TEST GENERATION FOR CONCURRENT PROGRAMS MODELED BY COMMUNICATING NONDETERMINISTIC FINITE STATE MACHINES[1]

Gang Luo, Gregor v. Bochmann and Anindya Das

Departement d'IRO, Universite de Montreal,

C.P. 6128,Succ.A, Montreal, P.Q., H3C 3J7, Canada

e-mail:luo@iro.umontreal.ca

Fax: (514) 343-5834

ABSTRACT We present a method of generating test cases for concurrent programs. The specifications of concurrent programs are modeled by a set of communicating nondeterministic finite state machines (CNFSMs), where the CNFSM model is a simplified model derived from CCITT SDL. A conformance relation, called trace-equivalence, is defined under this model, which serves as a guide to test generation. A test generation method for a single NFSM is developed, which is generalized from the W-method. For a set of CNFSMs, the test cases are generated in the following manner: A set of CNFSMs are first transformed into a single NFSM by reachability analysis; then the test cases are generated from the resulting NFSM by using the method for a single NFSM.

KEYWORDS: CCITT SDL, concurrent programs, finite state machines, 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).

1. INTRODUCTION

Concurrency and nondeterminism are two important features of formal specification languages for communication software, in particular, communication protocols. All the three major specification languages for communication software, LOTOS [Bolo87], ESTELLE [Budk87] and SDL [Beli89] support the description of concurrency and nondeterminism -- (SDL will support nondeterminism [SDL91] in the near future). However, concurrency raises a challenge in software testing. A set of deterministic finite state machines , communicating with each other by input queues and channels as described in SDL [Beli89], may have nondeterministic behavior. Furthermore, in general, they cannot be modeled as a global finite state machine which preserves the same relationship between input and output sequences. Several heuristic approaches for concurrent program testing, based on extended finite state machines [Luo89b, Kaln91, Arak91], have been reported, although they do not provide an analysis of the corresponding fault coverage. But these approaches did not address the issue of nondeterminism.

Some work on test generation from nondeterministic models has been done in the context of basic LOTOS [Brin88, Brin89] and labeled transition systems with a finite number of states [Fuji91, Fuji91b]. However, these methods are not for testing NFSMs (Nondeterministic Finite State Machines) where every transition of NFSMs is associated with an input/output pair. A method to compare two NFSMs based on IO trace inclusion has been given in [Cern92] under the condition that the internal structures of both NFSMs are available. It, however, is not suitable for test generation for conformance testing since in this context, implementations are assumed to be black-boxes. [Luo89] presented a method of generating test cases from NFSMs; but this method is not guided by a well-defined conformance relation and has very limited fault detection power.

We present in this paper test generation methods for a single NFSM and for a system of CNFSMs (Communicating NFSMs). The motivation of our work is test generation based on SDL

specifications [SDL91]. By neglecting interaction parameters, a set of SDL processes can be abstracted to a set of CNFSMs. Test cases are then developed from these CNFSMs.

We first define, in Section 2, NFSMs, and related notations which are similar to those for labeled transition systems [Brin88, Brin89, Fuji91, Fuji91b]. Then, an abstract testing framework for NFSMs is presented including a conformance relation, fault model and test run observation. The conformance relation between implementations and specifications , which is based on I/O traces, is given in the context of the black-box-testing strategy in which implementations are viewed as black boxes.

Guided by the given conformance relation, we present, in Section 3, a method for generating test cases from NFSMs. Our method is based on extending the state identification approach in the area of test generation for deterministic FSMs (Finite State Machine) [Chow78, Sabn85, Fuji91c] to NFSMs. We first transform a given NFSM with internal actions to a trace-equivalent NFSM without internal actions. We then transform the NFSM to an equivalent NFSM which has a lower degree of nondeterminism. The NFSM obtained by the transformation is deterministic in the sense that a state and an input/output pair uniquely determine the next state, while a state and an input alone does not, in general, determine a unique next state and an output. Test cases are generated from the resulting NFSMs by an approach similar to the W-method [Chow78].

In the Section 4 we consider a system of several CNFSMs. We first show that such a system, in general, cannot be modeled as a global finite state machine, and that even a set of *deterministic* FSMs may still contain nondeterminism. We then give two test generation methods for a system of CNFSMs, one of which is based on a restriction of the channel and queue lengths and the other on an assumption for the speed at which inputs are sent by the environments. We finally discuss the detection of faults related to queues and channels.

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

2. NOTATIONS AND ABSTRACT TESTING FRAMEWORK

We first give in this section the definition of a single NFSM. We then present a conformance relation for NFSMs, inspired by the work of defining conformance relations for labeled transition systems [Brin88, Brin89]. The relation is based on the black-box testing strategy where implementations are assumed to be black boxes. We next give a fault model for NFSMs. We finally formalize the concept of "test run observation", which provides a means for the formal presentation of our testing method. We also describe a *complete-testing* assumption without which no full fault coverage can be achieved for nondeterministic models.

2.1 Single NFSM

We first define in the following a single NFSM, which can be viewed as a single simplified SDL process without parameters. Spontaneous transitions, which are internal actions and cause nondeterminism, are included in our model. We first give a formal definition of NFSMs which is similar to the one given in [Star72]. We then define notations for NFSMs by using an approach similar to those for labeled transition systems [Brin88, Brin89, Fuji91, Fuji91b].

DEFINITION *Nondeterministic Finite State Machine* :

A Nondeterministic Finite State Machine (NFSM) is defined as a 5-tuple (St, Li, Lo, h , s0) where :

(1) St is a finite set of states.

(2) Li is a finite set of inputs.

(3) Lo is a finite set of outputs.

(4) *h* is a function.

h : d --> (powerset(St ∞ (Lo ≈ {ε})) - {ϕ})

where (a) *St* 6*Li* /d and d/*St* 6(*Li* ≈ {i_I}); (b) ε represents an empty output; and (c) ϕ denotes the empty set. Let P, P'□St , a□*Li* ≈ {i_I} and b□*Lo* ≈ {ε}. We write P-a/b->P' to denote (P', b)□*h* (P,a); P-a/b->P' is also called a transition from P to P' with label a /b.

(5) s0 is the initial state which is in St.

(6) a reset input *r* is assumed such that upon receiving *r* in any state the NFSM will return to the initial state.

[End of definition].

In the above definition, P-i_I/ε->P' represents an internal, non-observable transition. Furthermore, according to the above definition, the NFSM is completely defined, which means that, for every a□Li and every state, the NFSM has at least one transition with the input a. We note, therefore, that inputs are never blocked, as would be the case for labeled transition systems.

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

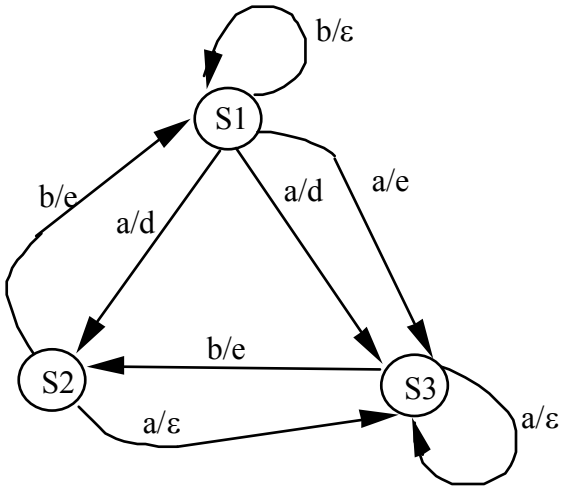

Figure 1. An example of a NFSM

For a NFSM, if there are no internal actions, and if no two outgoing transitions from the same state have the same input, then the NFSM is deterministic; and we denote it an FSM.

