GENERATING TESTS FOR COMMUNICATION SOFTWARE MODELED BY PARTIALLY-SPECIFIED FINITE STATE MACHINES[1]

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

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

ABSTRACT In order to test the control portion of communication software, specifications are usually first abstracted to state machines, then test cases are generated from the resulting machines. In practical applications, the state machines obtained from the specifications are often partially-specified, and sometimes nondeterministic. We come out with two methods for test generation. The first generates test suites for the software that is modeled by partially-specified finite state machines (PFSMs) with respect to a conformance relation, called *quasi-equivalence*. The method is a generalized version of the Wp-method and applicable also to completely-specified deterministic machines which are specific class of PFSMs. This method yields usually smaller test suites with full fault coverage for each class of machines than the existing methods for the same class which also assure full fault coverage. The second method generates test suites to test deterministic implementations against their partially-specified nondeterministic finite state machine (PNFSM) specifications, with respect to a conformance relation, which we call *determinization relation.*

KEYWORDS: Error detection, finite state machines, partially-specified nondeterministic finite state machines, protocol conformance testing, 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

In the area of communication software, systematic approaches have been developed for protocol conformance testing [Rayn87, Boch89], and the selection of appropriate test suites [Fuji91, Pitt90, Sidh89, Sari87, Chow78]. These approaches can produce significant economic benefits [Aho90, AT&T90]. Usually, the specifications of communication software are first abstracted to state machines, then test cases are generated from the resulting machines [Lee91, Roug89]. A considerable amount of work has been done to generate test cases for *completely-specified*, *deterministic* finite state machines (FSMs) [Fuji91, Sidh89, Sari84, Chow78, Gone70, Vuon89, Sabn85, Nait81, Vasi73]. However, the specifications of communication software are often *partially* (or incompletely) specified, and may be *nondeterministic*. For example, communication protocols are often partially specified [Sari84]. A major specification language for communication software, ESTELLE [Budk87, ISO9074] supports the description of partially-specified behavior. Furthermore, in practical applications, specifications may be nondeterministic but their implementations are deterministic. In nondeterministic state machine specifications, for a given state, several transitions may be associated with the same input, and they represent several choices which valid implementations can have. During implementation process, only one of the choices (transitions) is required to be implemented. Such an implementation process is also called determinization process. Some work has been reported on test generation for partially-specified deterministic machines [Petr91, Evtu89]; however, no test generation method has been reported for checking nondeterministic specifications against their implementations.

We study in this paper test generation for the finite state machines that could be both partially-specified and nondeterministic, guided by pre-defined conformance relations.

In the area of protocol conformance testing, the meaning of conformance between a specification and the valid implementations is specified either by informal description, or by precisely-defined conformance relations. Usually, the formally-defined conformance relations are preferable since they provide a means to direct the development of test generation methods and a basis to analyze the validity of the methods. For completely-specified deterministic finite state machines (FSMs), partially-specified deterministic finite state machines (PFSMs), there are commonly-defined conformance relations in the literature [Fuji91, Chow78, Vasi73, Star72, Gill62]. However, no conformance relation has been reported for partially-specified nondeterministic finite state machines (PNFSMs), except for some general study on the specialization of object behaviors and requirement specifications [Boch92].

In Section 2, after formally defining PNFSMs and several related notations, we first describe a conformance relation, called *quasi-equivalence*, for PFSMs [Gill62]. We then introduce a conformance relation between PNFSM specifications and their PFSM implementations, called *determinization* relation. The relation is defined in terms of input/output traces in accordance with black-box testing strategy. When the relation is applied to FSM and PFSM specifications, which are specific cases of PNFSMs, it coincides to the corresponding conformance relations given in the literature. We also define several concepts which are related to test generation.

Guided by the conformance relations, in Section 3, we come out with two test generation methods. The first method is to generate test suites from PFSMs (partially-specified *deterministic* finite state machines); and the resulting test suites can be used to test PFSM implementations against their specifications with respect to the quasi-equivalence relation. The method is a generalized version of the Wp-method [Fuji91], obtained by combining the ideas given in [Fuji91] and [Evtu89]. The second one is to adaptively generate test suites from so-called *observable* PNFSMs for testing the determinization relation. The OPNFSMs have the 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. The OPNFSMs are a specific class of PNFSMs that have a lower degree of nondeterminism.

In Section 4, we compare our methods with other test generation methods, on the basis of applicability, fault coverage and the size of test suites. This method yields usually smaller test suites with full fault coverage for each class of machines than the existing methods for the same class which also assure full fault coverage.

We conclude in Section 5 by discussing some extreme case of the length of test cases and the upper bound of the size of test suites, for partial machines.

2. NOTATIONS AND ABSTRACT TESTING FRAMEWORK

We first give in this section the definition of PNFSMs, then present conformance relations between specifications and implementations under the black-box testing strategy (where implementations are assumed to be black-boxes), and finally define several concepts which are related to testing.

2.1 Partially-specified nondeterministic finite state machines (PNFSMs)

We first define PNFSMs in a traditional form similar to that given in [Star72] for NFSMs. For the convenience of presentation, we then define additional notations for PNFSMs similar to that for labeled transition systems [Brin88, Fuji91b, Fuji91c].

DEFINITION *Partially-specified Nondeterministic Finite State Machine* :
A *Partially-specified Nondeterministic Finite State Machine* (PNFSM) is defined as 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 inputs.

(3) Lo is a finite set of outputs.

(4) h is a behavior function:

h : d => powerset(St ∞ Lo) \{□} where

(i) d⁄St 6 Li (PNFSM becomes completely specified if d=St 6 Li);

(ii) □ 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 called a *transition* from P to Q with label a /b.

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

We assume that a "reliable" reset input *r* is available in any implementation of a PNFSM such that upon receiving *r* in any state the implementation returns to the initial state.

We often use in the following the term "partial machine" to refer to a PNFSM, which may be deterministic or not. A partial machine 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 such a machine.

