# 동적 프로토콜 적합성 시험
## (Dynamic Protocol Conformance Test)

박 진 희 † 김 명 철 †† 최 지 영 ††† 유 상 조 ††††
(Jinhee Park) (Myungchul Kim) (Jiyoung Choe) (Sangjo Yoo)

**요 약** 프로토콜 적합성 시험은 프로토콜 명세에 내재되어 있는 애매성으로 인하여 벤더들이 구현한 프로토콜들이 상호운용되지 못하는 경우를 예방하기 위하여 시행하는 절차이다. 이 시험은 구현한 프로토콜이 프로토콜 명세에 적합하게 구현되어 있는지를 시험하는 것으로 ISO/IEC JTC1을 통해 국제 표준으로 제정되어 있다. 그러나 이 시험 방법은 고정적인 시험 시퀀스를 사용하기 때문에 정확한 시험 결과를 도출하지 못하는 경우가 종종 발생한다. 이런 문제는 프로토콜 FSM을 구성하는 여러 트랜지션들이 시험 시퀀스에 포함되어 시험 대상 트랜지션의 시험에 영향을 미치기 때문에 발생한다. 이 논문은 동적 적합성 시험 방법 (Dynamic Conformance Test Method: DCTM)을 제안하여 이런 문제를 해결하도록 한다. DCTM은 기존의 방법에서와 같은 고정적인 시험 시퀀스를 사용하는 것이 아니라 시험 중 동적으로 alternative 경로를 시험 시퀀스로 선택하는 방법으로 기존 적합성 시험 방법보다 fault coverage 면에서 향상된 결과를 나타낸다. 또한 제안한 DCTM이 기존 적합성 시험 방법보다 fault coverage가 향상된다는 것을 보이기 위해 시험 도구를 개발하여 TCP 프로토콜에 적용하여 본다.

***Abstract*** Protocol conformance test is used to promote interoperability of protocol implementatons developed by venders. Non-interoperability between protocol implementations may be caused by ambiguity and/or misinterpretation of the protocol specifications by vendors. Conventional method on protocol conformance test has been standardized by ISO/IEC JTC1 with the purpose of whether a protocol implementation conforms to its specification. However, sometimes the conventional method gives wrong test results because the test is based on static test sequences. This problem is caused by the fact as some failed transitions of a protocol FSM included in test sequences have an effect on the test result of transitions to be tested. In this paper, a new approach called Dynamic Conformance Test Method (DCTM) is proposed to solve this problem. DCTM dynamically selects test sequence durng testing depending on an information showing an alternative path without failed tranistions. As a result, the fault coverage of the DCTM is better than that of the conventional test method. In order to demonstrate the fault coverage of DCTM compared to that of the conventional method, a testing tool is implemented and appied to the TCP protocol.

## 1. 개 요

정보통신 서비스가 널리 확산되면서 이와 관련된 프로토콜의 연구, 개발, 제품화가 활발히 진행되고 있다. 프로토콜이란 통신하는 개체사이의 약속을 말하며 이 약속을 지킴으로써 원하는 서비스를 할수 있게 된다.

그러나 프로토콜 명세에 내재되어 있는 애매성과 이를 구현하는 벤더들의 서로 다른 해석에 따른 구현으로 인하여 프로토콜 구현물들 사이에 상호운용되지 않는 경우가 발생할 수 있다. 이런 문제를 해결하기 위하여 프로토콜 구현물이 프로토콜의 명세에 대해 정확히 구현되었는지를 시험할 필요가 있다. 이 시험을 적합성 시험 (conformance test)이라고 하는데 이것은 프로토콜 구현물, 즉Implementation Under Test (IUT)가 그 프로토콜 명세에 맞게 구현되었는지 시험하는 것을 목적으로 하며 이때 IUT는 입력 포트와 출력 포트를 가진

† 비 회 원 : 전자부품연구원 인터넷미디어연구센터 연구원
pjhe@keti.re.kr
†† 종신회원 : 한국정보통신대학원대학교 공학부 교수
mckim@icu.ac.kr
††† 비 회 원 : 아이디(주) 연구원
choeji@it.co.kr
†††† 비 회 원 : 인하대학교 정보통신전문대학원 교수
sjyoo@inha.ac.kr
논문접수 : 2000년 4월 24일
심사완료 : 2001년 5월 14일

블랙박스로 간주한다. IUT를 시험하기위한 시험 시퀀스는 프로토콜 명세로부터 얻어지며 입력과 출력의 쌍으로 이루어져 있다. 이 입력들이 IUT의 입력 포트로 들어가게 되고, 그 결과가 출력 포트로 나오게 되는데 이 때 그 결과는 시험 시퀀스의 출력과 일치하는지를 체크한다[1]. 시험 시퀀스는 서브 시험 시퀀스의 집합으로 이루어지며, 서브 시험 시퀀스는 preamble, 해당 트랜지션, postamble의 접합체로 만들어진다. preamble이란 초기상태에서 시험대상 트랜지션까지 거쳐야 하는 트랜지션들의 열을 말하고, postamble은 시험 시퀀스가 시험대상 트랜지션을 위한 것임을 증명하는 부분으로 트랜지션의 도착 상태를 확인하는 시퀀스를 말한다.

