SELECTING TEST SEQUENCES FOR PARTIALLY-SPECIFIED NONDETERMINISTIC 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. The state machines obtained from the specification are often *both* partially-specified *and* nondeterministic. We come out with a method of generating test suites for the software that is modeled by partially-specified nondeterministic finite state machines (PNFSMs). On the basis of intuitive notions, a conformance relation, called *quasi-equivalence*, is introduced for such machines, which serves as a guide to test generation. Our method is also applicable to completely-specified deterministic machines, partially-specified deterministic machines, and completely-specified nondeterministic machines, which are typical classes of PNFSMs. When applied to such classes of machines, this method usually yields smaller test suites with full fault coverage for each class of machines than the existing methods for the same class which also provide full fault coverage. The test suites generated by the method can be used to check a conformance relation between a specification and its implementations.

KEYWORDS: Finite state machines, partially-specified nondeterministic finite state machines, protocol conformance testing, protocol 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 [Rayn87, Boch89], and the selection of appropriate test suites [Fuji91, Pitt90, Sidh89, Sari87, Sari84, 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, Chow78, Gone70, Vuon89, Sabn85, Nait81, Vasi73]. However, the specifications of communication software often contain *both* nondeterministic *and* partially (or incompletely) specified behavior. For example, all the three major specification languages for communication software, LOTOS [Bolo87, ISO8807], ESTELLE [Budk87, ISO9074] and SDL [Beli89] support the description of nondeterminism (SDL will support nondeterminism in the near future [SDL91]); and ESTELLE and LOTOS can describe partially-specified behavior. Therefore, the state machines abstracted from the specifications may be *both* partially-specified *and* nondeterministic. There is a practical need for testing nondeterministic models [Witt92]; in particular, communication protocols, when tested under the ISO remote testing architecture, are often modeled as partially-specified and nondeterministic finite state machines.

Some work on test generation for nondeterministic models has been done in the context of LOTOS [Trip91, Pitt90, Brin88] and finite labeled transition systems [Fuji91b, Fuji91c], but they are not applicable to testing nondeterministic state machines where every transition is associated with an input/output pair. Furthermore, several results have been reported on test generation for either partially-specified deterministic machines [Petr91, Evtu89], or completely-specified nondeterministic machines [Luo89, Trip92, Kloo92]. The methods given in [Luo89, Trip92, Kloo92] are all based on the generalization of unique I/O sequences [Sabn85], even when applied to FSMs, a specific class of NFSMs, they still cannot guarantee full fault coverage, although full fault coverage for FSMs can be assured by many other methods. The reason is the same as pointed out in [Voun89]. Therefore, they have limited fault detection power. Furthermore, no work on test generation for *both* partially-specified *and* nondeterministic finite machines has been reported.

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), and completely-specified nondeterministic finite state machines (NFSMs), 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 introduce a conformance relation, called *quasi-equivalence*, for PNFSMs. The relation is defined in terms of input/output traces in accordance with black-box testing strategy. When the relation is applied to FSMs, NFSMs and PFSMs, 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 testing.

Guided by the conformance relations, in Section 3, we come out with a method for generating test cases from PNFSMs. We first transform a PNFSM to an equivalent one that has a lower degree of nondeterminism, called *observable* PNFSM (OPNFSM). 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. We then generate test suites from the resulting OPNFSM by a method which we call *Harmonized State Identification method* (HSI-method). As an example, we finally apply the method to generate a test suite for a communication protocol, called *Inres* [MUTE92], within the remote testing architecture.

In Section 4, we compare our method with other test generation methods, on the basis of applicability, fault coverage and the size of test suites. The main advantage of our method over the other methods is its broadest applicability with 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. We also discuss the application of the method to generating test cases for specifications written in SDL or ESTELLE.

2. NOTATIONS AND ABSTRACT TESTING FRAMEWORK

We first give in this section the definition of PNFSMs, then present conformance relations for PNFSMs 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 introduce additional notations for PNFSMs similar to that for labeled transition systems [Brin88, Fuji91b, Fuji91c]; we also define several specific classes of PNFSMs.

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:

$$h : \mathcal{D} \Rightarrow \text{powerset}(St \times Lo) \setminus \{\varnothing\} \qquad \text{where}$$

(i) $\mathcal{D} \subseteq St \times Li$ (PNFSM becomes completely specified if $\mathcal{D} = St \times Li$);

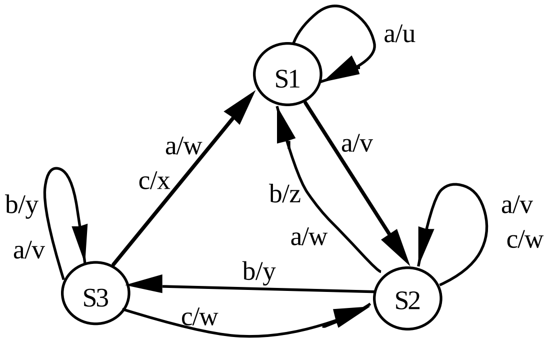
(ii) $\varnothing$ denotes the empty set.

Let $P, Q \in St$, $a \in Li$ and $b \in Lo$. We write P-a/b->Q to denote $(Q, b) \in h(P,a)$; 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.



Li = { a, b, c }

Lo = {u, v, w, x, y, z}

the initial state is S1.

Figure 1. An example of a partial NFSM

For a PNFSM, if no two outgoing transitions from the same state have the same input, then the machine is deterministic; and we call it a partial FSM (PFSM).

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

Table 1. Notation for PNFSMs

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 sequences over L; x denotes such a sequence.
	Note that $\varepsilon \in L^*$
P\-u->	For P, Q $\in$ St, not($\exists$Q(P-u->Q))
P=ε=>Q	P=Q
P=a/b=>Q	P-a/b->Q
P=x=>Q	$\exists P_1, ..., P_{k-1} \in St$ (P=P_0=u_1=>P_1...=u_k=>P_k=Q)
	where $u_{1,...,}u_k \in L$, and $x=u_1...u_k$
P=x=>	$\exists Q \in St$ (P=x=>Q)
Tr(P)	*Tr*(P)={ x | P=x=>}
x^{in}	For $x \in L^*$, x^{in} is an input sequence obtained by deleting all outputs in x
	(note that $x^{in} \in Li^*$)
V^{in}	For $V \subseteq L^*$, $V^{in}=\{x^{in} \mid x \in V\}$
Tr^{in}(P)	Tr^{in}(P)={ x^{in} | 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

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

In initially connected PNFSMs, every state is reachable from the initial state.

Without loss of generality, we assume that all PNFSMs considered in the rest of the paper are initially connected. If a given PNFSM 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 of the machines.

We now define several specific classes of PNFSMs, which are useful concepts for test generation. 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, $S\text{-}a/b\text{->}S_1$ & $S\text{-}a/b\text{->}S_2 \Longrightarrow S_1 = S_2$. □

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.

DEFINITION: *Reduced PNFSMs* :
An PNFSM is *reduced* iff $\forall S_i, S_j \in St\ (i \neq j \Longrightarrow Tr(S_i) \neq Tr(S_j))$. □

A PNFSM 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 $S_i \not\equiv S_j$, iff

$$\exists x \in Tr(S_i) \oplus Tr(S_j) \ \ (x^{in} \in Tr^{in}(S_i) \cap Tr^{in}(S_j))$$

where $Tr(S_i) \oplus Tr(S_j) = (Tr(S_i) \cup Tr(S_j)) \setminus (Tr(S_i) \cap Tr(S_j))$.