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

Table 1. Notation for NFSMs

notation	meaning
Li',	Li ≈ {i_i}
Lo',	Lo ≈ {ε}
i	i = i_i/ε , i denotes an internal action
L	(Li' 6 Lo') – {i}, a set of observable actions
L'	L ≈ {i}
L*	set of sequences over L; x denotes such sequences
P-u_1...u_n-> Q	For P, Q □ St, $u_{1,...,}$ u_n□ L, ∃ P_k□ St (P=P_0-u_1->P_1...-u_n->P_n=Q) where 0≤k≤n
P-u_1...u_n->	For P □ St, $u_{1,...,}$ u_n□ L, ∃ Q□St (P-u_1...-u_n->Q)
P=ε=>Q	P-i^n->Q (nε1) or P=Q (note: i^n means n times i)
P=a/b=>Q	For P, Q □ St, a/b □ L, ∃ P_1, P_2□St (P=ε=>P_1-a/b->P_2=ε=>Q)
P=u_1...u_n=>Q	For P, Q □ St, $u_{1,...,}u_n$□L, ∃P_k(P=P_0=u_1=>P_1...=u_n=>P_n=Q) where 0≤k≤n
P=x=>Q	For P, Q □ St, x□ L*, P=u_1...u_n=>Q with x=u_1...u_n
P=x=>	For P□ St, x□ L*, ∃ Q □ St (P=x=>Q)
P/=x=>	not (P=x=>)

In the following we often make no difference between a state within a given NFSM and a NFSM. In fact, a state S within a NFSM over a given set of actions L' may be considered to represent a transition system, also denoted by S, which has S as initial state and contains all those states and transitions which are reachable from this initial state. A given NFSM may actually consist of several unconnected components, for instance, such as a specification S and an implementation I.

2.2. Conformance relations for NFSMs

In black-box-testing, the only way to distinguish an implementation from its specification is to check the difference between the output sequences caused by the same input sequences. Based on input and output sequences (i.e. i/o traces), we give the following conformance relations.

DEFINITION *Trace preorder* :

The *trace preorder* relation between two states I and S, written

$I \leq_{trace} S$, holds iff $\forall x \in L^*$:

if $I =x\Rightarrow$ then $S=x\Rightarrow$

[End of definition].

It is easy to prove that the above relation is reflective and transitive. Therefore, the relation is a preorder.

DEFINITION *Trace equivalence* :

The *trace equivalence* relation between two states I and S, written

$I =_{trace} S$, holds iff $I \leq_{trace} S$ and $S \leq_{trace} I$

[End of definition].

Similarly, the trace-equivalence relation is an equivalence relation since it is also symmetric. This relation serves as a guide for test generation for NFSM specifications. For an FSM (deterministic finite state machine), the above relation becomes the ordinary equivalence relation as defined in [Chow78]. Since for NFSMs there is no blocking mechanism , as in the case of labeled transition systems, the trace-equivalence relation is the most discriminating conformance relation under the black-box testing assumption.

2.3. Fault model for a single NFSM

Like other state-machine based test generation, fault models, together with the above-defined trace-equivalence relation, serves as a guide to test generation and a basis for test coverage analysis. [Boch91] gave a general survey on a variety of fault models in testing. We now give a fault model for NFSMs. Let S and I be two NFSMs, with S being a specification and I its implementation. We assume that they have the same St, Li and Lo. The fault types are defined as follows:

(1) *Single output fault:* We say that a transition in I has a single output fault if,

(a) not($S=_{trace}I$);

(b) there exists S' such that $S=_{trace}S'$ and S' can be obtained from I by changing the output of the above transition.

(2) *Single transfer fault:* We say that a transition in I has a single transfer fault if,

(a) not($S=_{trace}I$);

(b)there exists S' such that $S=_{trace}S'$ and S' can be obtained from I by changing the ending state of the above transition.

(3) *Single extra (missing) transition fault:* We say that I has a single extra (missing) transition fault if,

(a) not($S=_{trace}I$);

(b) there exists S' such that $S=_{trace}S'$ and S' can be obtained from I by eliminating (adding) a transition from (to) I.

(4) *Multiple faults:* We say that I has multiple faults if,

(a) not($S=_{trace}I$);

(b)there exists S' such that $S=_{trace}S'$ and S' can be obtained from I by changing the outputs of certain transitions, changing the ending states of certain transitions, and eliminating and adding transitions on I.

In Section 4, we will discuss additional faults concerning the queues and channels in a system of CNFSMs.

2.4. Test run observation

We first define the concepts of test cases and test suites, which are needed for formalizing the concept of test run observation.

DEFINITION *Test case* and *test suite* :
For a given NFSM, t is a *test case* if t □ L* . A *test suite* is a set of test cases.
[End of definition].

During testing, after test sequences are applied to the given implementation, the resulting outputs are observed. The following definition formalizes the observation caused by applying a test case t to a given state P of a NFSM.

DEFINITION *Observation of test runs* :
For a given NFSM, *Observation of test runs* is a function O: St ∞ test cases --> {T, F} such that :

O(P@t) = T iff P=t => ; otherwise, (P@t) = F.

[End of definition].

Remember that t is a sequence of input/output pairs. Suppose that t_i is the sub-sequence of t , obtained by deleting all outputs in t; therefore, t_i is an input sequence. O(P@t) = T means that when the input sequence t_i is applied to the state P, the sequence of input/output pairs t may occur.

Complete-testing assumption: For a test case t and a given NFSM I (an implementation; I also represents the initial state of the implementation), suppose that O(I@t) = T holds and that t_i is the sub-sequence of t , obtained by deleting all outputs in t. We assume that a positive integer N can be found such that if the input sequence t_i is applied to I N times, the sequence of input/output pairs t will occur at least once.

For a given test case (an I/O sequence) t, this assumption states that it is possible, by applying the input sequence of the test case a finite of number of times, to exercise **all** possible execution paths of the implementation.

In practice, complete-testing can be achieved by obtaining an implementation with fairness by adopting some techniques such as those given in [Wu 91]. For a given input sequence, the probability that not all possible corresponding execution paths are exercised at least once, may be reduced to close to zero by applying the input sequence a sufficiently large number of times.

3. TEST GENERATION FOR A SINGLE NFSM

We present in this section a method of generating test cases for a single NFSM. Our method is based on state set identification, an approach similar to the state identification in the area of test generation for FSMs [Chow78, Sabn85, Fuji91c]. However, our method needs to cope with nondeterminism too. In a deterministic model, one input sequence can uniquely determine one output sequence, which is not always true in a nondeterministic model.

In our method, a NFSM with internal actions is first transformed to a trace-equivalent NFSM without internal actions. Then, the resulting NFSM is transformed to a so-called ONFSM (*observably nondeterministic* FSM) which is trace-equivalent to the original NFSM. The

corresponding ONFSM usually has a lower degree of nondeterminism than the original NFSM. In the corresponding ONFSM, each state corresponds to a set of states of the original NFSM. Next, based on the ONFSM, a set of I/O sequences, a so-called characterization set, is derived for state set identification. Finally, a test generation method is presented in terms of I/O sequences, ONFSMs, and characterization sets.

We first introduce in Section 3.1 two concepts: ONFSM and a so-called characterization set, in order to present the test generation method. We then consider test generation for ONFSMs without internal actions in Section 3.2. We next give an algorithm for transforming a NFSM to a trace-equivalent ONFSM in Section 3.3. An algorithm of transforming a NFSM with internal actions to a trace-equivalent NFSM without internal actions is given in Section 3.4. Incorporating the above two algorithms, the above test generation method can generate test cases for any NFSM.

3.1. ONFSM and characterization set

We first define ONFSM (*Observably nondeterministic FSMs)*, a concept originally described in [Cern92] which represents a restricted form of nondeterminism.

DEFINITION *Observably-nondeterministic NFSMs* :

A NFSM is said to be *observably-nondeterministic* if the following two statements hold:

(1) 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 and S-a/b->S_2" implies "S_1=S_2".

(2) the NFSM does not contain any internal action.

[End of definition].

ONFSMs are a subclass of NFSMs. They are called observably-nondeterministic since a state and an input/output pair can uniquely determine the next state. However an ONFSM may still be nondeterministic in the sense that given a state and an input, one cannot determine a unique next state and a unique output.

For example, a trace-equivalent ONFSM for the NFSM in Figure 1 is shown in Figure 2.