현존하는 프로토콜 적합성 시험은 고정적인 시험 시퀀스를 사용하기 때문에 종종 정확한 fault를 찾아내기가 어렵다. 즉 시험대상 트랜지션을 위한 시험 시퀀스 생성과정에서 preamble과 postamble 부분에 fault 트랜지션이 포함될 수 있는데 이것이 시험대상 트랜지션의 진단 결과에 영향을 미친다. 이 논문에서는 이러한 문제를 해결하기 위하여 소프트웨어 테스트 분야의 연구에서 얻어진 프로그램 dependence를 이용한 디버깅 개념을 프로토콜 적합성 시험에 이용하려고 한다. 프로그램 dependence는 프로그램 statement들 사이에 존재하는 관계를 말하며 대표적으로 "data dependence"와 "control dependence"가 있다. Data dependence는 프로그램의 변수에 값을 할당하는 statement가 다른 statement에 영향을 주는 것이고, control dependence는 conditional structure에서 condition 부분과 condition에 영향을 받는 statement 사이의 dependence를 말한다. 소프트웨어 테스트 분야에서는 이러한 dependence를 이용하여 하나의 큰 프로그램을 여러 개의 작은 독립적인 프로그램, 즉 slice로 나누어 오류나 버그 등을 쉽게 찾아내려는 연구가 진행되고 있다[2][3].

프로토콜 분야에서도 Abstract Test Case Relation Model (ATCRM)을 이용하여 시험 케이스를 동적으로 생성함으로써 이런 문제를 해결하려는 시도[4]가 있었다. 그러나 이 방법은 테스트 단위가 하나의 트랜지션이 아니라 트랜지션 집합이어서 에러를 지역화 하기가 어렵고, ATCRM이 IUT의 모든 가능한 행위들을 다루고 있기 때문에, 시험 대상들이 ATCRM안의 초기 상태에서 모든 가능한 경로들을 포함하고 있어서 시험해야 할 시험 케이스가 많아진다.

프로토콜 적합성 시험 중 시험 시퀀스 생성 과정에서 트랜지션 사이에 dependence가 발생된다. 여기서 dependence란 시험 시퀀스 중에 포함된 트랜지션이 시험대상 트랜지션의 진단 과정에 영향을 미치는 것을 말한다. 즉 시험 시퀀스 생성시 그 시퀀스에 preamble과 postamble 부분에 포함된 트랜지션들이 fault라면 시험대상 트랜지션이 제대로 구현되었다해도 진단 결과는 fail이다.

이 논문에서는 프로토콜 시험에서 시험 시퀀스 생성시 preamble, postamble과 시험대상 트랜지션 사이의 관계, 즉dependence 때문에 일어나는 문제를 해결하기 위하여 DCTM이라는 새로운 방법을 제시하고자 한다. 이 방법은 시험 시퀀스를 만들때 트랜지션이 가질 수 있는 모든 서브 시험 시퀀스를 나타내는 데이타 구조인 Test Sequence Tree(TST)를 사용하여 발견된 결함 트랜지션을 내포하지 않은 서브 시험 시퀀스를 동적으로 선택하려는 것이다. 즉 시험 시퀀스내의 preamble과 postamble 부분에 포함되어 있는 트랜지션과 시험대상 트랜지션 사이의 dependence를 고려하여 시험대상 트랜지션에 보다 정확한 진단을 내리기 위한 방법이다. 이런 진단을 local verdict이라고 하는데 이는 각 트랜지션이 올바로 구현되었는지 아닌지를 판단하는 것으로 본 논문에서 제안한 방법은 정확한 local verdict을 내게 되어 결국 시스템 전체의 conformance를 판단하는 global verdict의 향상을 가져오게 된다. 이 방법은 논문 [4]의 방법보다 더 적은 개수의 시험 케이스를 가지고도 동일한 fault coverage를 갖으며 시험 대상이 경로가 아니라 하나의 트랜지션에 집중되어 있어 에러를 지역화 하기가 쉬운 장점을 가지고 있다. DCTM이 기존의 적합성 시험 방법론보다 fault coverage가 향상된다는 것을 보이기 위해 방법론을 시험 도구로 개발하여 Transmission Control Protocol(TCP)에 적용하여 본다.

이 논문은2장에서 프로토콜 적합성 시험에서 사용되고 있는 기존의 방법과 문제점을 살펴보고, 3장에서는 2장에서 기존 방법을 사용할 때 발생하는 문제를 해결하여 더 좋은 fault coverage를 산출할 수 있는 DCTM을 제안하도록 한다. 4장에서는 이 두 방법을 비교해 보기 위해 시험 도구를 구현하여 TCP 프로토콜에 적용하고 그 결과를 비교해본 후 5장에서 결론을 맺도록 한다.

## 2. 기존 적합성 시험 방법과 문제점

ISO/IEC JTC1을 통해 국제 표준으로 제정되어 사용되고 있는 프로토콜에 대한 적합성 시험[5]은 그림 1과 같이 시험 시퀀스 생성, IUT에 시험 시퀀스 적용, 시험 그리고 시험 결과 분석의 여러 단계를 포함하고 있다.