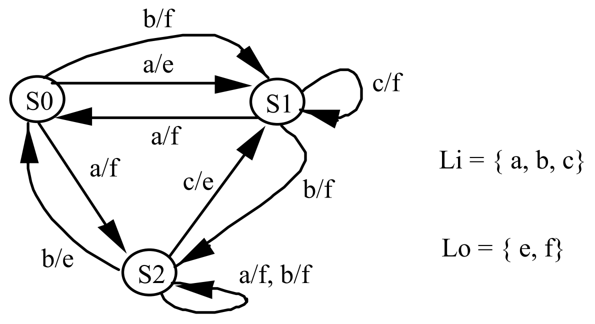

Figure 1. An example of a PNFSM

For the convenience of the presentation, we also introduce in Table 1 several notations.

Table 1. Notation for PNFSMs

notation	**meaning**
L	Li 6 Lo, a set of input/output pairs; u denotes such a pair
ε	ε is the empty sequence.
L*	set of sequences over L; x denotes such a sequence. Note that ε□L*
P\-u->	For P, Q □ St, not(∃Q(P-u->Q))
P=ε=>Q	P=Q
P=a/b=>Q	P-a/b->Q
P=x=>Q	∃P_1, ..., P_{k-1}□St (P=P_0=u_1=>P_1...=u_k=>P_k=Q) where $u_{1,...,}u_k$□L, and x=u_1...u_k
P=x=>	∃Q□St (P=x=>Q)
Tr(P)	*Tr*(P)={ x | P=x=>}
x^{in}	For x□L*, x^{in} is an input sequence obtained by deleting all outputs in x (note that x^{in}□Li*)
V^{in}	For V⁄L*, V^{in}={x^{in} | x□V }

Tr^{in}(P)	$Tr^{in}(P)=\{ x^{in} \mid P=x=> \}$, (note that $Tr^{in}(P)=Li^*$ for each state P of completely-specified NFSMs)

DEFINITION *Initially connected PNFSM*:

Given a PNFSM S (St, Li, Lo, h, S_0), S is said to be *initially connected* iff

$$\forall S_i \in St \; \exists x \in L^* \; (S_0=x=>S_i). \; \Box$$

In initially connected PNFSMs, every state is reachable from the initial state. Without loss of generality, we assume that all machines considered in the rest of the paper are initially connected. If a given machine S is not initially connected, we may consider only such a submachine which 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.

We first define so-called observable PNFSMs, a concept originally described in [Star72] for completely specified machines, which represents a restricted form of nondeterminism.

DEFINITION *Observable PNFSMs* (OPNFSMs) :

A PNFSM is said to be *observable* if for every state S $\in$St, and every input/output pair a/b $\in$L, there is at most one transition; that is, S-a/b->S_1 & S-a/b->S_2 ==> S_1=S_2. $\Box$

As an example, Figure 1 shows an OPNFSM. OPNFSMs are a subclass of partial machines. In observable machines, a state and an input/output pair can uniquely determine at most one next state. However, an OPNFSM may still be nondeterministic in the sense that a state and an input cannot determine a unique next state and a unique output. We note that all deterministic machines are observable.

Given a state P in a PNFSM, we say that P is *deterministic* if no two outgoing transitions from P have the same input; and we say that P is *nondeterministic* if P is not deterministic. For a PNFSM, if every state is deterministic, then the machine is deterministic; and we call it a partial FSM

(PFSM). We now define in the following several specific classes of PFSMs, which are useful concepts for test generation.

DEFINITION: *Reduced PFSMs* :

An PFSM is *reduced* iff $\forall S_i, S_j \square$ St ($i \square j$ ==> $Tr\,(S_i) \square Tr\,(S_j)$). $\Box$

A PFSM is reduced if and only if none of its states accept the same set of input/output sequences.

DEFINITION: *Distinguishable states*:

Given a pair of states S_i and S_j, S_i and S_j are *distinguishable*, written S_i— S_j, iff

$$\exists x \square Tr(S_i) \& Tr(S_j) \; (x^{in} \square Tr^{in}(S_i) \leftrightarrow Tr^{in}(S_j))$$

where $Tr(S_i) \& Tr(S_j) = (Tr(S_i) \approx Tr(S_j)) \setminus (Tr(S_i) \leftrightarrow Tr(S_j))$.

If a pair of states are not distinguishable, we say that they are *indistinguishable*. $\Box$

Two states are distinguishable if and only if there is an input/output sequence x such that x can be accepted by only one of the two states but the input sequence x^{in} can be accepted by the both of them.

DEFINITION: *Minimal PFSMs:*

A PFSM is *minimal* iff $\forall S_i, S_j \square$St ($i \square j$ ==> S_i—S_j). $\Box$

A PFSM is minimal if and only if every pair of states are distinguishable.

A minimal PFSM is reduced, but a reduced PFSM is not necessarily minimal. Given a minimal machine S, each state of P is distinguishable from all other states of P, which is not necessarily true for a reduced machine. If we consider a completely specified machine, then a reduced machine is also minimal.

We also need the following concepts for presenting our method.

DEFINITION: *prefix set* *pref*(V) *for a given set of sequences*:

Given a set of sequences $V \subseteq Li^*$,

pref(V)$=\{t1 \mid t2 \in Li^* \ \&\ t1.t2 \in V\}$ where t1.t2 is the concatenation of t1 with t2. □

DEFINITION: *Concatenation of sets of i/o sequences or input sequences*:

Assuming $V1, V2 \subseteq L^*$ (or $V1, V2 \subseteq Li^*$), *the concatenation of sets*, written ".", is defined as follows:

$V1.V2 = \{ t1.t2 \mid t1 \in V1 \ \&\ t2 \in V2\}$ where t1.t2 is the concatenation of t1 with t2.

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

2.2. Conformance relations

Before any study on how to generate test suites, the following question must first be answered: under the black-box testing strategy, what kind of conformance relation between a specification and the corresponding implementation is expected to hold ?

For (completely-specified, deterministic) FSMs, there is a widely-accepted conformance relation, called *equivalence*, (see, e.g., [Fuji91, Chow78, Vasi73, Star72, Gill62]), which requires that a specification and its implementation produce the same output sequence for every input sequence. We define in this section conformance relations for machines in terms of the relations between their initial states.

