12

An approach to dynamic protocol testing

Sangjo Yoo, Myungchul Kim, and Deukyoon Kang
Korea Telecom Research & Development Group
Sochogu Umyundong 17, Seoul, 137-792
{sjyoo, mckim, dykang}@sava.kotel.co.kr

Abstract

Protocol conformance testing aims at checking if a protocol implementation conforms to the standard (or specification) it is supposed to support. The results of testing can be classified into global verdict showing the tested system is either error-free or faulty, and local verdict indicating whether each element (e.g., a transition in the FSM) of the system is implemented correctly or not. In reality, the conventional protocol test procedure may give wrong local verdicts in the initial stages of testing because the procedure uses predetermined test sequence. In this paper, we propose a dynamic procedure for protocol testing using Test Sequence Tree (TST). The procedure allows us to get local verdicts more correctly than the conventional methods. The TST is reconfigured dynamically to obtain accurate verdicts for the untested elements by feedback of the local verdicts of the tested elements. The proposed technique was tested on the ITU-T Q.2931 signalling protocol. Our results showed that the fault coverage of our test procedure is better than the conventional methods. An extension of the proposed dynamic testing technique to the nondeterministic FSM is also discussed.

Keywords

Protocol testing, test sequence tree, test path selection

Testing of Communicating Systems, M. Kim, S. Kang & K. Hong (Eds)
Published by Chapman & Hall

1 INTRODUCTION

Protocol conformance testing determines the conformance of a protocol implementation to its specification (or standard). Conformance testing of implementations with respect to their standards before their deployment to networks is important to vendors and network operators to promote interoperability and to ensure correct behaviour of the implementations.

There has been much work on automatic test sequence generation methods from Finite State Machine (FSM) (S. Naito, 1981)(G. Gonenc, 1970)(Krishan Sabnani, 1988)(Tsun S. Chow, 1978). Among them, Transition tour (T), Distinguishing Sequence (DS), Unique Input/Output (UIO), and characteristic set (W) methods are well known. DS, UIO, and W methods have better fault coverage than the T method (Deepinder P. Sidhu, 1989). The test sequences generated by DS, UIO, and W methods provide both global and local verdicts. Global verdicts indicate whether the tested system is error-free or faulty, whereas local verdicts show whether each element (e.g., a transition in FSM) of the system is implemented correctly or not. Local verdicts also provide information on error locations (indicating where the faulty transitions of the system are) and the degree of conformance that are not provided in global verdicts. However, the conventional protocol test procedure may give wrong local verdicts to the Implementation Under Test (IUT) having faulty elements because it uses fixed test sequences that are predetermined. The fixed test sequences are usually obtained by the conventional test sequence generation method (e.g., DS, UIO, or W methods).

In this paper, we propose a new dynamic procedure using the Test Sequence Tree (TST) for protocol testing which provides more accurate local verdicts than the conventional one. The TST is reconfigured dynamically during testing to get correct verdicts for untested elements by feedback of the local verdicts of the tested elements.

The rest of the paper is organized as follows; Section 2 surveys test sequence generation methods and test procedures, and points out the problem of the conventional test procedure. Section 3 describes the principles of dynamic protocol testing illustrated with a simple FSM. Our algorithm for dynamic protocol test procedure is proposed in Section 4. As a case study, the proposed model is applied to the IUT-T Q.2931 protocol (B-ISDN signalling procedure) in Section 5. In Section 6, extension of the test procedure to nondeterministic FSM is discussed. Finally, Section 7 concludes the paper.

2 PROBLEM STATEMENT

2.1 Overview of test sequence generation and test procedure

Protocol conformance testing includes a number of steps as shown in Figure 1, i.e., generating test sequence, applying the test sequence to the IUT, and

analyzing the test results.

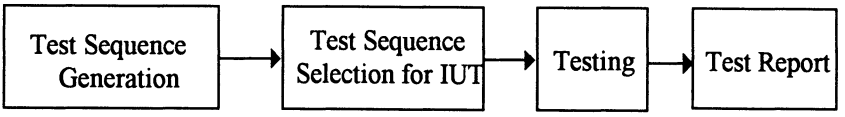


Figure 1 Conventional procedure for protocol conformance testing.

Test sequence generation methods based on FSM such as DS, UIO, and W methods are often used to produce the test sequences. In FSM, a transition consists of a head state, an input/output, and a tail state. The head state denotes the starting state of the transition and the tail state denotes the ending state of the transition. To test each transition of a machine, the following conventional procedure is usually applied to generate a sub test sequence for each transition:

1) Bring the IUT into a desired state starting from an initial state with the shortest path.
2) Apply the inputs to the IUT and observe the outputs from the IUT.
3) Verify that the IUT ends in the expected state.

Note that the well-known test sequence generation methods (e.g., DS, UIO, or W methods) are used in step 3). The test sequence for the entire machine is generated by concatenating the sub test sequences (Deepinder P. Sidhu, 1989) Myungchul Kim, 1995). In order to enhance efficiency, test sequence optimization using Chinese Postman algorithm (M. U. Uyar, 1986)(Alfred V. Aho, 1988), multiple UIOs (Shen Y. -N., F. Lombardi, 1989), and others techniques (Deepinder P. Sidhu, 1989)(Mon-Song Chen, 1990) have been proposed.