트랜지션은 출발 상태, 입력, 출력, 도착 상태로 구성되어 있는데 출발 상태는 트랜지션의 시작 상태를, 도착 상태는 트랜지션의 마지막 상태를 나타낸다. 기존의 시

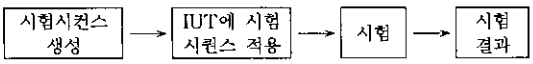



그림 1 프로토콜 적합성 시험을 위한 기존 방법의 절차

험 방법은 아래와 같은 방법으로 각 트랜지션의 서브 시험 절차를 실행하고 있는데, 여기서 (3)의 도착 상태를 검증하는 부분에서는 DS, W, UIO [1]방법 등이 사용될 수 있다.

***서브 시험 절차***

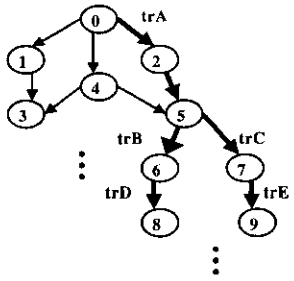
(1) IUT의 상태를 초기 상태에서 최단 경로를 따라 시험을 시작 하려는 상태로 놓는다.

(2) IUT에 시험 대상 트랜지션에 대한 입력을 적용하고 결과를 관찰한다.

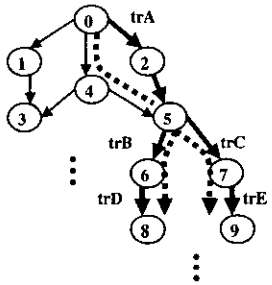
(3) IUT가 기대된 상태에서 끝났는지를 검증한다.

각 트랜지션을 위한 서브 시험 시퀀스는 위의 세 단계를 거친 후 만들어지는데 이 시퀀스들을 하나의 열로 합침으로써 프로토콜 전체를 시험하기 위한 시험 시퀀스가 만들어진다. 그러나 위 세 단계를 거쳐 만들어지는 시험 시퀀스들은 고정된 시험 시퀀스이기 때문에 정확한 진단 결과를 도출하지 못하게 되는 경우가 발생하는데 이를 설명하고자 한다.

그림 2에 여러 트랜지션들로 구성된 IUT의 예가 있는데, 여기서 원으로 표현된 부분은 상태를 나타내고, 화살표로 표현된 부분은 트랜지션을 나타내며 숫자와 영문자로 나타내어지는 레이블은 상태와 트랜지션의 이름을 나타낸다. 그림 2 (a)에서 처럼 만약 IUT의 트랜지션들 중 트랜지션 trA가 잘못 구현된다면 시험 시퀀스에 트랜지션 trA를 포함하는 아직 시험되지 않은 트랜지션들에게 영향을 미친다. 그림 2 (a)에서는 굵은 화살표로 이루어진 경로를 고정적으로 사용하게 되는데 이때 트랜지션 trA의 fault 트랜지션이라면 트랜지션 trB, trC, trD, trE는 올바른 판정을 받을 수 없게 된다. 만약 프로토콜이 올바로 구현된 트랜지션들로 구성된 alternative 경로를 가지고 있다면 trB, trC, trD, trE는 올바른 판정을 줄 수 있게 된다. 그림 2 (b)에서 점선의 화살표로 된 트랜지션이 alternative 경로를 나타내는데 이때 트랜지션 trA가 fault라도 alternative 경로를 적용하게 되면 트랜지션 trB, trC, trD, trE는 trA의 "fail" 판정의 영향을 피할 수 있게 된다. 따라서 alternative 경로를 고려하지 않고 미리 고정된 시험 시퀀스를 사용하는 기존 프로토콜 시험 절차는 잘못 구현된 트랜지션들이 시험 시퀀스에 포함되어 시험대상 트랜지션에 영향을 주기 때문에 잘못된 Local 판정을 낳게 된다. 즉 시험 시퀀스 생성 과정에서의 트랜지션 사이의 dependence 때문에 문제가 발생하는 것이다.




(a) 초기 상태에서 최단 경로 적용




(b) 초기 상태에서 alternative 경로 적용

그림 2 경로 결정에 의한 시험 결과 비교

그림 3은 임의의 FSM M을 보여주고 표1은 그림3의 FSM의 각 트랜지션들을 위한 서브 시험 시퀀스를 나타낸다. 이것들은 초기상태로부터 시험대상 트랜지션의 출발상태까지 최단 경로를 적용하고, 시험대상 트랜지션을 시험하고, UIO 방법을 사용하여 시험대상 트랜지션의 도착 상태를 검증하는 방식의 기존의 프로토콜 시험 방법을 사용함으로써 얻어진다. 여기에서 추가된 리셋(reset)동작은 서브 시험 시퀀스들이 항상 초기 상태에서 시작하도록 하기 위해 적용된다. 그리고 시험 시퀀스는 표1의 각 서브 시험 시퀀스를 하나로 합침으로써 만

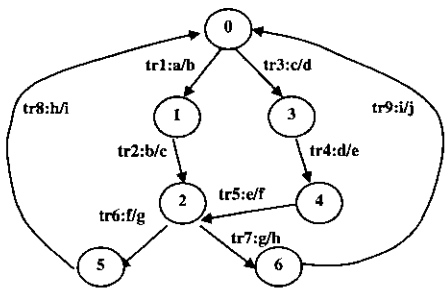



그림 3 유한 상태 기계M

들어지는데 이중에서 만약 트랜지션 trA의 서브 시험 시퀀스가 트랜지션 trB의 서브 시험 시퀀스에 포함된다면 트랜지션 trA를 위한 시험은 트랜지션 trB의 시험에 의해 수행될 수 있다. 이런 관계를 이용하여 시험 시퀀스의 최적화 작업이 수행되며, 그 결과는 표 2에 나타나 있다.