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

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 both of them.

DEFINITION: *Minimal PNFSMs:*

A PNFSM is *minimal* iff $\forall S_i, S_j \in St \ \ (i \neq j \Longrightarrow S_i \not\equiv S_j)$. □

A PNFSM is minimal if and only if every pair of states are distinguishable. A minimal PNFSM is reduced, but a reduced PNFSM is not necessarily minimal. Given a minimal machine S, each state is distinguishable from all other states; however, this is not necessarily true for a reduced machine. If we consider a completely specified machine, then a reduced machine is also minimal. The OPNFSM shown in Figure 1 is reduced, but not 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 $V \in Li^*$,

$pref(V) = \{t1 \mid t2 \in Li^* \ \& \ t1.t2 \in V \ \& \ t1 \neq \varepsilon\}$ 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 $|$ 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 for PNFSMs

Before any study on how to generate test suites for PNFSMs, 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 ? There are several conformance relations defined in the literature for FSMs, PFSMs and NFSMs. However, no conformance relation has been reported for PNFSMs.

Generalizing the conformance relations for FSMs, PFSMs and NFSMs on the basis of intuitive notions, we will define in this section conformance relations for PNFSMs in terms of the relations between their initial states.

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.

DEFINITION *Equivalence*:

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

P$\equiv$Q, holds iff *Tr*(P) =*Tr*(Q)

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

We say that an implementation I is *equivalent* to its specification S if and only if $S \equiv I$. The above definition is similar to that in [Fuji91, Chow78, Vasi73, Gill62], but it can also be applied to PNFSMs. The above relation is an equivalence relation since it is reflective, transitive and symmetric. It corresponds to the equivalent relation between NFSMs given in [Star72].

We now explain the intuitive notions for defining a conformance relation for partial machines. We say that a state machine is partial if its behavior function is not defined for all state/input combinations. The behavior function of a partial machine may not be completely specified for certain reasons. There are two basic interpretations for such an undefined state/input combination, namely "don't care" and "forbidden".

In the case of "don't care" interpretation, an undefined state/input combination means that the specification allows any further behavior of an implementation starting from a certain state under a certain input. Since an implementation can always be represented by a completely specified machine it actually completes a given partially specified machine. In other words, a partial machine represents a set of completely specified machines, and its implementation is required to conform to one of these machines.

In the second interpretation, an undefined state/input combination means that the input in the combination cannot be applied to the state, i.e., a transition cannot be executed, due to limitations imposed by the environment. For example, it is impossible to send data to a protocol machine via a connection until it has accepted this connection. Undefined "forbidden" state/input combinations will never occur in real executions. Thus, any method for executable test suite derivation should not consider these combinations.

Both interpretations require that the external behavior of an implementation is equal to that of its specification only for all those input sequences that can be accepted by a specification, instead of all possible sequences. For PFSMs (a specific class of PNFSMs), a conformance relation, called *quasi-equivalence*, was presented in [Petr91, Star72, Gill62], which is in accordance with the above intuitive notions. 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.*

Guided by the same intuitive notions, we generalize the quasi-equivalence to PNFSMs by requiring that, *for every input sequence that can be accepted by a specification, the specification and its implementation produce the same set of output sequences.* We formally define the generalized quasi-equivalence as follows.

DEFINITION *Quasi-equivalence*:

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

$P \leq_{quasi} Q$, holds iff

(a) $Tr(P) \subseteq Tr(Q)$, and

(b) $\forall\, x \in Tr(Q)\ (\, x^{in} \in Tr^{in}(P) \Longrightarrow x \in Tr(P)\,)$

Given two PNFSMs S and I with their initial states S_0 and I_0, we write $S \leq_{quasi} I$ (i.e., implementation I is *quasi-equivalent to* its specification) iff $S_0 \leq_{quasi} I_0$. □

In some situations [Boch92, Cern92], a weaker conformance relation, called *trace-inclusion*, is needed, which requires that the implementations accept all the input/output sequences that can be accepted by their specifications.

DEFINITION *Trace-inclusion*:

The *trace-inclusion* relation between two states P and Q in PNFSMs, written

$P \leq_{trace} Q$, holds iff $Tr(P) \subseteq Tr(Q)$, Given two PNFSMs S and I with their initial states S_0 and I_0, we write $S \le_{trace} I$ iff $S_0 \le_{trace} I_0$. □

It is easy to prove that the quasi-equivalence and trace-inclusion relation are reflective and transitive. Therefore, they are preorders.

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) $S \equiv I \iff S \le_{trace} I \;\&\; I \le_{trace} S$

(ii) $S \equiv I \iff S \le_{quasi} I \;\&\; I \le_{quasi} S$

(iii) $S \le_{quasi} I \implies S \le_{trace} I$

(iv) if S and I are (completely-specified) NFSMs, then $S \le_{quasi} I \iff S \equiv I$

(v) if S and I are (deterministic) PFSMs, then $S \le_{quasi} I \iff S \le_{trace} I$

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

$S \le_{quasi} I \iff S \equiv I \iff S \le_{trace} I$. □

The above theorem is evident from the corresponding definitions.

It is well-known that any nondeterministic finite automaton where each transition is associated with a single symbol (not with an I/O pair) can be modeled by an equivalent deterministic automaton [Hopc79]. However, nondeterministic finite state machines, where each transition is associated with an I/O pair, cannot be modeled by equivalent deterministic finite state machines. For example, in a NFSM with S_0-a/b-> and S_0-a/c->, we have $\{a/b, a/c\} \subseteq Tr(S_0)$. On the other hand, no deterministic FSM has $\{a/b, a/c\} \subseteq Tr(S_0)$. Therefore, nondeterministic finite state machines, in general, cannot be transformed to equivalent deterministic finite state machines for test generation.

2.3. Definitions related to testing

We define in this section several concepts which are related to tesing nondeterministic 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 $t \in Tr^{in}(S_0)$.

A *test suite* is a finite set of test cases. □

DEFINITION : *Trace-inclusion* with respect to a given input set.

The *trace-inclusion* relation between two states P and Q, with respect to a given input set $\prod \subseteq Li^*$, written $P \leq_{\prod} Q$, holds iff $(V \cap Tr(P)) \subseteq Tr(Q)$

$$\text{where } V=\{x \mid x \in L^* \ \& \ x^{in} \in \prod \cap Tr^{in}(P) \cap Tr^{in}(Q)\}.$$

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

We note: $S \leq_{trace} I$ iff $\forall \prod \subseteq Li^* \ (S \leq_{\prod} I)$.

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 $\prod \subseteq Li^*$, written

$P =_{\prod} Q$, holds iff $Q \leq_{\prod} P \ \& \ P \leq_{\prod} Q$

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

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

The relation is reflective and symmetric but not transitive. We note: (i) $S \equiv I$ iff $\forall \prod \subseteq Li^*$ $(S =_{\prod} I)$, and (ii) $S \leq_{quasi} I$ iff $\forall \prod \subseteq Tr^{in}(S)$ $(S =_{\prod} I)$.

In order to test nondeterministic implementations, one usually make a so-called *complete-testing assumption*: it is possible, by applying a given input sequence to a given implementation a finite of number of times, to exercise all possible execution paths of the implementation which are traversed by the input sequence [Fuji91b, Fuji91c, Luo89]. Without such an assumption, no test suites can guarantee full fault coverage (in terms of conformance relations) for nondeterministic implementations. In practice, 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

We present in this section a test generation method for PNFSMs, called *HSI-method*. The test suites generated by the HSI-method can be used to test PNFSM implementations against their specifications with respect to the quasi-equivalence or trace-inclusion relations.