DEFINITION *Equivalence:*

The *equivalence* relation between two states P and Q in PFSMs, written

$P \neq Q$, holds iff $Tr(P) = Tr(Q)$

Given two PFSMs S and I with their initial states S_0 and I_0, we write $S \neq I$ iff $S_0 \neq I_0$. □

A reduced machine has no equivalent states. For PFSMs, a conformance relation, called *quasi-equivalence*, was presented in [Gill62, Star72, Evtu89]. The relation requires that, for every input sequence that can be accepted by a specification, the specification and its implementation produce the same output sequence.

DEFINITION *Quasi-equivalence:*

The *quasi-equivalence* relation between two states P and Q in PFSMs, written

$P \leq_{quasi} Q$, holds iff $\mathit{Tr}(P)/\mathit{Tr}(Q)$

Given two PFSMs S and I with their initial states S_0 and I_0, we write $S \leq_{quasi} I$ iff $S_0 \leq_{quasi} I_0$. $\square$

The quasi-equivalence relation is not an equivalent relation since it is not symmetric.

However, it is quite often in practical applications that specifications are nondeterministic and their implementations are deterministic. For a state P and a set of transitions starting from P, we say that in state P, an input *a* *associates* the set of transitions if the input symbol of every transition in the set is *a*. In nondeterministic state machine specifications, for a given state, several transitions may be associated with the same input, and they represent several choices which valid implementations can have. During implementation process, only one of the choices is required to be implemented. Such an implementation process is also called determinization process. We now formalize the conformance relation between PNFSM specifications and their PFSM implementations, which we call *determinization relation.*

DEFINITION *Choice:*

The *choice* relation between a PNFSM S and a PFSM S', written

$S \varepsilon_{cho} S'$, holds iff S' can be obtained from S in the following fashion:

For every P□ St and every *a* □Li, if the input *a* associates more than one transitions from the state P, keep one of them and eliminate the rest of transitions. We say that S' *is chosen* from S. $\square$

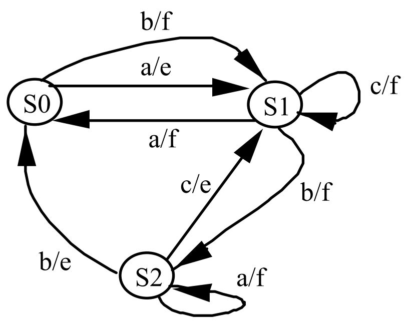

Figure 2. A PFSM chosen from the PNFSM of Figure 1

DEFINITION *Determinization*

The *determinization* relation between a PNFSM S and a PFSM I, written $S\varepsilon_{det}I$, holds iff $\exists S'(S\varepsilon_{cho}S' \ \& \ S'\leq_{quasi}I)$. □

We present in the following the relations among the above-defined conformance relations .

THEOREM 1: Given two PNFSMs S and I, assuming that they have common Li and Lo, we have the following statements:

(i) $S \neq I \iff S\leq_{quasi}I \ \& \ I\leq_{quasi}S$

(ii) if S and I are (deterministic) PFSMs, then $S\varepsilon_{det}I \iff S\leq_{quasi}I$

(iii) if S and I are (completely-specified, deterministic) FSMs, then

$S\leq_{quasi}I \iff S\neq I \iff S\varepsilon_{det}I$. □

The above theorem is evident from the corresponding definitions.

2.3. Definitions related to testing

We define in this section several concepts which are related to testing finite state machines.

DEFINITION *Test case* and *test suite* :

For a given PNFSM, a sequence t of a finite length is a *test case* if t□$Tr^{in}(S_0)$. A *test suite* is a finite set of test cases. □

DEFINITION : *Equivalence* with respect to a given input set:

The *equivalence* relation between two states P and Q, with respect to a given input set □⁄Li*, written P$=_{□}$Q, holds iff ¢x□*Tr*(P)&*Tr*(Q) (x□*Tr*^{in}(P)↔*Tr*^{in}(Q) ==> x^{in}□□).

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

The equivalence relation with respect to a given input set □ requires that, *for every input sequence in □that can be accepted by both a specification and its implementation, the specification and its implementation produce the same output sequence.*

We note: (i) S≠I iff ∀□⁄Li* (S $=_{□}$ I), and (ii) S$\leq_{quasi}$I iff ∀□⁄*Tr*^{in}(S) (S $=_{□}$ I).

3. TEST GENERATION

We present in this section two test generation methods. The first method is to generate test suites from PFSMs (partially-specified *deterministic* finite state machines); and the resulting test suites can be used to test PFSM implementations against their specifications with respect to the quasi-equivalence. The second one is to adaptively generate test suites from PNFSMs (partially-specified *nondeterministic* finite state machines) for testing the determinization relation.

3.1. Test generation for PFSMs with respect to quasi-equivalence

We first define several key concepts for presenting our method, then give an algorithm of generating test suites, and finally present a theorem for establishing the validity of the algorithm.

DEFINITION: *Characterization set* W:

Given an PFSM, a *characterization set* is a minimal set W⁄Li* such that:

∀S_i, S_j□St (S_i—S_j ==> ∃x□*Tr*(S_i)&*Tr*(S_j) (x^{in}□*Tr*^{in}(S_i)↔*Tr*^{in}(S_j)↔W)). □

The above definition is generalized from the concept of the characterization set for FSMs given in [Chow78] to PFSMs. We omit the algorithm for generating characterization sets because of the length limitation of the paper. One can develop such an algorithm by borrowing the ideas of generating characterization sets for FSMs pointed out by [Koha70, Chow78].

DEFINITION: *State identification sets* $\{W_0, W_1, ..., W_{n-1}\}$:
Given a PFSM with n states and a characterization set W, $\{W_0, W_1, ..., W_{n-1}\}$ is a tuple of *state identification sets* if, for i=0, 1, ..., n-1, W_i is a minimal set such that

(i) $W_i / \mathit{Tr}^{in}(S_i) \leftrightarrow \mathit{pref}(W)$, and

(ii) for j=0, 1, ..., n-1, $(S_i — S_j,$ ==> $\exists x□\mathit{Tr}(S_i)\&\mathit{Tr}(S_j)$ ($x^{in}□W_i$)). $\square$

The above definition is generalized from the concept for FSMs given in [Fuji91] to PFSMs.