표 1 UIO를 사용한 그림3의 기계 M을 위한 서브 시험 시퀀스

| Transition | Sub Test Sequence |
|---|---|
| tr1 | [*reset/null*, a/b, b/c] |
| tr2 | [*reset/null*, a/b, b/c, f/g] |
| tr3 | [*reset/null*, c/d, d/e] |
| tr4 | [*reset/null*, c/d, d/e, e/f] |
| tr5 | [*reset/null*, c/d, d/e, e/f, g/h] |
| tr6 | [*reset/null*, a/b, b/c, f/g, h/i] |
| tr7 | [*reset/null*, a/b, b/c, g/h, i/j] |
| tr8 | [*reset/null*, a/b, b/c, f/g, h/i, a/b] |
| tr9 | [*reset/null*, a/b, b/c, g/h, i/j, a/b] |

표 2 그림3의 기계 M을 위한 최적화된 시험 시퀀스

| 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] |

만약 여기에서 IUT의 트랜지션 tr1이 입력 a에 대해 올바른 출력을 내지 못한다면 트랜지션 tr1뿐만 아니라 시험 시퀀스에 트랜지션 tr1을 포함하고 있는 트랜지션 tr2, tr6, tr7, tr8, tr9가 올바르게 구현되었어도 fail 판정을 받게된다. 이와 같이 기존의 프로토콜 시험 방법은 시험 시퀀스를 구성하는 트랜지션들이 시험대상 트랜지션의 판정에 영향을 미쳐 올바른 시험을 방해하는 경우가 종종 발생한다.

## 3. 동적 프로토콜 시험 방법

2장에서는 이미 사용되고 있는 프로토콜 적합성 시험 방법과 문제점에 대해 살펴보았다. 이 방법은 고정된 시험 시퀀스를 사용하기 때문에 시험 대상 트랜지션에 부정확한 판정을 줄 수 있다. 이를 개선하기 위하여 이 논문에서는 DCTM이라는 새로운 방법을 제안한다. 이 방법은 시험 중 시험 판정에 따라 시험 경로를 동적으로 다시 선택하는 것으로 시험 시퀀스 생성시 관계하는 트랜지션 사이의 dependence를 분석하여 더욱 정확한 시험 결과를 도출한다. DCTM에서 고려하는 dependence는 크게 preamble 부분에 포함되는 트랜지션과의 dependence와 postamble 부분에 포함되는 트랜지션과의 dependence로 나누어 생각해 볼 수 있다. 이 장에서는 두 경우에 대해 각각 살펴볼 것이며 우선 다음과 같은 용어들을 정의하도록 한다.

***정의***

- 트랜지션의 집합 (Set of Transition : ST)
  ST는 FSM M안에 있는 모든 트랜지션들의 집합을 말하며 $t_i$는 i번째 트랜지션을 말한다.
  $ST = \{t_1, t_2, \ldots, t_i, \ldots, t_n\}$
  *($t_i$ =<a head state, an input/output, a tail state>, n = Machine M의 총Transition 개수)*
- 유일한 경로 (Unique Path : UP)
  초기 상태에서$t_i$까지 존재하는 단 하나의 경로이다.
  $UP_i$ = 트랜지션$t_i$ @ Verification ($t_i$의 도착 상태)
  *(여기에서 @는 concatnation으로 열을 합치는 것을 의미한다)*
- UP의 집합 (Set of Transition in UP : STU)
  $STU_i$는 $UP_i$의 모든 트랜지션들의 집합이다.
  $STU_i = \{t_1, \ldots, t_k\}$ $(0 < k \leq n)$.
- 경로 시험 시퀀스 (Path Test Sequence : PTS)
  $PTS_i$는 트랜지션$t_i$를 위한 시험 시퀀스이다. $PTS_i^q$는 다음의 순서로 생성된다.
  (1) IUT를 초기 상태에서$t_i$의 출발 상태로 가져간다.
  (2) $t_i$를 적용한다.
  (3) DS, UIO, W를 $t_i$의 도착 상태 검증에 적용한다.
  $PTS_i^q = Path_i^q@t_i@Verification$ ($t_i$의 도착 상태)
  *($Path_i^q$는$t_i$의 초기 상태로부터 q번째 경로의 Transition들의 Sequence)*
- 서브 시험 시퀀스 트리 (Test Sub-Sequence Tree : $TSST_i$)
  $TSST_i$는$t_i$의 모든 $PTS_i$의 집합이다.
  $TSST_i = \{PTS_i^1, \ldots, PTS_i^q, \ldots, PTS_i^j\}$
  *(j는 $t_i$의 초기 상태에서 가능한 모든 경로의 개수를 나타낸다)*
- 시험 시퀀스 트리 (Test Sequence Tree : TST)
  시험 시퀀스 트리 FSM M을 위한 모든 시험 시퀀스를 하나의 트리로 나타내는 구조이다.
  $TST_M = \{TSST_1, \ldots, TSST_i, \ldots, TSST_n\}$
- 다수의 postamble 집합 (Multiple Postamble Set : MPS)
  MPS는 어떤 상태에서 가질 수 있는 postamble을 구성하는 시퀀스들의 집합으로, 원소가 UIO이면 multiple

