12

Test sequence generation for adaptive interoperability testing

Sungwon Kang and Myungchul Kim
Korea Telecom Research Laboratories
Sochogu Umyundong 17, Seoul 137-792, Korea
e-mail: kangsw@sava.kotel.co.kr, mckim@sava.kotel.co.kr

Abstract

When testing communicating systems, nondeterminism makes it a more difficult and evasive process. Adaptive testing is an efficient approach to testing nondeterministic systems. In this paper, we develop an interoperability test generation method for adaptive testing. Also we define a measure of testing cost and compare our method with the conventional approach.

Keywords

Test case generation, adaptive interoperability testing, nondeterminism

1. INTRODUCTION

In order to ensure interoperability of communication networks, it is essential to verify correctness of implementations of communication protocols as well as to verify correctness of their specifications on which implementations are to be based. Specifications of implementations constituting a system should be such that correct implementations interoperate. The activity widely called validation and verification, however, is almost always incomplete for most nontrivial protocol specifications. Verifying implementation correctness is called conformance testing and is in comparing the behavior of implementation under test with respect to the expected behavior according to its specification. It also is incomplete for nontrivial protocols. Such incompleteness leaves the direct examination of system behavior as an essential and integral part of interoperability assurance activity.[1]

Recently as the need for systematic interoperabilty testing increases, there has been significant related work ranging from investigation from practitioner's point of view [Bonnes 90][Vermeer 94] to theoretical study with varying degree of formalism [Arakawa 92][Castanet

[1] Often a specification is given in a parameterized form so that the real specification is obtained by instantiating the parameters with concrete values. In such a case, inability of implementations to interoperate may be due to incompatibility of parameters. Ideally such incompatibility should not be regarded as a major concern of testing but of validation and verification.

A. Cavalli et al. (eds.), *Protocol Test Systems VIII*
© Springer Science+Business Media Dordrecht 1996

94][Kajiwara 94][Luo 94]. Ideally interoperability test sequence generation should be done, as with conformance test sequence generation, in such a way that minimizes the cost of the testing process and maximizes the quality of the test result, which tend to be conflicting goals. In general, approaches from practitioners emphasize the former aspect whereas approaches from theoreticians emphasize the latter aspect.

In this paper, we develop an interoperability test generation method which accommodates these conflicting goals. For the second goal our test generation method is based on state space analysis of the global system. And for the first goal our test generation method is directed to adaptive testing [Kim 94] and global system state space is compressed through the so called slow environment principle [Arakawa 92][Luo 94]. When dealing with a system of communicating entities, nondeterminism is a problem that we cannot get away with. Adaptive testing is an elegant approach to controlling and observing behaviors of nondeterministic systems. The slow environment assumption enables us to abstract from nondeterminism to a significant extent.

This paper is organized as follows. In Section 2, we define a system of communicating finite state machines model and define interoperability with respect to the model. In Section 3, we describe our test generation method for adaptive testing. In Section 4, the test generation method is combined with interoperability test generation to give test sequence generation for adaptive interoperability testing. Also an example of interoperability test generation is given and improvement over the conventional non-adaptive method is measured. In Section 5, we summarize contributions of this paper and suggest further research problems.

2. A SYSTEM OF COMMUNICATING IOSM'S AND INTEROPERABILITY

Definition 2.1 An IOSM M is a 5-tuple $<St, s_0, L_{in}, L_{out}, Tr>$ where:

(1) $St = \{s_0, ..., s_{n-1}\}$ is a set of states,
(2) $s_0 \in St$ is the initial state,
(3) $L_{in} = \{v_1, ..., v_m\}$ is a set of input symbols,
(4) $L_{out} = \{o_1, ..., o_p\}$ is a set of output symbols and
(5) $Tr \subseteq \{s\text{–}v/\mathbf{o}\rightarrow s' \mid s, s' \in St \wedge v \in L_{in} \wedge \mathbf{o} \in (L_{out})^* \}$

In spite of the name, the definition of IOSM here differs from that of [Phalippou 91]. Bold face letters are used to denote a sequence of symbols and the empty sequence is denoted as ε or –. L_{in} and L_{out} are respectively called *input alphabet* and *output alphabet*. Tr is a set of transitions of M. We call $v/\mathbf{o} \in (L_{in}/(L_{out})^*)$ a *label*. If $v_1/\mathbf{o}_1, ..., v_k/\mathbf{o}_k \in (L_{in}/(L_{out})^*)$, we denote by $s\text{–}\ v_1/\mathbf{o}_1 ... v_k/\mathbf{o}_k \rightarrow s'$ that

$$\exists s_1,...,s_{k-1} \in St: s\text{–}v_1/\mathbf{o}_1\rightarrow s_1 \wedge s_1\text{–}v_2/\mathbf{o}_2\rightarrow s_2 \wedge ... \wedge s_{k-1}\text{–}v_k/\mathbf{o}_k\rightarrow s'$$

In this paper, we adopt the *completeness assumption* which requires that transitions be defined for every input symbol, i.e.

$$\forall s \in St, v \in L_{in} : \exists s_1 \in St, \mathbf{o} \in (L_{out})^*: (s\text{–}v/\mathbf{o}\rightarrow s_1) \in Tr.$$

In the set Tr of transitions of a *deterministic* IOSM, there is only one transition for any state with the same input symbol, i.e., Tr satisfies the condition:

$$\forall s, s_1, s_2 \in St, v \in L_{in}, \mathbf{o}_1, \mathbf{o}_2 \in (L_{out})^*: (s\text{–}v/\mathbf{o}_1\rightarrow s_1 \wedge s\text{–}v/\mathbf{o}_2\rightarrow s_2) \rightarrow \mathbf{o}_1 = \mathbf{o}_2 \wedge s_1 = s_2.$$

Note that Definition 2.1 does not restrict IOSM to be deterministic.