We first describe in Section 3.1 how to generate test cases for OPNFSMs, a specific class of partial machines. We then give in Section 3.2 an algorithm for transforming an arbitrary PNFSM to a trace-equivalent OPNFSM. Incorporating methods given in Sections 3.1 and 3.2, we can generate test cases for **arbitraty** PNFSMs. As an example, in Section 3.3, we apply the method to generate a test suite for a communication protocol, called *Inres*.

3.1. Test generation for OPNFSMs

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 OPNFSM, a *characterization set* is a minimal set $W \subseteq Li^*$ such that:

$\forall S_i, S_j \in St \;\; (S_i \ncong S_j \implies \exists x \in \mathit{Tr}(S_i) \oplus \mathit{Tr}(S_j) \; (x^{in} \in \mathit{Tr}^{in}(S_i) \cap \mathit{Tr}^{in}(S_j) \cap W)).$ □

The above definition is generalized from the concept of the characterization set for FSMs given in [Chow78] to PNFSMs. The W-set is used to identify states in a given machine. An algorithm of generating characterization sets is given in Appendix II.

We find, however, that it is not neccesary to use the whole characterization set for state identification. We only use the subsets of this set, called *harmonized state identification sets*, for state identification.

DEFINITION: *Harmonized state identification sets* $\{D_0, D_1, ..., D_{n-1}\}$:

Given an OPNFSM with n states, $\{D_0, D_1, ..., D_{n-1}\}$ is a tuple of *harmonized state identification sets* if, for i=0, 1, ..., n-1, D_i is a minimal set such that

(i) $D_i \subseteq \mathit{Tr}^{in}(S_i) \cap \mathit{pref}(W)$, and

(ii) for j=0, 1, ..., n-1, $S_i \ncong S_j, \implies \exists x \in \mathit{Tr}(S_i) \oplus \mathit{Tr}(S_j) \; (x^{in} \in \mathit{pref}(D_i) \cap \mathit{pref}(D_j)).$ □

For the OPNFSM shown in Figure 1, $D_1=D_2=D_3=\{a.b\}$. An algorithm of generating harmonized state identification sets is given in Appendix II.

DEFINITION: *subscripts*(A) for a given state set:

For $A \subseteq St$, *subscripts*(A) is a string of integers i1, i2, ..., ik,

where $i1 < i2 < ... < ik$ and $A=\{S_{i1}, S_{i2}, ..., S_{ik}\}$. □

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 OPNFSMs. The states in a given set 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 in 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 an OPNFSM and a state $S_i \in St$, $f(S_i)$ is defined as a set $A \subseteq St$ such that:

(i) $S_i \in A$, and

(ii) $\forall S_k, S_j \in A \ (k \neq j \Longrightarrow S_k \ncong S_j)$, and

(iii) there is no $B \subseteq St$ such that

(i') $S_i \in B$, and

(ii') $\forall S_k, S_j \in B \ (k \neq j \Longrightarrow S_k \ncong S_j)$, and

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

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

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

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

Given an OPNFSM, we have $\delta = |\{f(S_i) \mid S_i \in St\}|$. □

According 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 OPNFSM. A fuzziness degree δ of a given OPNFSM influences the size of test suites and lengths of test cases.

DEFINITION : *Prime machine*:

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

We give in the following the test generation algorithm, which we call *Harmonized State Identification method* (HSI-method). This algorithm requires that the user previously estimates an *upper bound* on the number of states in the prime machine of the given NFSM implementation.

ALGORITHM 1: Test generation.

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

Output : A test suite Π.

Step 1: Determine the fuzziness degree δ of S.

Step 2: Let the number of states in S be n ($n \leq \delta m$). Find a set of harmonized state identification sets $\{D_0, D_1, ..., D_{n-1}\}$ from S.

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

Step 4: Construct a test suite Π such that:

$$\Pi = \bigcup_{\substack{S_0=x=>Si\ \& \\ x^{in}\in \mathbb{Q}.(\{\varepsilon\}\cup Li...\cup Li^{\delta m-n+1})}} \{x^{in}\}.D_i \qquad \Box.$$

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.

As an example, we derive a test suite $\prod$ for the PNFSM given in Figure 1 as follows:

$\mathbb{Q}=\{\varepsilon, a, a.b\}$, $D_1 = D_2 = D_3 = \{a.b\}$, $f(S_1)=\{S_1,S_3\}$, $f(S_2)=f(S_3)=\{S_2,S_3\}$, $\delta = 2$.

Assume that the prime machines of implementations do not have more than 2 states (i.e., m=2); then, we have $n \le \delta m$. We note that a test suite could be reduced by deleting each test case that is a prefix of another test case. The final test suite is as follows:

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

We note that a reset must be issued before the execution of each test case.

THEOREM 2: (Validity of the test generation method):
Consider a given specification S in the form of an OPNFSM, and any NFSM I. Suppose $n \le \delta m$ 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 the following:

(i) $S \le_{quasi} I$ iff $S =_{\prod} I$; (ii) $S \le_{trace} I$ iff $S \le_{\prod} I$.

Proof : (i) follows from Lemmas given in Appendix I. We omit the proof of (ii) since it is similar to the proof for (i). □

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 minimal forms for test generation.

In practical application, state machines that represent implementations, are always completely specified. Therefore, for a given OPNFSM specification S and a given test suite $\prod$, if the complete-testing assumption is satisfied by a given implementation NFSM I, then the relations "$I =_{\prod} S$" and "$S \leq_{\prod} I$" can be checked by repeatedly applying every test case to I a sufficient number of times. Thus, according to Theorem 2, the test suites generated by Algorithm 1 can be used to test NFSM implementations against their specification with respect to the quasi-equivalence or trace-inclusion relations.

3.2. Equivalent transformation to obtain OPNFSMs

We now present an algorithm to construct an equivalent OPNFSM from a given PNFSM. Combined with this algorithm, the test generation method given in Section 3.1 can be used to generate test cases for an arbitrary PNFSM.

ALGORITHM 2: Constructing an equivalent OPNFSM.

Input : A PNFSM S.

Output : An OPNFSM S'.

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

Step 2: If there is no unmarked node in the resulting graph G, then stop; G represents the OPNFSM S', and the node $\{S_0\}$ represents the initial state of S'. Otherwise,

(a) Find and mark an unmarked node M in G, where the label $M \subseteq St$;

(b) For every $u \in L$, first construct $M' = \{P' \mid P \in M\ (P = u \Rightarrow P')\}$. Then, if M' is not a node label in the resulting graph G, create an unmarked node with label M' and a directed edge from M to M' with label u; go to Step 2. □

The Figure 2 shows an example of using the above algorithm to construct an equivalent OPNFSM from a given partial machine.

