TEST SELECTION BASED ON COMMUNICATING NONDETERMINISTIC FINITE STATE MACHINES USING A GENERALIZED Wp-METHOD[1]

Gang Luo, Gregor v. Bochmann, and Alexandre Petrenko[2]

Departement d'IRO, Universite de Montreal,
C.P. 6128, Succ.A, Montreal, P.Q., H3C 3J7, Canada
e-mail:luo@iro.umontreal.ca Fax: (514) 343-5834

ABSTRACT We present a method of generating test sequences for concurrent programs and communication protocols that are modeled as communicating nondeterministic finite state machines (CNFSMs). A conformance relation, called trace-equivalence, is defined within this model, serving as a guide to test generation. A test generation method for a single nondeterministic finite state machine (NFSM) is developed, which is an improved and generalized version of the Wp-method that generates test sequences only for deterministic finite state machines. It is applicable to *both* nondeterministic *and* deterministic finite state machines. When applied to deterministic finite state machines, it yields usually smaller test suites with full fault coverage than the existing methods that also provide full fault coverage, when the numbers of states in implementation NFSMs are bounded by a known integer. For a system of CNFSMs, the test sequences are generated in the following manner: A system of CNFSMs is first reduced into a single NFSM by reachability analysis; then the test sequences are generated from the resulting NFSM using the generalized Wp-method.

INDEX TERMS: Concurrent programs, communications, nondeterministic finite state machines, protocol conformance testing, protocol engineering, SDL, software engineering, and software testing.

---

[1]This work was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at the University of Montreal (Canada)

[2] On leave from the Institute of Electronics and Computer Science, Riga, Latvia.

1. INTRODUCTION

The testing phase represents a large effort within the common software development cycle. In the area of communication software, systematic approaches have been developed for protocol conformance testing [31, 5], and the generation of appropriate test suites [11, 30, 36, 34, 10]. These approaches can produce significant economic benefits [1, 3]. A considerable amount of work has been done in the area of test generation for the software modeled by deterministic finite state machines (DFSMs) [11, 36, 29, 10, 15, 41, 33, 40]. However, relatively little work has been done to generate test sequences for nondeterministic models and concurrent models. There is a practical need for testing nondeterministic models [42]; for instance, some CSMA protocols for local area networks are nondeterministic [38]. *Nondeterminism* and *concurrency* are two important features of formal specification languages for communication software, in particular, communication protocols. All the three major specification languages for communication software, LOTOS [7], ESTELLE [9] and SDL [4] support the description of nondeterminism and concurrency (SDL will support nondeterminism in the near future [35]).

In order to generate test sequences for the control portion of software written in SDL or ESTELLE, usually, one first abstracts, by neglecting interaction parameters, a given specification into a system of state machines that communicate with each other over input queues and channels as described in SDL and ESTELLE. Test sequences are then developed from the resulting machines.

Some work on test generation for nondeterministic models has been done in the context of basic LOTOS [30, 8] and finite labeled transition systems [12, 13]. However, these approaches are not applicable to testing nondeterministic finite state machines (NFSM) where every transition is associated with an input and an output. Several methods of generating test sequences for NFSMs have been given in [23, 39, 20] and they are all based on the generalization of unique I/O sequences [33]. However, these methods are not guided by any pre-defined conformance relation and have limited fault detection power. Furthermore, even when the methods are applied to DFSMs, a specific class of NFSMs, they still cannot guarantee the full fault coverage that the W- and Wp-methods can provide; the reason is the same as pointed out in [41]. To

deal with concurrency, some heuristic approaches for concurrent program testing, based on extended finite state machines [24, 19, 2], have been reported, but they did not address the issue of nondeterminism.

We present in this paper a test generation method for a single NFSM and apply it to test a system of CNFSMs (Communicating NFSMs).

In Section 2 we first define NFSMs and related notations that are similar to those for labeled transition systems [8, 12]. Then, we define a conformance relation between specifications and their implementations, called trace-equivalence, for NFSMs. This relation is presented in the context of the black-box testing strategy under which implementations are viewed as black boxes. We finally present a fault model for NFSMs and several definitions related to testing.

Guided by the given conformance relation, we present, in Section 3, a method for generating test sequences from NFSMs. Our method is based on extending the state identification approach used for DFSMs [10, 33, 28, 11] to NFSMs. We first transform an NFSM to an equivalent one that belongs to a specific class of NFSMs, called observable NFSM (ONFSM). An ONFSM has a property that a state and an input/output pair uniquely determine the next state, while a state and an input alone do not necessarily determine a unique next state and an output. Test sequences are then generated from the resulting ONFSMs by a method generalized from the Wp-method [11]. As an example, we finally apply the method to generate a test suite for a communication protocol, called *Inres* [16], within the ISO remote testing architecture.

In the Section 4 we consider a system of several CNFSMs. We first illustrate that such a system, in general, cannot be modeled as a deterministic *finite* state machine with the same input/output behavior, and that even a system of communicating DFSMs may still behave in a *nondeterministic* manner. We then use two heuristic approaches to derive a reduced state machine from a system of CNFSMs, one of which is based on a restriction on the channel and queue lengths and the other on an assumption about the speed at which inputs are sent by the environment. The test generation method for a single NFSM is used to derive test sequences from the resulting machine.

We conclude by discussing the application of the method to generate test sequences from SDL specifications.

2. NOTATIONS AND ABSTRACT TESTING FRAMEWORK

We first give in this section the definition of NFSMs, the several typical classes of them, and additional notations related to them. We then present a conformance relation for NFSMs, on the basis of the black-box testing strategy under which implementations are viewed as black boxes. Finally, we present a fault model for NFSMs and give some definitions related to testing.

2.1 Nondeterministic finite state machine

DEFINITION *Nondeterministic Finite State Machine*.

A Nondeterministic Finite State Machine (NFSM) is a 5-tuple (St, Li, Lo, h, S_0) where:

(1) St is a finite set of states, St=$\{S_0, S_1, ..., S_{n-1}\}$.

(2) Li is a finite set of input

(3) Lo is a finite set of outputs, and it may include a special symbol λ that represents an empty output.

(4) h is a behavior function.

h : St Li --> powerset(St × Lo) \ {∅}

where ∅ denotes the empty set. Let P, Q∈ St, a∈ Li and b∈ Lo. We write P-a/b->Q to denote (Q,b)∈ h(P,a); P-a/b->Q is also called *a transition* from P to Q with label a /b.

(5) S_0 is the initial state which is in St. □

This definition is basically the same as the one given in [37]. An NFSM becomes *deterministic*, denoted as a DFSM, if none of transitions from the same state have the same input. We also assume that a "reliable" reset input *r* is available in any NFSM implementation such that upon receiving *r* in any state the machine returns to the initial state.

According to the above definition, the NFSM is completely defined, thus for every input and every state, the NFSM has at least one transition with this input. During testing, whether an input causes a transition with an empty output may be determined by waiting a certain period of time; if there is no output during that period, the transition invoked by the input is considered to produce an empty output λ. We do not include spontaneous transitions (or called internal actions) in our model, since the NFSMs that allow spontaneous transitions can be modeled by equivalent (i.e., "trace-equivalent" in Section 2.2) NFSMs without spontaneous transitions, using an approach similar to that given in [26]. In some situation, a single transition may be associated with several outputs, for instance, the transitions in SDL; in this case, the several outputs can be modeled by a new single output in NFSMs.

An NFSM can be represented by a directed graph in which the nodes are the states and the directed edges are transitions linking the states. Figure 1 shows an example of an NFSM.

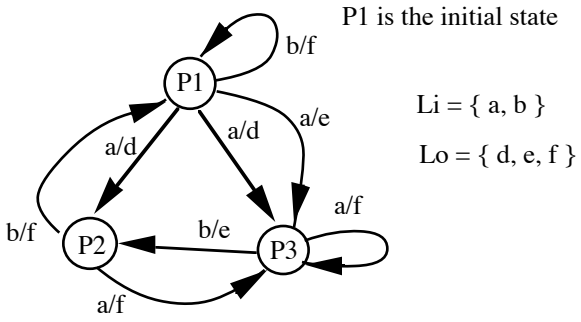


Figure 1. An example of an NFSM

We define in the following the concept of ONFSMs (*Observable Nondeterministic [illegible]nite State Machines*), originally described in [37], a specific class of NFSMs.

DEFINITION *Obser[illegible]ble NFSMs* (ONFSMs).

An NFSM is said [illegible]be *obse[illegible]able* if $\forall S \in St\ \forall u \in L\ (S\text{-}u\text{->}S_i\ \&\ S\text{-}u\text{->}S_j \quad i=j)$ [illegible]

where L= Li Lo. □

Given an ONFSM, for every state S in St, and every input/output pair a/b in L, there is at most one transition from S with label a/b. ONFSMs are a typical class of NFSMs where a state and an input/output pair uniquely determine the next state. However, an ONFSM may still be nondeterministic because given a state and an input, one cannot determine a unique next state and a unique output.

For the convenience of the presentation, we introduce in Table 1 several notations, which are similar to those in the labeled transition systems [8]. We treat an I/O pair as a single label and a sequence of I/O pairs as a sequence of labels, called *trac*[redacted] This provides us with a convenient means to capture the nature of nondeterminism.