DEFINITION: *subscripts*(A) for a given state set:
For A/St, *subscripts*(A) is a string of integers i1, i2, ..., ik,
where i1< i2< ...<ik and A=$\{S_{i1}, S_{i2}, ..., S_{ik}\}$. $\square$

Given two sets of states A and B, we say that the *subscripts* of A is *smaller* than that of B if *subscripts*(A) precedes *subscripts*(B) in lexicographic order. The notation *subscripts*(A) for a given set of states A is needed for defining a so-called *maximal set of pairwise-distinguishable states* $f(S_i)$ *for a given state* S_i for PFSMs. A set of states are *pairwise-distinguishable* if and only if every pair of states in the set are distinguishable. *A maximal set of pairwise-distinguishable states* is a set such that it is not contained by any other set of pairwise-distinguishable states. A maximal set of pairwise-distinguishable states $f(S_i)$ for a given state S_i is the set with the smallest subscript among the maximal sets of pairwise-distinguishable states that contains S_i, which is formally defined as follows.

DEFINITION: *Maximal set of pairwise-distinguishable states* f(S_i) *for a given state* S_i:

Given a PFSM and a state S_i□St, f(S_i) is defined as a set A⁄St such that:

(i) S_i□A, and

(ii) $\forall S_k$, S_j□A (k□j ==> S_k—S_j), and

(iii) there is no B⁄St such that

(i') S_i□B, and

(ii') $\forall S_k$, S_j□B (k□j ==> S_k—S_j), and

(iii') $|B| > |A|$ or

$|B|=|A|$, and *subscripts*(B) precedes *subscripts*(A) in lexicographic order. $\square$

Given a minimal machine, for every state S_i, we have f(S_i)=St. For a given PFSM, we denote the number of all different maximal sets of pairwise-distinguishable states f(S_i)'s as *fuzziness degree* δ, as defined below.

DEFINITION: *Fuzziness degree* δ *for a given* PFSM:

Given a PFSM, we have $\delta = |\{f(S_i) \mid S_i$□$St\}|$. $\square$

According to the above definition, every state S_i has only one maximal set of pairwise-distinguishable states f(S_i). Therefore, it is easy to see that $1 \leq \delta \leq |St|$, and $\delta=1$ for any minimal PFSM.

DEFINITION : *Prime machine*:

For a given PFSM S (St, Li, Lo, h_S, S_0), the *prime machine* of S is a reduced (not necessarily minimal) PFSM M (St_M, Li, Lo, h_M, M_0) such that $S \neq M$. $\square$

We give in the following the test generation algorithm. This algorithm requires that the user previously estimates an *upper bound* on the number of states in the prime machine of the given FSM implementation, that is, the number of states in its minimal form.

ALGORITHM 1: Test generation.

Input : A specification S in the form of a (arbitrary) PFSM (St, Li, Lo, h, S_0), and the upper bound m on the number of states in the prime machine of the given FSM implementation.

Output : a test suite □.

Step 1: Determine the fuzziness degree δ of S.

Step 2: Let the number of states in S be n ($n \le \delta m$). Construct a characterization set W, and a tuple of state identification sets $\{W_0, W_1, ..., W_{n-1}\}$.

Step 3: Construct a minimal set $\mathbb{Q}$⁄Li* such that: $\forall S_i$□St $\exists x$□L* (x^{in}□$\mathbb{Q}$ & S_0=x=>S_i).

Step 4: Let $\mathbb{P}$=$\mathbb{Q}$.($\{\varepsilon\} \approx$ Li)) and $\mathbb{R}$=$\mathbb{P}\backslash\mathbb{Q}$

Step 5: Assume: (i) for V⁄Li*, V@W= $\bigcup_{\substack{S_0=x=>Si \\ \& \, x^{in} \in V}} \{x^{in}\}.(W \leftrightarrow Tr^{in}(S_i))$.

(ii) for V⁄Li*, V8$\{W_0, W_1, ..., W_{n-1}\}$= $\bigcup_{\substack{S_0=x=>Si \\ \& \, x^{in} \in V}} \{x^{in}\}.W_i$

Construct a test suite □ in the following manner:

□ = □1≈□2

where □1 = $\mathbb{Q}$.($\{\varepsilon\} \approx Li \approx Li^2 \approx ... \approx Li^{\delta m-n}$)@W, and

□2 =$\mathbb{R}$. $Li^{\delta m-n}$8$\{W_0, W_1, ..., W_{n-1}\}$. $\square$

In the above algorithm, the given specification is not required to be reduced. However, a much smaller test suite will be obtained if we use its reduced form.

For example, assuming that the implementation is a minimal FSM and will not have more than 3 states, we generate a test suite for the PFSM of Figure 2 as follows:

W = {a, b}, W_0={a}, W_1={a, b}, W_2={ b}

$\mathbb{Q}$ = {ε, a, a.b}, $\mathbb{P}$ ={ε, a, b, c, a.a, a.b, a.c, a.b.a, a.b.b, a.b.c},

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

δ = 1,

□1= {ε, a, a.b}@{a, b}={a, b, a.a, a.b, a.b.a, a.b.b}

□2= ℝ8{W_0, W_1, W_2}

={b}.W_1≈ {a.a}.W_0≈{a.c}.W_1≈ {a.b.a}.W_2≈ {a.b.b}.W_0≈ {a.b.c}.W_1

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

□ = □1≈□2 = {a, b, a.a, a.b, a.b.a, a.b.b, b.a, b.b, a.a.a, a.c.a, a.c.b, a.b.a.b,

a.b.b.a, a.b.c.a, a.b.c.b}

Furthermore, a test suite could be reduced by deleting each test case that is a prefix of another test case. The final test suite is { b.a, b.b, a.a.a, a.c.a, a.c.b, a.b.a.b, a.b.b.a, a.b.c.a, a.b.c.b}. A reset input must be sent before a test case is applied.

THEOREM 2: (Validity of the test generation method):
Consider a given specification S in the form of a PFSM, and any FSM I. Suppose $n \le \delta m$ where n is the number of states in S, and m is the upper bound of number of states in the prime machine of I. Let □ be the test suite generated for S using Algorithm 1. We have $S \le_{quasi} I$ iff $S =_{□} I$.