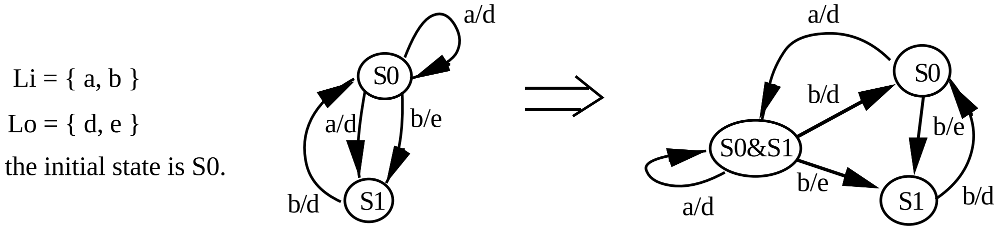


Figure 2. Transformation to obtain an equivalent OPNFSM

3.3. Test generation for the Inres protocol

As an application example of using the HSI-method to generate test suites, we consider the Inres protocol (Initiator-responder protocol) from [MUTE92] which has already been used as a reference in many publications. Under the ISO remote testing architecture, we construct a NFSM for the system under test which consists of a Responder and a User, as shown in Figure 3. An FSM model of the Responder can be easily constructed from the state tables [Kroo92] and is not presented in this paper. The nondeterministic model shown in Figure 4 is assumed for the User. The user may disconnect by sending IDISreq only in response to an ICONind or IDATind. We assume that there is a certain control over the User's behavior in such a way that, during the test campaign, the User executes each option sufficiently often.

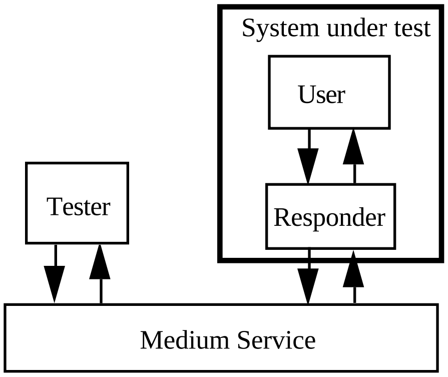


Figure 3. Remote testing of Inres-responder

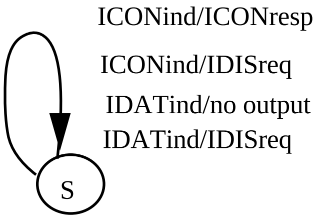
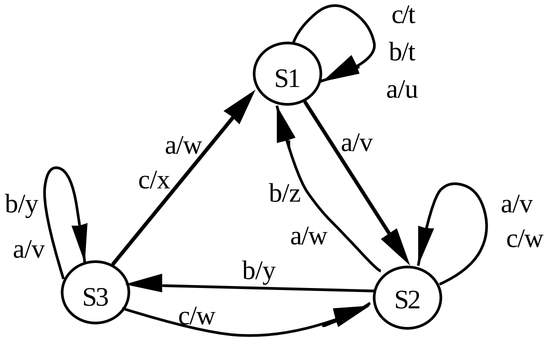


Figure 4. User's behavior

The behavior of the system under test is described by the completely specified minimal ONFSM with three states shown in Figure 5. Interpretation of inputs, outputs and states is given in Table 2.



the initial state is S1.

Figure 5. NFSM for the system under test

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 Π as follows:

$\mathbb{Q}=\{\varepsilon, a, a.b\}$, $D_1 = D_2 = D_3 = \{b\}$, $\delta = 1$

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

4. COMPARISON WITH OTHER RELATED WORK

Since FSMs, NFSMs and PFSMs are specific classes of PNFSMs, the HSI-method can be applied to them, to test the equivalence and quasi-equivalence relations, respectively (see Theorem 1). We compare in this section our HSI-method for PNFSMs with the other test generation methods for different machines [Fuji91, Vuon89, Sabn85, Nait81, Chow78, Vasi73, Petr91, Petr92, Trip92, Kloo92, Luo89], which also require a "reliable" reset in the implementations (note, that simple experiments or checking sequences do not use this assumption). The main advantage of the HSI-method over the other test generation methods is its broader applicability with full fault coverage (w.r.t. conformance relations), as shown in Figure 6.

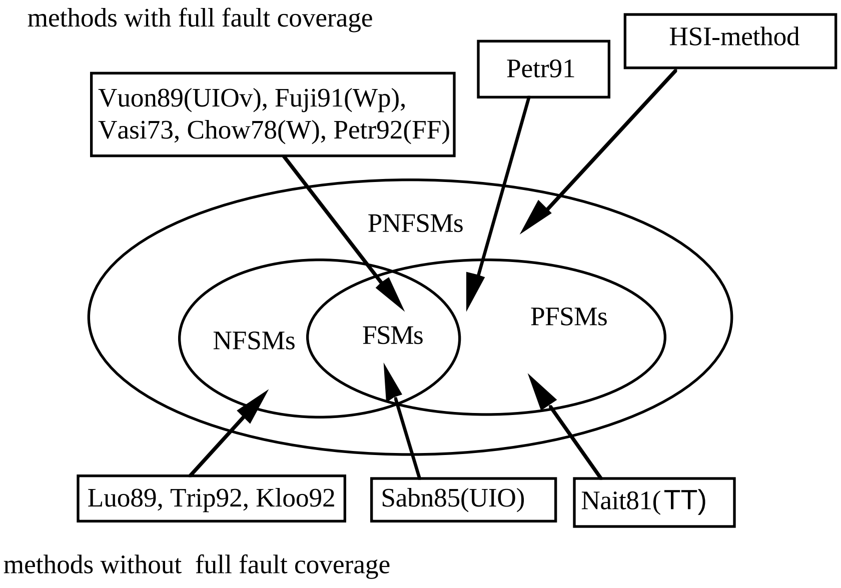

methods with full fault coverage
Vuon89(UIOv), Fuji91(Wp),
Vasi73, Chow78(W), Petr92(FF)
Petr91
HSI-method
PNFSMs
NFSMs
FSMs
PFSMs
Luo89, Trip92, Kloo92
Sabn85(UIO)
Nait81(TT)
methods without full fault coverage

Figure 6. General comparison based on applicability and fault coverage

4.1. Pure FSMs

When the HSI-method is applied to FSMs, the conformance relation to be checked 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. Since the W-, Wp- and HSI- methods detect all faults that may even increase the state number up to the given bound, we need to compare our method with W- and Wp- methods only.

We first describe the W- and Wp- methods in our formalism. These methods assume that specifications are minimal (completely-specified) FSMs. We note that an FSM is minimal if and only if it is reduced.

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

$$\text{for } j=0, 1, ..., n-1, \Big(j\neq i \implies \exists x\in Tr(S_i)\oplus Tr(S_j) \; (x^{in}\in W_i) \Big). \;\square$$

The test suite generated by the W-method is $\prod_W = \bigcup_{\substack{S_0=x=>S_i \ \& \\ x^{in}\in \mathbb{P}.(\{\varepsilon\}\cup Li...\cup Li^{m-n})}} \{x^{in}\}.W$

where $\mathbb{P}= \mathbb{Q}.(\{\varepsilon\}\cup Li)$, and $\mathbb{Q}$ is constructed according to Step 3 of Algorithm 1.