Table 1. Notation for NFSMs

notation	meaning
L	Li Lo, a set of input/output pairs; u denotes such a pair
ε	ε is the empty sequence
L*	set of sequen[redacted]otes such a sequence. Note that ε∈ L*
P=ε=>Q	P=Q
P=x=>Q	$\exists P_1, ..., P_{k-1} \in St$ $(P=P_0\text{-}u_1\text{->}P_1 ... \text{-}u_k\text{->}P_k=Q)$ where $u_1,...,u_k \in L$, and $x=u_1...u_k$
P=x=>	∃Q∈ St ([redacted]=x=>Q)
Tr(P)	*Tr*(P)={ x | [redacted]=x=>} (note that *Tr*(P)={ x | P=x=>Q for some Q∈ St })
x^{in}	For x∈ L*, x^{in}[redacted]nce obtained by deleting all outputs in x (note that $x^{in} \in Li^*$)
V^{in}	For V L*[redacted]

Given an NFSM S, we say that S is *initially connected* if every state is reachable from the initial state by a directed path in the machine; i.e., $\forall S_i \in St\ \exists x \in L^*$ $(S_0=x=>S_i)$. Without loss of generality, we assume that all NFSMs considered in the rest of the paper are initially connected. If a given NFSM S is not initially connected, we need to consider only such a sub-machine that is a portion of S consisting of all states and transitions that are reachable from the initial state of S. The unreachable states and transitions of machines do not affect the behavior.

2.2. Conformance relation for NFSMs

In black box testing, the only way to distinguish an implementation from its specification is to check the difference between their output sequences caused by input sequences. We assume in the rest of the paper that the NFSMs of interest have the same Li and Lo if we do not specify them explicitly.

We now discuss a conformance relation for NFSMs. For DFSMs, there is a widely accepted conformance relation, (see, e.g., [11, 10, 40, 37, 14]), which requires that a specification and its implementation produce the same output sequence for every input sequence. A generalized version of this relation was defined in [37], which we call *trace-equivalence*, applicable not only to DFSMs but also to NFSMs. Trace-equivalence requires that *a specification and its implementation produce the same set of possible output sequences for every input sequence*.

DEFINITION *Trace-equivalence*.

The *trace-equivalence* relation between two states P and Q in NFSMs, written

$P =_{trace} Q$, holds iff $Tr(P)$ $Tr(Q)$

Given two NFSMs S and I with their initial states S_0 and I_0, we write $S =_{trace} I$ iff $S_0 =_{trace} I_0$. □

The trace-equivalence relation is an equivalence relation since it is reflective, transitive and symmetric. This relation serves as a guide to test generation for NFSM specifications. For a DFSM, the above relation becomes the ordinary equivalence relation as defined in [21, 10, 11].

It is well-known that any nondeterministic finite automaton ion is associated with a single symbol (no) can be modeled by an e ivalent deterministic automaton [18]. However, nondet ministic finite state machines where each tr sition is associated with an I/O pair cannot be model by trace-equivalent deterministic finite state machi , assuming a∈ Li, in an NFSM with S_0=a/b >, we have {a/b, a/c} Tr e of DFSMs can have {a/b, a/c} Tr

2.3. Fault model for an NFSM

In the area of test generation for DFSMs, a fault model serves as a basis for fault coverage analysis [10, 41, 29, 11]. A general survey on a variety of fault models in testing was given in [6]. We now give a fault model for NFSMs, which is a generalized version of the fault model given in [10]. Let S and I be two NFSMs, with S being a specification and I its implementation. Assume that they have the same Li and Lo. If $S=_{trace}I$, we say that I has no fault with respect to S; and if not($S=_{trace}I$), we define the fault types as follows:

(1) *Single output fault:* We say that a transition in I has a single output fault if there exists S' such that (i) $S=_{trace}S'$, and (ii) S' can be obtained from I by changing the output of this transition.

(2) *Single transfer fault:* We say that a transition in I has a single transfer fault if there exists S' such that (i) $S=_{trace}S'$, and (ii) S' can be obtained from I by changing the ending state of the above transition.

(3) *Single extra (missing) transition fault:* We say that I has a single extra (missing) transition fault if there exists S' such that (i) $S=_{trace}S'$, and (ii) S' can be obtained from I by eliminating (adding) a transition from (to) I.

(4) *Multiple faults:* We say that I has multiple faults if there exists S' such that (i) $S=_{trace}S'$, (ii) S' can be obtained from I by changing the outputs of certain transitions, changing the ending states of certain transitions, and eliminating and/or adding transitions on I, and (iii) the fault type of I is not one of the above (1)-(3).

According to this definition, if $S=_{trace}I$, then I has no fault with respect to S. For NFSMs, if a testing method can check the trace-equivalence between specifications and their implementations, then it can detect any types of faults in this model; such a testing method is said to be able to provide full fault coverage.

We note that the development of our test generation method is only guided by the conformance relation. However, in the area of test generation for state machines, fault models are widely used for evaluating the

goodness of test generation methods in terms of detectable faults [10, 12, 33, 41, 11, 6]. Therefore, we give this fault model to show that our method when applied to DFSMs can detect all types of faults that can be detected by the former methods.

2.4. Definitions related to testing

We define in the following several concepts that are related to testing NFSMs.

Given an NFSM, an inpu[redacted]d to be a *test sequence*; and a finite set of test sequences is a *test su[redacted]*.

DEFINITION: *Concate[redacted]put sequences.*

Assuming V1, V2 L*[redacted]*on of sets*, written ".", is defined as follows:

V1.V2 = { t1.t2 | t1∈V1 & t2∈V2} where t1.t2 is the concatenation of t1 with t2.

We write $V^n = V.V^{n-1}$ for n>1 and $V^1 = V$. □

This notation is required for presenting te[redacted]suites.

DEFINITION : *V-equivalence of sta[redacted]an[redacted]*

Given two states P and Q, and a set V [redacted]

$P =_V Q$ holds iff $\forall x \in L[redacted](x^{in} \in V [redacted] x \notin [redacted](P) \; Tr(Q)[redacted]$

where $Tr(P)$ $Tr(Q)$ [redacted](Q)) \ ($Tr(P) \cap Tr(Q)$)

Given two NFSMs S and I with their initial states S_0 and I_0, we write S[redacted]I iff $S_0 =_V I_0$. □

The V-equivalence relation requires that, for every input sequen[redacted]e in V, the specification and its implementation produce the same set of possible output sequences. We nee[redacted]r analyzing the validity of our test generation method. We have: $P =_{trace} Q$ iff $\forall V$ Li[redacted]

In order to test nondeterministic implementations, one usually makes a so-called *complete-testing assumption*: it is possible, by applying a given input sequence t to a given implementation a finite number of times, to exercise all possible execution paths of the implementation that are traversed by t [12, 13, 23]. Without such an assumption, no test suites can guarantee full fault coverage for nondeterministic implementations. This assumption is similar to the one of so-called "all weather conditions" for nondeterministic systems of Milner [27, page 11]. For testing nondeterministic models, as pointed in [8], the quality of testing increases with the number of repetitions of test sequence application; in actual testing, this number is limited by practical and economical considerations. Ideally, for an implementation and a given input sequence, the probability that not all possible corresponding execution paths are exercised at least once, may be reduced to close to zero by applying the input sequence a sufficiently large number of times.

3. TEST GENERATION FOR A SINGLE NFSM

A method of generating test sequences for a single NFSM is presented in this section. This method is based on state identification approach [11, 10, 33] for DFSMs; however, it has to cope with an additional problem, nondeterminism. In a deterministic model, an input sequence uniquely determines one output sequence, which is not necessarily true in a nondeterministic model. We first present in Section 3.1 a method of generating test sequences for ONFSMs; and in Section 3.2 we then apply this method to generate a test suite for an example, the communication protocol called *Inres*. Finally, we discuss in Section 3.3 the transformation of an NFSM into a trace-equivalent ONFSM; incorporating such transformation, our test generation method can generate test sequences for any NFSM.

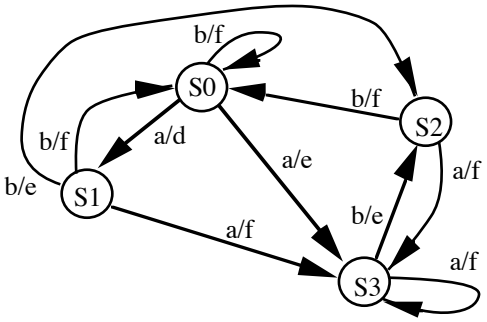
3.1. Test generation

We first introduce several concepts that are needed for presenting our test generation algorithm.

DEFINITION: *Minimality of ONFSMs.*

An ONFSM is *minimal* if none of its states are trace-equivalent. □

For test generation, we first transform general NFSMs into trace-equivalent minimal ONFSMs, then generate test sequences from the resulting minimal ONFSMs. For example, a trace-equivalent minimal ONFSM for the NFSM in Figure 1 is shown in Figure 2.