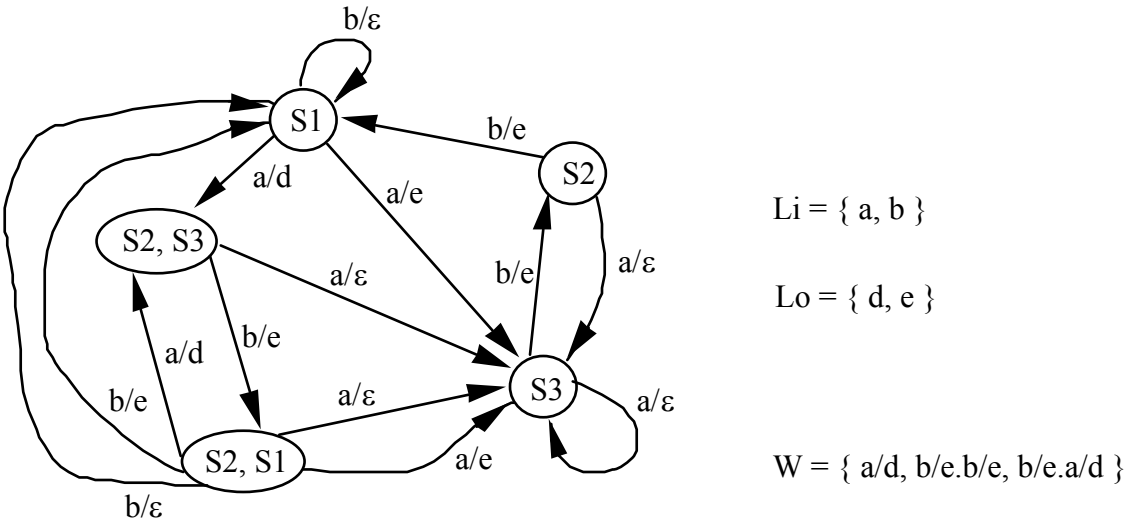

Figure 2. The trace-equivalent minimal ONFSM of the NFSM

In order to present our test generation algorithm, we define in the following the concepts of characterization set and the minimality of NFSMs.

DEFINITION: *Characterization set* :

For an ONFSM, with the state set St, a *Characterization set* is a test suite W such that :

$\forall$ P1, P2 $\in$ St : $\exists$ t $\in$ W , O(P1@t) $\neq$ O(P2@t)

[End of definition].

DEFINITION: *Minimality of ONFSMs* :

An ONFSM is minimal if no two states are trace-equivalent.

[End of definition].

According to the above definitions, if a given ONFSM is minimal, there exists a characterization set for it. The ONFSM shown in Figure 2 is minimal, the corresponding W set is also shown in Figure 2. The following table shows the results of the application of the W set to every state of the ONFSM. The first row lists all the states, and the first column lists all input/output sequences of the W set. For a input/output sequence and a state, their cross in the table being "accepted" means that the input/output sequence will be observed if the corresponding input sequence is sent repeatly under the complete testing assumption. Note that "S2,S3" is a name of a single state, and so is "S2,S1".

Table 2. The results of the application of the W set { a/d, b/e.b/e, b/e.a/d}

	"S1"	"S2"	"S3"	"S2,S3"	"S2,S1"
a/d	**accepted**	not accepted	not accepted	not accepted	**accepted**
b/e.b/e	not accepted	not accepted	**accepted**	**accepted**	not accepted
b/e.a/d	not accepted	**accepted**	not accepted	**accepted**	**accepted**

The W set (Characterization set) is used to determine which state the given ONFSM is in. Take the ONFSM of Figure 2 for example. Assume that the machine is first led to the same unknown state, say Q, before each application of the W set. We want to use the W set to determine which state is the state Q. We first derive the set of the three input sequences {a, b.b, b.c} from the W set of Figure 2 by neglecting all outputs. We then apply each input sequence in {a, b.b, b.c} to the state Q many times (up to N times according the complete testing assumption). If, as a result, the input/output sequences b/e.b/e and b/e.a/d are observed and input/output sequence a/d is not observed, we know that the Q is the state "S1, S2" under the complete testing assumption.

Although we do not give any algorithm to generate a characterization set from a given ONFSM, one can easily develop such an algorithm by using an approach similar to the generation of the W set in the W-method [Chow78].

3.2. Test generation

We first present the following notation for describing the concatenation of test cases.

DEFINITION: *Concatenation of sets of test cases* :
Assuming that V1 and V2 are two sets of test cases, the *concatenation* , written ".", is defined as follows:

V1.V2 = { t1.t2 | t1 □ V1, t2 □ V2 } where t1.t2 is the concatenation of the test case t1 with t2.

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

[End of definition].

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 which may be contained in the trace-equivalent minimal ONFSM for the given NFSM implementation.

ALGORITHM 1: *Test generation* .

Input : A specification S in the form of a minimal ONFSM, and the upper bound m on the number of states which may be contained in the trace-equivalent minimal ONFSM for the given NFSM implementation.

Output : a test suite □.

Step 1: Construct a characterization set W from S.

Step 2: Construct a set of test cases P from S such that:

for each transition Si -u-> Sj in S,

$\exists$ x.u $\in$ P such that there exists a path from the initial state S0 to Si where x is the sequence of labels of the edges along this path.

Step 3: Let the number of states in S be n ($n \leq m$). Construct a test suite Φ such that:

$$\Phi = \{ r \}.(P \approx P.L).Z$$

where r is the reset input, and $Z = (\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^{m-n}).W$

[End of algorithm].

The above algorithm may be considered a generalization to the W-method [Chow78] since it has a similar form.

For example, we generate in the following a test suite for the ONFSM of Figure 2. The ONFSM has 5 states. We first derive from the ONFSM in Figure 2 the following characterization set W :

$$W = \{ a/d, b/e.b/e, b/e.a/d \}.$$

We then derive the set P as follows:

$$P = \{ b/\varepsilon, a/d, a/e, a/d.b/e, a/d.a/\varepsilon, a/e.b/e, a/e.a/\varepsilon, a/d.b/e.b/\varepsilon, a/d.b/e.a/d, a/d.b/e.b/e, a/d.b/e.a/\varepsilon, a/d.b/e.a/e, a/e.b/e.b/e, a/e.b/e.a/\varepsilon \}.$$

Finally, assuming that the implementation is a minimal ONFSM and will not have more than 5 states, the test suite is the following:

$$\Phi = \{ r \}.(P \approx P.L).Z$$
$$= \{ r \}.P.(\{\varepsilon\} \approx L).Z$$
$$= \{ r \}.\{ b/\varepsilon, a/d, a/e, a/d.b/e, a/d.a/\varepsilon, a/e.b/e, a/e.a/\varepsilon, a/d.b/e.b/\varepsilon, a/d.b/e.a/d, a/d.b/e.b/e, a/d.b/e.a/\varepsilon, a/d.b/e.a/e, a/e.b/e.b/e, a/e.b/e.a/\varepsilon \}.(\{\varepsilon\} \approx L).\{ a/d, b/e.b/e, b/e.a/d \}.$$

We use the set $(P \approx P.L).Z$, instead of only using the set P .Z, in the above algorithm for the following reasons: (i) the set P.Z is used to check whether all transitions of the specification exist and have the correct next states, and (ii) the set P.L.Z is used to ensure that a state in a given

implementation has no extra transitions than those specified in the specification. For example, Figure 3 shows a faulty implementation of the specification shown in Figure 2, which has an extra transition fault (see the transition of bold line in Figure 3). For the sets W and P given above, it is easy to see from Figure 3 that this fault is not necessarily detected if □ = { r }. P .Z .