Proof : The theorem follows from Lemmas given in Appendix. □

As shown in Algorithm 1, test suites for minimal partial machines can be constructed in the same way as for completely specified minimal machines since δ is equal to one for minimal machines. However, if a partial machine has indistinguishable states, then the machine cannot be transformed into its minimal form to generate test suite with respect to the quasi-equivalence relation. The reason is that the transformation of a partial machine into a minimal form by merging states will result in the appearance of new traces that are not defined in the original machine. In turn, this results in that some valid implementations may not pass a test suite derived from the minimal form, and that some test cases in such a test suite may be not acceptable in the original machine. Therefore, partial machines should not be transformed into a minimal form for test generation.

3.2. Test generation for OPNFSMs with respect to determinization relation

We present in this section an adaptive testing procedure to generate test suites from OPNFSMs for testing the determinization relation.

ADAPTIVE TESTING PROCEDURE:

Input : An OPNFSM specification S and an FSM implementation I (the internal structure of implementation is not available).

Output : Report "($S\varepsilon_{det}I$)" or "not($S\varepsilon_{det}I$)".

Step 1: Let an OPNFSM variable SS be S initially. If the SS is not a deterministic PFSM, then go to Step 2. Otherwise, go to Step 7.

Step 2: For the SS, find $x□L^*$ and a nondeterministic state S_i such that

S_0=x=>S_i and $\forall y□$pref({x}) (y□x & S_0=y=>S_j ==> S_j is deterministic.)

Step 3: Find $a□L^{in}$ such that the multiplicity of the set {u | u^{in}=a & S_i=u=> } is over one.

Step 4: Apply the test sequence r.x^{in}.a to the implementation (r is the reset input). If the output sequence observed after applying r.x^{in} is not the one produced by applying the test sequence to the specification, then report "not($S\varepsilon_{det}I$)" (i.e., the implementation fails to pass the testing), and stop. Otherwise, assume the last output in the resulting output sequence is d, and do Step 5.

Step 5: If not(S_i=a/d=>), then report "not($S\varepsilon_{det}I$)", and stop. Otherwise, do Step 6.

Step 6: Construct a new OPNFSM SS from the original by deleting all transitions from S_i with the input a except for the transition with the label a/d. If the resulting SS is not a deterministic PFSM, then go to Step 2. Otherwise, go to Step 7.

Step 7: Generate a test suite from SS using Algorithm 1, then apply the resulting test suite □ to the implementation. If $SS=_{□}I$, then report "($S\varepsilon_{det}I$)"; otherwise, report "not($S\varepsilon_{det}I$)". Stop. □

We say the above procedure to be adaptive because the test selection depends on the result of application of the formerly selected test cases. We use an example to explain the testing procedure. Assume that the specification is the OPNFSM shown in Figure 1, and that the implementation I satisfies $S' \leq_{quasi} I$ where S' is the PFSM shown in Figure 2. First, we find a nondeterministic state S_0 in Step 2 with $x=\varepsilon$. By Steps 3 and 4, we derive a test sequence r.a . The last output in the resulting output sequence must be e; then construct a new machine SS by deleting the transition from S_0 to S_2 with the label a/f. After repeating the above steps once again, we delete the transition from S_2 to S_2 with the label b/f. At this point, the resulting SS is the PFSM shown in Figure 2. According to Step 7, we generate a test suite from the SS, as shown in Section 3.1.

4. COMPARISON WITH OTHER RELATED WORK

Since FSMs and PFSMs are specific classes of OPNFSMs, our methods can be applied to them, to test the equivalence and quasi-equivalence relations, respectively (see Theorem 1). We compare in this section our method for OPNFSMs with the other test generation methods for different machines [Fuji91, Vuon89, Sabn85, Nait81, Chow78, Vasi73, Petr91, Petr92, Evtu89], which also require a "reliable" reset in the implementations (note, that simple experiments or checking sequences do not use this assumption), as shown in Figure 3. When applied to such classes of machines, this method yields usually smaller test suites with full fault coverage for each class of machines than the existing methods for the same class which also assure full fault coverage.

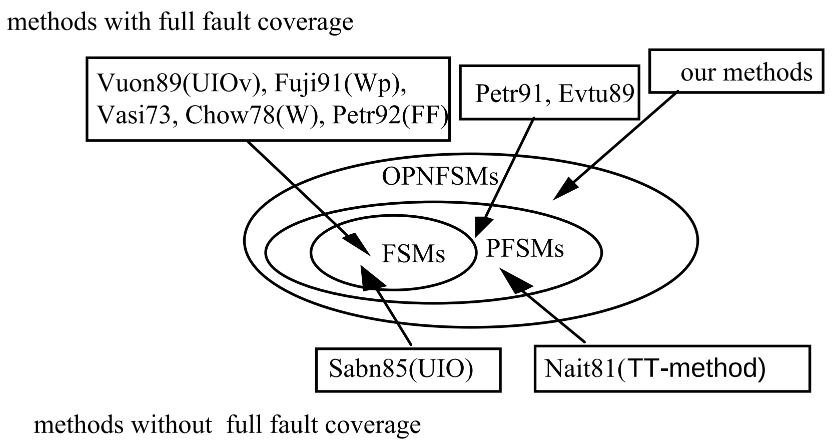
methods with full fault coverage
Vuon89(UIOv), Fuji91(Wp), Vasi73, Chow78(W), Petr92(FF)
Petr91, Evtu89
our methods
OPNFSMs
FSMs
PFSMs
Sabn85(UIO)
Nait81(TT-method)
methods without full fault coverage

Figure 3. General comparison based on applicability and fault coverage

When our method is applied to FSMs, the conformance relation to be tested is the equivalence, the same as in the W-method [Vasi73, Chow78], the Wp-method [Fuji91], the UIO-method [Sabn85], the UIOv-method [Vuon89], the FF-method [Petr92] and the TT-method (Transition tour) [Nait81]. The UIO-method does not guarantee full fault coverage, as it has been pointed out in [Vuon89]; neither does the TT-method. These methods have been justified by simulation on the basis of percentage of fault coverage. UIOv- and FF- methods guarantee full fault coverage (i.e., check equivalence) only if no malfunction causes an increase in the number of states. The W- and Wp-methods detect all faults that may even increase the state number up to the given bound, and the latter produces, in general, smaller test suite than the former [Fuji91].