A paper (Samuel T. Chanson, 1992) which is closely related with our work makes use of the Abstract Test Case Relation Model (ATCRM) to derive test cases dynamically. In this paper, test purposes are obtained from the sequence of transitions from the initial state to all reachable states. There are some disadvantages to this approach:

1) Because the unit of test is a test purpose (i.e., a set of transitions) rather than focusing on one transition, it is harder to localize the errors.
2) The number of test purposes is large because the ATCRM covers all possible behaviors of the IUT and the test purposes consist of all possible paths from the initial state in ATCRM.

While Chanson and Li's work could be a solution to the problem stated, our method has the following advantages:

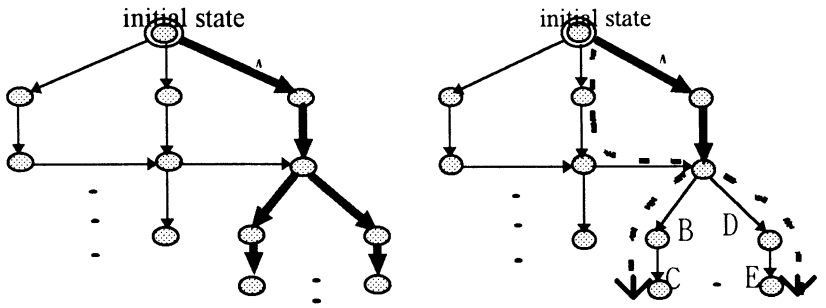
1) The number of test purposes is less with the same fault coverage
2) The ability of error localization is better because our model focuses on transitions not paths.

2.2 Problem of conventional protocol test procedure

The conventional protocol test procedure uses fixed test sequences. This gives rise to inefficiency. With reference to Figure 2-a, if transition A of IUT is implemented incorrectly, then the test sequence will give fail verdicts to all untested transitions following A (given by the bold arrows).

If the protocol has an alternative path consisting of transitions that are implemented correctly, then Figure 2-b shows that transitions B, C, D and E should not fail the test. Thus, the protocol test procedure using fixed test sequence may give wrong local verdicts to transitions that are implemented correctly because faulty transitions may affect subsequent transitions.

The conventional test procedure consists of deriving sub test sequences for each transition, optimizing the length of test sequence for machine, applying it to IUT, and then analyzing the test results. Table 1 shows the sub test sequences for each transition of the FSM in Figure 3 by getting the shortest path from the initial state to the head state of a transition to be tested, executing the transition, and verifying the tail state of the transition using the UIO method. The reset operation is added before each sub test sequence in order to ensure that sub test sequences always start at the initial state. Among the sub test sequences of Table 1, if the sub test sequence of transition i is included in the sub test sequence j, the testing for the transition i can be performed by the sub test sequence j as well. Thus the optimized test sequence can be derived as given in Table 2.



a) using shortest path from the initial state

b) using other correctly implemented path from the initial state

: state —→ : transition

: transition assigned as "fail" because of the faulty transition A

: path using correctly implemented transitions

Figure 2 Test result comparison by path selection.

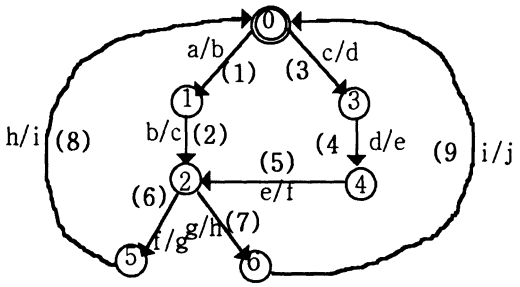


Figure 3 Finite State Machine M.

Table 1 Sub test sequences using UIO for the machine M in Figure 3

Transition	*Sub test sequence*	*Transition*	*Sub test sequence*
(1)	[reset/null, a/b, **b/c**]	(6)	[reset/null, a/b, b/c, f/g, **h/i**]
(2)	[reset/null, a/b, b/c, **f/g**]	(7)	[reset/null, a/b, b/c, g/h, **i/j**]
(3)	[reset/null, c/d, **d/e**]	(8)	[reset/null, a/b, b/c, f/g, h/i, **a/b**]
(4)	[reset/null, c/d, d/e, **e/f**]	(9)	[reset/null, a/b, b/c, g/h, i/j, **a/b**]
(5)	[reset/null, c/d, d/e, e/f, **g/h**]		

* The transition presented in **bold characters** is the UIO sequence for the transition.

Table 2 Optimized test sequence for the machine M in Figure 3

Test sequence for the machine M
[reset/null, a/b, b/c, f/g, h/i, a/b, reset/null, c/d, d/e, e/f, g/h, reset/null, a/b, b/c, g/h, i/j, a/b]

However, if the IUT has a faulty implementation of transition (1) (e.g., generating the output 'd' for the input 'a'), not only transition (1) is assigned a "fail" verdict, transitions (2), (6), (7), (8), and (9) will also be assigned the "fail" verdicts even though they are implemented correctly because transition (1) is part of the test sequence for testing those transitions. If transitions (3), (4), and (5) are implemented correctly, then we may adopt the path consisting of transitions (3), (4), and (5) as an alternative path for testing transitions (6), (7), (8), and (9). In this way, we can provide more accurate test results by isolating the effect of the faulty transition (1).