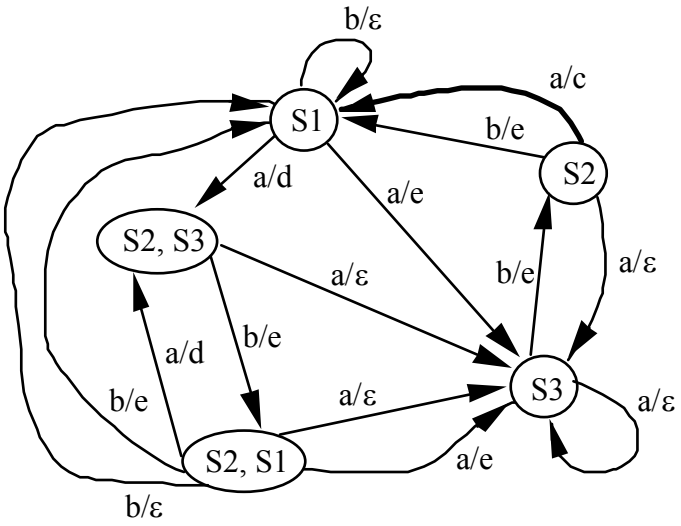

Figure 3. The faulty implementation

If the ONFSM to be tested is in fact an FSM, the set (P ≈ P.L).Z in the above algorithm can be replaced by a smaller the set P.Z; then the above method is the same as the W-method of [Chow78]. Furthermore, this method can be easily modified to a method like the Wp-method [Fuji91c] or the UIO method [Sabn85], obtaining a smaller test suite with the same fault detection power.

THEOREM 1: (Test generation)

Consider a specification S in the form of a minimal ONFSM, and a NFSM I. Suppose n ≤ m where n is the number of states in S, and m is the number of states which is contained in the trace-equivalent minimal ONFSM of I. Under the fault model given in Section 3.2, assuming that the test suite □ is generated for S by using Algorithm 1, we have the following:

$I =_{trace} S$ iff $\forall t \square \square, O(I@t) = O(S@t)$

Proof :

It follows from Lemmas 0 - 4 given in Appendix .

[End of theorem].

3.3. Trace-equivalent transformation to obtain ONFSMs

We now present an algorithm to construct a trace-equivalent ONFSM for an arbitrary (comletely specified) NFSM which has no internal actions. This algorithm can be used to generate test cases for the NFSM by using the test generation method described in Section 3.2.

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

Input : a NFSM S which does not have any internal action.

Output : an ONFSM S'

Step 1: Build a graph G consisting initially of a single node, labeled { P0 }, where P0 is the initial state of S (the node {P0} is unmarked initially).

Step 2: If there is no unmarked node in the resulting graph G, stop; G is the ONFSM S' and the node { P0 } represents the initial state of S'.

Otherwise:

(a) find and mark a unmarked node $\underline{P}$ in G, where the label $\underline{P}$ is a set of states of S ;

(b) for every u □ L, first construct $\underline{P}'=\{P' \mid P \square \underline{P}, P\text{-}u\text{->} P'\}$, then, if $\underline{P}'$ is not a node label in the resulting graph G, create a node with label $\underline{P}'$ and a directed edge from $\underline{P}$ to $\underline{P}'$ with label u; go to Step 2.

[End of algorithm].

For example, the above algorithm constructs the ONFSM shown in Figure 2 from the NFSM in Figure 1. It is easy to see from the above algorithm that the resulting machine is a completely-

specified and observably-nondeterministic . This is, in general, as stated by the following theorem.

THEOREM 2: For a given NFSM S which does not have any internal action, the S' constructed by using Algorithm 2 is an ONFSM which is trace-equivalent to S.

Proof:

See Appendix. [End of proof].

We note that the ONFSM obtained through Algorithm 2 is reduced in general. One can easily develop an algorithm for reducing a given ONFSM to a minimal form, by using an approach similar to the state minimization for FSMs.

3.4. Trace-equivalent transformation to avoid internal actions

We present in the following an approach to transforming a given NFSM with internal actions to a trace-equivalent NFSM without internal actions, in order to generate test cases for the given NFSM by incorporating the test generation method given in Section 3.2 and the transformation method given in Section 3.3.

For a given NFSM, if the input/output pairs are viewed as labels, the NFSM can be viewed as a labeled transition system. Then, the transformation method given in [Luo91b] can be applied to obtain a trace-equivalent NFSM without internal actions.

4. TEST GENERATION FOR A SYSTEM OF CNFSMs

We present in this section the application of test generation method given in Section 3 to generate test cases for a system of CNFSMs. We first present a formal definition for a system of CNFSMs. We then show that for concurrent software, even if each individual sequential program is modeled by a *deterministic finite* state machine, the whole system may be nondeterministic and may not be able to be modeled by a state machine with a finite number of states. This means that testing concurrent programs requires to handle the problems related to nondeterminism and state space explosion. By applying the test generation method given in Section 3 for a single NFSM, we next present test generation methods based on some reduced global state machines since it is impossible to construct a trace-equivalent global NFSM to represent a system of CNFSMs. We finally discuss the detection of faults in queues and channels.

4.1. CNFSMs

Concurrent programs, especially in the area of communication software and communication protocols, are easily modeled by a system of CNFSMs where the NFSMs communicate with each other using queues and channels. We assume here that the communication mechanism of a system of CNFSMs is the same as described in SDL [Beli89]. This means that asynchronuous communication is assumed. We now define in the following a system of CNFSMs formally.

DEFINITION *A system of CNFSMs* :

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

(1) Every individual CNFSM has an input queue of infinite length. Inputs pass through a queue in FIFO (first-in and first-out) fashion.

(2) Every pair of CNFSMs, say F1 and F2, has two channels, one for conveying signals (here the signals are outputs for F1 and inputs for F2) from F1 to F2, the other from F2 to F1. Signals

pass through channels in FIFO (first-in and first-out) fashion, then are put into the corresponding input queue (see Figure 2). Channels can contain an unlimited number of signals.

(3) Execution time of transitions is arbitrary.

(4) The inputs can remain in a queue for an arbitrary period of time.

(5) The signals can remain in a channel for an arbitrary period of time.

[End of definition].

Figure 4 shows a system of CNFSMs consisting of two CNFSMs.

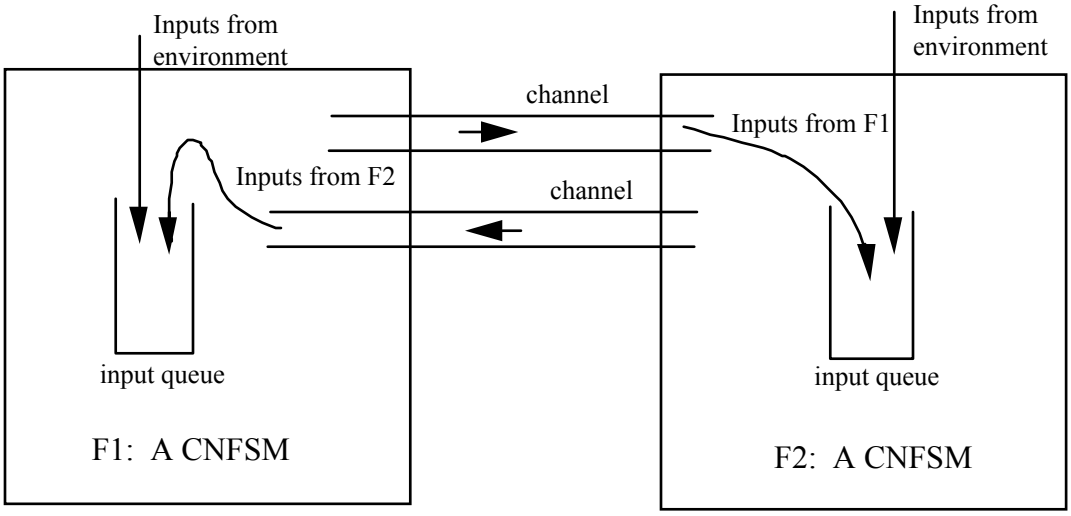

Figure 4. A system of two CNFSMs

4.2. State space explosion problem

We show in this section, by the example of Figure 5, that (i) a system of CNFSMs (even if each of them is deterministic) cannot be modeled, in general, by a trace-equivalent state machine which

contains only a finite number of states. (ii) Moreover, a system of CFSMs (even if each of them is deterministic), may be nondeterministic.