The method given in Section 3.1 may be considered a generalization of the Wp-method [Fuji91], and is, in particular, applicable to deterministic FSMs, which are a specific class of PFSMs. We note that even for FSMs, our method is slightly different from the Wp-method. We do not require Wi/W as in the Wp-method, instead, we only require Wi/*pref*(W). Since Wi/W implies Wi/*pref*(W) but not vice versa, our method may produce shorter test cases than W- and Wp-methods.

Test generation for partial FSMs has received much less attention than that for completely-specified FSMs. However, practical communication software are often modeled as partial machines. Some authors proposed to complete the "don't care" state/input combinations of partial machines in accordance with a so-called *completeness assumption* [Sabn85, Vuon89]. The assumption states that a machine should be constructed in such a way that, for every state/input combination representing "don't care", it produces a *null* or *error* output and either remains in the *same* state or goes into an *error* state. However, in many cases, implementations are not constructed in the above way. In this case, the completeness assumption cannot be satisfied. Methods for test suite generation from a deterministic partial FSM were reported in [Evtu89, Petr91]. We note that neither the method given in Section 3.1 nor method given in [Petr91] necessarily produces smaller test suites than the other. However, our methods combine the ideas given in [Evtu89] and [Fuji90], can generate usually smaller test suites than the method given in [Evtu89], and are applicable to testing deterministic machine implementations against their nondeterministic specifications.

5. CONCLUSION

We first present in this paper a method of generating test cases from partially-specified finite state machines specifications. If a given PFSM is not minimal and its fuzziness degree δ is more than one, then the lengths of test cases produced by our method grow rapidly when δ increases. Let n and m be the numbers of states in a specification and its implementation, respectively. In the extreme case, when $\delta = n$, the length of a test case can reach n6m, as shown in [Evtu89]. As to the multiplicity of test suites produced by the method given in Section 3.1, the order is $O(n^3|Li|^{\delta m-n+1})$. We then give a method of generating test suites to check OPNFSM deterministic implementations against their nondeterministic specifications. Our methods can be applied to test generation for the control portion of the software written in SDL [Beli89] or ESTELLE.

APPENDIX : VALIDITY OF TEST METHOD

For the convenience of presentation, we make several conventions and definitions; then we give several lemmas which are required for proving the Theorem 2.

Given a PFSM S (St_S, Li, Lo, h_S, S_0) and an FSM 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 $\delta m \geq n$.

(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) A characterization set is W. A tuple of state identification sets is $\{W_0, W_1, ..., W_{n-1}\}$.

(6) the fuzziness degree of S is δ.

(7) a set $\mathbb{Q}$/Li* constructed from S such that: $\forall S_i$□St $\exists x$□L* (x^{in}□$\mathbb{Q}$ & S_0=x=>S_i).

(8) $\mathbb{P}=\mathbb{Q}.(\{\varepsilon\} \approx \mathrm{Li})$) and $\mathbb{R}=\mathbb{P}\backslash\mathbb{Q}$.

(9) A test suite □ which is constructed using Algorithm 1:

$$□ = □1 \approx □2$$

where $□1 = \mathbb{Q}.(\{\varepsilon\} \approx \mathrm{Li} \approx \mathrm{Li}^2 \approx ... \approx \mathrm{Li}^{\delta m-n})@W$, and

$□2 = \mathbb{R}.\ \mathrm{Li}^{\delta m-n}8\{W_0, W_1, ..., W_{n-1}\}$. $\square$

Definitions of several notations

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

It is easy to see $\mathbb{D}$r⁄$\mathbb{D}$ and $|\mathbb{D}\mathrm{r}| \leq |\mathbb{D}| \leq$n6m.

LEMMA 1: For V⁄Li*, assume $|[S_0,M_0]$-*after*-V$|$ εk .

If $|\mathbb{D}|$>k, then $|[S_0,M_0]$-*after*-V.$(\{\varepsilon\} \approx \mathrm{Li})|$ εk+1; and if $|\mathbb{D}| \leq$k, then

$[S_0,M_0]$-*after*-V.$(\{\varepsilon\} \approx \mathrm{Li})$ = $[S_0,M_0]$-*after*-V.

Proof:

(I) To prove that the lemma holds when $|\mathbb{D}|$>k.

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

statements	reasons
(1) $\|\mathbb{D}\| > k$	hypothesis
(2) $\|[S_0,M_0]$-*after*-V$\|$ =k	hypothesis
(3) $[S_0,M_0]$-*after*-V⁄$\mathbb{D}$	definition of $\mathbb{D}$

(4) $\exists[S_i,M_i]\square\mathbb{D}\backslash[S_0,M_0]$-*after*-V (1) & (2) & (3)

(5) $\exists[S_{k-1},M_{k-1}]\square[S_0,M_0]$-*after*-V

$\exists[S_k,M_k],[S_i,M_i]\square\mathbb{D}\backslash[S_0,M_0]$-*after*-V

$\exists u\square L\ \exists x,y\square L^*$ such that: $x^{in}\square V$ &

$([S_0,M_0]{=}x{=}{>}[S_{k-1},M_{k-1}]$-u-$>[S_k,M_k]{=}y{=}{>}[S_i,M_i]$ (4)

(6) $\exists[S_k,M_k]\square([S_0,M_0]$-*after*-V.$(\{\varepsilon\}\approx Li))\backslash[S_0,M_0]$-*after*-V (5)

(7) $|[S_0,M_0]$-*after*-V.$(\{\varepsilon\}\approx Li)|\ \varepsilon k+1$ (6)

(II) To prove that the lemma holds when $|\mathbb{D}|\leq k$.