3 DYNAMIC SELECTION OF TEST PATH

We now propose a new test procedure for selecting an appropriate path dynamically from the initial state to the transition to be tested depending on the local errors in the IUT. The dynamic selection of test path makes it possible to get more accurate intermediate test results. Before proposing our dynamic path selection method, it is necessary to define some terms:

Definitions

- A **Set of Transitions (ST)** is the set of all transitions in a FSM M.
  $ST = \{t_1, t_2, \ldots, t_i, \ldots, t_n\}$ *where*
  t_i =<*a head state, an input/output, a tail state*> *and*
  n = *the total number of transitions of machine M.*
- A **Unique Path (UP_i)** is a path including transition t_i and a transition to verify the tail state of t_i, if there is only one possible path from the initial state to t_i.
- A **Set of Transitions in UP_i (STU_i)** is the set of all transitions in Up_i.
  $STU_i = \{t_1, \ldots, t_k\}$ $(0 < k \leq n)$.
- A **Path Test Sequence (PTS_i)** is the test sequence for transition t_i. PTS_i^q, the test sequence for t_i, is generated as follows: 1) apply the q-th path (if it exists) which brings the IUT from the initial state to the head state of t_i, 2) apply t_i, and 3) apply DS, UIO, or W methods to verify the tail state of t_i.

  $PTS_i^q = Path_i^q @ t_i @ Verification$(*for the tail state of the* t_i)
  where, @ : concatenation of sequence and
  $Path_i^q$ = *sequence of transitions of the q-th path from the initial state to* t_i.
- A **Test Sub-Sequence Tree ($TSST_i$)** is the set of all PTS_is for t_i.
  $TSST_i = \{PTS_i^1, \ldots, PTS_i^q, \ldots, PTS_i^j\}$
  where j = *the # of possible paths from the initial state to* t_i.
- A **Test Sequence Tree (TST)** for FSM M is
  $TST_M = \{TSST_1, \ldots, TSST_i, \ldots, TSST_n\}$.

Let us demonstrate how TST is set up initially and reconfigured dynamically during testing based on the results of local verdicts. For the Finite State Machine M in Figure 4, by using UIO sequence for tail state verification, the TST_M (Test Sequence Tree) for testing each transition in M is given in Figure 5.

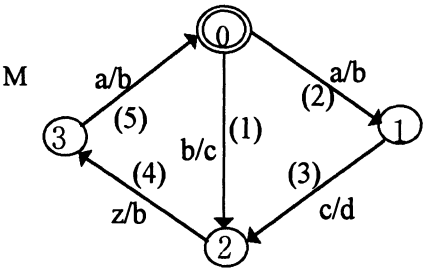


Figure 4 Finite state machine M.

Table 3 UIO sequences for machine

State	*UIO sequence*
0	[b/c]
1	[c/d]
2	[z/b]
3	[a/b, a/b]

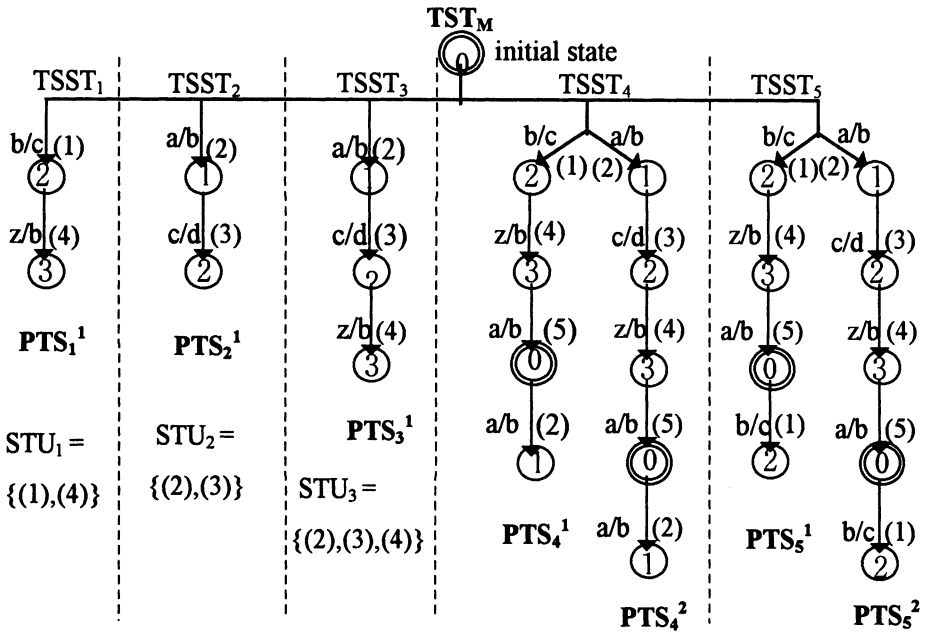


Figure 5 TST_M before testing.

For testing transition (1), there is only one path from the initial state to the transition. Therefore, $TSST_1$ for transition (1) is obtained by concatenating [b/c] in transition (1) and UIO sequence [z/b] for state 2 that is the tail state of transition (1).

For testing transition (4), there are two possible paths to bring the IUT to transition (4) from the initial state. The first one is [b/c] in transition (1) that is the shortest path from the initial state to transition (4), and the second one is [a/b, c/d] passing through transitions (2) and (3). (Notation '[a/b, c/d]' stands for concatenation of 'a/b' and 'c/d'.) Therefore, $TSST_4$ for testing transition (4) consists of PTS_4^1 = [b/c, z/b, a/b, a/b] and PTS_4^2 = [a/b, c/d, z/b, a/b, a/b]. As shown in Figure 5, note that $TSST_1$, $TSST_2$, and $TSST_3$ have STUs.