The general class of nondeterministic IOSM's poses challenges to testing since controllability based upon observation of external behavior is very limited. So in this paper we confine our discussion to the class of observable IOSM's defined as follows.

Definition 2.2 (*n*-observable IOSM) Let M be as in Definition 2.1. Then M is *n*-observable if and only if there is $n \geq 1$ such that

$$\forall s, s_i, s_j \in St: \exists\, \mathbf{x} \in (L_{in}/(L_{out}*))^n: (s\text{–}\mathbf{x}\rightarrow s_i \wedge s\text{–}\mathbf{x}\rightarrow s_j) \rightarrow s_i = s_j.$$

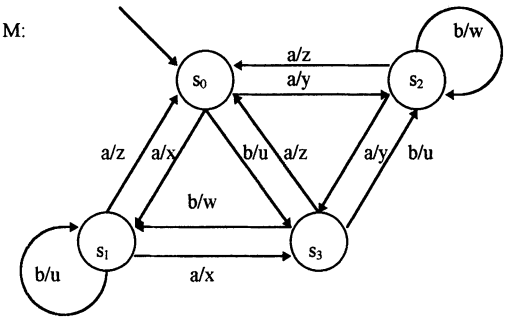

Figure 1 An observable IOSM.

We call 1-observable IOSM *observable IOSM*. Figure 1 depicts an observable IOSM.

Definition 2.3 Let M be as in Definition 2.1. Let $v, v_1, ..., v_k \in L_{in}$ and $s, s' \in St$. Then:

(1) $\sigma(s,v) = \{(s',v/\mathbf{o}) \mid s\text{–}v/\mathbf{o}\rightarrow s' \in Tr\}$
(2) $\sigma(s,\mathbf{v}) = \{(s',v_1/\mathbf{o}_1...v_k/\mathbf{o}_k) \mid s\text{–}v_1/\mathbf{o}_1...v_k/\mathbf{o}_k\rightarrow s' \in Tr \wedge \mathbf{v} = v_1...v_k\}$
(3) $s(\mathbf{v}) = \{v_1/\mathbf{o}_1...v_k/\mathbf{o}_k \mid (s', v_1/\mathbf{o}_1...v_k/\mathbf{o}_k) \in \sigma(s,\mathbf{v}) \wedge \mathbf{v} = v_1...v_k\}$
(4) $M(\mathbf{v}) = s_0(\mathbf{v})$

A member of $M(\mathbf{v})$ is a sequence of labels and is called a *run* (or *trace*) of M for the input sequence **v**.

Definition 2.4 Let M_1 and M_2 be two IOSM's with the same input alphabet L_{in}. Then M_1 and M_2 are *observationally equivalent* ($M_1 \equiv_o M_2$) if and only if

$$\forall \mathbf{v} \in L_{in}*: M_1(\mathbf{v}) = M_2(\mathbf{v}).$$

The above definition stipulates so called *trace equivalence*, i.e. two IOSM's are said to be observationally equivalent if and only if, given any input sequence, the set of traces for that sequence are the same. A non-observable IOSM can always be transformed into a trace equivalent observable IOSM [Luo 94]. Various methods were developed to determine conformance by testing for deterministic IOSM's [Sidhu 89]. Trace equivalence was often used as a conformance relation between two IOSM models, one for a specification and the other for an implementation.

IOSM defined as a machine that interacts with environment is not adequate for modeling situations where more than one IOSM's communicate with each other. So we now introduce the notion of communicating IOSM.

Definition 2.5 A *Communicating IOSM*(= CIOSM) M is a 5-tuple <St, s_0, L_{in}, {M}×N×L_{out}, Tr> where St, s_0, L_{in} and L_{out} are as in Definition 2.1 and N = N′∪{Env} where N′ is the set of identifiers for CIOSM's such that M∈N′. The set of transitions is now:

$$Tr \subseteq \{s\text{–}v/\mathbf{u}\rightarrow s' \mid s,s' \in St \wedge v \in L_{in} \wedge \mathbf{u} \in (\{M\}\times N\times L_{out})^*\}$$

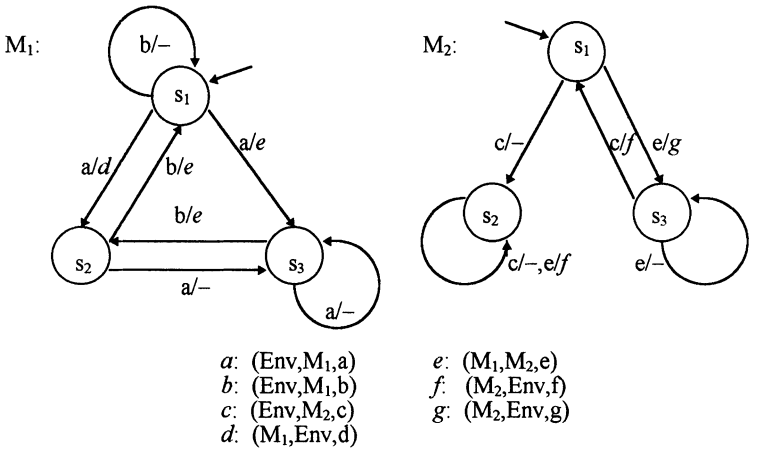

Figure 2 CIOSM's M_1 and M_2.

The difference between IOSM and CIOSM is that CIOSM sends messages of the form (*sender*, *receiver*, *symbol*) to a specific receiver, more than one message in a single transition. Other notions defined for IOSM can be extended for CIOSM in an obvious way.