(1) $|\mathbb{D}|\leq k$ hypothesis

(2) $|[S_0,M_0]$-*after*-V $|\ \varepsilon k$ hypothesis

(3) $[S_0,M_0]$-*after*-V/$\mathbb{D}$ definition of $\mathbb{D}$

(4) $[S_0,M_0]$-*after*-V.$(\{\varepsilon\}\approx Li) = [S_0,M_0]$-*after*-V (1) & (2) &(3). $\square$

LEMMA 2: Assume $S_0{=}_{\mathbb{Q}}M_0$. If $|\mathbb{D}|{>}\delta 6m$, then $|[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}|\ \varepsilon\delta 6m$;

and if $|\mathbb{D}|\leq\delta 6m$, then $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n} = \mathbb{D}$.

Proof:

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

(I) To prove that the lemma holds when $|\mathbb{D}|{>}\delta 6m$.

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

(2) $|\mathbb{D}|{>}\delta 6m$ hypothesis

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

(4) $|[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}|\ \varepsilon\delta 6m$ (2) & (3) & apply Lemma 1 δ6m-n times

(II) It is evident from Lemma 1 that the lemma also holds when $|\mathbb{D}|\leq\delta 6m$. $\square$

LEMMA 3: For $S_i\square St_S$, $|\mathbb{D}_i|\leq m$

Proof:

(1) $|St_M| = m$ hypothesis

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

(3) $\exists[S_j,M_k],[S_l,M_k]□\mathbb{D}_i$ ($j□l$ & $S_j=_W M_k$ & $S_l=_W M_k$) (1) & (2)

(4) $S_j=_W S_l$ (3) & M is completely specified

(5) (4) is not true "definition of W" & "S_j, $S_l□f(S_i)$"

(6) $|\mathbb{D}_i| \le m$ (2) causes the contradiction between (4) and (5). $\square$

LEMMA 4: $|\mathbb{D}r| \le \delta 6m$.

Proof:

Let $E = \{f(S_i) \mid S_i□St_S\}$.

(1) $\delta=|E|$ definition of δ

(2) $\mathbb{D}r/\ \bigcup^{f(Si)\in E} \mathbb{D}_i$ definition of $\mathbb{D}r$

(3) $\forall f(S_i)□E\ (|\mathbb{D}_i|\le m)$ Lemma 3

(4) $|\mathbb{D}r| \le \sum^{f(Si)\in E} |\mathbb{D}_i|$ (2)

(5) $|\mathbb{D}r| \le \delta 6m$ (1) & (3) & (4). $\square$

LEMMA 5: If $S_0=_{□1}M_0$, then $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n} = \mathbb{D}r$.

Proof:

When $|\mathbb{D}| \le \delta 6m$, the lemma is evident from Lemma 2. Now consider the case that $|\mathbb{D}| > \delta 6m$.

(1) $S_0=_{□1}M_0$ hypothesis

(2) $|\mathbb{D}| > \delta 6m$ hypothesis

(3) $|\,[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}\,|\ \varepsilon\delta 6m$ (2) & Lemma 2

(4) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}/\mathbb{D}r$ (1)

(5) $|\,[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}\,| \le |\mathbb{D}r| \le \delta 6m$ (4) & Lemma 4

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

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

LEMMA 6: If $S_0=_{□1}M_0$, then $\forall[S_i,M_k]□\mathbb{D}\ (\exists[S_j,M_k]□\mathbb{D}r)$.

Proof:

Let E = $\{f(S_i) \mid S_i□St_S\}$, thus $\delta=|E|$.

(1)	$S_0=_{□1}M_0$	hypothesis
(2)	$S_0=_{\mathbb{Q}}M_0$	(1)
(3)	if $|\mathbb{D}|>|[S_0,M_0]$-*after*-$V|$, then $|[S_0,M_0]$-*after*-$V.(\{\varepsilon\} \approx Li)| \varepsilon |[S_0,M_0]$-*after*-$V|+1$	the same reason as for Lemma 1
(4)	$|[S_0,M_0]$-*after*-$\mathbb{Q}| \varepsilon n$	"S is initially-connected" & (2)
(5)	not($\forall[S_i,M_k]□\mathbb{D}$ $(\exists[S_j,M_k]□\mathbb{D}r)$)	assumption
(6)	$[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}/\mathbb{D}r/\mathbb{D}$ & $\mathbb{D}r□\mathbb{D}$	(1) & (5)
(7)	$|[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n}| \varepsilon\delta 6m$	(4) & (6) & "apply (3) δm-n times"
(8)	$|\mathbb{D}r|\varepsilon\delta 6m$	(6) & (7)
(9)	$\exists M_l□St_M \forall[S_i,M_j]□\mathbb{D}r\ (l□j)$	(5)
	consider such a M_l	make a convention
(10)	$\mathbb{D}r/ \bigcup_{f(Si)\in E} \mathbb{D}_i$	definition of $\mathbb{D}r$
(11)	$\forall f(S_i)□E\ (|\mathbb{D}_i|\leq m-1)$	(9) & the similar reason for Lemma 3
(12)	$|\mathbb{D}r| \leq \sum_{f(Si)\in E} |\mathbb{D}_i|$	(10)
(13)	$|\mathbb{D}r| \leq\delta 6(m-1)$	(11) & (12) & $\delta=|E|$.
(14)	$\forall[S_i,M_k]□\mathbb{D}\ (\exists[S_j,M_k]□\mathbb{D}r)$	(5) causes the contradiction between (8) and (13). $\square$

LEMMA 7: If $S_0=_{□1}M_0$, then $\forall[S_i,M_k]□\mathbb{D}\ ([S_i=_{W_i}M_k] \iff [S_i=_W M_k])$.

Proof:

(1)	$S_0=_{□1}M_0$	hypothesis
(2)	$[S_i,M_k]□\mathbb{D}$ & $S_i=_{W_i}M_k$	assumption
(3)	$S_j=_W M_k$	(2) & (1) & Lemma 6
(4)	$S_i=_{W_i}S_j$	(2) & (3) & $W_i/pref(W)$
(5)	$i=j$	(4) & definition of W_i
(6)	$\forall[S_i,M_k]□\mathbb{D}\ ([S_i=_{W_i}M_k] \Longrightarrow [S_i=_W M_k])$	(2) ==> (3) & (5)

(7) $\forall[S_i,M_k]\square\mathbb{D}$ ($[S_i=_{W_i}M_k]$ <== $[S_i=_{W}M_k]$) definition of W_i

(8) $\forall[S_i,M_k]\square\mathbb{D}$ ($[S_i=_{W_i}M_k]$ <==> $[S_i=_{W}M_k]$) (6) & (7). $\square$

LEMMA 8: If $S_0=_{\square}M_0$, then $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n} = \mathbb{D}r = \mathbb{D}$.