In using the TST in Figure 5, we start with testing transition (1) which is closest to the initial state. If a fault on the transition is detected; the test result of transition (1) is assigned a "fail" verdict; the transition is registered as a "faulty transition". To reconfigure TST_M as a result of the failure of transition (1), TST_M is searched to find a STU having transition (1) as its element. If a STU_i has transition (1) as an element of the set, the corresponding transition t_i is assigned a "fail" verdict automatically and does not have to be tested. In the case of TST_M in Figure 5, there is no transition having transition (1) as an element of its STU.

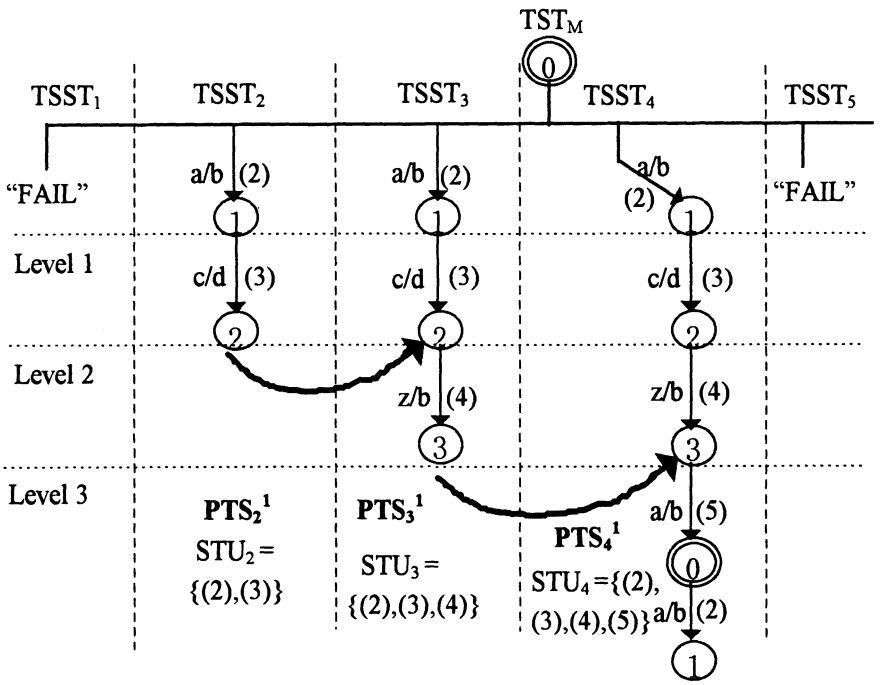


Figure 6 Reconfigured TST_M' right after detecting transition (1) is faulty.

The next step is to eliminate all PTSs which contain transition (1) from TST_M. In Figure 5, PTS_4^1 of $TSST_4$ for transition (4) and PTS_5^1 and PTS_5^2 of $TSST_5$ for transition (5) include transition (1). Therefore these three paths are eliminated from TST_M. As a result, we have only one PTS for transition (4) so that STU_4 must be created. In case of $TSST_5$ for transition (5), there is no remaining PTS for the transition so that the "fail" verdict is assigned automatically without testing. Figure 6 shows the reconfigured TST_M' after the testing of transition (1).

As shown in Figure 6, if the "pass" verdict is assigned to transition (2) as the result of observation [a/b, c/d], PTSs are searched to locate those having the same transition path (namely, [a/b, c/d]) up to level 2 from the initial state. As a result, PTS_3^1 of $TSST_3$ and PTS_4^1 of $TSST_4$ are selected. Of those, PTS_3^1 of $TSST_3$ having shorter length from the initial state is chosen for the next test. Since the transitions up to level 2 of PTS_3^1 are the same as those of PTS_2^1, only the test sequence corresponding to the transition [z/b] is applied for testing transition (3) of the IUT.

In the example given in Figure 4, if we do not use the proposed test procedure, transition (4) would be assigned a "fail" verdict because of the faulty implementation of transition (1) regardless of whether transition (4) is implemented correctly or not. The test procedure proposed can be depicted as in Figure 7.

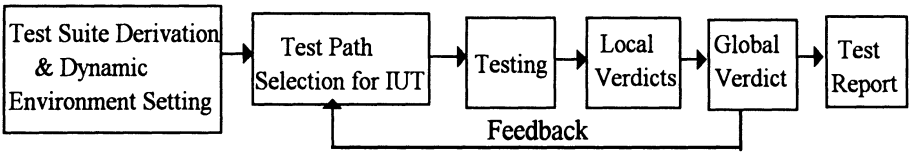


Figure 7 The proposed test procedure using dynamic selection of test path.

By using the result of every local verdict, TST is dynamically reconfigured to select an appropriate path test sequence.

The proposed dynamic test procedure has the following properties:

1) It uses an alternative path (if it exists) during testing when there is a problem (e.g., a faulty transition is detected) in the preamble path which brings the IUT to the transition to be tested.
2) If all possible paths from the initial state to a transition to be tested include faulty transitions, then the transition is automatically given a "fail" verdict without testing.
3) If a transition has passed the test, testing of further transitions starting from the current IUT state condition is performed without reset (i.e., restarting from the initial state) to minimize testing effort.

4 PROPOSED TEST PATH SELECTION ALGORITHM

In this section, we give an algorithm for the dynamic test path selection procedures proposed. It consists of two steps: initial construction of TST and Dynamic Test Path Selection.