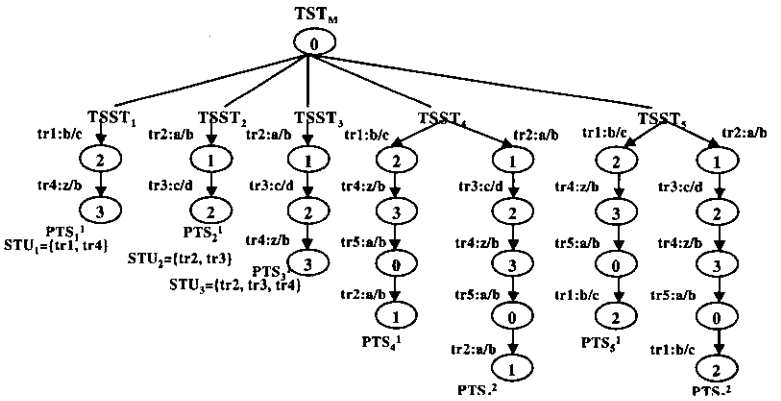



그림 5 시험의 TSTM

UIO, DS이면 Multiple DS, W이면 Multiple W이다.

$MPS_i = \{a_1, a_2, , a_j, , a_n\}$

*($MPS_i$는 i번째 상태에 대한 MPS를 말하며 $a_j$는 j번째 postamble 시퀀스를 말한다. 여기서 n은 하나의 상태에서 존재할 수 있는 postamble시퀀스의 개수를 나타낸다.)*

### 3.1 Preamble에 대한 DCTM

시험 시퀀스를 생성할 때 preamble 부분에 포함된 트랜지션들과 시험대상 트랜지션과의 dependence란 preamble 부분에 포함된 트랜지션들이 시험대상 트랜지션의 시험 판정에 영향을 주는 현상을 말한다. Preamble에 대한 DCTM은 이 dependence 문제를 해결하기 위한 방법으로 초기상태에서 시험대상 트랜지션까지의 경로 즉, preamble 부분의 시퀀스를 최단 경로 하나만 선택하는 것이 아니라 가능한 모든 경로를 탐색한다. 여기서 가능한 모든 경로를 생각할 때 preamble 부분에 초기상태가 두번 이상 포함되는 루프(loop)현상이 발생할 수 있는데, 여기서는 경로 선택에서 루프를 제외하도록 한다. 이렇게 하여 만들어지는 시험 시퀀스들은 트리 형태(TST)로 저장되며 시험 시퀀스를 동적으로 재구성하는데 중요한 자료가 된다.

이제부터 TST가 초기에 어떻게 구성되고 Local 판정에 따라 테스트 중 어떻게 동적으로 재설정되는지 살펴보도록 하겠다. 그림 4의 FSM M의 도착 상태 검증

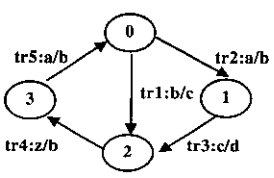



그림 4 Finite State Machine M

을 위한 UIO 시퀀스로 표 3의 내용을 적용하면 그림 5와 같은 각 트랜지션을 테스트하기 위한 $TST_M$을 구할 수 있다.

표 3 Machine M의 UIO Sequences

| State | UIO Sequence |
|---|---|
| 0 | [b/c] |
| 1 | [c/d] |
| 2 | [z/b] |
| 3 | [a/b, a/b] |

그림 5를 보면 트랜지션 tr1, tr2, tr3는 하나의 시험 경로를, 트랜지션 tr4, tr5는 두 개의 가능한 시험 경로를 가지고 있다. 트랜지션 tr1의 경우는 초기 상태에서부터 그 트랜지션까지 시험 경로가 단 하나만 존재하므로 트랜지션 tr1을 위한 $TSST_1$은 트랜지션 tr1인 [b/c]와 트랜지션 tr1의 도착 상태인 상태2의UIO 시퀀스 [z/b]를 붙임으로써 시험 시퀀스를 생성한다. 트랜지션 tr4의 경우는 초기상태에서부터 트랜지션 tr4까지의 최단 경로와 트랜지션 tr2와tr3을 포함하는 또 하나의 경로, 즉 두 개의 가능한 시험 경로를 가지고 있다. 그러므로 트랜지션 tr4를 시험하기 위한 $TSST_4$는 [b/c, z/b, a/b, a/b]인 $PTS_4^1$와 [a/b, c/d, z/b, a/b, a/b]인 $PTS_4^2$로 구성될 수 있다. 그림5에서 볼 수 있듯이 트랜지션 tr1, tr2, tr3는 하나의 시험 경로를 가지므로 $TSST_1$, $TSST_2$, $TSST_3$ 가 $STU_i$를 가지고 있다.