Li = { a, b
Lo =

Relat
states
S0
S1 {P
S2 {P2}
S3 {P3}

Figure 2. Minimal ONFSM trace-equivalent to the NFSM in Figure 1

DEFINITION: *Prime machine.*

For a given NFSM S (St, Li, Lo, h_S, S_0), the *prime machine* of S is a minimal ONFSM M (St_M, Li, Lo, h_M, M_0) such that $S =_{trace} M$. □

In test generation, the role of prime machines for NFSMs is the same as that of minimal DFSMs for DFSMs. For two given NFSMs S1 and S2, if $S1 =_{trace} S2$, then their prime machines, say M1 and M2, are isomorphic to one another. It is easy to prove that the trace-e n is the isomorphism between M1 and M2. The isomorphism DFSMs and re studied in [10] and [37], respectively.

DEFINITION: *Characterization set*

Given an ONFSM, a *characterizatio*

$\forall S_i, S_j \in St$ ($i \neq j$ $\exists x \in$ (S_i) $Tr(S_j)$

where $Tr(S_i)$ $Tr(S_j)$ $r(S_j)) \setminus (Tr(S_i) \cap Tr(S_j))$. □

According to this definition, if and only if a given ONFSM is minimal, there exists a characterization set for it. The W set is introduced to distinguish states from each other in a minimal ONFSM; when it is applied to different states, different sets of possible output sequences are produced. *Intuitively, given a minimal ONFSM, for any two different states, there exists an input sequence in W such that two different sets of possible output sequences are produced when this input sequence is applied to these states, respectively.* The ONFSM shown in Figure 2 is minimal; and the set {a, b}, for instance, is a characterization set. Although we do not present the algorithm for generating characterization sets, one can develop such an algorithm by borrowing the ideas o[redacted]cterization sets for deterministic machines pointed out in [21, 10].

DEFINITION: *prefix set pref(V)[redacted] of sequences.*

Given a set of sequences V Li*[redacted]

pref(V)={t1 | t2∈Li* & t1.t2∈V & t1≠ε} where t1.t2 is the concatenation of t1 with t2. □

We note that for x≠ε, x∈pref({x}). This notation enables us to present the following definition precisely.

DEFINITION *A tuple of state identification sets* {W0, [redacted]}.

Given an ON[redacted]M and a characterization set W [redacted]{W0, W1, ..., Wn-1} is said to be a tuple of *state identification sets* [redacted]n-1, Wi is a (pref[redacted]ch that

(i) Wi pre[redacted]

(ii) for j=0, 1, ..., n-1, (j≠i ∃x∈[redacted]r(Si) *Tr*(Sj[redacted]). □

According to this definition, if a given ONFSM is minimal, there exists a tuple of state identification sets for it. The motivation of introducing state identification sets is as follows. A state identification set W_i can determine whether an ONFSM is in the given state S_i. The idea of state identification sets intuitively captures the following notion: *Given a state S_i, for any other state S_j, there must exist an input sequence in W_i such that two different sets of possible output sequences are produced when this input sequence is applied to these states, respectively.* In an extreme case, the W set can serve as a state identification set for

any state. As an example, for the ONFSM shown in Figure 2, we have state identification sets $W_0=\{a\}$, $W_1=W_3=\{b\}$, $W_2=\{a, b\}$.

We note: All existing test generation methods for DFSMs that ensure full fault coverage (for examples, the W- and Wp- methods [10, 11]), require that the user previously estimates an *upper bound* on the number of states in the prime machine of the given DFSM implementation. Similarly, our test generation method also requires that the user previously estimates an *upper bound* on the number of states in the prime machine of the given NFSM implementation. In the simplest case, one may assume that this number is equal to the number of states of the specification. Without the assumption of such an upper bound, no method can check the trace-equivalence between specifications and their implementations for DFSMs, as pointed out in [14]; therefore, for NFSMs, this assumption is also necessary. We now present the test generation algorithm as follows.

ALGORITHM 1: Test generation.

Input : A specification S in the form of a minimal ONFSM with n states, and the upper bound m ($n \le m$) on the number of states in the prime machine of the given NFSM implementation.

Output : a test suite $\prod$.

Step 1: Construct a characterization set W, and a tuple of state identification sets

$\{W_0, W_1, ..., W_{n-1}\}$.

Step 2: Construct a (preferably minimal) set $\mathbb{Q}$ = {ε} ∪ Li* such that

$\forall S_i \in St$ [illegible] $\in L^*$ ($x^{in} \in \mathbb{Q}$ & [illegible] $= x => S_i$).

Step 3: Construct two sets $\mathbb{P}$ and $\mathbb{R}$ such that $\mathbb{P}=\mathbb{Q}.(\{\varepsilon\}\cup Li)$ and $\mathbb{R}=\mathbb{P}\backslash\mathbb{Q}$.

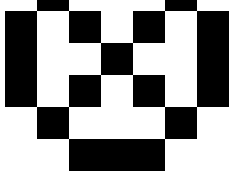
Step 4: First, define operator [illegible] as follows:

for V Li* [illegible] $\{W_0, W_1, ..., W_{n-1}\} = \bigcup_{S_0=x=>Si \ \&\ x^{in}\in V} \{x^{in}\}.W_i$.

Then, construct a test suite $\prod$ in the following manner:

$\prod=\prod 1 \cup \prod 2$

where $\prod 1=\mathbb{Q}.(\{\varepsilon\}\cup Li\cup Li^2\cup...\cup Li^{m-n}).W$,



$\prod 2=\mathbb{R}.Li^{m-n}\ \{W_0, W_1, ..., W_{n-1}\}$. □

Remarks on Algorithm 1:

(1) W_i can be derived from W by checkin the sequences in W one by one, and by eliminating the unnecessary sequences for the state identification set. It is not difficult for one to develop a detailed algorithm for generating W_i.

(2) $\mathbb{Q}$ is a so-called *state cover*. A set $\mathbb{Q}$ Li , for every state S_i, $\mathbb{Q}$ contains an input sequence that may lead the machine from the initial state S_0 to S_i; and $\mathbb{Q}$ is preferably minimal but not necessarily. To avoid cumbersome presentation, we do not give the method of deriving a minimal $\mathbb{Q}$.

Given an ONFSM S, a state cover $\mathbb{Q}$ can be derived as follows:

(i) Construct a tree such that (a) the tree is a subgraph of the machine S, and it contains all nodes of S with the node labeled S_0 being the root; and (b) for every node in the tree, there is a directed path in the tree from the root to this node.

(ii) $\mathbb{Q}$ contains all input sequences each of which is path from the root to a node in the tree (including the root).

We note: For DFSMs, if $\mathbb{Q}$ is derived usin this approach, then it is minimal; however, it is not necessarily minimal for NFSMs.

(3) $\mathbb{P}$ is a so-called *transition cover*. A set $\mathbb{P}$ Li er if, for every transition S_i-u->S_j, $\mathbb{P}$ contains an input sequence $x^{in}.u^{in}$ such that x^{in} and $x^{in}.u^{in}$ may lead the machine from the initial state S_0 to S_i and S_j, respectively.

We note: For DFSMs, if $\mathbb{Q}$ is derived using the approach in (2), then $\mathbb{P}\backslash\{\varepsilon\}$ is a minimal transition cover.

(4) We intuitively explain the validity of Algorithm 1 as follows:

Consider a given specification S in the form of an ONFSM, and any NFSM I; let M be the prime machine of I. Assume that the number of states in M is bounded by m. Assume $S=\prod I$. Then,

(i) the set $\mathbb{Q}.(\{\varepsilon\}\cup Li\cup Li^2\cup...\cup Li^{m-n})$ is a state cover for M; therefore, $\prod 1$ is used to verify that every state in M is W-equivalent to a state in S. In other words, if $S=\prod_1 I$, each state in M is W-equivalent to one and only one state in S. In this sense, checking $S=\prod_1 I$ is a kind of state verification.

(ii) The set $\mathbb{Q}.(\{\varepsilon\}\cup Li\cup Li^2\cup...\cup Li^{m-n})\cup\mathbb{R}.Li^{m-n}$ is a transition cover for M; $\prod 1\cup\prod 2$ is used to verify transitions in M. For every state M_i in M that is W-equivalent to a state in S_j, if $S=_{\prod}I$, then (a) for any input a, the same set of possible outputs is produced when a is applied to M_i and S_j, respectively; (b) if $M_i=a/c=>M_l$ and $S_j=a/c=>S_k$, then M_l is W_k-equivalent to S_k.

In this sense, when $S=_{\prod 1}I$, checking $S=_{\prod 2}I$ is a kind of transition verification.

Note: $\mathbb{P}.(\{\varepsilon\}\cup Li\cup Li^2\cup...\cup Li^{m-n})=\mathbb{Q}.(\{\varepsilon\}\cup Li\cup Li^2\cup...\cup Li^{m-n})\cup\mathbb{R}.Li^{m-n}$.

(iii) The state and transition verifications together check $S=_{trace}M$; therefore, they check $S=_{trace}I$ because of $I=_{trace}M$.

The validity of Algorithm 1 is formally presented in Theorem 1 given hereafter, and is proved in the Appendix 1.