The test suite generated by the Wp-method is

$$\prod_{Wp} =\prod 1 \cup \prod 2 \quad \text{where} \quad \overline{\mathbb{W}} = W_0\cup W_1\cup ...\cup W_{n-1}$$

$$\prod 1 = \bigcup_{\substack{S_0=x=>S_i \ \& \\ x^{in}\in \mathbb{Q}.(\{\varepsilon\}\cup Li...\cup Li^{m-n})}} \{x^{in}\}.\overline{\mathbb{W}}$$

$$\Pi 2 = \bigcup_{\substack{S_0=x=>Si\ \ \& \\ x^{in}\in(\mathbb{P}\backslash\mathbb{Q}).(\{\varepsilon\}\cup Li\ldots\cup Li^{m-n})}} \{x^{in}\}.W_i$$

For reduced FSMs, since δ=1, the test suite generated by the HSI-method is

$$\Pi = \bigcup_{\substack{S_0=x=>Si\ \ \& \\ x^{in}\in\mathbb{P}.(\{\varepsilon\}\cup Li\ldots\cup Li^{m-n})}} \{x^{in}\}.D_i$$

We note that $D_i \sqsubseteq \overline{\mathbb{W}}$, i=0,1,..., n-1, but $W_i \sqsubseteq D_i$. Therefore, neither the Wp-method nor the HSI-method necessarily produces smaller test suites than the other. For a given a characterization set W, there must be a set of harmonized state identification sets $\{D_0, D_1, ..., D_{n-1}\}$ such that $D_i \sqsubseteq W$, i=0,1,..., n-1. It is easy to see $|\Pi| \le |\Pi_W|$; that is, the HSI-method produces usually smaller (but never larger test suites) than the W-method.

4.2. Partial FSMs

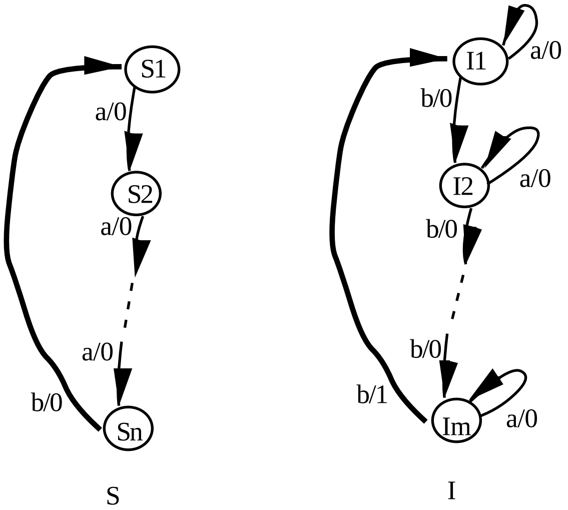
Test generation for partial FSMs has received much less attention than that for completely-specified FSMs. However, practical communication software is often modeled as partial machines. Some authors proposed to complete the "don't care" state/input combinations of partial machines in accordance with 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. Therefore, the completeness assumption is not always satisfied. Methods for test suite generation from a deterministic partial FSM were proposed in [Evtu89, Petr91]. The HSI-method combines the ideas of these methods with the concept of harmonized state identifiers, and further generalizes them to nondeterministic machines.

4.3. Nondeterministic FSMs

When we consider completely specified, nondeterministic FSMs, the conformance relation to be checked is the equivalence. In this context, some test generation methods for NFSMs based on UIO-sequences have been presented [Luo89, Trip92, Kloo92]. However, these methods cannot guarantee full fault coverage (i.e., equivalence). The reason is the same as pointed out in [Voun89]. Therefore, they have limited fault detection power. The main advantage of the HSI-method over these methods is that it guarantees full fault coverage.

5. CONCLUSION

We present in this paper a uniform method, called the HSI-method, for generating test suites from different types of state machines, ranging from pure FSMs to arbitrary partially-specified, even nonminimal, nondeterministic finite state machines. Unfortunately, if a given OPNFSM is not minimal and its fuzziness degree δ is more than one, then the lengths of test cases produced by the HSI-method grow rapidly when δ increases.



S1 and I1 are initial states of S and I, respectively.

Figure 7. An example of the worst case

Let n and m be the numbers of states in a specification and its implementation, respectively. In the extreme case, when δ =n, the length of a test case can reach n×m. This upper bound holds even in the deterministic case. As shown in [Evtu89], for each pair n and m, there is a partial FSM with n states, for which the shortest test case has the length n×m for its implementations with no more than m states. Figure 7 gives an example of such machines, where δ=n for the PFSM S. Since $((a/0)^{n-1}.b/0)^m \in Tr(S)$ and $((a/0)^{n-1}.b/0)^m \notin Tr(I)$, we have not($S \leq_{trace} I$). It is easy to see that $(a^{n-1}.b)^m$ is the shortest test sequence of the length of n×m for distinguishing the two machines.

In spite of this bound, the HSI-method yields much smaller test suites for states machines that are less fuzzy. As to the size of test suites produced by the HSI-method, its order is $O(n^3|Li|^{\delta m-n+1})$.

This method can be applied to test generation for the control part of specifications written in SDL or ESTELLE. In such cases, we can first abstract SDL processes or ESTELLE modules to PNFSMs by neglecting parameters; we then apply the test generation method for the resulting PNFSMs. In the situation of testing concurrent programs specified in SDL or ESTELLE, even though individual processes are deterministic, the whole system usually is nondeterministic; therefore, there is a need for methods to test nondeterministic machines. As far as implementation of test generation tools is concerned, the advantage of our method is that we need to implement only one test generation method -- the HSI-method -- for PNFSMs, instead of implementing several individual methods for FSMs, PFSMs and NFSMs since they are specific cases of partially-specified nondeterministic finite state machines.

APPENDIX I: 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 an OPNFSM 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) S has n states with $n \geq 2$.
(2) the fuzziness degree of S is δ.
(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 tuple of harmonized state identification sets of S is $\{D_0, D_1, ..., D_{n-1}\}$.

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

(7) a test suite $\prod$ is constructed such that:

$$\prod = \bigcup_{\substack{S_0=x=>Si\ \& \\ x^{in}\in\mathbb{Q}.(\{\varepsilon\}\cup Li...\cup Li^{\delta m-n+1})}} \{x^{in}\}.D_i = \mathbb{Q}.(\{\varepsilon\}\cup Li\cup ...\cup Li^{\delta m-n+1})\otimes\{D_0, D_1, ..., D_{n-1}\}$$

where for $V\subseteq Li^*$, $V\otimes\{D_0, D_1, ..., D_{n-1}\}= \bigcup_{\substack{S_0=x=>Si \\ \&\ x^{in}\in V}} \{x^{in}\}.D_i$.

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 $V\subseteq Li^*$ $[S_i,M_i]$-*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*-L*
$\mathbb{D}$r	$\mathbb{D}r=\{[S_i,M_j]\mid[S_i,M_j]\in\mathbb{D}\ \&\ S_i=_{Di}M_j\}$
$\overline{Li}^k$	$\overline{Li}^k=\{\varepsilon\}\cup Li\cup ...\cup Li^k$, when $k\geq1$; and $\overline{Li}^0=\{\varepsilon\}$.

