IEEE Trans. On S. E. , vol. 15. No. 4 , 1989.

[Ural 87] H. Ural, "A Test Derivation Method for Protocol Conformance Testing", Proc. of the 7th IFIP Symposium on Protocol Specifiction, Testing and Verification, Zurich, May 5-8 1987.

[Vuon 89] S. T. Vuong, W. W. L. Chan and M. R. Ito, "The UIOv-Method for protocol test sequence generation", in the 2-nd International Workshop on Protocol Test Systems, Berlin, Germany, Oct. 3-6, 1989.

[Weze 89] C. D. Wezeman, "The CO-OP method for Compositional Derivation of canonial testers", in E. Brinksma, G. Scollo, C. A. Vissers (eds), Protocol Specification, Testing and Verification, IX, (North-holland, Amsterdam, 1989).

[Zeng 86] H.X. Zeng et al., "New Advances in Ferry Testing Approaches", Computer Networks and ISDN Systems, Vol. 15, No. 1, 1988.

Appendix

In this appendix, we will prove that the partial W-method (Wp-method) has the same power of fault detection as the W-method. In the following, S and I represent two FSMs. S usually represents a protocol specification and I an implementation.

S is assumed to be minimal and having n states. W is a characterization set of S. S and I are assumed to be deterministic, completely specified. The reachability of all states from initial one is assumed in S and I. S and I have the same input set X. It is assumed that the number of states in I is bounded by an integer m. Z is a distinguishing set X[m-n].W where $X[k] = \{\varepsilon\} \cup X \cup X^2 \cup \ldots \cup X^k \quad (k >= 0)$.

Definition A.0

An isomorphism from S to I is a function f which maps states in S to states in I, such that: a) f is one-to-one and onto, b) if Si-x/y-Sj is in S, then f(Si)-x/y->f(Sj) is in I.

Given a set V of input sequences, a relation V-equivalence (see Section 2) is said to be an isomorphism from S to I (written as S isom_V I), if it is a graph of a function which is an isomorphism from S to I. If f is a function, its graph is a relation $\approx_V$, such that Ik $\approx_V$ Si if and only if Ik=f(Si).

The following two lemmas are given in [Chow 78]. The first lemma implies that equivalence between

FSMs can be verified by an isomorphism with respect to V-equivalence. The second gives the necessary and sufficient condition for a Z-equivalence to be an isomorphism.

Lemma A.1 [Chow 78]
S is equivalent to I ($S \approx I$)
$\Leftrightarrow$
V-equivalence is an isomorphism from S to I for some V. ($S \text{ isom}_V I$)

Lemma A.2 [Chow 78]
Z-equivalence is an isomorphism from S to I. ($S \text{ isom}_Z I$)
$\Leftrightarrow$
(1) for every state Si of S, there is a state Ik of I such that Ik is Z-equivalent to Si. In particular, Io is Z-equivalent to So.
(2) if Si-x/y->Sj, then there are states Ik and Il of I such that Ik, Il are Z-equivalent to Si and Sj, respectively, and Ik-x/y->Il.

It is noted that in [Chow 78], I is also assumed to be minimal. In our model, however, we do not assume the minimality for I. In case I is not minimal, we can reduce I into I' which is minimal and equivalent to I and has m' states. Since $m' \leq m$ holds we can still use m as the bound. Therefore the whole proof in this appendix can be applied to I' which is equivalent to I.

The next lemma states that for any state in I there exists a state in S which is W-equivalent if the condition (1) of Lemma A.2 holds. In other words, W-equivalence partitions I into exactly n classes in the same way as S.