THEOREM 1: (Validity of the test generation method)

Consider a specification S in the form of a minimal ONFSM, and any NFSM I. Suppose $n\leq m$ where n is the number of states in S, and m is the upper bound on the number of states in the prime machine of I. Let $\prod$ be the test suite generated for S using Algorithm 1. We have $S=_{trace}I$ iff $S=_{\prod}I$.

Proof: It follows from Lemmas given in Appendix 1. (The skeleton of the proof is given above in the remarks on Algorithm 1). □

According to Theorem 1, provided that the number of states in the prime machine of a given NFSM implementation is bounded by m ($n\leq m$), the test suite produced using Algorithm 1 can detect any type of faults given in the fault model, including multiple faults; in this sense, our method can guarantee full fault coverage, with respect to the given bound m.

Test application: If the complete-testing assumption is satisfied by the given implementation NFSM I, for a given NFSM specification S and a given test suite $\prod$, then the relation "$S=_{\prod}I$" can be checked as follows. For every test sequence t in $\prod$, let A and B be the two sets of possible output sequences produced when t is applied to S and I, respectively. A is easy to be derived from S. B can be obtained through

applying the test sequence t a sufficient number of times if the complete-testing assumption is satisfied by I. If for every test sequence t in $\prod$, A=B, then "$S=_{\prod}I$". Thus, according to Theorem 1, the test suites generated by Algorithm 1 can be used to test NFSM implementations against their specifications with respect to the trace-equivalence.

We now give an example of the application of this algorithm. Assuming that the implementation is a minimal ONFSM and will not have more than 4 states, we generate a test suite for the ONFSM of Figure 2 as follows:

$W = \{a, b\}$ $W_0=\{a\}$, $W_1=W_3=\{b\}$, $W_2=\{a, b\}$

$\mathbb{Q}=\{\varepsilon, a, b\}$ $\mathbb{R}=\{\varepsilon, b, a.a, a.b, a.b.a, a.b.b\}$, $\mathbb{R}=\{b, a.a, a.b.a, a.b.b\}$

$\prod 1=\{\varepsilon, a, b\}.\{a, b\}=\{a, b, a.a, a.b, a.b.a, a.b.b\}$

$\prod 2= \mathbb{R} \ \{W_0, W_1, ..., W_3\}$

$=\{b\}.W_0 \cup \{a.a\}.W_3 \cup \{a.b.a\}.(W_3 \cup W_1) \cup \{a.b.b\}.W_0$

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

$\prod=\prod 1 \cup \prod 2=\{a, b, a.a, a.b, a.b.a, a.b.b, b.a, a.a.b, a.b.a.b, a.b.b.a\}$

We note that in this example, $\mathbb{Q}$ is not minimal since it is derived using the approach of Remark (2) given before; and $\mathbb{Q}$ could be $\{a, a.b\}$. Furthermore, a test suite could be reduced by deleting each test sequence that is a prefix of another test sequence. The final test suite for the $\prod$ given above is $\{b.a, a.a.b, a.b.a.b, a.b.b.a\}$. The expected output sequences for the final test suite are listed as follows:

input sequences	**the corresponding output sequences**
b.a	f.d, f.e
a.a.b	d.f.e, e.f.e
a.b.a.b	d.f.d.f, d.f.d.e, d.f.e.e, d.e.f.e, e.e.f.e
a.b.b.a	d.f.f.d, d.f.f.e, d.e.f.d, d.e.f.e, e.e.f.d, e.e.f.e

A reset input must be sent before each test sequence of the test suite is applied.

We now illustrate how the resulting test suite can reveal faults. Consider the NFSM shown in Figure 3, which is a faulty implementation of the NFSM shown in Figure 2. This machine contains a single transfer

fault, as shown by the bold line in Figure 3. Assume that the complete-testing assumption is satisfied. When the input sequence a.a.b of the test suite is applied to the machine a number of times, the set of output sequences observed will be {d.f.e, e.f.f}, which is different from the expected set {d.f.e, e.f.e}. Therefore, a fault is discovered.

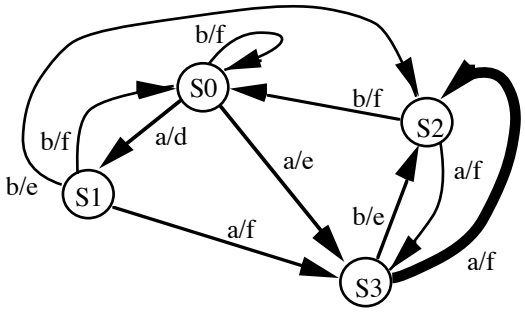


Figure 3. A faulty implementation of the ONFSM [illegible] 2

The above method may be considered a generalization of the Wp-method [11], and is, in particular, applicable to [illegible] FSMs, which are a class of NFSMs. We note that even [illegible], our method is slightly different from the [illegible]. (1) We [illegible] method, instead, we only require W [illegible] W [illegible] versa, our method may produce shorter test sequences than W- and Wp-methods. (2) We do not require $\prod 2 = \mathbb{R}.(\{\varepsilon\} \cup Li \cup [illegible] \cup ... \cup Li^{m-n}) \{W_0, W_1, ..., W_{n-1}\}$ as in the Wp-method; instead, we [illegible] $= \mathbb{R}.Li^{m-n} \{W_0, W_1, ..., W_{n-1}\}$, because of $\mathbb{R}.(\{\varepsilon\} \cup Li \cup Li^2 \cup ... \cup Li^{m-n-1}) \mathbb{Q}.([illegible] ... \cup Li^{m-n})$.

3.2. Test generation for Inres protocol

As an application example, we consider the Inres protocol (Initiator-responder protocol) from [16], which has already been used as a reference in many publications. To show how a test suite for the protocol under the ISO remote testing architecture can be obtained, we construct an NFSM for the system under test that consists of Responder and User (see Figure 4a); User refers to the upper layer protocols.

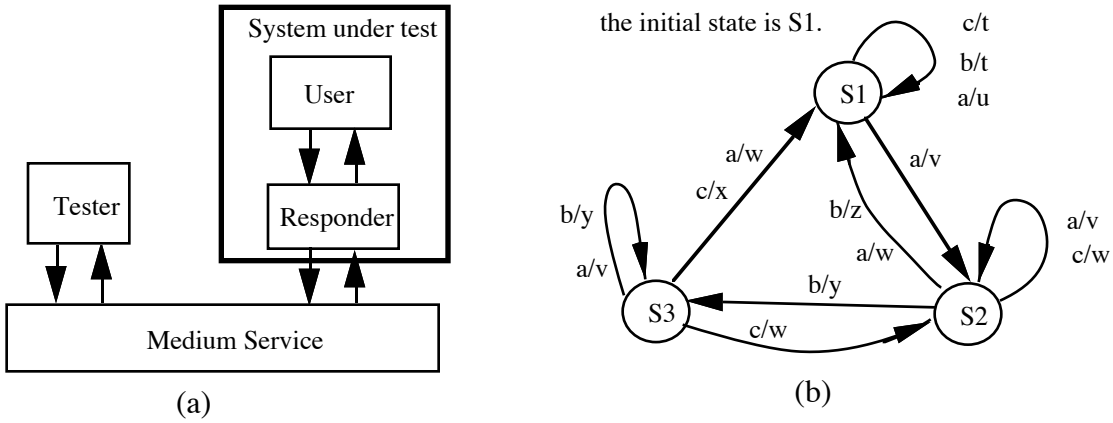


Figure 4. (a) Remote testing of Inres-responder, (b) NFSM for System under test

The DFSM model of Responder can be easily constructed from state tables [22] and is not presented in this paper. We assume that User may disconnect by sending IDISreq only in response to an ICONind or IDATind, and that it may send ICONresp when receiving ICONind, and send λ when receiving IDATind. During a test campaign, User executes each option sufficiently often. We also assume that Tester sends inputs slowly; it sends inputs to Responder only when no transitions in both User and Responder will be executed without new inputs sent from Tester (in this case, the FIFO queue between User and Responder will not contain more than one input (output)). This assumption is called slow-environment assumption, discussed in more detail in Section 4. Then, the behavior of the system under test is described by a minimal ONFSM with three states, as shown in Figure 4b. Interpretation of inputs, outputs and states is given in Table 2 [16].

Table 2. Interpretation of inputs, outputs and states

Inputs: Li = {a,b,c}.
a - "CR PDU", b - "DT_1 PDU", c - "DT_0 PDU";

Outputs: Lo = {t,u,v,w,x,y,z}.
t - "no output", u - "DR PDU", v - "CC PDU", w - "AK_0 PDU",
x - "AK_0 PDU followed by DR PDU", y - " AK_1 PDU",
z - "AK_1 PDU followed by DR PDU".

States: St = $\{S_1,S_2,S_3\}$.
S_1 - "disconnected" (initial state), S_2 - "data transfer & dat_nr=1", S_3 - "data transfer & dat_nr=0".

We derive a test suite ∏ as follows:

$\mathbb{Q}=\{\varepsilon, a, a.b\}$, $W=W_1=W_2=W_3=\{b\}$.

Assuming that a prime machine of any implementation does not have more than 3 states (i.e., m=3), we have the final test suite: $\prod = \{a.a.b,\ a.b.a.b,\ a.b.b.b,\ a.b.c.b,\ a.c.b,\ b.b,\ c.b\}$.