It is easy to see $\mathbb{D}r\subseteq\mathbb{D}$ and $|\mathbb{D}r|\leq|\mathbb{D}|\leq n\times m$. Since 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 $V\subseteq Li^*$, assume $|[S_0,M_0]$-*after*-$V|\geq k$.
If $|\mathbb{D}|>k$, then $|[S_0,M_0]$-*after*-$V.(\{\varepsilon\}\cup Li)|\geq k+1$; and if $|\mathbb{D}|\leq k$, then
$[S_0,M_0]$-*after*-$V.(\{\varepsilon\}\cup Li) = [S_0,M_0]$-*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)	$\mid\mathbb{D}\mid > k$	hypothesis
(2)	$\mid[S_0,M_0]$-*after*-$V\mid=k$	hypothesis
(3)	$[S_0,M_0]$-*after*-$V\subseteq\mathbb{D}$	definition of $\mathbb{D}$
(4)	$\exists[S_i,M_i]\in\mathbb{D}\backslash[S_0,M_0]$-*after*-V	(1) & (2) & (3)
(5)	$\exists[S_{k-1},M_{k-1}]\in[S_0,M_0]$-*after*-V $\exists[S_k,M_k],[S_i,M_i]\in\mathbb{D}\backslash[S_0,M_0]$-*after*-V $\exists u\in L\ \exists x,y\in L^*$ such that: $x^{in}\in V\ \&$ $([S_0,M_0]=x=>[S_{k-1},M_{k-1}]$-u->$[S_k,M_k]=y=>[S_i,M_i]$	(4)

(6) $\exists[S_k,M_k]\in([S_0,M_0]\text{-}\mathit{after}\text{-}V.(\{\varepsilon\}\cup Li))\backslash[S_0,M_0]\text{-}\mathit{after}\text{-}V$ (5)

(7) $|[S_0,M_0]\text{-}\mathit{after}\text{-}V.(\{\varepsilon\}\cup Li)|\geq k+1$ (6)

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

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

(2) $|[S_0,M_0]\text{-}\mathit{after}\text{-}V|\geq k$ hypothesis

(3) $[S_0,M_0]\text{-}\mathit{after}\text{-}V\subseteq\mathbb{D}$ definition of $\mathbb{D}$

(4) $[S_0,M_0]\text{-}\mathit{after}\text{-}V.(\{\varepsilon\}\cup Li) = [S_0,M_0]\text{-}\mathit{after}\text{-}V$ (1) & (2) &(3). □

LEMMA 2: Assume $S_0=_{\mathbb{Q}}M_0$. If $|\mathbb{D}|>\delta\times m$, then $|[S_0,M_0]\text{-}\mathit{after}\text{-}\mathbb{Q}.\overline{L}i^{\delta m\text{-}n}|\geq\delta\times m$;
and if $|\mathbb{D}|\leq\delta\times m$, then $[S_0,M_0]\text{-}\mathit{after}\text{-}\mathbb{Q}.\overline{L}i^{\delta m\text{-}n} = \mathbb{D}$.

Proof:

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

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

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

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

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

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

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

LEMMA 3: If $S_i=_{Di} M_k$, then $S_i\not\equiv S_j \Longrightarrow$ not($S_j=_{Dj}M_k$)

Proof:

(0) for $V\subseteq Li^*$, $S_i=_V M_k$ iff $S_i=_{\mathit{pref}(V)} M_k$ evident

(1) $S_i=_{Di} M_k$ hypothesis

(2) $S_i\not\equiv S_j$ hypothesis

(3) $S_j=_{Dj} M_k$ assumption

(4) $S_i=_{\mathit{pref}(Di)} M_k$ (0) & (1)

(5) $S_j=_{\mathit{pref}(Dj)} M_k$ (0) & (3)

(6) $\exists x\in \mathit{Tr}(S_i)\oplus\mathit{Tr}(S_j)\ (x^{in}\in\mathit{pref}(D_i)\cap\mathit{pref}(D_j))$ definition of D_i & (2)

(7) let x be a sequence such that
$x\in\mathit{Tr}(S_i)\oplus\mathit{Tr}(S_j)(x^{in}\in\mathit{pref}(D_i)\cap\mathit{pref}(D_j))$
in the following making a definition based on (6)

(8) $x\in\mathit{Tr}(S_i)\backslash\mathit{Tr}(S_j)$ or $x\in\mathit{Tr}(S_j)\backslash\mathit{Tr}(S_i)$ (7)

(9) $x^{in}\in\mathit{Tr}^{in}(M_k)$ The NFSM M is completely specified

(10) $x\notin\mathit{Tr}(S_i)\backslash\mathit{Tr}(S_j)$ (4) & (5) & (7) & (9)

(11) $x\notin\mathit{Tr}(S_j)\backslash\mathit{Tr}(S_i)$ (4) & (5) & (7) & (9)

(12) (8) is not true (10) & (11)

(13) not($S_j=_{Dj} M_k$) (3) causes the contradiction between (8) and (12).□

LEMMA 4: For $S_i\in St_S$, $|\{[S_k,M_k]|\ [S_k,M_k]\in\mathbb{D}r\ \&\ S_k\in f(S_i)\}|\leq m$

Proof:

Let $\mathbb{D}_i = \{[S_k,M_k] \mid [S_k,M_k] \in \mathbb{D}r \ \& \ S_k \in f(S_i)\}$.

(1) $|St_M| = m$ hypothesis

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

(3) $\exists [S_j,M_k],[S_l,M_k] \in \mathbb{D}_i$ ($j \neq l$ & $S_j =_{Dj} M_k$ & $S_l =_{Dl} M_k$) (1) & (2)

(4) (3) is not true Lemma 3

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

LEMMA 5: $|\mathbb{D}r| \leq \delta \times m$.

Proof:

Let $E = \{f(S_i) \mid S_i \in St_S\}$, and $\mathbb{D}_i = \{[S_k,M_k] \mid [S_k,M_k] \in \mathbb{D}r \ \& \ S_k \in f(S_i)\}$.

(1) $\delta = |E|$ definition of δ

(2) $\mathbb{D}r \subseteq \bigcup_{f(Si) \in E} \mathbb{D}_i$ definition of $\mathbb{D}r$

(3) $\forall f(S_i) \in E \ (|\mathbb{D}_i| \leq m)$ Lemma 4

(4) $|\mathbb{D}r| \leq \sum_{f(Si) \in E} |\mathbb{D}_i|$ (2)

(5) $|\mathbb{D}r| \leq \delta \times m$ (1) & (3) & (4). □

LEMMA 6: If $S_0 =_{(\mathbb{Q}.\overline{L}i^{\delta m-n}) \otimes \{D_0,\ldots,D_{n-1}\}} M_0$, then $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n} = \mathbb{D}r$.

Proof:

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

(1) $S_0 =_{(\mathbb{Q}.\overline{L}i^{\delta m-n}) \otimes \{D_0,\ldots,D_{n-1}\}} M_0$ hypothesis

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

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

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

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

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

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

LEMMA 7: If $S_0 =_{\Pi} M_0$, then $[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n+1} = \mathbb{D}r = \mathbb{D}$.

Proof:

When $|\mathbb{D}| \leq \delta \times m$, the lemma is evident from Lemma 2. Now we argue that $|\mathbb{D}| > \delta \times m$ does not hold if $S_0 =_{(\mathbb{Q}.\overline{L}i^{\delta m-n+1}) \otimes \{D_0,\ldots,D_{n-1}\}} M_0$ holds.

(1) $S_0 =_{(\mathbb{Q}.\overline{L}i^{\delta m-n+1}) \otimes \{D_0,\ldots,D_{n-1}\}} M_0$ hypothesis

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

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

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

(5) $|[S_0,M_0]$-*after*-$\mathbb{Q}.\overline{L}i^{\delta m-n+1}| \leq |\mathbb{D}r| \leq \delta \times m$ (4) & Lemma 5

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

LEMMA 8: If $S_0 =_{\prod} M_0$, then

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

Proof: $S_0 =_{\prod} M_0$ implies $S_0 =_{(\mathbb{Q}.\overline{L}i^{\delta m\text{-}n}) \otimes \{D_0,...,D_{n-1}\}} M_0$. Then, because of $S_0 =_{\prod} M_0$ and $S_0 =_{(\mathbb{Q}.\overline{L}i^{\delta m\text{-}n}) \otimes \{D_0,...,D_{n-1}\}} M_0$, the lemma follows Lemmas 6 and 7. □

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

Proof:

(I) To show $Tr(S_0) \subseteq Tr(M_0)$.