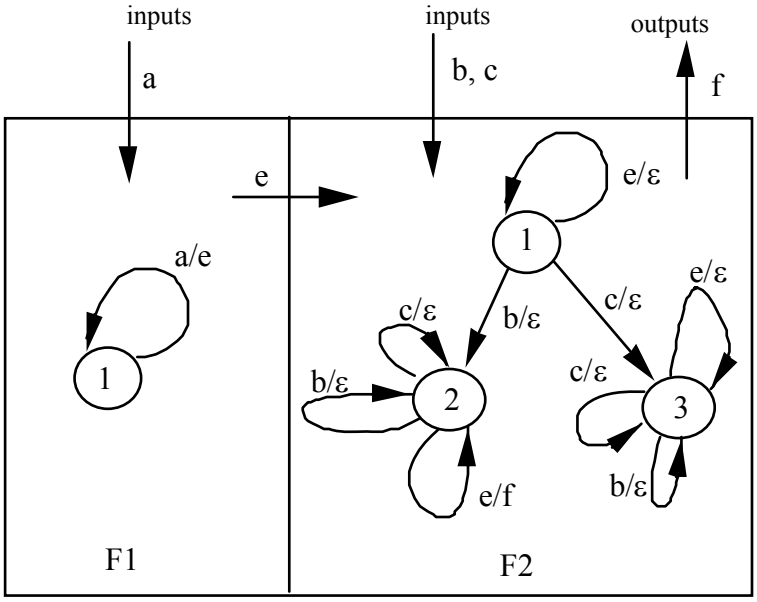

Figure 5. An example to explain state space explosion

Suppose that F1 and F2 have the input queues q_1 and q_2 respectively, and that c_{12} is the channel from F1 to F2 and c_{21} from F2 to F1. Furthermore, we use a tuple $< s_1, s_2, q_1, q_2, c_{12}, c_{21} >$ to represent a global state of the system, where s_1 and s_2 indicate the states of F1 and F2 respectively. Suppose that we send the input sequence $a^n.b$ to the system shown in Figure 5; let us examine what kinds of the sequence of input/output pairs can be observed at the boundary of the system. Assuming that the system is initially in the state <1, 1, [], [], [], [] > where [] stands for the empty queue or channel, the following cases may happen after the system receives the sequence $a^n.b$ (n=0, 1, 2, ...):

(1) the system first enters the state <1, 1, [], $[e^n.b]$, [], [] >, then transfers to the state <1, 2, [], [], [], [] > without any output.

(2) the system first enters the state <1, 1, [], $[b.e^n]$, [], [] >, then transfers to the state <1, 2, [], [], [], [] > generating f^n as an output sequence.

(3) other cases vary between the above two extremes.

Therefore, the system is nondeterministic. Furthermore, in case (2), we observe the input/output sequence $a^n.b.f^n$. Assume that the system is modeled by an trace-equivalent state machine F. Since the output sequence f^n can be produced only after b is received, and since the number of f's is also n , the state machine F has to contain at least n states to remember the input sequence a^n. In addition , since the number n can be any positive integer, the state machine F must contain an infinite number of states. Thus we can conclude that a system of CNFSMs, in general, cannot be modeled by a trace-equivalent global NFSM. Consequently, the test generation method for NFSMs given in Section 4 cannot be applied to a system of CNFSMs directly.

4.3. Generating test cases by applying the method for single NFSMs

We show in the following how to apply the test generation method for a single NFSM in order to generate test cases for concurrent programs modeled by a system of CNFSMs. We first use the reachability analysis to obtain a single NFSM from a system of CNFSMs. We then apply the test generation method for a single NFSM to the resulting NFSM.

4.3.1. Method based on bounded queues and channels

Since unbounded queues or unbounded channels cannot be implemented in a practical application, it is reasonable to assume upper bounds for queue lengths and channel lengths. In case of bounded queues and channels, a system of CNFSMs can be transformed into a trace-equivalent global NFSM by reachability analysis. After the transformation, the test generation method for NFSMs given in Section 4 can be applied to the transformed NFSM which models the system of CNFSMs. However, even if bounded queues and bounded channels are assumed, a trace-equivalent global NFSM for a given system of CNFSMs still may be very large. Therefore, we impose an additional constraint on our model.

4.3.2. Method assuming a slow environment

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

For a given system of CNFSMs com(F1, F2,....., Fn), consisting of n NFSMs named F1, F2,....., Fn, if the system contains no live-lock and runs only in a slow environment, then the system can be modeled by a trace-equivalent global NFSM with M16M26,... ...,6Mn state where M1,M2, ... Mn are the numbers of states in F1, F2,....., Fn, respectively. For such a system of CNFSMs, the trace-equivalent global NFSM can be obtained by using the reachability analysis.

Figure 6 gives an example of a system of two CNFSMs without live-lock. Figure 7 shows the trace-equivalent global NFSM of the system obtained by using the reachability analysis.