3.3. Trace-equivalent transformation to obtain ONFSMs

We now discuss the method of transforming arbitrary NFSMs into trace-equivalent ONFSMs. Combined with this transformation, the test generation method described in Section 3.1 can be used to generate test sequences for arbitrary NFSMs. As discussed in Section 2.2, in general, NFSMs cannot be transformed into trace-equivalent DFSMs, but they can be transformed into trace-equivalent ONFSMs. The method of transforming nondeterministic finite automata into equivalent deterministic automata given in [18] can be modified to transform arbitrary NFSMs into trace-equivalent ONFSMs. The main idea of the modification is as follows: (1) Given an NFSM F1, by viewing input/output pairs in F1 as labels, consider the NFSM F1 as a nondeterministic finite automaton A1. (2) Apply the method given in [18] to A1 to obtain an equivalent deterministic finite automaton A2. (3) By viewing the labels in A2 as original input/output pairs, derive an ONFSM F2 from A2. The details of a transformation algorithm can be found in Appendix 2.

4. APPLICATION OF THE GENERALIZED Wp-METHOD TO A SYSTEM OF CNFSMs

We first describe a system of CNFSMs. We then illustrate that for concurrent software, even if each individual sequential program is modeled by a *deterministic finite* state machine, the whole system may be *nondeterministic*, and cannot be necessarily modeled by a *state machine* with only a *finite* number of states; the definition of *state machines* is the same as that of NFSMs except that St could be infinite. We finally show that our test generation method for a single NFSM can be applied to generate test sequences for a system of CNFSMs, incorporating exhaustive reachability analysis [17].

4.1. Communicating NFSMs

The control portion of concurrent programs, especially in the area of communication software and communication protocols, can be modeled by a system of CNFSMs where the NFSMs communicate with each other over queues and channels. We assume that asynchronous communication mechanism is used in a system of CNFSMs, as described in SDL [4]. We describe such a system of CNFSMs as follows:

A system of CNFSMs, denoted by com(F1, F2, ..., Fn), consists of a number of CNFSMs, F1, F2, ..., Fn, where:

(1) Each individual CNFSM is an NFSM plus an input FIFO (first-in and first-out) queue; the NFSM only consumes the inputs in the queue.

(2) Each pair of machines may have two FIFO channels for communication; each channel is designated for one direction of communication. A channel connects only two machines.

(3) If a pair of machines, say F1 and F2, can communicate with one another through the channels between them, then signals from one machine pass through a FIFO channel and enter the input queue in the other machine (see Figure 5).

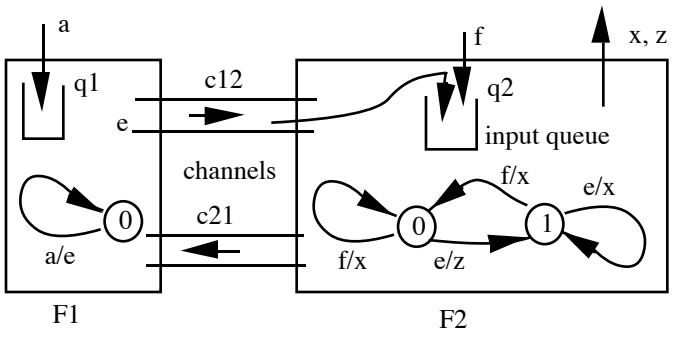


Figure 5. A system of two CNFSMs

Channels can contain an unlimited number of signals. The signals can remain in a queue or in a channel, for an arbitrary period of time. Individual CNFSM has an input queue of an infinite length. Figure 5 shows a system consisting of two communicating state machines.

4.2. Nondeterministic behavior of communicating DNFMs

We illustrate in this section, by an example, that (i) A system of communicating DFSMs may have *nondeterministic* behavior. (ii) Moreover, a system of communicating NFSMs cannot be modeled, in general, by a trace-equivalent state machine that contains only a finite number of states.

Consider the system of CDFSMs F1 and F2 in Figure 5. The two machines have the input queues q_1 and q_2, respectively; and c_{12} is the channel from F1 to F2. We use a tuple ($\mathbf{c_{12}}$, $\mathbf{c_{21}}$, $\mathbf{q_1}$, $\mathbf{q_2}$, $\mathbf{s_1}$, $\mathbf{s_2}$) to represent a global state of this system, where $\mathbf{c_{12}}$, $\mathbf{c_{21}}$, $\mathbf{q_1}$, $\mathbf{q_2}$ denote the contents of c_{12}, c_{21}, q_1, q_2, and $\mathbf{s_1}$ and $\mathbf{s_2}$ denote the states of F1 and F2, respectively (note: we use bold characters for contents, and plain characters for names to avoid confusing). In the system shown in Figure 5, however, F1 has only one state, c_{21} is not used, and q_1 receives inputs only from the environment. Therefore, for simplicity, we use a tuple ($\mathbf{c_{12}}$, $\mathbf{q_2}$, $\mathbf{s_2}$) to represent a global state.

Suppose that the environment sends the input sequence a.f; let us examine the output sequences that can be observed at the boundary of the system. A state machine that models a system of CNFSMs is called a *global state machine* of the system. Figure 6 shows a portion of the global state machine of the system consisting of F1 and F2. Assume that the system is initially in the state ([], [], 0) where [] stands for the empty queue or channel. The following cases may happen along with other cases when the system receives the sequence a.f:

(1) the system first enters the state ([], [f,e], 0), then transfers to the state ([], [], 0), issuing the output sequence z.x.

(2) the system first enters the state ([e], [f], 0), then transfers to the state ([], [], 1), generating x.z as an output sequence.

Therefore, this system is *nondeterministic*, so is its global state machine. The existing test generation methods for DFSMs are not applicable to such nondeterministic machines, therefore test generation methods for handling nondeterministic machines are required.

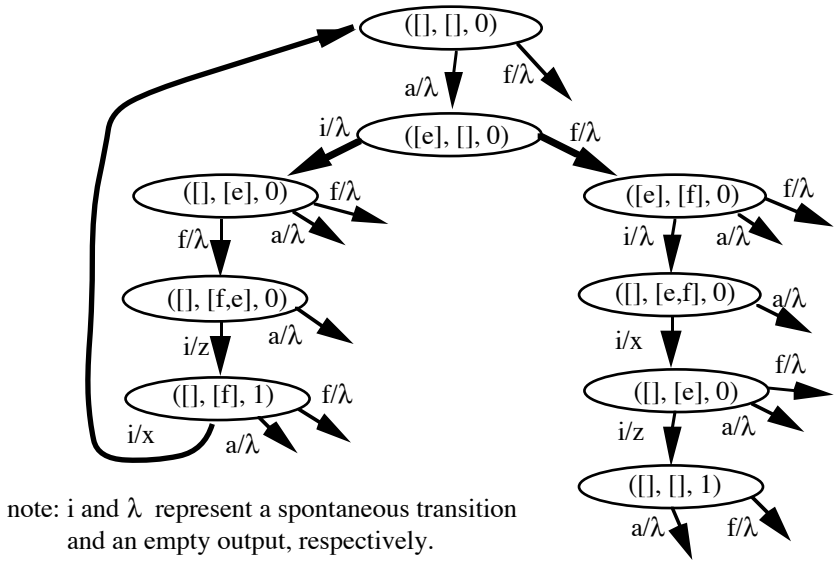


Figure 6. The nondeterministic behavior of the system in Figure 5

It is obvious from the above example that the number of states in a global state machine becomes infinite in case of unlimited channels and/or queues, since their contents are parts of global states. Consequently, our test generation method for a single NFSM, in general, cannot be applied to a system of CNFSMs directly, unless bounded channels and queues are assumed.

4.3. Generating test sequences by generalized Wp-method for a system of CNFSMs

We show in the following that the above developed method for a single machine can be applied to generate test sequences for a system of CNFSMs.

4.3.1. Approach based on bounded queues and channels

Since unbounded queues or unbounded channels cannot be implemented in practical application, it is reasonable to assume queue and channel to be of limited lengths. In case of bounded queues and channels, we model a system of CNFSMs as a trace-equivalent NFSM using exhaustive reachability analysis [17]. Then, we apply the generalized Wp-method to the resulting machine, with the upper bound m being chosen, for instance, as the number of states in the resulting machine. However, even if bounded queues

and channels are assumed, a trace-equivalent NFSM for a given system of CNFSMs may still be very large. In such cases, we use in the following another, further restricted approach.

4.3.2. Approach assuming a slow environment

For a system of CNFSMs, we say that the system runs in a *slow environment* if inputs can be sent from the environment to the system only in situations where all the queues and all the channels are empty. We say that the system has a *live-lock* if it is possible to execute an infinite number of transitions without further inputs. We assume in the following that the system has no livelock. The assumption of a slow environment is, for instance, satisfied for the communication protocols with handshake, such as connection and disconnection phases, for instance, in the ISO transport protocol.