(Step 1 : Initial Construction of TST before Testing)

begin
construct TSST;
setup TST;
compute STU;
end

For each transition in ST={t_1, . . . , t_i, . . . , t_n}, $TSST_i$ is constructed using DS, UIO, or W method for tail state verification. For testing of t_i, all PTS_is are generated. The PTS_is are rearranged in increasing order of length from the initial state to t_i. As a result, we obtain $TSST_i$ ={PTS_i^1, . . . , PTS_i^q, . . . , PTS_i^j}. Using TSST for each transition of FSM, the Test Sequence Tree (TST) is setup. Let the set of TSSTs be ordered according to the distance from the initial state. Therefore, we have TST ={$TSST_1$, . . . , $TSST_i$, . . . , $TSST_n$}. If there is only one path from the initial state to T_i, then compute STU_i for $TSST_i$.

(Step 2 : Testing and Dynamic Test Path Selection)

begin
for t_1 **to** t_n **in** $t_i \in$ ST **do**
begin

```
if "pass" or "fail" verdict is already assigned for t_i then
        continue;
    else
    begin
      for q:=1 to q:=j do ——— (1)
      begin
          execute transitions in PTS_i^q ;
          if unexpected output is observed then
          begin
              if PTS_i^q is the last path of t_i then
              begin
                  assign "fail" verdict to t_i test;
                  if any TSST_k of TST has STU and t_i is an element
                      of STU_k then assign "fail" to t_k test of TST;
                  break;
              end
          end
          else
          begin
              assign "pass" verdict to t_i test;
              If PTS_k exists then——— (2)
              begin
                  jump into level p of PTS_k;
                  break;
              end
          end
      end
    end
end
```

For each transition in ST, we obtain a local verdict. In statement (1), j is the number of all possible paths from the initial state to t_i in the q-th PTS. As a result of executing transitions in PTS_i^q, if unexpected output is detected and PTS_i^q is the last path of t_i, then the verdict of t_i is "fail" and $TSST_k$ having t_i in its STU_k is also assigned "fail" without testing. If the output is correct, a "pass" verdict assigned to t_i. In statement (2), we try to find PTS_k with the shortest sequence matching the sequence from the initial state to level p that is the last position of the currently executed t_i. If the PTS_k exists, then jump to level p of PTS_k.

5 COMPARISON OF EXPERIMENTAL TEST RESULT FOR B-ISDN Q.2931 SIGNALLING PROTOCOL

In this section, we compare our new test procedure using dynamic test path

selection method with the conventional one by applying both of them to real communication protocol testing. Figure 8 shows the simplified FSM for the call establishment and clearing procedure for the user side of ITU-T Q.2931 protocol. ITU-T Q.2931 is a recommendation for the User Network Interface (UNI) signalling protocol that is used in Asynchronous Transfer Mode network (ATM) and Broadband Integrated Digital Network (B-ISDN). The FSM of Figure 8 has 7 states and 15 input/output transitions.

Table 4 shows the UIO sequence for each state of the FSM in Figure 8. Table 5 and Table 6 list the test sequences for transitions using shortest path and Test Sequence Tree (TST) according to the proposed method, respectively. In the conventional test procedure, the test sequence for each transition presented in Table 5 is fixed by using only one path from the initial state and is not changed during testing. However in case of the proposed test procedure, multiple paths for transition (10), (11), (12), (13), (14), and (15) are allowed as shown in Table 6, and the path to be used is dynamically selected during testing.

To compare our test procedure with the conventional one for the FSM in Figure 8, a fault model is used. Generally, faults for FSM can be classified into three cases (Deepinder P. Sidhu and Ting-kau Leung, 1989) :

1) Produce an unexpected output for a given input and move to an expected state.
2) Produce an expected output for a given input and move to an unexpected state.
3) Produce an unexpected output for a given input and move to an unexpected state.

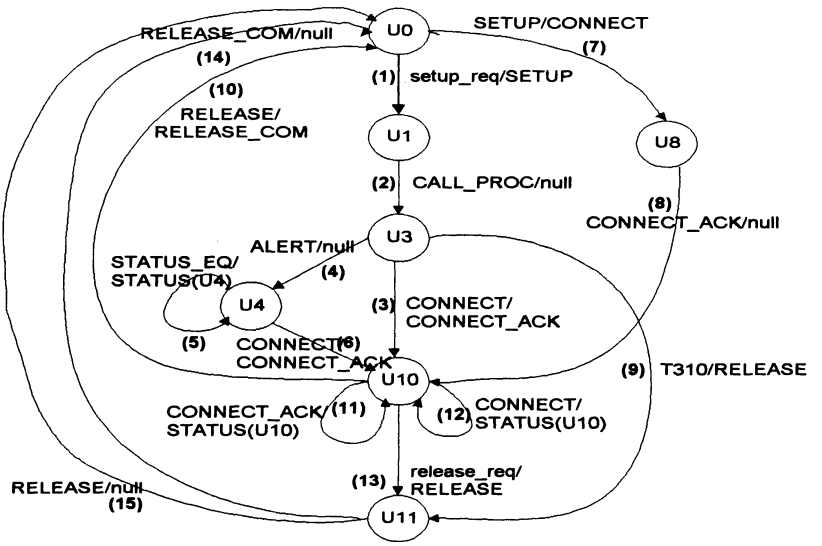


Figure 8 Simplified FSM of ITU-T Q.2931 signalling user side protocol for call establishment and clearing.