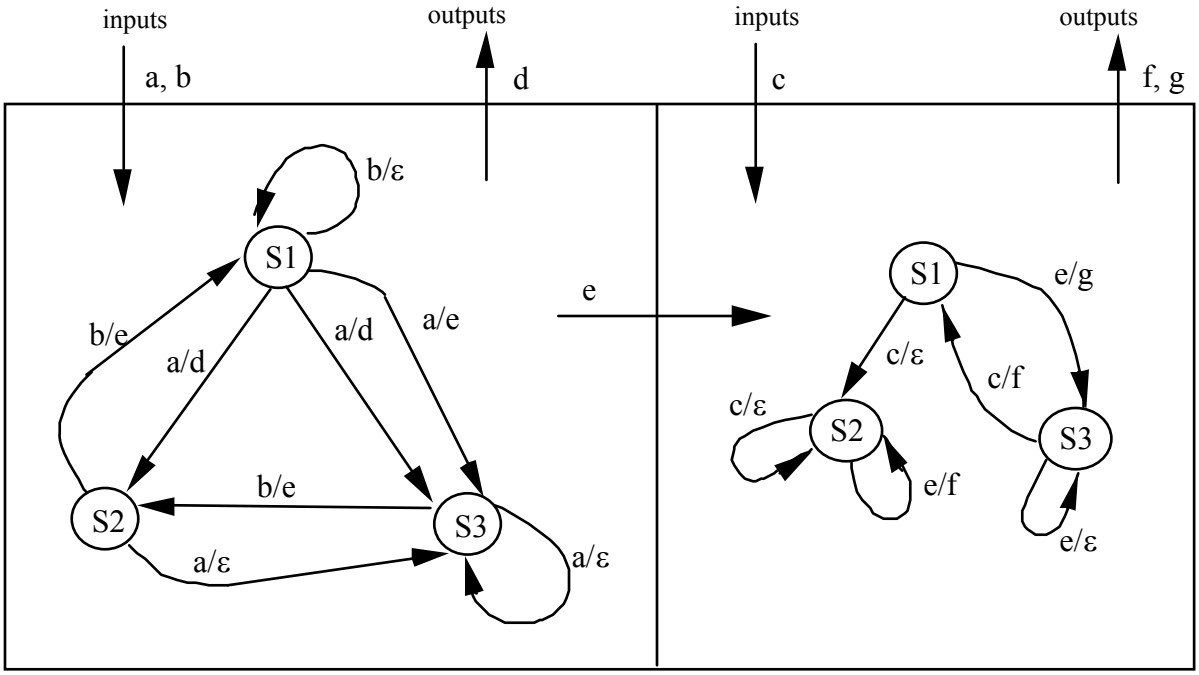

Figure 6. An example of a system of two CNFSMs

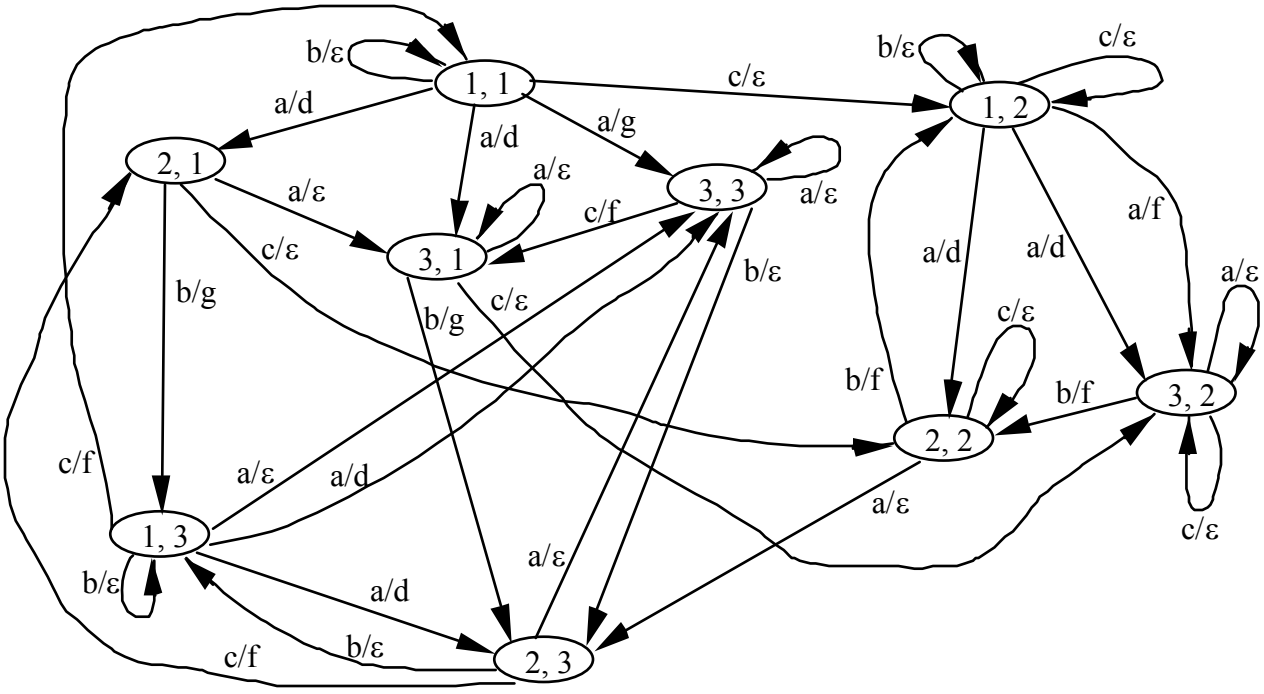

Figure 7. The global NFSM generated by using reachability analysis

After the global NFSM has been obtained from a given system of CNFSMs, we apply the test generation method given in Section 3 to the global NFSM, to generate test suite.

4.4. Detecting faults of queues and channels

We first define in this section an additional fault model concerning queues and channels, and then present a test selection framework which takes these fault types into account. Similar to SDL semantics, we assume unbounded queues (and channels) in the specifications modeled by CNFSMs. The implementation model, however, has to limit the size of each queue (and channel) to a prescribed "maximum length" [Boch91] since no unbounded queue (nor channel) can be implemented in practical applications. We therefore consider the following additional fault types. Let the specification be a system of CNFSMs com(F1, F2,....., Fn), and the implementation a system of CNFSMs com(F1', F2',....., Fn') with the same interconnection structure. We also assume the following: for a queue (channel) of length M, when the queue (channel) already contains n inputs, if an input is put into the queue (channel), then it will be lost.

The additional fault types are defined as follows:

(1) *Maximum length fault of queue* : We say that the queue in Fi' (i = 1, 2, ...,n) has a maximum length fault if the maximum length implemented is less than the one specified in Fi.

(2) *Maximum length fault of channel* : We say that a channel from Fi' to Fj' (i ,j = 1, 2, ...,n; and j□i) has a maximum length fault if the maximum length implemented is less than the one specified for the channel from Fi to Fj.

(3) *Ordering fault* : We say that a queue (channel) has an ordering fault if the FIFO ordering is not preserved [Boch91].

Further study is needed for an systematic procedure to describe how to find the test cases step by step. However, one usually has no difficulty to find the test cases by using intuition and heuristic approaches. For example, guided by the fault model, in order to detect the maximum length faults of queues and channels, we find test cases in the following manner:

Fault model guided test selection:

Given a system of CNFSMs com(F1, F2,....., Fn), for every queue and every channel, generate a test case t (i.e. an I/O sequence of inputs and outputs) for the queue (or channel) such that :

(a) O(com(F1, F2,....., Fn)@t) = T and

(b) O(com(F1', F2',....., Fn')@t) = F if the corresponding maximum queue length (or maximum channel length) in the implementation is less than the one specified and if no other fault exists in com(F1', F2',....., Fn').

[End].

For the example shown in Figure 5, assuming that the maximum length of channel c_{12} is 4 in the specification, by using the above procedure, a test case $t = a^4.b.f^4$ is generated. Following the arguments given in Section 4.2, the reader may check O(com(F1', F2')@t) = F easily if the maximum length of channel c_{12} is less than 4 in the implementation com(F1', F2'). This means that if the implementation com(F1', F2') is correct, it will produce the output sequence f^4 after receiving $a^4.b$ sometime if $a^4.b$ is sent to com(F1', F2') in the state <1, 1, [], [], [], [] > infinitely often. On the other hand, if the maximum length of channel c_{12} in the implementation com(F1', F2') is less than 4, it cannot produce the output sequence f^4 after receiving $a^4.b$ sometime even if $a^4.b$ is sent to com(F1', F2') in the state <1, 1, [], [], [], [] > infinitely often, as shown in Figure 8.

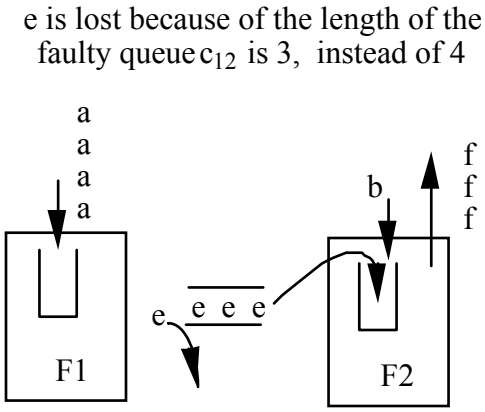

Figure 8. Detecting a channel fault.

Further study is needed for detecting the above ordering fault.

5. CONCLUSION

We present in this paper a method of generating test cases for concurrent programs from a system of CNFSMs. This method can be applied to test generation for the control part of SDL processes. In order to apply this method, SDL processes are first abstracted to a system of CNFSMs by neglecting the parameters and by the transformation to avoid the SAVE constructs [Luo91]. The test generation method are then applied to the system of the resulting CNFSMs. Further study may find a better solution to the state space explosion problem. The test generation method may also be useful in the area of VLSI circuit testing [Cern92] since some VLSI circuits can be viewed as NFSMs.

APPENDIX: VALIDITY OF TEST METHOD

We first introduce several definitions, then we give several lemmas which are required for proving the theorem of test generation.

DEFINITION : *Grouping function h* :

For a given NFSM I, suppose that (i) I_m is the trace-equivalent minimal ONFSM of I, (ii) St and St_m are the state sets of I and I_m respectively. Then,

grouping function h of I : St_m -> powerset(St) is defined as follows:

$h(s)=\{ g \in St \mid \exists x \in L^*, I_m=x=>s, I=x=>g \}$

[End of definition].

DEFINITION : *Multistates, prime multistate set* :

For a given NFSM I, suppose that (i) I_m is the trace-equivalent minimal ONFSM of I, (ii) St and St_m are the state sets of I and I_m respectively. , and (iii) h is the grouping function of I. Then, a *multistate* is a set of states in St; and a *prime multistate set* $\mathbb{D}$ of I is defined as follows:

$\mathbb{D} =\{ h(s) \mid s \in St\}$

[End of definition].

It is easy to see that the grouping function h, from St_m to $\mathbb{D}$, is one-one and onto.

DEFINITION : *Prime machine* :

For a given NFSM I, suppose that (i) the prime multistate set is $\mathbb{D}$. Then, a *prime machine* of I is a graph such that (i) the number of its nodes is $|\mathbb{D}|$, (ii) each node in it is labeled by a multistate in $\mathbb{D}$, and (iii) For each pair of multistates M_i and M_j in $\mathbb{D}$, there is a directed edge labeled u from M_i to M_j if and only if $\exists g_i \in M_i$, $\exists g_j \in M_j$, $\exists u \in L^*$ such that $g_i=u=>g_j$ in the given I.