Lemma A.3
For every Si in S, there exists Ik in I such that Ik is Z-equivalent to Si.
$\Rightarrow$
For every Il in I, there exists Sj in S such that Sj is W-equivalent to Il.
[proof]
Let IS be the set of states of I such that $IS = \{ Ik \mid \exists Si, Si \approx_Z Ik \}$. S has n states and Z distinguishes every state in S because W is included in Z. Therefore, IS includes at least n states of I, and particularly a state Io such that $So \approx_Z Io$. Let Il be any state in I. If $Il \in IS$ then Il is Z-equivalent to a certain state in S. It means Il is W-equivalent to a certain state in S because $W \subseteq Z$. Now assume $Il \notin IS$. Since IS includes at least n states, and there is at most m states in the implementation, the number of states which are not included in IS is at most m-n. Since all states in I are reachable from the initial one (i. e. Io), there exists a minimal sequence (among other sequences) λ such that Io-λ->Il. First, assume that the length of λ is less or equal to (m-n). Since S is completely specified, it follows that there exists in S, a state Sl, reachable

by λ from So (i. e. So -λ-> Sl). Since Io $\approx_Z$ So (and particularly Io $\approx_{\{\lambda\}.W}$ So), it follows Il $\approx_W$ Sl. Now, if the length of λ is bigger than m-n, then there is at least (m-n+1) states between Io and Il (Note: λ is minimal). Since there is at most m-n states in I which are not in IS, it follows that there exists, at least, a state Ik ∈ IS such that Io -λ1-> Ik and Ik -λ2-> Il with λ1 and λ2 minimal, λ = λ1.λ2 and the length of λ2 is less or equal to m-n. Since Ik ∈ IS, there exists Sk in S such that Ik $\approx_Z$ Sk. S is completely specified, then there exists a state Sl such that Sk -λ2-> Sl. As in the first case, we have Ik $\approx_Z$ Sk, Ik -λ2-> Il, Sk -{λ2}-> Sl, and the length of is less or equal to m-n, it follows that Ik $\approx_{\{\lambda 2\}.W}$ Sk, and finally Il $=_W$ Sl.
[end of proof]

The next lemma states that if the condition (1) of Lemma A.2 holds, then Zi-equivalence is sufficient for Z-equivalence where Zi is defined as follows.
Zi $=_{def}$ {$p_1.p_2$| $p_1 \in X[m\text{-}n]$, $p_2 \in Wj$, Si -p_1->Sj, Wj is an identification set of Sj}

Lemma A.4
Suppose that for every state Si of S, there is a state Ik of I such that Ik is Z-equivalent to Si. Then it follows that Ik of I is Z-equivalent to Si of S if and only if Ik is Zi-equivalent to Si, that is,
Ik $\approx_Z$ Si ⇔ Ik $\approx_{Zi}$ Si
[proof]
(⇒)
Z = X[m-n].W = {$p_1.p_2$| $p_1 \in X[m\text{-}n]$, $p_2 \in W$ }
Zi = {$p_1.p_2$| $p_1 \in X[m\text{-}n]$, $p_2 \in Wj$, Si-p_1->Sj }
Since Wj ⊆ W, it follows that Zi ⊆ Z.
(⇐)
Let Il and Sj be the states reached by applying an input sequence λ to Ik and Si respectively, where λ∈ X[m-n], that is, Ik-λ->Il and Si -λ->Sj. From the definition of Zi, Ik $\approx_{Zi}$ Si means that Il $\approx_{Wj}$ Sj. Now we want to prove that for any λ∈ X[m-n], Il $\approx_W$ Sj holds. Assume that Il $\approx_W$ Sj does not hold. From Lemma A.3, there exists a state Sj' that is W-equivalent to Il. Since Il $\approx_{Wj}$ Sj, it follows that Sj $\approx_{Wj}$ Sj'. But this contradicts the definition of Wj. Therefore Il $\approx_W$ Sj must hold. [end of proof]

By using lemma A.4, we can rewrite Lemma A.2 as follows.

Lemma A.5
Z-equivalence is an isomorphism from S to I. (S isom_Z I)
⇔
(1) for every state Si of S, there is a state Ik of I such that Ik is Z-equivalent to Si. In particular, Io is Z-equivalent to So.

(2') if Si-x/y->Sj, then there are states Ik and Il of I such that Ik is Zi-equivalent to Si and Il is Zj-equivalent to Sj, and Ik-x/y->Il.