Table 4 UIO sequence for each state

State	*UIO sequence*	*State*	*UIO sequence*
U0	setup_req/SETUP	U8	CONNECT_ACK/null
U1	CALL_PROC/null	U10	release_req/RELEASE
U3	T310/RELEASE	U11	RELEASE_COM/null
U4	STATUS_EQ/STATUS(U4)		

Table 5 Test sequence for each transition using shortest path from the initial state

Transition	*Test sequence*	*Transition*	*Test sequencefor*
(1)	(1)-(2)	(9)	(1)-(2)-(9)-(14)
(2)	(1)-(2)-(9)	(10)	(7)-(8)-(10)-(1)
(3)	(1)-(2)-(3)-(13)	(11)	(7)-(8)-(11)-(13)
(4)	(1)-(2)-(4)-(5)	(12)	(7)-(8)-(12)-(13)
(5)	(1)-(2)-(4)-(5)-(5)	(13)	(7)-(8)-(13)-(14)
(6)	(1)-(2)-(4)-(6)-(13)	(14)	(7)-(8)-(13)-(14)-(1)
(7)	(7)-(8)	(15)	(7)-(8)-(13)-(15)-(1)
(8)	(7)-(8)-(13)		

Table 6 Test Sequence Tree (TST) structure using the proposed method

TSST	*PTS*	*Test sequence*	*TSST*	*PTS*	*Test sequence*
$TSST_1$	PTS_1^1	(1)-(2)	$TSST_{12}$	PTS_{12}^1	(7)-(8)-(12)-(13)
$TSST_2$	PTS_2^1	(1)-(2)-(9)		PTS_{12}^2	(1)-(2)-(3)-(12)-(13)
$TSST_3$	PTS_3^1	(1)-(2)-(3)-(13)		PTS_{12}^3	(1)-(2)-(4)-(6)-(12)-(13)
$TSST_4$	PTS_4^1	(1)-(2)-(4)-(5)	$TSST_{13}$	PTS_{13}^1	(7)-(8)-(13)-(14)
$TSST_5$	PTS_5^1	(1)-(2)-(4)-(5)-(5)		PTS_{13}^2	(1)-(2)-(3)-(13)-(14)
$TSST_6$	PTS_6^1	(1)-(2)-(4)-(6)-(13)		PTS_{13}^3	(1)-(2)-(4)-(6)-(13)-(14)
$TSST_7$	PTS_7^1	(7)-(8)	$TSST_{14}$	PTS_{14}^1	(7)-(8)-(13)-(14)-(1)
$TSST_8$	PTS_8^1	(7)-(8)-(13)		PTS_{14}^2	(1)-(2)-(9)-(14)-(1)
$TSST_9$	PTS_9^1	(1)-(2)-(9)-(14)		PTS_{14}^3	(1)-(2)-(3)-(13)-(14)-(1)
$TSST_{10}$	PTS_{10}^1	(7)-(8)-(10)-(1)		PTS_{14}^4	(1)-(2)-(4)-(6)-(13)-(14)-(1)
	PTS_{10}^2	(1)-(2)-(3)-(10)-(1)	$TSST_{15}$	PTS_{15}^1	(7)-(8)-(13)-(15)-(1)
	PTS_{10}^3	(1)-(2)-(4)-(6)-(10)-(1)		PTS_{15}^2	(1)-(2)-(9)-(15)-(1)
$TSST_{11}$	PTS_{11}^1	(7)-(8)-(11)-(13)		PTS_{15}^3	(1)-(2)-(3)-(13)-(15)-(1)
	PTS_{11}^2	(1)-(2)-(3)-(11)-(13)		PTS_{15}^4	(1)-(2)-(4)-(6)-(13)-(15)-(1)
	PTS_{11}^3	(1)-(2)-(4)-(6)-(11)-(13)			

For simplicity, we use only the fault model given by case 1) in this paper. For the FSM in Figure 8, we assume that all transitions of the FSM are possible faulty transitions. Also, for a given number of faulty transitions, we compute the possible

faulty machines as shown in Table 7. For example, there are 105 different FSM implementations in case the FSM has two faulty transitions. Because there are 15 testing of transitions for each faulty FSM implementation, we have 1,575 test results in total. For the ideal tester that can identify perfectly all faulty and correct transitions, the number of "pass" transitions is 13 because there are two faulty transitions among the fifteen transitions. When we apply the test sequences of Table 5 generated by the conventional method, the average number of "pass" transitions is 8.219 for the implementation that has two faulty transitions since we have 863 "pass" results. On the other hand, our proposed test procedure using dynamic test path selection method get 9.514 average "pass" transitions because we obtain 999 "pass" results. Table 7 shows that the dynamic testing method produce more accurate test results for the individual transitions.