Definition 2.6 A system Σ of n CIOSM's is <{(M_i,Q_i) | 1 ≤ i ≤ n}, {Env}×N×$L_{\Sigma,in}$, N×{Env}×$L_{\Sigma,out}$, $s_{\Sigma,0}$> where:

(1) M_i, 1 ≤ i ≤ n, is a CIOSM <St_i, $s_{i,0}$, $L_{i,in}$, $L_{i,out}$, Tr_i> as in Definition 2.5.

(2) Q_i, 1 ≤ i ≤ n, is an input queue for M_i.

(3) $L_{\Sigma,in}$ is a set of external input symbols.

(4) $L_{\Sigma,out}$ is a set of external output symbols.

(5) The initial state of the system is $s_{\Sigma,0}$ = <($s_{1,0}$, Q_0), ...,($s_{n,0}$, Q_n)> where Q_i=ε and $s_{i,0}$ is the initial state of M_i, 1≤ i ≤n.[2]

The term *external* indicates association with the environment Env. In this model, there is one explicitly defined input queue for each process and implicit communication channels between

[2] Definition 2.6 shows that though we see a system of CIOSM's mostly as a black-box, it is assumed that the names of the individual CIOSM's are known.

each pair of processes. Since a system Σ can contain more that one CIOSM's, an actual input from the environment is of the form (Env,i,v) where i is the identifier of the receiving CIOSM and v is an external input symbol. External outputs have Env as its unique receiver.

A system of CIOSM's Σ is said to be *closed* if no transitions consume inputs from or produce outputs to the environment. We call system state changes caused by consumption of an external input by a CIOSM or of an external output by the environment *observable* (or *visible*) *state changes.* Other system state changes are called *internal* (or *invisible*) *state changes.*

Figure 2 which is a slight modification of an example in [Luo 94] describes a system consisting of two CIOSM's, M_1 and M_2. The input queues are not shown. In the next definition, we make precise the global behavior of a system of CIOSM's.

Definition 2.7. Let Σ be a system of CIOSM's as in Definition 2.6. Let $w \in N \times N \times L_{\Sigma,in}$, $u \in N \times N \times L_{\Sigma,out}$, $\mathbf{w} \in (N \times N \times L_{\Sigma,in})^*$, $\mathbf{u} \in (N \times N \times L_{\Sigma,out})^*$, s be a state of Σ. Then

(1) $\sigma(s,(Env,i,v)) = \{(s',(Env,i,v)/\mathbf{u}) \mid s = <...,(s_{i,j},Q_i.v),...> \wedge$
$\exists i' \in N: (s_{i,j} - (i',i,v)/(i,i_1,o_1)...(i,i_m,o_m) \rightarrow s_{i,j'}) \in Tr_i \wedge$
$s' = \mu(s,\{(s_{i,j'},Q_i)/(s_{i,j},Q_i.v)\} \cup \{(s_{ik,jk},Q.o_j)/(s_{ik,jk},Q) \mid 1 \le k \le m \wedge i_k \neq Env\})$
$\mathbf{u} = \varphi((i,i_1,o_1)...(i,i_m,o_m))\}$

where $\mu(s,\varnothing) = s$
$\mu(<...,(s,Q),...>, P \cup \{(s',Q')/(s,Q)\}) = \mu(<...,(s',Q'),...>,P)$
$\varphi(\varepsilon) = \varepsilon$
$\varphi((i,Env,o).\mathbf{u}) = (i,Env,o).\varphi(\mathbf{u})$
$\varphi((i,i',o).\mathbf{u}) = \varphi(\mathbf{u})$ if $i' \neq Env$

(2) $\sigma(s,\varepsilon) = \{(s,\varepsilon/\varepsilon)\}$
$\sigma(s_1,w_1...w_{m+1}) = \{(s_{m+2},\mathbf{w}w_{m+1}/\mathbf{u}u_{m+1}) \mid (s_{m+1},\mathbf{w}/\mathbf{u}) \in \sigma(s_1,w_1...w_m) \wedge$
$(s_{m+2},w_{m+1}/u_{m+1}) \in \sigma(s_{m+1},w_{m+1})\}$

(3) $s(\mathbf{w}) = \{\mathbf{w}/\mathbf{u} \mid (s',\mathbf{w}/\mathbf{u}) \in \sigma(s,\mathbf{w})\}$

(4) $\Sigma(\mathbf{w}) = s_{\Sigma,0}(\mathbf{w})$

In (1), σ with a single external input defines a set of new states reached by processing of a single input from the environment followed by a sequence of data movements (including a sequence of internal state changes) out of the receiving CIOSM. The reception of a message and sending messages as a response to it constitutes an atomic action. Thus input from the environment is instantly placed into the input queue of the receiving CIOSM. A system state in which input queues are all empty is called a *stable state*.[3] μ is a function that updates states. φ extracts a sequence of external inputs or outputs from a sequence of inputs or outputs, respectively.

(2) extends σ to a sequence of external inputs. (2)-(4) for systems of CIOSM's are similar to (2)-(4) for IOSM's of Definition 2.2. In a system defined by Definition 2.7, *deadlock* is said to occur when any combination of (external and internal) inputs cannot change system state. *Livelock* is said to occur when system state changes forever and diverges.

[3] This contrasts with the definition of stable state in [1] in which the system waits for input from the environment regardless of whether inputs queues and channels are empty or not.

Definition 2.8 Let Σ_1 and Σ_2 be systems of CIOSM's with the same set of identifiers for CIOSM's N and the same input alphabet L_{in}. Then

Σ_1 and Σ_2 are *observationally equivalent* (denoted $\Sigma_1 \equiv_o \Sigma_2$)
if and only if
$\forall \mathbf{w} \in (\{Env\} \times N \times L_{\Sigma,in})^*$: $\Sigma_1(\mathbf{w}) = \Sigma_2(\mathbf{w})$.