Proof:

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

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

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

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

(5) $[S_0,M_0]$-*after*-$\mathbb{R}.Li^{\delta m-n}/\mathbb{D}r$ (3) & (2) & Lemma 7

(6) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n+1}/\mathbb{D}r$ (5) & (2) & $\mathbb{R}\approx\mathbb{Q}=\mathbb{Q}.(\{\varepsilon\}\approx Li)$

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

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

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

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

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

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

(13) $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n} = \mathbb{D}r = \mathbb{D}$ (12) & Lemma 5. $\square$

LEMMA 9: If $S_0=_{\square}M_0$, then $S_0\leq_{quasi}M_0$.

Proof:

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

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

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

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

(ii) $\forall a\square Li$ ($S_j=_{\{a\}}M_j$)) (1) & Lemma 8

(3) not ($S_0\leq_{quasi}M_0$) assumption

(4) $\exists y$□L^* $\exists u$□L $\exists [S_i,M_i]$□𝔻r such that y^{in}□ ℚ.$\overline{L}i^{\delta m-n}$ &

$[S_0,M_0]$=y=>$[S_i,M_i]$ & S_i-u-> & M_i\-u-> (3) & (1) & Lemma 8 &

(5) $S_0 \leq_{quasi} M_0$ (3) causes the contradiction between (2) and (4). □

LEMMA 10: S_0=□M_0 iff S_0=□I_0 .

Proof: Since I_0=□M_0, S_0=□M_0 implies S_0=□I_0. By the same reason,S_0=□I_0 implies S_0=□M_0. □

REFERENCES

[Aho90] 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.

[AT&T90] AT&T Technical Journal, Special Issue on Protocol Testing and Verification, Vol.69, No.1, 1990.

[Boch89] Gregor v. Bochmann, "Trace Analysis for Conformance and Arbitration Testing", IEEE Transactions on Software Engineering, Vol. SE-15, No.11, 1989.

[Boch91] G.v. Bochmann, A. Das, R. Dssouli, M.Dubuc, A.Ghedamsi, and G.Luo, "Fault Models in Testing", IFIP Transactions, Protocol Testing Systems IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp.17-30.

[Boch92] G.v. Bochmann and Reinhard Gotzhein, "Specialization of Object Behaviors and Requirement Specifications", in preparation.

[Beli89] F. Belina and D. Hogrefe, "The CCITT-Specification and Description Language SDL", Computer Networks and ISDN Systems, Vol. 16, pp.311-341, 1989.

[Brin88] 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.

[Budk87] 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.

[Chow78] T.S.Chow, "Testing Software Design Modeled by Finite-State Machines, IEEE Transactions on Software Engineering, Vol. SE-4, No.3, 1978.

[Evtu89] N. V. Evtushenko and A.F. Petrenko, "Fault-Detection Capability of Multiple Experiments", Automatic Control and Computer Science, Allerton Press, Inc., New York, Vol.23, No.3, 1989, pp.7-11.

[Fuji91] S.Fujiwara, G. v. Bochmann, F.Khendek, M.Amalou and A.Ghedamsi, "Test Selection Based on Finite State Models", IEEE Transactions on Software Engineering, Vol SE-17, No.6, June, 1991, pp.591-603.

[Fuji91b] 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.

[Fuji91c] S. Fujiwara and G. v. Bochmann, "Testing Nondeterministic Finite State Machine", Publication #758 of D.I.R.O, University of Montreal, January 1991.

[Gill62] A. Gill, Introduction to the Theory of Finite-State Machines, New York: McGraw-Hill, 1962.

[Gone70] G. Gonenc, " A Method for Design of Fault Detection Experiments", IEEE Transactions on Computer, Vol C-19, June, 1970, pp.551-558.

[Hopc79] John E.Hopcroft, Jeffery D.Ullman, Introduction to Automata Theory, Languages, and Computation, 1979, Addison-Wesley Publishing Company, Inc., 418p.

[ISO9074] ISO, Estelle - A Formal Description Technique Based on an Extended Finite State Transition Model, IS 9074.

[Lee91] D.Y. Lee and J.Y. Lee, "A Well-Defined Estelle Specification for the Automatic Test Generation", IEEE Transactions on Computers, Vol.40, No.4, April, 1991, pp.526-542.

[Nait81] S.Naito and M.Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", in Proc. FTCS (Fault Tolerant Comput. Syst.), 1981, pp.238-243.

[Petr91] Alexandre Petrenko, "Checking Experiments with Protocol Machines", 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.83-94.

[Petr92] Alexandre Petrenko and Nina Yevtushenko, "Test Suite Generation for a FSM with a Given Type of Implementation Errors", IFIP 12th International Symposium on Protocol Specification, Testing, and Verification, (participant's proceedings), U.S.A., 1992.

[Pitt90] 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.

[Rayn87] D. Rayner, "OSI Conformance Testing", Computer Networks and ISDN Systems, Vol.14, No.1, 1987, pp.79-89.

[Sabn85] K.Sabnani & A.T.Dahbura, "A New Technique for Generating Protocol Tests", ACM Computer Communication Review, Vol.15, No.4, 1985, pp.36-43.

[Sari84] Behcet Sarikaya and Gregor v. Bochmann, "Synchronization and Specification Issues in Protocol Testing", IEEE Transactions on Communications, Vol. COM-32, No.4, April 1984, pp.389-395.

[Sari87] 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.

[Sidh89] D. P. Sidhu and T. K. Leung, "Formal Methods for Protocol Testing: A Detailed Study", IEEE Transactions on Software Engineering, Vol SE-15, No.4, April, 1989, pp.413-426.

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

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

[Vuon89] 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.