Table 7 Experimental test results using the fault model

The # of Faulty transitions	*Total # of different FSM implement-ations*	*The # of all possible test results*	***Conventional test method using fixed test sequence***		***Proposed test method using dynamic test path selection***		***Ideal tester***
			Total # of "pass" results	*The # of average "pass" transitions*	*Total # of "pass" results*	*The # of average "pass" transitions*	*The # of "pass" transitions*
1	15	225	168	11.2	182	12.133	14
2	105	1,575	863	8.219	999	9.514	13
3	455	6,825	2,692	5.916	3,171	6.969	12
4	1,365	20,475	5,690	4.17	7,111	5.21	11
5	3,003	45,045	8,610	2.867	10,834	3.608	10
6	5,005	75,075	9,606	1.919	11,923	2.382	9
7	6,435	96,525	8,016	1.2456	9,642	1.4984	8
8	6,435	96,525	5,019	0.78	5,778	0.898	7
9	5,005	75,075	2,340	0.4675	2,566	0.5127	6
10	3,003	45,045	795	0.2647	834	0.2777	5
11	1,365	20,475	188	0.1377	191	0.1399	4
12	455	6,825	28	0.0615	28	0.0615	3
13	105	1,575	2	0.019	2	0.019	2
14	15	225	0	0	0	0	1

As shown in Figure 9, the fault coverage of the proposed test method is closer to the ideal tester than that of the conventional method. This shows that the proposed test procedure using dynamic test path selection method can be used more efficiently and effectively in testing for product implementation or in acceptance testing for procurement. This is particularly useful when the proposed test procedure is used for debugging in the protocol implementation phase.

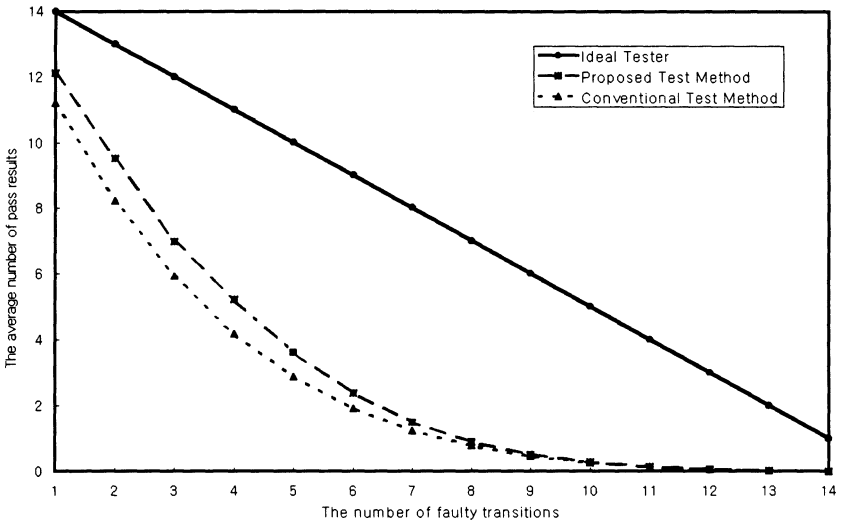


Figure 9 Test results comparison with ideal tester.

6 EXTENDING TO NONDETERMINISTIC FSM

In each state of the machine, if only one transition rule is executable, the machine moves to a new control state in a deterministic way. On the other hand, if more than one transition rules are executable for the same input, a nondeterministic choice is made to select a transition. The machine that can make such choices is called a nondeterministic machine. Most test sequence generation methods assume the FSM is deterministic. However many communication protocols exhibit nondeterminis. In this section, we present a method to extend our test procedure proposed in Section 3 to the observable nondeterministic FSM. The nondeterminism of ATM/B-ISDN may arise from the following reasons:

1) nondeterminism caused by options allowed in the specifications (e.g., in the ITU-T Q.2931 recommendation and ATM Forum UNI specification, sending a CALL-PROCEEDING message as the response of receiving a SETUP message is optional.).
2) nondeterminism caused by the messages which can be sent and received at any time for error report, check, or recovery (e.g., in the ITU-T Q.2931 recommendation and ATM Forum UNI specification, the STATUS-ENQUIRY message to ask for the state of peer entity can be sent in any state and at any time except the null state.).

For the nondeterministic FSM given in Figure 10, assume the next transition in state 5 is decided in a nondeterministic way. The output of either 'y' or 'z' can be observed as the response to the input 'x'. In this case, we say that transition (2) and transition (3) are in "companion relationship" in state 5 and state 5 is a

"nondeterministic node". The TST of the FSM in Figure 10 is constructed in Figure 11. The transitions connected by the dashed lines are in companion relationship with each other. If the expected output 'y' for the input 'x' in transition (2) of PTS_m^1 for the *transition m* test is not observed, but the unexpected output 'z' in transition (3) in state 5 is observed instead, then, move to the next transition level (i.e., level j) of the path PTS_n^1 that uses the same path transition as PTS_m^1 up to level *i* and apply the remaining test sequence to the IUT. In case of testing *transition n*, if the output 'y' is observed instead of 'z' in transition (3) of PTS_n^1, move to level *j* of PTS_m^1 and continue the testing. By using the above procedure, nondeterministic FSMs can also be tested efficiently based on the proposed Test Sequence Tree and dynamic test path selection method.

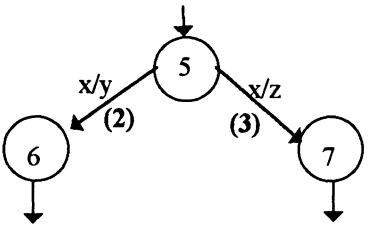


Figure 10 A part of nondeterministic FSM.

In addition, our approach avoids testing of duplicate paths at nondeterministic nodes as illustrated in the example above. During testing, if we get one of the outputs allowed against an input given at a nondeterministic node, the testing proceeds to the transition matching the output. The original transition is marked as "not tested yet". On the other hand, the transition in companion relationship with the original one is tested and it's verdict is given. This approach avoids duplicate testing on nondeterministic nodes, and thus provides more effective protocol testing to nondeterministic FSM.