Definition 2.9 Let Σ_1 be an implementation of a specification Σ_2 of a system of CIOSM's. Then

Σ_1 *interoperate with respect to* Σ_2 (denoted $\underline{interop}_{\Sigma 2}(\Sigma_1)$) if and only if $\Sigma_1 \equiv_o \Sigma_2$

This is a relative definition of interoperability. An absolute definition either should be contained in a given specification explicitly or should be agreed upon in advance by specification providers and implementors. An implication of the definition is that interoperability testing of implementations is a conformance testing of the global state space defined by the whole system. This conformance testing of interoperability is restricted by the observability and controllability that are allowed in a particular system configuration (or architecture) and here we assert that the practical limit on observability and controllability is set by the slowness of the environment which justifies us in focusing only on stable states. Then the additional ingredient interoperability testing provides is to fill the gap left by usual conformance testing. The real work of interoperability assurance should come at the stage of specification development. But the fine points that may have been missed at the time of specification writing (including validation and prototyping) can be augmented through testing.

3. TEST SEQUENCE GENERATION FOR ADAPTIVE TESTING

In this section, we concentrate on test generation for nondeterministic IOSM's. In Section 4, we will apply the test generation method developed here for adaptive testing to interoperability test generation.

Definition 3.1 A *path tree* of a state s_i (denoted T_{Pi}) of IOSM M is a tree such that

(1) The root of T_{Pi} corresponds to the initial state of M
(2) Each node corresponds to a state of M
(3) Each edge of T_{Pi} corresponds to some transition of M with the matching departure and destination states and the matching label.
(4) All leaf nodes of T_{Pi} correspond to s_i.

For the IOSM M in Figure 1, we may select the path trees as in the T_{Pi} column of Table 1. Thus for example, a/x.a/x and a/y.a/y are two branches of {a/x.a/x, a/y.a/y}[4] and traversal of either branch takes M to s_3. A path tree may not be unique. In P_i column are selected paths to the states of M. In this example, a path tree that exhibits the essential difference of the adaptive method from the conventional approach is T_{P3}. In the conventional approach, only one path to a state is selected at the time of test generation. In the adaptive approach, more that one paths to a state are selected and, depending on the responses from the implementation under test, different branches are followed.

[4] We use set notation to represent path trees as well as state identification trees.

	P_i	W_i	TP_i	Ts_i
i = 0	ε	{a/x, a/y}	{ε}	{{a/x.a/x, a/y.a/y}}
i = 1	a/x	{b/u.a/x.b/w}	{a/x}	{{b/u.a/x.b/w}}
i = 2	a/y	{b/w.a/y}	{a/y}	{{b/w.a/y}}
i = 3	**a/x.a/x**	**{b/u.a/y}**	**{a/x.a/x, a/y.a/y}**	**{{b/u.a/y, b/w.a/x}}**

Table 1 Path trees and state identification trees for M in Figure 1.

For the purpose of state identification, notions of varying generality such as Distinguishing Sequence, UIO sequence, W set, Wp sets and generalized Wp sets have been used [Chan 89][Chow 78][Sabnani 88][Luo 94]. Depending on whether a single sequence is used or a set of sequences is used for state identification purpose, it may be called *state identification sequence* or *state identification set*. We now introduce a further generalization along the line of state identification set.

Definition 3.2 A *state identification tree* of a state s_i (denoted Ts_i) of IOSM M is a tree such that

(1) The root of Ts_i corresponds to s_i
(2)-(3) are as in Definition 3.1
(4) Each path from the root to a leaf node is a state identification sequence.

The notion of state identification tree can be generalized to the notion of state identification forest for the cases where a single state identification sequence does not exist. A *state identification forest* of a state s_i (denoted Fs_i) of IOSM M is a set of trees such that each tree satisfies (1)-(3) of Definition 3.2 and

$$\{P \mid P \text{ is a path from the root to a leaf node of } T \wedge T \in Fs_i\}$$

is a state identification set.

The notions of *UIO* and *Partial UIO tree* were used in [Kim 94] for a similar purpose. Main differences of our approach are that we do not require that all transitions with the same input symbol should appear in the tree as branches and we use trees not only for state identification but also for tours to states. Thus even with the presence of the status message feature which makes use of input/output pairs of status enquiry/status name, our method can still give efficiency improvement.

For the IOSM M in Figure 1, we may select the state identification trees as in the Ts_i column of Table 1. By selecting W_0 as in Table 1, it takes two sequences of labels to distinguish s_0 from the other states. However, Ts_0 uses a tree each branch of which alone is enough to distinguish s_0 from the other states. But the two sequences were chosen so that the first labels of them constitute all possible transitions that can be taken upon receiving the input a. For state s_3, a sequence of length 2, i.e. b/u.a/y, is a state identification sequence but, in the adaptive approach, two sequences b/u.a/y and b/w.a/x are used. But again they were selected so that the first label has the common input b and u and w are all possible outputs for b.

For testing, the cost of test suite execution is more important than the cost of test suite generation since once generated a test suite is used repeatedly. In order to measure efficiency of a test suite execution, we define a sort of worst-case test case execution function $\xi_w(M,t)$ where M is the name of an IOSM or a system and t is a test case. Since tree in our method a

test case is a of which a branch is selected dynamically and a case is finished when a leaf node is reached, in order to apply ξ_w, we make the following assumptions about execution behavior:

(i) Once a path is selected, the same path is not taken again and all other possible paths have been traversed once and

(ii) Once a correct subsequence is taken, then the subsequence of it is always followed (with no need of backtracking) in the next traversal.

(iii) Only after all other possibilities are exhausted obeying the above two restrictions, the responses expected for the test case being executed are obtained.

What would happen in the real world differs from this. However, the model gives some idea about test suite execution efficiency. We illustrate this point with the example in Figure 1.