그림 5에서 TST를 시험할 때 먼저 초기상태에서 가장 가까운 트랜지션 tr1을 시험하기로 가정하자. 시험 중 만약 잘못 구현된 트랜지션이 발견된다면 트랜지션

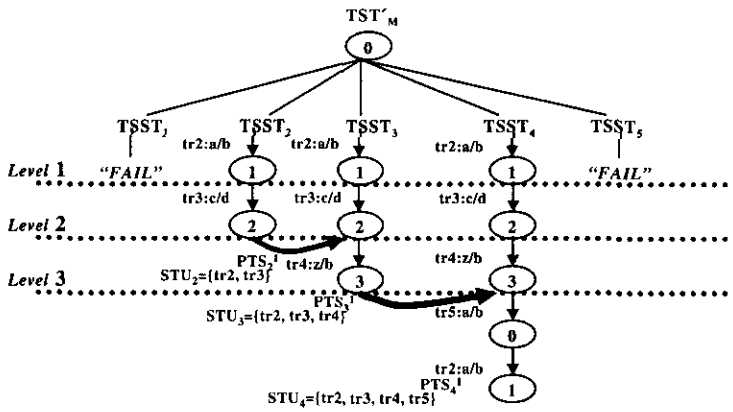



그림 6 트랜지션 tr1이 결함이라는 것을 탐지한 후에 재구성된 TSTM

tr1의 시험 결과 fail 판정을 낸다. 그리고 트랜지션 tr1의 시험 결과로 $TST_M$의 재구성을 위해 $TST_M$에서 트랜지션 tr1을 STU의 요소로 가지고 있는 트랜지션을 찾는다. 만약 트랜지션 tr1을 요소로 가지고 있는 $STU_i$가 있다면, 트랜지션$t_i$는 자동적으로 fail을 판정 받고 트랜지션 $t_i$는 더 이상 시험되지 않는다. 그러나 그림 5의 $TST_M$경우에서는 트랜지션 tr1를 포함하는 STU를 가지고 있는 트랜지션은 없으므로 tr1을 위한 더 이상의 연산은 필요없다. 다음 단계는 $TST_M$으로부터 트랜지션 tr1을 포함하는 모든 PTS들을 제거한다. 그림 5에서 살펴보면 트랜지션 tr4를 위한 $TSST_4$의 $PTS_4^1$와 트랜지션 tr5를 위한 $TSST_5$의 $PTS_5^1$과 $PTS_5^2$는 트랜지션 tr1을 포함하고 있다. 이것들은 시험 결과가 당연히 fail이므로 이 세 경로들은 시험해볼 필요 없이 제거된다. 그 결과로, 트랜지션 tr4가 단 하나의 PTS를 갖게 되기 때문에 $STU_4$가 생성되고 트랜지션 tr5의 $TSST_5$는 PTS가 없으므로 시험하지 않고 fail 판정을 내게 된다.

그림 6은 트랜지션 tr1의 시험 후에 재구성된 $TST_M$을 보여준다. 만약 그림6에서 보여지는 것처럼 트랜지션 tr2가 pass 판정을 받게 된다면, 초기 상태로부터 Level 2까지 같은 트랜지션 경로를 갖는 PTS들을 검색한다. 그 결과로 $TSST_3$의 $PTS_3^1$과 $TSST_4$의 $PTS_4^1$이 선택된다. 이들 중 초기 상태에서 가장 짧은 길이를 갖는 $TSST_3$의 $PTS_3^1$이 다음 시험을 위해 선택된다. $PTS_3^1$의 Level 2까지의 트랜지션이 $PTS_2^1$과 같기 때문에 트랜지션 tr4와 관련된 시험 시퀀스가 IUT의 트랜지션 tr3의 시험에 적용된다.

### 3.2 Postamble에 대한 DCTM

Postamble 부분에 속한 트랜지션과 시험대상 트랜지션과의 dependence는 트랜지션의 도착상태를 검증하기 위한 시퀀스가 시험대상 트랜지션의 시험 판정에 영향을 주는 현상을 말한다. 이 도착 상태 검증을 위한 시퀀스인 postamble은 시험대상이 블랙박스로 된 IUT이기 때문에 필요한 부분으로 T, UIO, W, DS 방법 [1]등이 사용된다. 이들 방법은 각각 다른 특성과 장점을 가지고 있다. 우선T방법은 테스트하려는 기계의 모든 트랜지션들이 시험되도록 랜덤하게 입력을 적용하여 시험 시퀀스를 적용하는 것으로 가장 간단하지만 입력을 적용하는데 있어서 중복 현상이 있을 수 있고 루프 현상에 빠질 수 있으며 에러를 진단하는데 성능이 떨어진다. UIO 방법은 각 상태들이 서로 다른 출력을 낼 수 있도록 각각에 입력을 적용하여 시험 시퀀스를 만든다. 이때 각 상태에 적용하는 입력이 같을 필요는 없다. DS 방법은 상태들이 서로 다른 출력을 내는 입력을 결정하여 모든 상태에 이미 결정한 입력을 적용하여 시험 시퀀스를 만든다. W방법은 각 상태가 서로 다른 출력을 내도록 하는 다수 개의 입력을 찾아 그것들을 모두 적용하여 시험 시퀀스를 만드는 것으로 이 경우 다른 방법들보다 에러 진단 능력이 뛰어날 수 있으나 시험 시퀀스가 커지는 단점이 있다. 이 논문에서는 postamble을 만드는 여러 방법 중 UIO를 선택하여 적용해보기로 한다.