If a given system of CNFSMs com(F1, F2,..., Fn) contains no live-lock and runs in a slow environment, then it can be modeled by a trace-equivalent NFSM with up to M1 M2 ... Mn states where M1, M2,..., Mn are the numbers of states in F1, F2,..., Fn, respectively. For such a system of CNFSMs, the trace-equivalent NFSM can be obtained using the reachability analysis. Figure 7 gives an example of a system of two CNFSMs without live-lock, and Figure 8 shows the global NFSM of the system.

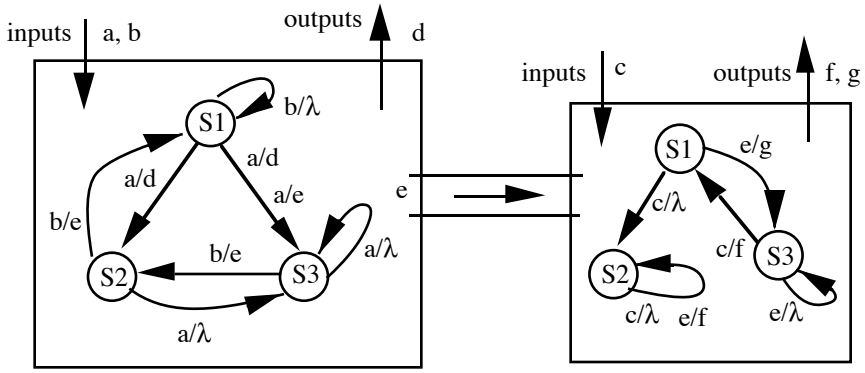


Figure 7. An example of a system of two CNFSMs

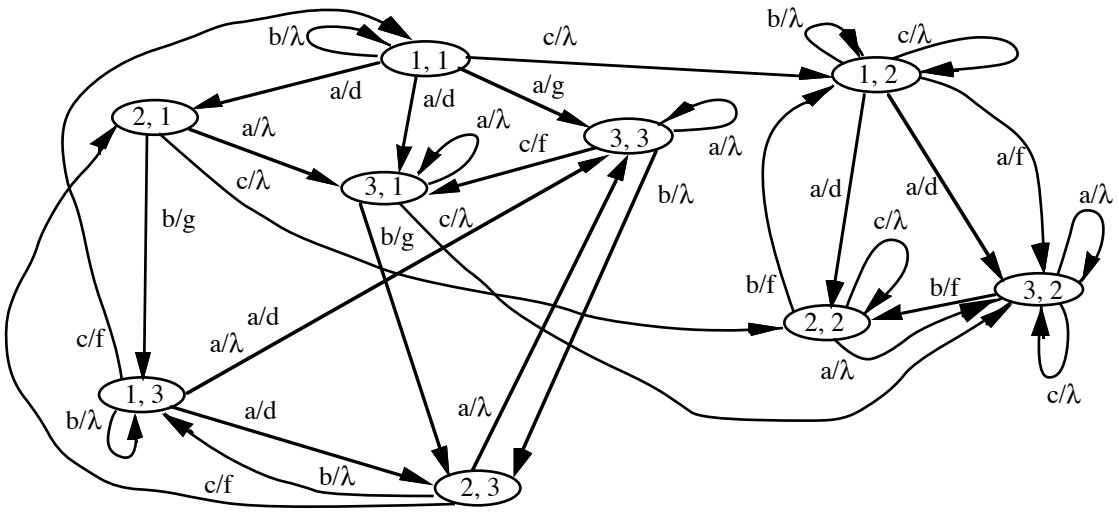


Figure 8. The global NFSM generated by using reachability analysis

After the global machine has been obtained from a given system of CNFSMs, we apply the generalized Wp-method to the resulting machine.

5. CONCLUSION

We presented in this paper a method of generating test sequences for NFSMs (nondeterministic finite state machines). This method is a generalization of the Wp-method [11]. It can ensure full fault coverage under the assumption of a limited number of states in the implementations, which has not been provided by other known test generation methods for NFSMs. It is also applicable to DFSMs (deterministic finite state machines); in this case, it reduces to a slightly improved version of the Wp-method. We also applied this method to generate test sequences for a system of communicating NFSMs.

We note that our method can be used to perform test generation for the control part of SDL processes. In order to apply this method, SDL processes are first abstracted, by neglecting parameters, into a system of communicating nondeterministic finite state machines with the SAVE constructs [32], called SDL-machines [25]. Although SDL-machines cannot be necessarily modeled as trace-equivalent finite state machines, most machines derived from practical SDL specifications can be modeled as trace-equivalent finite state machines. We use the algorithm given in [25] to obtain the trace-equivalent machines from the SDL-

machines, avoiding the SAVE constructs. Then, the test generation methods described in this paper can be applied.

APPENDIX 1: VALIDITY OF TEST METHOD

We first introduce several definitions, then we prove several lemmas that lead to Theorem 1.

Given a minimal ONFSM S (St_S, Li, Lo, h_S, S_0) and a NFSM I (St_I, Li, Lo, h_I, I_0), we assume in the following:

(1) All states of S and I are reachable from the initial states S_0 and I_0, respectively.

(2) S has n states with $n \geq 2$.

(3) M (St_M, Li, Lo, h_M, M_0) is the prime machine of I, and may have at most m states with $m \geq n$.

(4) S_i, S_j, S_k, S_l, and M_i, M_j, M_k, M_l represent the states of S and M, respectively.

(5) W, $\{W_0, W_1, ..., W_{n-1}\}$, $\mathbb{Q}$, $\mathbb{P}$, and $\mathbb{R}$ are produced using Algorithm 1.

(6) A test suite $\prod$ is constructed using Algorithm [illegible].

Then, $\prod = \prod 1 \cup \prod 2$ where $\prod 1 = \mathbb{Q}.(\{\varepsilon\} \cup Li \cup Li^2 \cup ... \cup Li^{m-n}).W$, and

$$\prod 2 = \mathbb{R}.Li^{m-n} \{W_0, W_1, ..., W_{n-1}\}.$$

Definitions of several notations

notation	meaning
$[S_i,M_i]$ -u-> $[S_j,M_j]$	For $u \in L$, S_i-u->S_j and M_i-u->M_j
$[S_i,M_i]$ =x=> $[S_j,M_j]$	For $x \in L^*$, S_i=x=>S_j and M_i=x=>M_j
$[S_i,M_i]$-*after*-V	Given a pair of states $[S_i,M_i] \in St_S \times St_M$, and a set V Li*
	$[S_i,M_i]$-*after*-V=$\{[S_j,M_j] \mid \exists x \in L^* (x^{in} \in V \ \& \ [S_i,M_i] =x=> [S_j,M_j])\}$
$\mathbb{D}$	$\mathbb{D}$ =$[S_0, M_0]$-*after*-Li*
$\mathbb{D}r$	$\mathbb{D}r = \{[S_i,M_j] \mid [S_i,M_j] \in \mathbb{D} \ \& \ S_i =_W M_i\}$
$\overline{Li}^k$	$\overline{Li}^k = \{\varepsilon\} \cup Li \cup ... \cup Li^k$, when $k \geq 1$; and $\overline{Li}^0 = \{\varepsilon\}$.

It is easy to see $\mathbb{D}r$ $\mathbb{D}$ [illegible] D D [illegible] e both S and M are observable, given $[S_i,M_i] \in \mathbb{D}$ and $x \in L^*$, if there is a pair $[S_j,M_j] \in \mathbb{D}$ such that $[S_i,M_i]$=x=>$[S_j,M_j]$, then $[S_j,M_j]$ is the only pair satisfying $[S_i,M_i]$=x=>$[S_j,M_j]$.

LEMMA 1: For V Li* *after*-V|≥k .

If |𝔻|>k, then |[S_0,M_0]-*after*-V.({ε} ∪ Li)| ≥k+1; and if |𝔻|≤k, then

[S_0,M_0]-*after*-V.({ε} ∪ Li) = [S_0,M_0]-*after*-V.

Proof:

(I) To prove that the lemma holds when |𝔻|>k.

The lemma holds when |[S_0,M_0] ow consider the case that |[S_0,M_0]-*after*-V|=k.

	statements	**reasons**
(1)	|𝔻| > k	hypothesis
(2)	|[S_0,M_0]-*after*-V| =k	hypothesis
(3)	[S_0,M_0]-*after*-V 𝔻	definition of 𝔻
(4)	∃[S_i,M_i]∈𝔻\[S_0,M_0]-*after*-V	(1) & (2) & (3)
(5)	∃[S_{k-1},M_{k-1}]∈[S_0,M_0]-*after*-V ∃[S_k,M_k]∈𝔻\[S_0,M_0]-*after*-V ∃x∈L* & x^{in}∈ V ∃u∈L such that: ([S_0,M_0]=x=>[S_{k-1},M_{k-1}]-u->[S_k,M_k]	(4)
(6)	∃[S_k,M_k]∈([S_0,M_0]-*after*-V.({ε} ∪ Li))\[S_0,M_0]-*after*-V	(5)
(7)	|[S_0,M_0]-*after*-V.({ε}	(6)

(II) To prove that the lemm olds when |𝔻|≤k.