With the assumptions (i), (ii) and (iii), in order to reach s_3 via P_3 it takes three tours from the initial state, i.e. a/y, a/x.a/z and a/x.a/x in that order. We denote this as

a/y
a/x.a/z
a/x.a/x

or in a linear fashion as (a/y, a/x.a/z, a/x.a/x).[5] In contrast, with T_{P_3} it takes two tours, e.g. (a/x.a/z, a/x.a/x). Then ξ_w(M, a/x.a/x) = (a/y, a/x.a/z, a/x.a/x) and ξ_w(M, {a/x.a/x, a/y.a/y}) = (a/x.a/z, a/x.a/x). And if we let |x| denote a function that counts the number of labels in x, |(a/y, a/x.a/z, a/x.a/x)| = 5 and |(a/x.a/z, a/x.a/x)| = 4. Thus the path tree approach is more efficient for the tour to s_3.[6]

Similar efficiency improvement can be achieved though state identification trees under the same assumptions. So when identifying state s_3 using Ts_3 instead of a singleton identification set W_3, Ts_3 requires only 2 trials (b/w.a/z, b/w.a/x) instead of 3 trials with W_3, i.e. (b/w, b/u.a/z, b/u.a/x). When traversal to s_3 is combined with the state identification of s_3, the following comparison can be made:

P_3 combined with W_3	T_{P_3} combined with Tw_3
a/y	a/x.a/z
a/x.a/z	a/x.a/x.b/w.a/z
a/x.a/x.b/w	**a/x.a/x.b/w.a/x**
a/x.a/x.b/u.a/z	
a/x.a/x.b/u.a/y	

If we compute test case execution effort in terms ξ_w, then the cost of adaptive testing is 10 whereas that of the conventional testing is 14.

When selecting T_{P_i} and Tw_i, however, discretion should be used. A helpful heuristic is to avoid a very long identification sequence while trying to minimize backtracking to the initial state due to a nondeterministic selection of unwanted transitions. Thus as many transitions from a state as possible should become branches of the state identification tree as long as they have the same input symbol. When a node has an incomplete set of children for an input due to absence of a single state identification sequence or efficiency consideration, then the node should be

[5] Return to the initial state can be achieved by appending pre-calculated paths to the initial state or by appending the reset primitives at the end of each test case. But this is an issue that can be treated independently from the subject of this paper.

[6] One might consider that after a/x.a/z we could get a/y. This suggests using only the assumptions (i) and (iii) but then analysis is far more complicated.

marked with the output sequences not in the branches so that those output sequences are not considered as faults.

Some approach uses *tree structured test suite* [Kim 94]. Such an approach is efficient for finding faulty implementations but is rather difficult to give analysis of the cause of failure. Also when an implementation passes a test suite, it is no more efficient than *tabular test suite* approach such as ours in which test cases have corresponding predetermined test purposes.

4. INTEROPERABILITY TEST GENERATION FOR ADAPTIVE TESTING

In this section, we assume the systems of CIOSM's under consideration (both as specifications and implementations) are free of livelocks. A similar assumption was made for the systems of CNFSM's in [Luo 94]. The assumption makes it possible to generate test suites which can detect all interoperation errors in a finite amount of time.

In [Luo 94], a system of communicating finite state machines was assumed to operate in a *slow environment*. In a slow environment, inputs can be sent from the environment to the system only in situations where all the queues are empty. [Arakawa 92] assumes the *single stimulation principle* which requires that a single stimulation must be given at stable states where no change to the system can occur in a stable state without a further stimulation. We assume the slow environment assumption.

Interoperability Test Generation Algorithm

Input: A system Σ of n CIOSM's as defined in Definition 2.6 which has no livelocks.
Output:An interoperability test suite Π.

Step 1: Model Σ as a trace-equivalent nondeterministic IOSM M using the slow environment assumption.
Step 2: Derive a minimal trace-equivalent observable IOSM M′ from M.
Step 3: For each state s_i of M′, construct a path tree T_{Pi} and a state identification tree Ts_i (or a forest Fs_i).
Step 4: Let k be the number of states of M′ and m be the upper bound m ($\geq$ k) on the number of states in the global state space of the system implementation under test. Then a state cover Π_1 and a transition cover Π_2 are constructed as follows:

$\Pi_1 = \{\mathbf{x}.\mathbf{y}.Ts_j \mid s_0 - \mathbf{x} \rightarrow s_i \wedge \mathbf{x} \in T_{Pi} \wedge s_i - \mathbf{y} \rightarrow s_j \wedge \mathbf{y} \in \bigcup_{0 \leq i \leq m-k} L^i \wedge s_i, s_j \text{ are states of } M'\}$

$\Pi_2 = \{\mathbf{x}.\mathbf{z}.Ts_j \mid s_0 - \mathbf{x} \rightarrow s_i \wedge \mathbf{x} \in T_{Pi} \wedge s_i - \mathbf{z} \rightarrow s_j \wedge \mathbf{z} \in L^{m-k+1} \wedge s_i, s_j \text{ are states of } M'\}$

where $L^0 = \{\varepsilon\}$ and $L^{q+1} = L^q.(L_{in}/(L_{out})^*)$

Step 5: Let Π_Σ be the result of optimizing $\Pi_1 \cup \Pi_2$ by repeatedly applying the rule:
Remove a test case t if
(1) all branches of t are subsequences of some deterministic test case or
(2) a test case is a subtree of another test case.

Without the slow environment assumption, a global state space graph with a potentially large number of nodes could be constructed in Step 1. With the assumption, the number of states in

M is bounded by $|St_1|\times|St_2|\times...\times|St_n|$. In Step 2, by transforming into a minimal observable IOSM, a further reduction in the size of state space may be possible.

"complete-testing assumption" [Luo 94] or "all weather conditions" [Milner 80] is the assumption that all possible paths of reachability graph can be traversed, i.e. with sufficiently many attempts all alternatives of nondeterministic behavior can be selected. With the assumption, observational equivalence of a system of CIOSM's and hence interoperability become properties that can be verified.