그림 4에서 트랜지션 tr2가 fault 트랜지션이라고 가정하자. 이때 그림 4에서 postamble로 트랜지션 tr2를 포함하는 상태는 표 3에 의해 상태 3이라는 것을 알 수 있다. 그림 4에서 보면 트랜지션 tr4의 도착상태는 상태 3이므로 트랜지션 tr4를 위한 시험 시퀀스에 postamble로 트랜지션 tr2가 포함된다. 이것이 그림 5에 나와있는데, $TSST_4$는 트랜지션 tr1으로 시작되는 시험 경로와

트랜지션 tr2로 시작되는 시험 경로, 즉 두 개의 시험 경로를 가지고 있다. 그러나 트랜지션 tr2가 fault 트랜지션이므로 트랜지션 tr1으로 시작되는 시험 경로를 선택한다고 해도 이것도 역시 postamble로 트랜지션 tr2를 가지기 때문에 결과는 fail이다. 즉 preamble부분의 트랜지션과 시험대상 트랜지션 사이의 dependence를 해결한 방법도 postamble 부분을 고정적인 시퀀스로 사용한다면 시험대상 트랜지션의 정확한 시험결과를 얻을 수 없다는 결론이 나온다. 그러므로 postamble 부분에 포함되는 트랜지션과의 dependence를 해결하는 것이 필요하다. 이를 위하여 그림4에 대해 표 4와 같이 각 상태에 대해 MPS를 구할 수 있는데 이 MPS는 Multiple UIO이다. 표에서 상태 0와 상태 3은 두개의 postamble을 가지고 있으므로 postamble부분의 트랜지션과 시험대상 트랜지션 사이에 존재하는 dependence 문제를 피할수 있게 된다.

표 4 그림4의 가계 M의 MPS

| State | MPS |
|---|---|
| 0 | [b/c], [a/b] |
| 1 | [c/d] |
| 2 | [z/b] |
| 3 | [a/b, b/c], [a/b, a/b] |

그림 7은 표 4의 MUIO를 이용하여 TST를 재구성한 것이다. 이제는 트랜지션 tr2가 fault 트랜지션이라도 트랜지션 tr4를 위한 시험 시퀀스로 트랜지션 tr2를 포함하지 않는 시험 경로 $PTS_4^2$를 선택할 수 있게 된다. 이때 주의할 것은 MPS를 구함에 있어서 각 상태를 다른 상태에 대해 구별할 수 있는 최대한 길이가 작은 것을 선택해야 한다는 것이다. 이것은 시험 비용을 최소화 시키는데 중요한 문제이다.

### 3.3 알고리즘

DCTM은 그림 8과 같이 도식화할 수 있다. 이것은 기존의 적합성 시험 방법과는 다르게 IUT의 요소 (eg. 시험대상 트랜지션)가 fault인지 아닌지를 판단하는 Local 판정이 fail이면 시험 시퀀스 선택 모듈로 돌아가 TST에서 시험 시퀀스를 동적으로 다시 선택할 수 있는 feedback과정이 있다.

그림 8은 아래와 같이 알고리즘으로 표현될 수 있는데, 크게 TST 구성과 이를 이용한 시험 경로 동적 선택으로 구성되어 있다.

ST={$t_1$, . . . , $t_i$, . . . , $t_n$}안의 각 트랜지션들에서, $TSST_i$가 도착 상태 검증을 위한 방법으로 DS, UIO, W를 선택, 사용하여 구성된다. $t_i$의 시험을 위해 모든 $PTS_i$가 생성되고 이 $PTS_i$들은 초기상태에서 $t_i$까지 길이가 증가하는 순서로 재배열하여 $TSST_i$ ={$PTS_i^1$, . . . , $PTS_i^q$, . . . , $PTS_i^j$}를 얻는다. FSM의 각 트랜지션