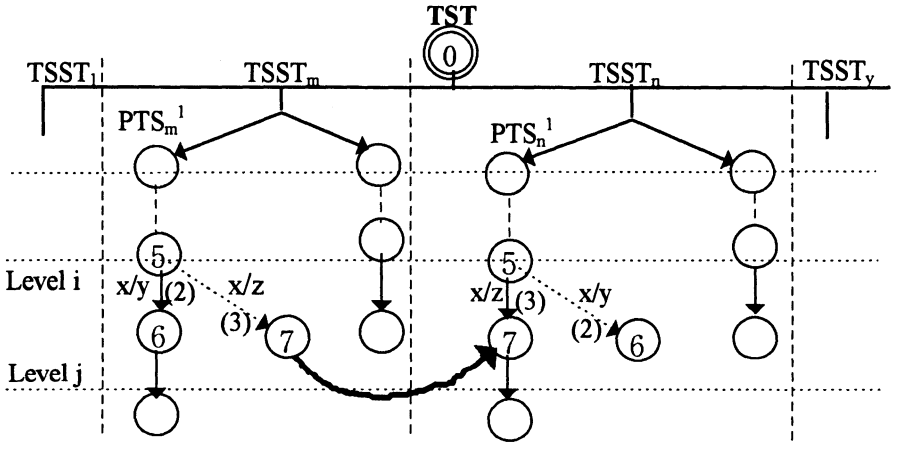


Figure 11 Dynamic test path selection method for the FSM of Figure 10.

7 CONCLUSION

In this paper, we have proposed a new dynamic protocol testing procedure that produces more correct local verdicts which helps to reduce the testing overhead. The Test Sequence Tree (TST) is the basic data structure used. The TST is reconfigured dynamically for the untested transitions during testing based on the results of local verdicts of already tested elements. We have applied our proposed dynamic test path selection algorithm to the FSM describing ATM/B-ISDN signalling protocol and compared it with the conventional method in terms of fault coverage. The results showed that the proposed test procedure generates more accurate verdicts and can be used more efficiently and effectively in testing. This method can be used for product implementation or in acceptance testing for procurement. Finally we have also presented some initial ideas in extending our proposed test procedure to deal with nondeterministic FSMs.

ACKNOWLEDGEMENTS

The authors would like to thank to Dr. Samuel T. Chanson and Dr. Sungwon Kang for their helpful comments on this paper.

8 REFERENCES

S. Naito and M. Tsuoyama (1981), "Fault detection for sequential machines by transition tours", IEEE 11th Fault Tolerant Comp. Symp., 238-43.

G. Gonenc (1970), "A method for the design of fault detection experiments", IEEE Trans. on Computer., vol C-19, pp. 551-558.

Krishan Sabnani and Anton Dahbura (1988), "A Protocol Test Generation Procedure", Computer Networks and ISDN Systems, vol.15, 285-97.

Tsun S. Chow (1978), "Testing Software Design Modeled by Finite-State Machines", IEEE Trans. on Software Eng., vol.SE-4, no.3, 178-87.

Deepinder P. Sidhu and Ting-kau Leung (1989), "Formal Methods for Protocol Testing: A Detailed Study", IEEE Trans. on Software Eng., vol.15, no.4, 413-26.

Myungchul Kim, S.T. Chanson, and Sangjo Yoo (1995), "Design for testability for protocols based on formal specifications", The 8th Int'l Workshop on Protocol Test Systems, 257-69.

M. U. Uyar and A. T. Dahbura (1986), "Optimal Test Sequence Generation for Protocols: The Chinese Postman Algorithm Applied to Q.931", IEEE Global Telecom. Conf.

Alfred V. Aho, Anton T. Dahbura, David Lee, and M. Umit Uyar (1988), "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours", Protocol Specification, Verification, and Testing, VIII, 75-86.

Shen, Y.-N., F. Lombardi, and A. T. Dahura (1989), "Protocol Conformance

Testing Using Multiple UIO Sequences", Protocol Specification, Verification, and Testing, IX, 131-43.

Mon-Song Chen, Yanghee Choi, and Aaron Kershenbaum (1990), "Approaches Utilizing Segment Overlap to Minimize Test Sequences",Protocol Specification,Verification,and Testing, IX,85-98.

S. T. Chanson and Qin Li (1992), "On Static and Dynamic Test Case Selections in Protocol Conformance Testing", The 5th Int'l Workshop on Protocol Test Systems, 225-67

9 BIOGRAPHY

Sangjo Yoo received BA in electric communication engineering from Hanyang Univ. in 1988 and MS in electrical engineering from the Korea Advanced Institute of Science and Technology in 1990. Currently he is with the Korea Telecom R&D Group as a member of technical staff.

Myungchul Kim received BA in electronics engineering from Ajou Univ. in 1982, MS in computer science from the Korea Advanced Institute of Science and Technology in 1984, and Ph.D in computer science from the Univ. of British Columbia in 1992. Currently he is with the Korea Telecom Research and Development Group as a managing director, Chairman of Profile Test Specification-Special Interest Group of Asia-Oceania Workshop, and is the Co-Chair of the 10th IWTCS'97. His research interests include protocol engineering on telecommunications and multimedia.

Deukyoon Kang received BA in electronics engineering from Kumoh Nat'l Institute of Technology in 1993 and MS in computer science from Pohang Institute of Science and Technology in 1995. Currently he is with the Korea Telecom R&D Group as a member of technical staff.