Proposition Let Σ_1 be an implementation of a specification Σ_2 of a system of CIOSM's, $t\in((N\times N\times L_{\Sigma,in})/(N\times N\times L_{\Sigma,out})^*)^*$, $\underline{exec}(\Sigma_1,t)$ the resulting set of traces of executing a test case t sufficiently many times with respect to Σ_1 and $(v_1/w_1.v_2/w_2.....v_q/w_q)^{in} = v_1v_2...v_q$. Then

$$\underline{interop}_{\Sigma 2}(\Sigma_1) \text{ if and only if } \forall t\in\Pi_{\Sigma 2}\text{: } \underline{exec}(\Sigma_1,t) = \Sigma_2(t^{in})$$

Thus the above algorithm generates an interoperabililty test suite that guarantees the interoperation of the system implementation with respect to the system specification but passing interoperabilty test alone does not guarantee that individual components of the system have been correctly implemented.

We illustrate the algorithm with the system of CIOSM's in Figure 2. To keep the example to a manageable size, we let m = k in this example.

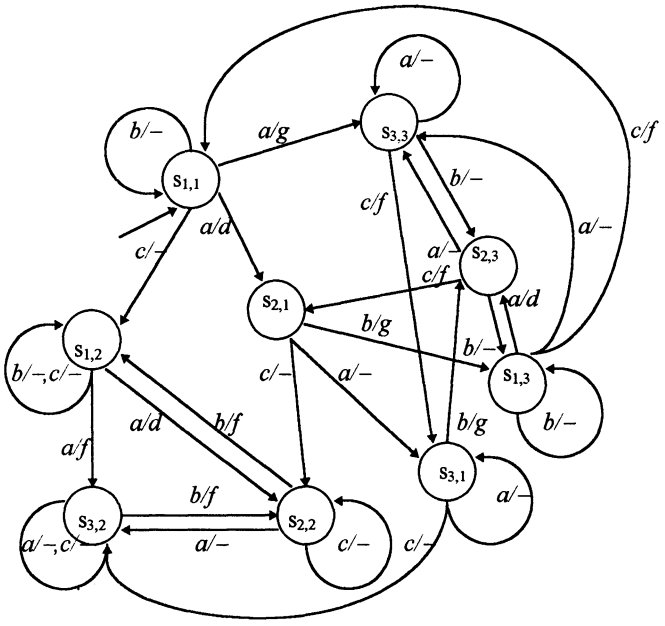

Figure 3 System state space as an observable IOSM.

	Conventional Method		Adaptive Method	
	P_i	W_i	T_{Pi}	T_{Wi}
$s_{1,1}$	ε	{***a/g***}	{ε}	{{***a/d.b/g, a/g***}}
$s_{1,2}$	*c/–*	{***a/d, a/f***}	{*c/–*}	{{***a/d.b/f, a/f***}}
$s_{1,3}$	*a/d.b/g*	{***c/f.a/g***}	{*a/d.b/g*}	{{***c/f.***{***a/d.b/g, a/g***}}}
$s_{2,1}$	*a/d*	{***b/g.c/f.a/g***}	{*a/d*}	{{***b/g.c/f.***{***a/d.b/g, a/g***}}}
$s_{2,2}$	*c/¬.a/d*	{***b/f.a/d***}	{*c/¬.a/d*}	{{***b/f.***{***a/d,a/f***}}}
$s_{2,3}$	*a/g.b/–*	{***b/¬.c/f.a/g***}	{*a/g.b/–*}	{{***b/¬.c/f.*** {***a/d.b/g, a/g***}}}
$s_{3,1}$	***a/d.a/–***	{*b/g.c/f.b/g*}	{***a/d.a/¬, a/g.c/f***}	{{*b/g.c/f.b/g*}}
$s_{3,2}$	*c/¬.a/f*	{*b/f.a/–*}	{*c/¬.a/f* }	{{*b/f.a/–*}}
$s_{3,3}$	*a/g*	{*b/¬.c/f.b/g*}	{*a/g*}	{{*b/¬.c/f.b/g*}}

Table 2 Path trees and state identification trees for M_1 and M_2 in Figure **2.**

By Step 1, a global state space graph with a large number of nodes is constructed. In Step 2, by transforming into a minimal observable IOSM and applying the slow environment assumption, the number of states is reduced to 9. Figure 3 is the result of Step 2. Step 3 is where we apply the adaptive method of Section 3 and the results are in the rightmost two columns of Table 2.

In Table 2, we emphasized the differences between the conventional method and the adaptive method by bold face letters. To build the particular table, we observe that the path to $s_{3,1}$ is the only reasonable case for adaptive tour to a state and $s_{1,1}$ and $s_{1,2}$ are definitely the cases for adaptive state verifications. Based upon adaptive state verifications of the two states, verification of many other states can be naturally done in an adaptive way. However, trying to do so for $s_{3,1}$, $s_{3,2}$ or $s_{3,3}$ would result in too lengthy test cases and was avoided here.

By Step 4, state covers and transition covers are constructed. The resulting state covers are shown in Appendix A and contain 10 test cases for the conventional method and 9 for the adaptive method. The transition covers are shown in Appendix B and contain 34 and 30 test cases, thus together with state covers totaling 44 and 39 test cases, respectively. The result of Step 5 is shown in Appendix B with bold face letters. In the final compressed test suite obtained with the adaptive method there are 24 test cases, which is fewer by 3. For the particular example, all the test sequences in the state cover happened to be subsequences of those in the transition covers.

Appendix C contains examples of applying ξ_w to both approaches together with testing costs for all groups of test cases measured by it. Our adaptive method requires significantly less effort (155) compared with the conventional approach (229).