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

(2) $\forall x \in L^*($ if $x^{in} \in \mathbb{Q}.\overline{L}i^{\delta m\text{-}n}$ 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 8 & "S, M are observable"

(3) not ($Tr(S_0) \subseteq Tr(M_0)$) assumption

(4) $\exists y \in L^* \ \exists u \in L \ \exists [S_i,M_i] \in \mathbb{D}r$ such that $y^{in} \in \mathbb{Q}.\overline{L}i^{\delta m\text{-}n}$ &
$[S_0,M_0]$=y=>$[S_i,M_i]$ & S_i-u-> & M_i\-u-> (3) & (1) & Lemma 8 & "S, M are observable"

(5) $Tr(S_0) \subseteq Tr(M_0)$ (3) causes the contradiction between (2) and (4)

(II) To show $\forall \ x \in Tr(M_0)$ ($x^{in} \in Tr^{in}(S_0) \implies x \in Tr(S_0)$)

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

(2) $\forall x \in L^*($ if $x^{in} \in \mathbb{Q}.\overline{L}i^{\delta m\text{-}n}$ 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 8 & "S, M are observable"

(3) not ($\forall \ y \in Tr(M_0)$ ($y^{in} \in Tr^{in}(S_0) \implies y \in Tr(S_0)$)) assumption

(4) $\exists w \in Tr(M_0)$ ($w^{in} \in Tr^{in}(S_0)$ & $w \notin Tr(S_0)$) (3)

(5) $\exists z \in L^* \ \exists u \in L \ \exists [S_i,M_i] \in \mathbb{D}r$ such that $z^{in} \in \mathbb{Q}.\overline{L}^{\delta m\text{-}n}$ &
$[S_0,M_0]$=z=>$[S_i,M_i]$ & S_i\-u-> & $u^{in} \in Tr^{in}(S_i)$ & M_i-u-> (4) & (2) & Lemma 8

(6) $\forall \ x \in Tr(M_0)$ ($x^{in} \in Tr^{in}(S_0) \implies x \in Tr(S_0)$)
(3) causes the contradiction between (2) and (5). □

LEMMA 10: $S_0 =_{\prod} M_0$ iff $S_0 =_{\prod} I_0$.

Proof: Since $I_0 =_{\prod} M_0$ and since both I and M are completely specified, $S_0 =_{\prod} M_0$ implies $S_0 =_{\prod} I_0$. By the same reason, $S_0 =_{\prod} I_0$ implies $S_0 =_{\prod} M_0$. □

We note that Lemmas 9 and 10 lead Theorem 2(i). The proof for Theorem 2(ii) is similar to that for Theorem 2(i).

APPENDIX II: GENERATION OF CHARACTERIZATION SET AND HARMONIZED STATE IDENTIFICATION SETS

DEFINITION: operators *partition* and *refine* **:**

For $x \in L^*$, Class$\in$powerset(St), Classes$\in$powerset(powerset(St)), we define

(1) *partition* is a function: $L^* \times$ powerset(St) --> powerset(powerset(St))

x *partition* Class = $\{A, B\}$ where A and B are decided as follows:

$$A=\{P \mid P \in Class \ \& \ (x \in \mathit{Tr}(P) \text{ or } x^{in} \notin \mathit{Tr}^{in}(P)\}$$

$$B=\{P \mid P \in Class \ \& \ (x \notin \mathit{Tr}(P)\}$$

(2) *refine* is a function: $L^* \times$ powerset(powerset(St)) --> powerset(powerset(St))

$$x \ \mathit{refine} \ Classes = \bigcup_{Class \in Classes} x \ \mathit{partition} \ Class$$

.

□

ALGORITHM : *Characterization set generation* .

Input : An OPNFSM (St, Li, Lo, h, S_0) with n states.

Output : a characterization set W.

```
variables  Class: powerset(St);
           Classes: powerset(powerset(St));
           K: integer;
           x:  L*
           V: powerset(L*)
           W: powerset(Li*)
Step 1:  Classes:={St};  k:=0; W:=∅; V:=∅.                    "∅ is an empty set"
Step 2:  REPEAT
            FOR ALL x (|x|=k & x∈L* )   DO1                   "|x|  is the length of x"
              IF |x refine  Classes| >|Classes|  THEN
                  DO2    Classes:=x  refine  Classes;    V:= V∪{x}    ENDDO2   ENDIF;
              IF   ∀Class∈Classes(|Class| =1)  THEN  GOTO Step 3   ENDIF
            ENDDO1;
            k:=k+1
         ENDREPEAT UNTIL k=n(n-1)/2;
Step 3:    W:=V^in. □
```

Given an OPNFSM (St, Li, Lo, h, S_0), for any pair of distinguishable states, there must be a $t \in Li^*$ of a length not more than $n(n-1)/2$ such that t distinguishes them [Star72]. Therefore, there must be a characterization set W such that $\forall x \in W \ (|x| \leq n(n-1)/2)$. For a completely specified minimal machines, the above algorithm terminates before k=n.

ALGORITHM : *Generation of harmonized state identification sets.*

Input : An OPNFSM (St, Li, Lo, h, S_0), a characterization set W.

Output : Harmonized state identification sets $\{D_0, D_1, ..., D_{n-1}\}$.

```
variables  i,  j: integer;
           x:  L*
           D_0, D_1, ..., D_n-1: powerset(Li*)
Step 1:   Let i:=0.  Choose a minimal D_0 such that
      (i)  $D_0 \subseteq \mathit{pref}(W) \cap \mathit{Tr}^{in}(S_0)$, and
      (ii)  for  j=1, ..., n-1, ( if  $S_j \not\equiv S_0$, $\exists x \in \mathit{Tr}(S_0) \oplus \mathit{Tr}(S_j)$  ( $x^{in} \in D_0 \cap \mathit{Tr}^{in}(S_j)$ ) ).
Step 2:   If i=n-1,  stop.  Otherwise, let i:=i+1, and choose a minimal D_i such that
```

(i) $D_i \subseteq \mathit{pref}(W) \cap \mathit{Tr}^{in}(S_i)$, and

(ii) for $j=0, 1, ..., i-1$, (if $S_j \ncong S_i$, $\exists x \in \mathit{Tr}(S_i) \oplus \mathit{Tr}(S_j)$ ($x^{in} \in \mathit{pref}(D_i) \cap \mathit{pref}(D_j)$)), and

(iii) for $j=i+1, ..., n-1$, (if $S_j \ncong S_i$,
$\exists x \in \mathit{Tr}(S_i) \oplus \mathit{Tr}(S_j)$ ($x^{in} \in D_i \cap \mathit{Tr}^{in}(S_j)$)).

Go to Step 2. □

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.

[Cern92] E. Cerny, "Verification of I/O Trace Set Inclusion for a Class of Nondeterministic Finite State Machines", ICCD'92 Conference, Cambridge, Mass., October 1992.

[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, 270p.

[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.

[ISO8807] ISO, Lotos - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, IS-8807, 1989.