[End of definition].

According the above definition, a prime machine is just a graph, not a NFSM. However, for a given NFSM I, if its prime machine is viewed as the graphic representation of a NFSM F, then the F is the trace-equivalent minimal ONFSM of I.

Table 3. Notation for some relations between multistates

notation	meaning
M_i~u-> M_j	$\exists\, g_i \in M_i$, $\exists\, g_j \in M_j$, $\exists\, u \in L^*$ such that g_i=u=>g_j in the given I
M_i~u->	there exists a M_j such that M_i~u-> M_j
M_i~u_1...u_n-> M_j	$\exists\, g_i \in M_i$, $\exists\, g_j \in M_j$, $\exists\, u_k \in L^*$ (k= 1,2, ...,n) such that g_i=u_1...u_n=>g_j in the given I
M_i~u_1...u_n->	there exists a M_j such that M_i~u_1...u_n-> M_j

Note that M_1~u_1-> M_2 and M_2~u_2-> M_3 together imply M_1~$u_1.u_2$-> M_3 in a prime machine.

We extend the observation of test runs to include the situation for multistates

DEFINITION *Observation of test runs for multistates* :

For a given NFSM, *Observation of test runs* is a function O:

(St ≈ powerset(St)) ∞ test cases --> {T, F} such that :

(i) for P ∈ St

O(P@t) = T iff P=t => ; otherwise, (P@t) = F.

(ii) for M ∈ powerset(St),

O(M@t) = T iff ∃ P∈ M such that P=t =>; otherwise, (M@t) = F.

[End of definition].

DEFINITION : *V-equivalent for states and multistates* :

For a given NFSM, let Qi and Qj, each be either a state or a multistate. V is a test suite. Qi is said to be V-equivalent to Qj if ∀ t∈ V, O(Qi@t) = O(Qj@t) . Otherwise, Qi and Qj are said to be V-distinguishable.

[End of definition].

We extend in the following the trace-equivalence relation to include the situation for multistates.

DEFINITION : *Trace-equivalent for states and multistates* :

For a given NFSM, let Qi and Qj, each be either a state or a multistate. Qi is said to be trace-equivalent to Qj if $\forall$ t□ L*, O(Qi@t) = O(Qj@t) . Otherwise, Qi and Qj are said to be trace-distinguishable.

[End of definition].

DEFINITION : *Isomorphism between two ONFSMs* :

For a pair of given ONFSMs I and S, an *isomorphism from* I to S is a function ∂ which maps states in S to states in I, such that : a) ∂ is one-one and onto, b) if Si-u-> Sj is in S, then ∂(Si)-u-> ∂(Sj) is in I, and c) if ∂(Si)-u-> ∂(Sj) is in I, then Si-u-> Sj is in S. Furthermore, S is said to be *isomorphic* to I if there exists an isomorphism between them.

[End of definition].

For a given NFSM I, if its prime machine is viewed as the graphic representation of a NFSM F, then the F is isomorphic to the trace-equivalent minimal ONFSM of I.

Please notice that the b) in the above definition, in general, does not imply c), while it does when the given ONFSMs are FSMs.

LEMMA 0: For two minimal ONFSMs I and S, I is trace-equivalent to S iff I and S are isomorphic with the root of I corresponding to the root of S. For a minimal ONFSM S and a NFSM I, with I_m being the trace-equivalent minimal ONFSM of I, I is trace-equivalent to S iff I_m and S are isomorphic with the root of I_m corresponding to the root of S.

We omit the proof for the above lemma since it is straightforward.

We assume in the following:

(1) S is a minimal ONFSM and I is a NFSM. S and I have the same input and output alphabets Li and Lo.

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

(3) I_m is the trace-equivalent minimal ONFSM of I, and I_m may have at most m states with $m \geq n$.

(4) W is a characterization set, and $Z = (\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^{m-n}).W$.

(5) P is a set of test cases such that : for each transition Si -u-> Sj in S, with S0 being the root of the ONFSM,

$\exists$ x.u $\square$ P such that there exists a path from S0 to Si in S with x being the sequence of labels of the edges along this path.

(6) $\mathbb{D}$ is the prime multistate set of I, M0 is the root label of the prime machine of I.

(7) St_S, St_I and St_m are the state sets of S, I and I_m respectively.

(8) h is the grouping function of I.

LEMMA 1: Trace-equivalence is an one-one and onto function from I_m to $\mathbb{D}$; and for every pair of M_i and M_j (i$\square$j) in $\mathbb{D}$, M_i is not trace-equivalent to M_j .

Proof:

Evident from the corresponding definitions.

[End of proof].

LEMMA 2: Suppose that W-equivalence partitions the states in $\mathbb{D}$ into at least n classes. Z will distinguish every pair of states in $\mathbb{D}$.

Proof:

Induction hypothesis **:** $(\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^i)$.W-equivalence partitions states in $\mathbb{D}$ into at least i+n classes for i=0,...,m-n.

Induction base **:** i=0, it is true by the hypothesis of the lemma.

Induction step **:** Assume induction hypothesis true for $i \varepsilon 0$. We show that it holds for i+1. If $(\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^i)$.W-equivalence has already partitioned states in $\mathbb{D}$ into at least i+1+n classes, then it is obviously true. Otherwise, from Lemma 1, there must be a pair of multistates in $\mathbb{D}$, M_i and M_j, such that M_i and M_j are L^k .W-distinguishable but $(\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^{k-1})$.W-equivalent, for some $k \geq i$. This implies that there must be a pair of states M_i' and M_j' such that M_i' and M_j' the (k-i-1)th successors of M_i and M_j , and M_i' and M_j' are L^{i+1} .W-distinguishable, but $(\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^i)$.W-equivalent. Therefore, $(\{\varepsilon\} \approx L \approx L^2 \approx ..... \approx L^{i+1})$.W partitions the states in $\mathbb{D}$ into i+1+n classes. By induction, we have the lemma.

[End of proof].

In the following, we use $\cdot$ to denote the composition of functions; i.e., $h \cdot f(x)$ means $h(f(x))$. For one-one and onto function, we also use h^{-1} to denote the inverse function of h; i.e., $h^{-1}(y)=x$ means $y=h(x)$.

LEMMA 3: S is isomorphic to I_m iff Z-equivalence satisfies the following statements :

(1) for every state $S_i \in St_S$, there is a multistate $M_i \in \mathbb{D}$ such that M_i is Z-equivalent to S_i. In particular, M0 is Z-equivalent to S0.

(2) if S_i-u-> S_j, then there are M_k and M_l in $\mathbb{D}$ such that M_k and M_l are Z-equivalent to Si and Sj, respectively, and M_k~u-> M_l .

(3) if $\exists M_k , M_l \in \mathbb{D}$ such that M_k~u-> M_l, then there are S_i and S_j,such that M_k and M_l are Z-equivalent to Si and Sj, respectively, and S_i-u-> S_j .

Proof:

(==>) : Assume that S is isomorphic to I_m and that the isomorphism is denoted by f. Then, for every $S_i \in St_S$, we have that S_i, $f(S_i)$, and $h \cdot f(S_i)$ are trace-equivalent since S is isomorphic to I_m.

Therefore, S_i, $f(S_i)$, and $h \cdot f(S_i)$ are Z-equivalent. Since Z-equivalence can distinguish every pair of states in St_S, it is easy to see that Z-equivalence satisfies the above statements (1)-(3) .

(<==) : Assume that Z-equivalence satisfies the above statements.

(I) To show that Z-equivalence is a function from St_S to $\mathbb{D}$, and is one-one and onto:

We first argue that Z-equivalence is a function. Because of (1), and W/Z, for every $S_i \in St_S$, there is a multistate $M_i \in \mathbb{D}$ such that M_i is Z-equivalent to S_i. Since S has n states, from the definition of W, it follows that W-equivalence has partitioned the multistates of $\mathbb{D}$ into at least n classes. By Lemma 2, Z-equivalence will distinguish every pair of states in $\mathbb{D}$. Therefore, there must be at most one state in $\mathbb{D}$ which is equivalent to a state in St_S.