In the following, Q is a state cover set of S, and R is a set (P - Q) where P is a transition cover set of S. In the next lemma, it is proved that the conditions (1) and (2') of Lemma A.5 can be satisfied by testing the Q.Z and (R.X[m-n] ⊗ $\mathcal{W}$) equivalence between S and I, where $\mathcal{W}$ is the set of sets Wi. The set of sequences (R.X[m-n] ⊗ $\mathcal{W}$) is defined as follows:

$$R.X[m-n] \otimes \mathcal{W} = \bigcup_{p1 \in R} \{p1\} . \left(\bigcup_{p2 \in X[m-n]} \{p2\}.Wj \right)$$

where Wj is the identification set of Sj in $\mathcal{W}$, Sj is reached by p2 from Si (i.e. Si -p2->Sj), and Si is reached by p1 from So (i.e. So -p1-> Si).

(Note that $\bigcup_{p2 \in X[m-n]} \{p2\}.Wj = Zi$).

Lemma A.6

The conditions (1) and (2') of Lemma A.5 are true if and only if S and I are Q.Z-equivalent and (R.X[m-n] ⊗ $\mathcal{W}$)-equivalent.

[proof]

(⇒)

From Lemma A.5 and A.1, it follows that I ≈ S. Therefore, I and S are V-equivalent for any set V.

(⇐)

a) S is Q.Z-equivalent to I implies condition (1): From the definition of a state cover set Q, for every state Si of S, there exists pi in Q such that So-pi->Si. Since I is completely specified and has the same input set as S, there exists a state Ik in I which is reached by applying pi to Io. Since Io is Q.Z-equivalent to So, Ik is Z-equivalent to Si. By taking ε∈ Q, we have So $\approx_Z$ Io in particular. Therefore, condition (1) is satisfied.

b) S is Q.Z-equivalent and (R.X[m-n] ⊗ $\mathcal{W}$)-equivalent to I implies condition (2'): From the definition of a transition cover set P = Q ∪ R, for each transition Si-x/y->Sj, there are sequences pi and pi.x in (Q ∪ R) such that So-pi->Si and So-pi.x->Sj. Since I is completely specified and has the same input set as S, there exists Ik and Il in I which are reached by applying pi and pi.x to Io, respectively.

In case pi∈ Q, since Io $\approx_{\{pi\}.Z}$ So, Ik is Z-equivalent to Si. It means that Ik is Zi-equivalent to Si. In case pi∈ R, since Io $\approx_{\{pi\}.Zi}$ So, Ik is Zi-equivalent to Si. Therefore Ik is Zi-equivalent to Si in both cases. The same discussion also holds for pi.x, and Il is Zj-equivalent to Sj.

Since Io-pi->Ik and Io-pi.x->Il and I is deterministic, it follows that Ik-x->Il. Furthermore, the output is equal to the output y produced by Si in response to input x. Therefore Ik-x/y->Il holds. [end of proof]

Finally, lemmas A.1, A.5, and A.6 directly lead to the following theorem.

Theorem A.7

S is equivalent to I (i.e. $S \approx I$)

$\Leftrightarrow$

S and I are Q.Z-equivalent and ($R.X[m-n] \otimes W$)-equivalent (i.e. $I \approx_{Q.Z} S \wedge I \approx_{R.X[m-n] \otimes W} S$)

The test sequence sets Q.Z and ($R.X[m-n] \otimes W$) correspond to the Phases 1 and 2, respectively. If both Phases 1 and 2 are successful, the Theorem A.7 guarantees that I is equivalent to S. This means the Wp-method can detect any output and transfer fault as long as the number of states in I is not larger than m.

As shown above, the Wp-method is based on the notion of V-equivalence. It means we need to apply different sequences of V to the same state in I. A means for returning to the same state is the reset operation followed by the corresponding transfer sequence. Therefore the reset operation is assumed to be correctly implemented in the implementation.