(1)	|𝔻| ≤ k	hypothesis
(2)	|[S_0,M_0]-*after*-V| ≥k	hypothesis
(3)	[S_0,M_0]-*after*-V 𝔻	definition of 𝔻
(4)	[S_0,M_0]-*after*-V.({ε} ∪ Li) = [S_0,M_0]-*after*-V	(1) & (2) &(3).□

LEMMA 2: Assume $S_0 =_{\mathbb{Q}} M_0$. If |𝔻|>m, then |[S_0,M_0]-*after*-ℚ.$\overline{L}i^{m-n}$| ≥m;

and if |𝔻|≤m, then [S_0,M_0]-*after*-ℚ.$\overline{L}i^{m-n}$ = 𝔻.

Proof:

Since m≥n, $\overline{L}^{m-n}$ is always defined.

(I) To prove that the lemma holds when |𝔻|>m.

(1) $S_0 =_{\mathbb{Q}} M_0$ hypothesis

(2) $|\mathbb{D}| > m$ hypothesis

(3) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}| \geq n$ S is initially connected & (1)

(4) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n}| \geq m$ (2) & (3) & "apply Lemma 1 m-n times"

(II) It is evident from Lemma 1 that the lemma also holds when $|\mathbb{D}| \leq m$. □

LEMMA 3: $|\mathbb{D}r| \leq m$.

Proof:

(1) $|St_M| = m$ hypothesis

(2) $|\mathbb{D}r| > m$ assumption

(3) $\exists [S_j,M_k],[S_i,M_k] \in \mathbb{D}r$ ($j \neq i$ & $S_j =_W M_k$ & $S_i =_W M_k$) (1) & (2)

(4) (3) is not true not($S_j =_W S_i$) by definition of W

(5) $|\mathbb{D}r| \leq m$ (2) causes the contradiction between (3) and (4). □

LEMMA 4: If $S_0 =_{\Pi 1} M_0$, then $[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n} = \mathbb{D}r$.

Proof:

When $|\mathbb{D}| \leq m$, the lemma is evident from [redacted] consider the case that $|\mathbb{D}| > m$.

(1) $S_0 =_{\Pi 1} M_0$ hypothesis

(2) $|\mathbb{D}| > m$ hypothesis

(3) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n}| \geq m$ [redacted] (2) & (1) & Lemma 2

(4) $[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n}$ $\mathbb{D}$[redacted] (1)

(5) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n}| \leq |\mathbb{D}r| \leq m$ (4) & Lemma 3

(6) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n}| = |\mathbb{D}r| = m$ (3) & (5)

(7) $[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\bar{L}i^{m-n} = \mathbb{D}r$ (4) & (6). □

LEMMA 5: If $S_0 =_{\Pi 1} M_0$, then $\forall [S_i,M_k] \in \mathbb{D}$ ($\exists [S_j,M_k] \in \mathbb{D}r$).

Proof:

(1) $S_0 =_{\Pi 1} M_0$ hypothesis

(2) $S_0 =_{\mathbb{Q}} M_0$ (1)

(3) if $|\mathbb{D}| > |[S_0,M_0]\text{-}after\text{-}V|$,

then $|[S_0,M_0]\text{-}after\text{-}V.(\{\varepsilon\} \cup$ [illegible]$)| \geq |[S_0,M_0]\text{-}after\text{-}V|+1$ the same reason as for Lemma 1

(4) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}| \geq n$ "S is initially connected" & (2)

(5) not($\forall [S_i,M_k] \in \mathbb{D}$ ($\exists [S_j,M_k] \in$ [illegible] assumption

(6) $[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\overline{Li}^{m-n}$ $\mathbb{D}r$ [illegible] (1) & (5)

(7) $|[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\overline{Li}^{m-n}| \geq m$ (4) & (6) & "apply (3) m-n times"

(8) $|\mathbb{D}r| \geq m$ (6) & (7)

(9) $|\mathbb{D}r| = m$ (8) & Lemma 3

(10) $\exists [S_j,M_k],[S_l,M_k] \in \mathbb{D}r$ ($j \neq l$ & $S_j =_W M_k$ & $S_l =_W M_k$) (5) &

(11) (10) is not true not [illegible] definition of W

(12) $\forall [S_i,M_k] \in \mathbb{D}$ ($\exists [S_j,M_k] \in \mathbb{D}r$) (5) causes the co[illegible]radiction between (10) and (11). □

LEMMA 6: If $S_0 =_{\Pi 1} M_0$, then $\forall [S_i,M_k] \in \mathbb{D}$ ($[S_i =_{Wi} M_k]$ $[S_i =_W M_k]$).

Proof:

(1) $S_0 =_{\Pi 1} M_0$ hypothesis

(2) $[S_i,M_k] \in \mathbb{D}$ & $S_i =_{Wi} M_k$ assumption

(3) $S_j =_W M_k$ (2) & (1) & Lemm[illegible]

(4) $S_i =_{Wi} S_j$ (2) & (3) & W_i pre[illegible]

(5) $i=j$ (4) & definition of W_i

(6) $\forall [S_i,M_k] \in \mathbb{D}$ ($[S_i =_{Wi} M_k]$ [illegible]) (2) (3) & [illegible]

(7) $\forall [S_i,M_k] \in \mathbb{D}$ ($[S_i =_{Wi} M_k]$ [illegible]) definition of W_i

(8) $\forall [S_i,M_k] \in \mathbb{D}$ ($[S_i =_{Wi} M_k]$ $[S_i =_W M_k]$) (6) & (7). □

LEMMA 7: If $S_0 =_{\Pi} M_0$, then $[S_0,M_0]\text{-}after\text{-}\mathbb{Q}.\overline{Li}^{m-n} = \mathbb{D}r = \mathbb{D}$.

Proof:

(1) $S_0 =_{\Pi} M_0$ hypothesis

(2) $S_0 =_{\Pi 1} M_0$ (1)

(3) $S_0=\Pi^2 M_0$ (1)

(4) $S_0=_{\mathbb{Q}} M_0$ (1)

(5) $[S_0,M_0]$-*after*-$\mathbb{R}.Li^{m-n}$ $\mathbb{D}$ [illegible] Lemma 6

(6) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n+1}$ $\mathbb{D}$ [illegible] Lemma 4

(note: $\mathbb{R}\cup\mathbb{Q}=\mathbb{Q}.(\{\varepsilon\}\cup Li)$

$\mathbb{Q}.Li^{m-n+1}\cup\mathbb{Q}.Li^{m-n}=\mathbb{R}.Li^{m-n}\cup\mathbb{Q}.Li^{m-n}$

From (2) and Lemma 4, $[S_0,M_0]$-*after*-$\mathbb{Q}.Li^{m-n}$ $\mathbb{D}$ [illegible]

From (5) and Lemma 4, $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n+1}$ $\mathbb{D}$ [illegible]

(7) $|\mathbb{D}|>m$ assumption

(8) $|[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n+1}|\geq m+1$ (7) & (4) & Lemma 2 & [illegible]

(9) (7) is not true (6) & Lemma 3

(10) $|\mathbb{D}|\leq m$ (7) causes the contradiction between (7) and (9)

(11) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n+1}=\mathbb{D}$ (10) & (4) & Lemma 2

(12) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n+1}=\mathbb{D}r=\mathbb{D}$ (6) & (11) & $\mathbb{D}r$ $\mathbb{D}$ [illegible]

(13) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n}=\mathbb{D}r=\mathbb{D}$ (12) & Lemma 4. □

LEMMA 8: If $S_0=_{\Pi}M_0$, then $S_0=_{trace}M_0$.

Proof:

Note that $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{Li}^{m-n+1}=[S_0,M_0]$-*after*-$\mathbb{P}.\overline{Li}^{m-n}$.

(1) $S_0=_{\Pi}M_0$ hypothesis

(2) $\forall x\in L^*$(if $x^{in}\in\mathbb{Q}.\overline{Li}^{m-n}$ and $[S_0,M_0]=x=>[S_j,M_j]$,

then (i) $[S_j,M_j]$ is unique, and

(ii) $\forall a\in Li\ (S_j=_{\{a\}}M_j)$) (1) & Lemma 7 &

"S, M are observable"

(3) not $(S_0=_{trace}M_0)$ assumption

(4) $\exists z\in L^*\ \exists a\in Li\ \exists[S_i,M_i]\in\mathbb{D}$ such that

$[S_0,M_0]=z=>[S_i,M_i]$ & not$(S_i=_{\{a\}}M_i)$

where $[S_i,M_i]$ is unique for z (3) & "S, M are observable"

(5) $\exists y\in L^*\ \exists a\in Li\ \exists[S_i,M_i]\in\mathbb{D}r$ such that

$y^{in} \in \mathbb{Q}.\overline{L}i^{m-n}$ & $[S_0,M_0]=y=>[S_i,M_i]$ & not$(S_i=_{\{a\}}M_i)$ (4) & (1) & Lemma 7

(note: From (1) & Lemma 7, $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{m-n} = \mathbb{D}r = \mathbb{D}$)

(6) $S_0= {}_{trace}M_0$ (3) causes the contradiction between (2) and (5). □

LEMMA 9: $S_0=_\Pi M_0$ iff $S_0=_\Pi I_0$.