From the definition of W and W/Z, Z-equivalence is one-one. It also is easy to see that the mapping is onto, because I has the same input set as S. Hence, according (2), every reachable multistate in $\mathbb{D}$ is Z-equivalent to some state in St_S.

(II) To show that S is isomorphic to I_m:

Let λ denote the Z-equivalence; i.e., λ is a function from St_S to $\mathbb{D}$, and is one-one and onto. h^{-1}, the inverse function of the grouping function h, from $\mathbb{D}$ to St_m is one-one and onto. Therefore, $\lambda \cdot h^{-1}$ is a function from St_S to St_m ; and it also is one-one and onto. Then, from (2) and (3), it is easy to see that $\lambda \cdot h^{-1}$ is an isomorphism from S to I_m.

[End of proof].

LEMMA 4: Z-equivalence satisfies the following three statements, iff I and S are (P ≈ P.L).Z-equivalent.

(1) for every state $S_i \in St_S$, there is a multistate $M_i \in \mathbb{D}$ such that M_i is Z-equivalent to S_i. In particular, M0 is Z-equivalent to S0.

(2) if S_i-u-> S_j, then there are M_k and M_l in $\mathbb{D}$ such that M_k and M_l are Z-equivalent to Si and Sj, respectively, and M_k~u-> M_l .

(3) if $\exists M_k , M_l \in \mathbb{D}$ such that M_k~u-> M_l, then there are S_i and S_j,such that M_k and M_l are Z-equivalent to Si and Sj, respectively, and S_i-u-> S_j .

Proof:

(==>) : Assume that Z-equivalence satisfies the following three statements. From Lemma 3, S is isomorphic to I_m . Hence, it is easy to see that I and S are (P ≈ P.L).Z-equivalent.

(<==) : Assume that I and S are (P ≈ P.L).Z-equivalent.

(I) For every S_i □ St_S, there is a x □ P such that S0-x->S_i . Let M_j be the multistate reached by x in I. Since we cannot distinguish S and I with respect to x.Z, S_i must be Z-equivalent to M_j.

(II) If S_i-u-> S_j, there is a x □ P such that S0-x-> S_i. We also have x.u □ P such that S0-x.u-> S_j. Let M_k and M_l denote the multistates reached by x, x.u in I. Since we cannot distinguish S and I with respect to x.Z and x.u.Z, S_i and S_j are Z-equivalent to M_k and M_l , respectively, with M_k~u-> M_l.

(III) From the above (I) and (II), and from the " if part (I)" of the proof of Lemma 3, Z-equivalence is a function from St_S to 𝔻, and is one-one and onto. Then, for every M_k □ 𝔻 , there must be a S_i □ St_S which is Z-equivalent to M_k , there must be a x □ P such that S0-x-> S_i ; hence, we have M0~x-> M_k since we cannot distinguish S and I with respect to x.Z. This means that for every M_k □ 𝔻, there a x □ P such that M0~x-> M_k. For every M_k~u-> M_l, we have x □ P such that M0~x-> M_k, and x.u □ P.L (Here please notice that x.u is not necessarily in P). Let S_i and S_j □ St_S denote the states reached by x, x.u in S. Then, we have S_i-u-> S_j . Since we cannot distinguish S and I with respect to x.Z and x.u.Z, S_i and S_j are Z-equivalent to M_k and M_l , respectively.

[End of proof].

THEOREM 2: For a given NFSM S which does not have any internal action, the S' constructed by using Algorithm 2 is an ONFSM which is trace-equivalent to S.

Proof:

It is easy to see from the above algorithm that the resulting machine is completely-specified and observably-nondeterministic .

Suppose that S is a given NFSM and S’ the ONFSM obtained by applying Algorithm 1.

Let $x=a_1/b_1.a_2/b_2.....a_n/b_n \in L^*$,

and $S\text{-}a_1/b_1\text{->}S_1\text{-}a_2/b_2\text{->}S_2.....\text{-}a_n/b_n\text{->}S_n$ ..... (1)

From (1) and Algorithm 1 Step 2(a) and (b), there exist $S'_1, S'_2, ..., S'_n$ such that

$S'\text{-}a_1/b_1\text{->}S'_1\text{-}a_2/b_2\text{->}S'_2.....\text{-}a_n/b_n\text{->}S'_n$ ..... (2)

Similar to the above arguments, $S'=x\Rightarrow$ implies $S=x\Rightarrow$ ..... (3)

The theorem holds because of (2) and (3).

[End of proof].

Acknowledgments: The authors would like to thank Prof. Rachida Dssouli, Prof. Alexandr Petrenko who read the draft and gave us useful comments, and Mr. C. Wu, Mr E.H.Htite, Mr. M. Yao and Mr. A.Ghedamsi for helpful discussions and comments.

REFERENCES:

[Arak91] Noriyasu Arakawa, Terunao Soncoka, "A test Case Generation Method for Concurrent Programs", Proceedings of International Workshop on Protocol Testing Systems, Oct. 15-17th, 1991, the Netherlands.

[Boch91] G.v. Bochmann, A. Das, R. Dssouli, M.Dubuc, A.Ghedamsi, and G.Luo, "Fault Model in Testing", Proceedings of International Workshop on Protocol Testing Systems, Oct. 15-17th, 1991, the Netherlands.

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

[Bolo87] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language Lotos", Computer Networks and ISDN Systems, vol. 14, no. 1, pp.25-59, 1987.

[Brin88] Ed Binsksma, "A Theory for the Derivation of Tests", IFIP Protocol Specification, Testing, and Verification VIII.

[Brin89] Ed Brinksma, Rudie Alderden, Rom Langerak, Jeroen van de Lagemaat and Jan Tretmans, "A Formal Approach to Conformance Testing", 2nd International Workshop on Protocol Test System", Berlin(West), Germany, Oct. 3-6, 1989, pp.311-325.

[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, pp.3-23, 1987.

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

[Cern92] E. Cerny, "Verification of I/O Trace Set Inclusion for a Class of Nondeterministic Finite State Machines", Internal Report, D.I.R.O, University of Montreal, 1992.

[Fuji91] Susumu Fujiwara and Gregor von Bochmann, "Testing Nondeterministic Finite State Machine", Publication#758 of D.I.R.O, University of Montreal, January 1991.

[Fuji91b] Susumu Fujiwara and Gregor von Bochmann, "Testing Nondeterministic Finite State Machine with Fault Coverage", 4th International Workshop on Protocol Testing Systems, Oct. 1991, Leidschendam, Netherlands.

[Fuji91c] S.Fujiwara, Gregor von Bochmann,F.Khendek,M.Amalou & A.Ghedamsi, "Test Selection Based on Finite State Models", IEEE Trans. on Software Engineering, Vol SE-17, No.6, June, 1991, pp.591-603.

[Kaln91] A. Kalnins, "Global State Based Automatic Test Generation for SDL", SDL'91: Evolving Methods (Proceedings of 5th SDL Forum), North-Holland, 1991, pp.303--312.

[Luo89]. Gang Luo & Junliang Chen , "Generating Test Sequences For Communication Protocol Modeled by CNFSM", Proc. of the 3rd Pan Pacific Computer Conference, Aug. 16- 19, 1989, Beijing, China.

[Luo89b]. Gang Luo & Junliang Chen, "Test Design for SDL Described Concurrent Communication Software", International Conference on Communication Techniques'89, Beijing, China.

[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.

[Luo91b] Gang Luo, Gregor von Bochmann, Anindya Das, and Cheng Wu, "Failure-Equivalent Transformation of Transition System to Avoid Internal Actions", Publication#789 of D.I.R.O, University of Montreal, September 1991.

[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.

[SDL91] SDL newsletter, Dec. 1991.

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

[Tret89] Jan Tretmans, "Test Case Derivation from LOTOS Specifications", Forte'89, pp.469-488.

[Wu91] C. Wu, G.v. Bochmann and M. Yao, Fairness of N-Party Synchronization and Its Implementation in a Distributed Environment, Submitted to Distributed Computing Systems Conf. 92 in Japan.