5. CONCLUSION

In this paper, we developed a test sequence generation method which can be applied to IOSM's with observable nondeterminism and applied it to interoperability test generation. Since systems of CIOSM's almost unavoidably show nondeterministic behavior, any useful interoperability test generation approach should cope with it. Since non-observable IOSM's can be transformed into equivalent observable machines, our method works for any systems of CIOSM's as long as the assumption of no livelocks is met.

Also in this paper, we defined a measure of worst case test execution cost and used it to compare the results of applying our approach and the conventional approach to a non-trivial example. For that example, the comparison shows 11% improvement in the number of test cases and 32% improvement of the testing cost in terms of ξ_w. Since an adaptive approach is particularly suitable for dealing with nondeterminism, we believe the improvement will be greater for systems with highly nondeterministic behavior.

In comparison with the conventional approaches, our method tends to require more effort at the test generation step. For the overall cost of testing, however, test generation cost is much less important than the cost of actual testing itself. For once a test suite is generated, it usually is used over and over again.

In relation to interoperabiltiy test generation for adaptive testing, the following are among the further research problems:

(i) Other than the slow environment assumption, what assumption can contribute to reducing the size of global system space without decreasing (or significantly decreasing) controllability and observability which are crucial to test result analysis?

(ii) Random state space analysis to validation and verification is a particularly efficient method for realistic protocols. Since nondeterministic behavior can be considered "random" in some sense, how could the adaptive method be combined with random state space analysis approach to yield a good interoperability test generation method?

REFERENCES

[Arakawa 92] Arakawa, N. and Soneoka, T., "A Test Case Generation Method for Concurrent Programs", *Protocol Test Systems, IV*, J. Kroon, R. J. Heijink and E. Brinksma (Eds.), Elsevier Science Publishers B. V. (North-Holland), 1992.

[Bonnes 90] Bonnes, G., "IBM OSI Interoperability Verification Services", The 3rd IWPTS, 1990.

[Castanet 94] Castanet, R. and Kone, O., "Deriving Coordinated Testers for Interoperability", *Protocol Test Systems , VI(C-19)*, O. Rafiq(Ed.), Elsevier Science B. V. (North-Holland), IFIP, 1994.

[Chan 89] Chan, W. Y. L., Vuong, S. T. and Ito, M. R., "An Improved Protocol Test Generation Procedure Based on UIOs", *SIGCOMM '89 Symposium: Communication Architecture and Protocols in Computer Comm. Review 19(4)*, , pp. 283-294, September 1989.

[Chow 78] Chow, T. S., "Testing Software Design Modeled by Finite-State Machines", *IEEE Trans. on S.E., Vol. SE-4, No. 3*, May 1978.

[Kajiwara 94] Kajiwara, M., Ichikawa, H., Itoh, M. and Yoshida, Y., "Specification and Verification of Switching Software", *IEEE Transactions on Communications, Vol. Com-33, No. 3*, March 1985.

[Kim 94] Kim, B. and Chun, W., "Generating Test Cases for Nondeterministic FSMs Using UIOTrees and PUIOtrees", Technical Report, Comp. Eng., Chungnam National University, 1994.

[Luo 94] Luo, G., Bochmann G. and Petrenko, A., "Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method", *IEEE Transactions on S.E., Vol 20, No. 2*, February 1994.

[Milner 80] Milner, R., "A Calculus of Communicating Systems," LNCS, Vol. 92, 1980.

[Phalippou 91] Phalippou, M., "The Limited Power of Testing", Proceedings of the 4th International Workshop on Protocol Test Systems, The Hague, October 1991.

[Sabnani 88] Sabnani, K. and Dahbura, A., "A Protocol Test Generation Procedure", Computer Networks and ISDN Systems, Vol. 15, pp. 285-297, 1988.

[Sidhu 89] Sidhu, D. P. and Leung, T. K., "Formal Methods for Protocol Testing: A Detailed Study", *IEEE Trans. on S.E., Vol. 15, No. 4*, April 1989.

[Vermeer 94] Vermeer, G.S. and Blik, H., "Interoperabilty Testing: Basis for the Acceptance of Communication Systems", *Protocol Test Systems, VI(C-19),* Elsevier Science Publishers B. V. (North-Holland), 1994.

Appendix A. Comparison of state covers

	Conventional Method (10 test cases)	Adaptive Method (9 test cases)
$s_{1,1}$	*a/g*	{ε}.{*a/d.b/g, a/g*}
$s_{1,2}$	*c/¬. a/d* *c/¬. a/f*	{*c/¬*}.{*a/d.b/f, a/f*}
$s_{1,3}$	*a/d.b/g. c/f.a/g*	{*a/d.b/g*}.{*c/f.*{*a/d.b/g, a/g*}}}
$s_{2,1}$	*a/d. b/g.c/f.a/g*	{*a/d*}.{*b/g.c/f.* {*a/d.b/g, a/g*}}
$s_{2,2}$	*c/¬.a/d. b/f.a/d*	{*c/¬.a/d*}.{*b/f.*{*a/d, a/f*}}}
$s_{2,3}$	*a/g.b/¬. b/¬.c/f.a/g*	{*a/g.b/¬*}.{*b/¬.c/f.*{*a/d.b/g, a/g*}}
$s_{3,1}$	*a/d.a/¬. b/g.c/f.b/g*	{*a/d.a/¬, a/g.c/f*}.{*b/g.c/f.b/g*}
$s_{3,2}$	*c/¬.a/f. b/f.a/¬*	{*c/¬.a/f*}.{*b/f.a/¬*}
$s_{3,3}$	*a/g. b/¬.c/f.b/g*	{*a/g*}.{*b/¬.c/f.b/g*}