Proof: Since $I_0=_\Pi M_0$, $S_0=_\Pi M_0$ implies $S_0=_\Pi I_0$. By the same reason, $S_0=_\Pi I_0$ implies $S_0=_\Pi M_0$. □

APPENDIX 2: TRANSFORMATION TO OBTAIN ONFSMS

We now present an algorithm to construct a trace-equivalent ONFSM for an arbitrary NFSM.

ALGORITHM 2 : Constructing a trace-equivalent ONFSM for a given NFSM.

Input : an NFSM S.

Output : an ONFSM S'.

Step 1: Build a graph G consisting initially of a single unmarked node, labeled M_0, where $M_0=\{S_0\}$.

Step 2: If there is no unmarked node in the graph G, stop; G is the ONFSM S' and the node M_0 represents the initial state of S'. Otherwise:

(a) find and mark a unmarked node M in G, where the label M is a set of states of S.

(b) for every $u \in L$, (i) first construct $M'=\{Q \mid P \in M\ (P=u=>Q)\}$, (ii) if M' is not a node label in the resulting graph G, then create a node with label M', (iii) create a directed edge from M to M' with label u.

(c) Go to Step 2. □

It is not difficult to prove that the NFSM S' resulting from the above algorithm is trace-equivalent to S. For example, this algorithm constructs the machine shown in Figure 2 from the NFSM in Figure 1. In general, the number of states in an ONFSM grows exponentially as the number of states in the original NFSM does. However, in practical application, NFSMs are usually not so nondeterministic (i.e., there are only a few states each of which has more than one transitions associated with the same input.); in this case, the number of states in ONFSMs will not grow exponentially. We note that the ONFSM obtained using Algorithm 2

may not be a minimal machine. In this case, it should be reduced to a minimal form using an approach similar to the state minimization for DFSMs [21].

REFERENCES:

[1] Alfred V. Aho, Barry S. Bosik and Stephen J.Griesmer, "Protocol Testing and Verification within AT&T", AT&T Technical Journal, Vol.69, No.1, 1990, pp.4-6.

[2] Noriyasu Arakawa, Terunao Soncoka, "A test Case Generation Method for Concurrent Programs", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.95-106.

[3] AT&T Technical Journal, Special Issue on Protocol Testing and Verification, Vol.69, No.1, 1990.

[4] F. Belina and D. Hogrefe, "The CCITT-Specification and Description Language SDL", Computer Networks and ISDN Systems, Vol. 16, 1989, pp.311-341.

[5] Gregor v. Bochmann, Rachida Dssouli, and J. R. Zhao, "Trace Analysis for Conformance and Arbitration Testing", IEEE Trans. on Software Engineering, Vol. SE-15, No.11, 1989, pp.1347-1356.

[6] G.v. Bochmann, A.Das, R.Dssouli, M.Dubuc, A.Ghedamsi, and G.Luo, "Fault Model in Testing", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.17-30.

[7] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language Lotos", Computer Networks and ISDN Systems, Vol. 14, No. 1, 1987, pp.25-59.

[8] Ed Brinksma, "A Theory for the Derivation of Tests", IFIP Protocol Specification, Testing, and Verification VIII, Ed. by S. Aggarwal and K. Sabnani, Elsevier Science Publishers B.V. (North-Holland), 1988, pp.63-74.

[9] S. Budkowski and P. Dembinski, "An Introduction to Estelle: A Specification Language for Distributed Systems", Computer Networks and ISDN Systems, Vol. 14, No.1, 1987, pp.3-23.

[10] T.S.Chow, "Testing Software Design Modeled by Finite-State Machines, IEEE Trans. on Software Engineering, Vol. SE-4, No.3, 1978, pp.178-187.

[11] S.Fujiwara, G. v. Bochmann, F.Khendek, M.Amalou and A.Ghedamsi, "Test Selection Based on Finite State Models", IEEE Trans. on Software Engineering, Vol SE-17, No.6, June, 1991, pp.591-603.

[12] Susumu Fujiwara and Gregor von Bochmann, "Testing Nondeterministic Finite State Machine with Fault Coverage", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems,1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.267-280.

[13] Susumu Fujiwara and Gregor von Bochmann, "Testing Nondeterministic Finite State Machine", Publication #758 of D.I.R.O, University of Montreal, January 1991.

[14] A. Gill, Introduction to the Theory of Finite-State Machines, New York: McGraw-Hill, 1962, 270p.

[15] G. Gonenc, "A Method for Design of Fault Detection experiments", IEEE Trans. on Computer, Vol C-19, June, 1970, pp.551-558.

[16] D. Hogrefe, MUTEST: OSI Formal Specification Case Study: the Inres protocol and Service.

[17] G.J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991, 500p.

[18] John E.Hopcroft, Jeffery D.Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, Inc., 1979, 418p.

[19] A. Kalnins, "Global State Based Automatic Test Generation for SDL", SDL'91: Evolving Methods (Proceedings of 5th SDL Forum), North-Holland, 1991, pp.303-312.

[20] Hans Kloosterman, "Test Derivation from Nondeterministic Finite State Machines", IFIP Transactions, Protocol Test Systems,V (Proceedings of 5th International Workshop on Protocol Testing Systems, Montreal, Canada, 1992), Ed. by G.v. Bochmann, R. Dssouli and A. Das, 1993, North-Holland, pp.297-308.

[21] Z.Kohavi, Switching and Finite Automata Theory, McGraw-Hill Computer Science Series, New York, 1970.

[22] J. Kroon, Inres State Tables, Private Communication, 1992.

[23] Gang Luo & Junliang Chen, "Generating Test Sequences For Communication Protocol Modeled by CNFSM", Information Technology: Advancement, Productivity and International Cooperation (Proc. of the 3rd Pan Pacific Computer Conference), Vol.I, Ed. by Chen Liwei et al, International Academic Publishers, 1989, pp.688-694.

[24] Gang Luo & Junliang Chen, "Test Design for SDL Described Concurrent Communication Software", International Conference on Communication Techniques'89, Beijing, China, International Academic Publishers, 1989, pp.207-210.

[25] Gang Luo, Anindya Das, and Gregor von Bochmann, "Test Selection Based on SDL specification with Save", SDL'91: Evolving Methods (Proceedings of 5th SDL Forum), North-Holland, 1991, pp.313-324.

[26] Gang Luo, Gregor von Bochmann, Anindya Das, and Cheng Wu, "Failure-Equivalent Transformation of Transition Systems to Avoid Internal Actions", Information Processing Letters, Vol. 44, No.6, December, 1992, Elsevier Science Publishers B.V.(North-Holland), pp.333-343.

[27] Robin Milner, A Calculus of Communicatiing Systems, Lecture Notes in Computer Science Vol.92, 1980, Sringer-Verlag, 171p.

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

[29] Alexandre Petrenko and Nina Yevtushenko, "Test Suite Generation for a FSM with a Given Type of Implementation Errors", Proceedings of IFIP 12th International Symposium on Protocol Specification, Testing, and Verification, U.S.A., North-Holland, 1992, pp.229-243.

[30] D. H. Pitt and D. Freestone, "The Derivation of Conformance Tests from Lotos Specifications", IEEE Transactions on Software Engineering, Vol.16, No.12, Dec. 1990, pp.1337-1343.

[31] D. Rayner, "OSI Conformance Testing", Comput. Networks & ISDN Syst., Vol.14, 1987, pp.79-89.

[32] Anne Bourguet-Rouger & Pierre Combes, "Exhaustive Validation and Test Generation in Elivis", SDL Forum'89: The Language at Work, Ed. by Ove Faergemand and Maria Manuela Marques, North-Holland, 1989, pp.231-245.

[33] K.Sabnani & A.T.Dahbura, "A Protocol Test Generation Procedure", Computer Networks and ISDN, Vol.15, No.4, 1988, North-Holland, pp.285-297.

[34] B. Sarikaya, G.v. Bochmann, and E. Cerny, "A Test Design Methodology for Protocol Testing", IEEE Transactions on Software Engineering, Vol.13, No.9, Sept. 1987, pp.989-999.

[35] SDL Newsletter, Dec. 1991.

[36] D. P. Sidhu and T. K. Leung, "Formal Methods for Protocol Testing: A Detailed Study", IEEE Trans. on Software Engineering, Vol SE-15, No.4, April, 1989, pp.413-426.

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

[38] Andrew S. Tanenbaum, Computer Networks, Second Edition, Prentice Hall, 1988, 658p.

[39] Piyu Tripathy and Kshirasagar Naik, "Generation of Adaptive Test Cases from Nondeterministic Finite State Models", IFIP Transactions, Protocol Test Systems,V (Proceedings of 5th International Workshop on Protocol Testing Systems, Montreal, Canada, 1992), Ed. by G.v. Bochmann, R. Dssouli and A. Das, 1993, North-Holland, pp.309-320.

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

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

[42] M. F. Witteman, R. C. van Wuijtswinkel and S.Ruud Berkhout, "Nondeterministic and Default Behaviour", IFIP Transactions, Protocol Test Systems,V (Proceedings of 5th International Workshop on Protocol Testing Systems, Montreal, Canada, 1992), Ed. by G.v. Bochmann, R. Dssouli and A. Das, 1993, North-Holland, pp.275-288.