[Kloo92] Hans Kloosterman, "Test Derivation from Nondeterministic Finite State Machines", Participants' Proceedings of 5th International Workshop on Protocol Testing Systems, Ed. by G.v. Bochmann, R.Dssouli and A. Das, 1992, Montreal, Canada, (to be appeared in IFIP Transactions, Protocol Testing Systems V, North-Holland), pp.254-265.

[Kroo92] J. Kroon, Inres State Tables, Private Communication, 1992.

[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.

[Luo89] 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, 1989, International Academic Publishers, pp.688-694.

[Luo91] 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.

[MUTE92] D. Hogrefe, MUTEST: OSI Formal Specification Case Study: the Inres Protocol and Service.

[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", Comput. Networks & ISDN Syst., Vol.14, 1987, pp.79-89.

[Roug89] Anne Bourguet-Rouger & Pierre Combes, "Exhaustive Validation and Test Generation in Elivis", SDL Forum'89: The Language at Work, North-Holland, 1989.

[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.

[Trip91] Piyu Tripathy and Behcet Sarikaya, "Test Generation from LOTOS Specification", IEEE Transactions on Computer, Vol C-40, No.4, 1991, pp.543-552.

[Trip92] Piyu Tripathy and Kshirasagar Naik, "Generation of Adaptive Test Cases from Nondeterministic Finite State Models", Participants' Proceedings of 5th International Workshop on Protocol Testing Systems, Ed. by G.v. Bochmann, R.Dssouli and A. Das, 1992, Montreal, Canada, (to be appeared in IFIP Transactions, Protocol Testing Systems V, North-Holland), pp.266-279.

[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.

[Witt92] M. F. Witteman, R. C. van Wuijtswinkel and S.Ruud Berkhout, "Nondeterministic and Default Behaviour", Participants' Proceedings of 5th International Workshop on Protocol Testing Systems, Ed. by G.v. Bochmann, R.Dssouli and A. Das, 1992, Montreal, Canada, (to be appeared in IFIP Transactions, Protocol Testing Systems V, North-Holland), pp.241-252.

APPENDIX III: PROPERTIES OF THE CONFORMANCE RELATIONS

LEMMA: Given three states S, P, and I, if $S \leq_{ref} P$ and $P \leq_{ref} I$, then $S \leq_{ref} I$; that is, $\leq_{ref}$ is transitive.
Proof:

Part I: It is evident that $Tr(S) \subseteq Tr(I)$ from the hypothesis.

Part II: To prove $\forall\, x \in Tr(I)\ (\, x^{in} \in Tr^{in}(S) \Longrightarrow x \in Tr(S)\ )$

(1) $S \leq_{ref} P$ ...... hypothesis

(2) $P \leq_{ref} I$ ...... hypothesis

(3) $x \in Tr(I)$ ...... hypothesis

(4) $x \in Tr(P)$ ...... (2) & (3)

(5) $x^{in} \in Tr^{in}(S)$ ...... hypothesis

(6) $x \in Tr(S)$ ...... (1) & (4) & (5)

(7) $x^{in} \in Tr^{in}(S) \Longrightarrow x \in Tr(S)$ ...... (5) & (6) & "$\Longrightarrow_{+}$ "

(8) $\forall\, x \in Tr(I)\ (\, x^{in} \in Tr^{in}(S) \Longrightarrow x \in Tr(S)\ )$ ...... (3) & (7) & "$\forall_{+}$ "

The lemma holds from Parts I and II.
[End of proof]

ALGORITHM : *State identification set generation* .
Input : A PNFSM (St, Li, Lo, h, s_0), a given state P

Output : A state identification set ID for P.

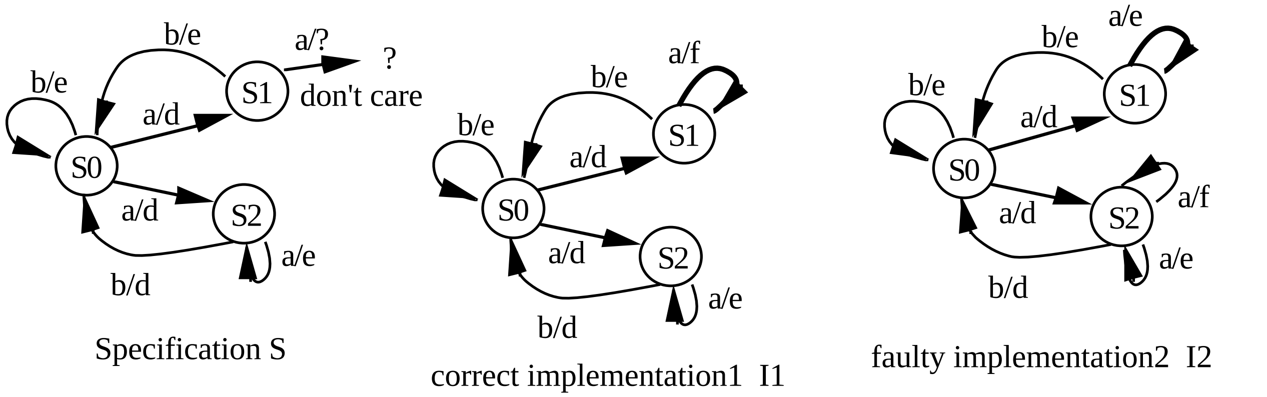
```
variables  Class: powerset(St);
           Classes: powerset(powerset(St));
           K: integer;
           x:  L*
           ID: powerset(L*)
Method:
Step 1:  Classes:={St};  k:=1; ID:=ϕ.
Step 2:  REPEAT
             FOR ALL x (|x|=k &x∈L* )  DO1
               IF ∃Class∈Classes ({P}∈Class  & |x partition  Class| >1)  THEN
                   DO2    Classes:=x refine  Classes;    ID:= ID∪{x}   ENDDO2   ENDIF;
               IF  ∀Class∈Classes ({P}∈Class ==⇒ |Class| =1)  THEN  STOP  ENDIF
             ENDDO1;   K:=K+1
           ENDREPEAT;
[End of algorithm].
```

DEFINITION : *Product graph* $\mathbb{G}$:
A *product graph* is a directed graph $\mathbb{G}$ such that:

(i) The graph $\mathbb{G}$ has $|\,\mathbb{D}\,|$ nodes. Each node is labeled a pair <Si,Mi> in $\mathbb{D}$. Different nodes have different labels.

(ii) For each pair of nodes <Si,Mi> and <Sj,Mj>, there is a directed edge from <Si,Mi> to <Sj,Mj> with a label u if and only if <Si,Mi> -u-> <Sj,Mj>.
[End of definition].

DEFINITION: *Unique state identification sets* $\{W_0, W_1, ..., W_{n-1}\}$:
Given an OPNFSM and a set of harmonized unique state identification sets $\{D_0, D_1, ..., D_{n-1}\}$, a set of *unique state identification sets* is a set of sets $\{Wi \mid Si \in St \ \& \ Wi \subseteq Di\}$ such that :
$\forall Si \in St$ ($i \neq j$ $\Longrightarrow$
if Si is distinguishable from Sj,
then $\exists x \in Wi$ ($x^{in} \in Tr^{in}(Si) \cap Tr^{in}(Sj)$ & $x \in Tr(Si) \oplus Tr(Sj)$) .
[End of definition].



Specification S

correct implementation1 I1

faulty implementation2 I2

Li = { a, b } Lo = { d, e, f } I1 cannot be distinguished from I2 by black-box testing.

Figure 2. Relation between specification and implentations