Appendix B. Transition covers and compressed test suites

	Conventional Method (34 (**27**) test cases)	Adaptive Method (30 (**24**) test cases)
$s_{1,1}$	*a/d. b/g.c/f.a/g*	{ε}. *a/d.* {*b/g.c/f.*{*a/d.b/g, a/g*}}
	a/g. b/¬.c/f.b/g	{ε}. *a/g.* {*b/¬.c/f.b/g*}
	b/¬. a/g	{ε}. ***b/¬.*** {***a/d.b/g, a/g***}
	c/¬. a/d	
	c/¬. a/f	{ε}. *c/¬.* {*a/d.b/f, a/f*}
$s_{1,2}$	*c/¬. a/d. b/f.a/d*	{*c/¬*}. *a/d.* {*b/f.*{*a/d, a/f*}}
	c/¬. a/f. b/f.a/¬	{*c/¬*}. ***a/f.*** {***b/f.a/¬***}
	c/¬. b/¬. a/d	
	c/¬. b/¬. a/f	{*c/¬*}. ***b/¬.*** {***a/d.b/f, a/f***}
	c/¬. c/¬. a/d	
	c/¬. c/¬. a/f	{*c/¬*}. *c/¬.* {***a/d.b/f, a/f***}
$s_{1,3}$	***a/d.b/g. a/d. b/¬.c/f.a/g***	{***a/d.b/g***}. ***a/d.*** {***b/¬.c/f.***{***a/d.b/g, a/g***}}
	a/d.b/g. a/¬. b/¬.c/f.b/g	{***a/d.b/g***}. ***a/¬.*** {***b/¬.c/f.b/g***}
	a/d.b/g. b/¬. c/f.a/g	{***a/d.b/g***}. ***b/¬.*** {***c/f.***{***a/d.b/g, a/g***}}
	a/d.b/g. c/f. a/g	{*a/d.b/g*}. *c/f.* {*a/d.b/g, a/g*}
$s_{2,1}$	***a/d. a/¬. b/g.c/f.b/g***	{***a/d***}. ***a/¬.*** {***b/g.c/f.b/g***}
	a/d. b/g. c/f.a/g	{***a/d***}. ***b/g.*** {***c/f.***{***a/d.b/g, a/g***}}
	a/d. c/¬. b/f.a/d	{***a/d***}. ***c/¬.*** {***b/f.***{***a/d, b/f.a/f***}}
$s_{2,2}$	***c/¬.a/d. a/¬. b/f.a/¬***	{*c/¬.a/d*}. ***a/¬.*** {***b/f.a/¬***}
	c/¬.a/d. b/f. a/d	
	c/¬.a/d. b/f. a/f	{*c/¬.a/d*}. ***b/f.*** {***a/d.b/f, a/f***}
	c/¬.a/d. c/¬. b/f.a/d	{*c/¬.a/d*}. *c/¬.* {***b/f.a/d, b/f.a/f***}
$s_{2,3}$	***a/g.b/¬. a/¬. b/¬.c/f.b/g***	{***a/g.b/¬***}. ***a/¬.*** {***b/¬.c/f.b/g***}
	a/g.b/¬. b/¬. c/f.a/g	{*a/g.b/¬*}. *b/¬.* {*c/f.*{*a/d.b/g, a/g*}}
	a/g.b/¬. c/f. b/g.c/f.a/g	{***a/g.b/¬***}. ***c/f.*** {***b/g.c/f.***{***a/d.b/g, a/g***}}
$s_{3,1}$	***a/d.a/¬. a/¬. b/g.c/f.b/g***	{***a/d.a/¬, a/g.c/f***}. ***a/¬.*** {***b/g.c/f.b/g***}
	a/d.a/¬.b/g. b/¬.c/f.a/g	{***a/d.a/¬, a/g.c/f***}. ***b/g.*** {***b/¬.c/f.*** {***a/d.b/g, a/g***}}
	a/d.a/¬. c/¬. b/f.a/¬	{***a/d.a/¬, a/g.c/f***}. *c/¬.* {***b/f.a/¬***}

$s_{3,2}$	*c/¬.a/f. a/¬. b/f.a/–* *c/¬.a/f. b/f. b/f.a/d* *c/¬.a/f. c/¬. b/f.a/–*	*{c/¬.a/f}. a/¬. {b/f.a/–}* *{c/¬.a/f}. b/f. {b/f.{a/d, a/f}}* *{c/¬.a/f}. c/¬. {b/f.a/–}*
$s_{3,3}$	*a/g. a/¬. b/¬.c/f.b/g* *a/g. b/¬. b/¬.c/f.a/g* *a/g. c/f. b/g.c/f.b/g*	*{a/g}. a/¬. {b/¬.c/f.b/g}* *{a/g}. b/¬. {b/¬.c/f.{a/d.b/g, a/g}}* *{a/g}. c/f. {b/g.c/f.b/g}*

Appendix C. Comparison of testing costs in terms of ξ_w

	Conventional Method (ξ_w = 229)	Adaptive Method (ξ_w = 155)
$s_{1,1}$	4	3
$s_{1,2}$	30	14
$s_{1,3}$	37	28
$s_{2,1}$	24	17
$s_{2,2}$	39	20
$s_{2,3}$	20	15
$s_{3,1}$	*a/g* ***a/d.a/¬. a/¬. b/g.c/f.b/g*** *a/g* *a/d.a/¬.b/g. b/¬.c/f.a/d* ***a/d.a/¬.b/g. b/¬.c/f.a/g*** *a/g* ***a/d.a/¬. c/¬. b/f.a/–*** 26	***a/g.c/f. a/¬. b/g.c/f.b/g*** ***a/g.c/f. b/g. b/¬.c/f.a/d.b/g*** ***a/d.a/¬. c/¬. b/f.a/–*** 18
$s_{3,2}$	26	21
$s_{3,3}$	23	19