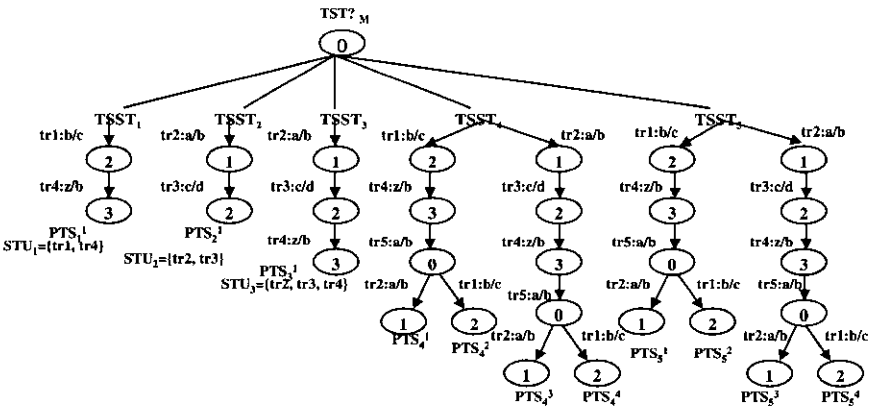



그림 7 Multiple UIO를 사용한 TST″M

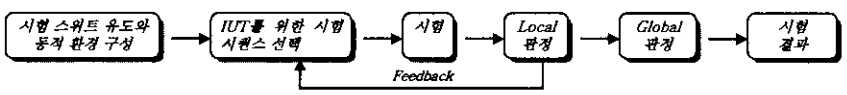



그림 8 동적 시험 경로 선택을 사용한 DCTM 절차

*알고리즘*

```
1: construct TSST;
2: setup TST;
3: compute STU;
4: construct MPSi;
5: construct TST;
6: for t1 to tn in ti ST do begin
7:     if pass or fail verdict is already assigned for ti then continue;
8:     else begin
9:         for q:=1 to q:=j do begin
10:            if appropriate PTSiq exist then execute transitions in PTSiq ;
11:            if unexpected output is observed then begin
12:                if PTSiq is the last path of ti then begin
13:                    assign fail verdict to ti test;
14:                    if any TSSTk of TST has STU and ti is an element of STUk
15:                    then assign fail to tk test of TST;
16:                    break;
17:                end if
18:            else begin
19:                assign pass verdict to ti test;
20:                if PTSk exists then begin
21:                    jump into level p of PTSk;
22:                    break;
23:                end if
24:            end if-else
25:        end for
26:    end if-else
27: end for
```

의 TSST를 사용하여 Test Sequence Tree (TST)가 만들어진다. 초기상태에서 가까운 순서로 TSST의 집합을 정렬하여 TST={$TSST_1$, ..., $TSST_i$, ..., $TSST_n$}를 얻는다. 만약 초기 상태에서 $t_i$로 경로가 하나만 존재한다면 $TSST_i$의$STU_i$를 계산한다.

1행에서 5행까지는 TST 초기 구성에 관한 내용이고, 6행부터 ST안의 각 트랜지션의 Local 판정을 얻는다. 9행에서 *j*는 q번째 PTS에서 초기 상태에서 $t_i$로의 모든 가능한 경로의 개수이다. $PTS_i^q$안에서 실행하는 트랜지션들의 결과로써 만약 기대하지 않은 출력이 탐지되고, $PTS_i^q$가 $t_i$의 마지막 경로라면 $t_i$의 판정은 fail이고 그것의 $STU_k$안의 $t_i$를 갖는 $TSST_k$는 시험없이 fail을 할당한다. 만약 출력이 올바르다면, pass 판정이 $t_i$에 할당된다. 10행에서 바람직한 TST 중 $PTS_i^q$를 찾아야 하는데 이때 Multiple DS, Multiple W방법을 사용할 때는 문제가 없지만 Multiple UIO를 사용하는 경우에 각 상태들이 현재 사용하는 UIO가 상태들 사이에서 성립이 되는지 살펴보아야 한다. 20행에서 초기 상태에서 현재 실행된 $t_i$의 마지막 위치인 Level p까지 일치되는 시퀀스 중 최단 시퀀스를 가진 $PTS_k$를 찾아야 한다. 만약 $PTS_k$가 존재한다면 $PTS_k$의Level p까지 점프한다.

## 4. 시험 도구 개발 및 적용

3장에서 제안한 DCTM을 증명하기 위하여 제안된 방법을 시험 도구로 개발하여 TCP에 적용하여 본다.

### 4.1 시험 도구 개발

DCTM이 기존의 방법보다 fault coverage 면에서 얼마나 좋은 결과를 보이는지 알아보기 위하여 시험 도구를 구현하였다. 이 시험 도구는 Windows NT 환경의 C++언어로 작성되어 Visual C++ 6.0로 컴파일 하였다. 이 시험 도구는 기존 적합성 시험 방법과 DCTM을 포함하고 있다. 이 시험 도구는 프로토콜을 입력으로 받아 기존 적합성 시험의 방법과 DCTM을 동시에 수행하고 각각의 경우의 결과를 출력한다. 이 도구에서 사용한 postamble은 Multiple UIO이다.

그림 9처럼 이 시험 도구는 FSM 형태의 프로토콜을 입력으로 받는다. 이 도구의 첫번째 작업은 입력 처리 모듈로서 입력으로 받은 프로토콜을 프로그램의 다른 모듈에서 사용하기 쉬운 형태로 변환하는 작업이다. 특히 트랜지션 중심의 형태와 상태 중심의 형태의 정보로 재가공하는데, 이때 정보들은 링크드리스트로된 트리 구조로 저장된다. 이 작업으로 인해 프로토콜은 트랜지션 중심 형태와 상태 중심 형태로 정보가 데이타베이스화 되어 저장, 유지된다. 데이타베이스 중 상태 중심 형태의 정보는 UIO 시퀀스 생성에, 트랜지션 중심 형태의 정보는 시험 시퀀스 생성과 처리 모듈에서 사용된다. Sub Test Sequence Generation I의 모듈에서는 트랜지션 중심 형태의 데이타베이스를 이용하여 preamble과 시험대상 트랜지션으로 이루어지는 서브 시험 시퀀스를 생성하게 된다. 이때 이 모듈에서는 기존 적합성 시험 방법을 위해 최단 경로로 구성되는 preamble과 DCTM을 위해 alternative 경로까지 고려한 다수의 preamble을 각각 구하게 된다. 그리고 Sub Test Sequence Generation II 모듈에서는 상태 중심 형태의 데이타베이

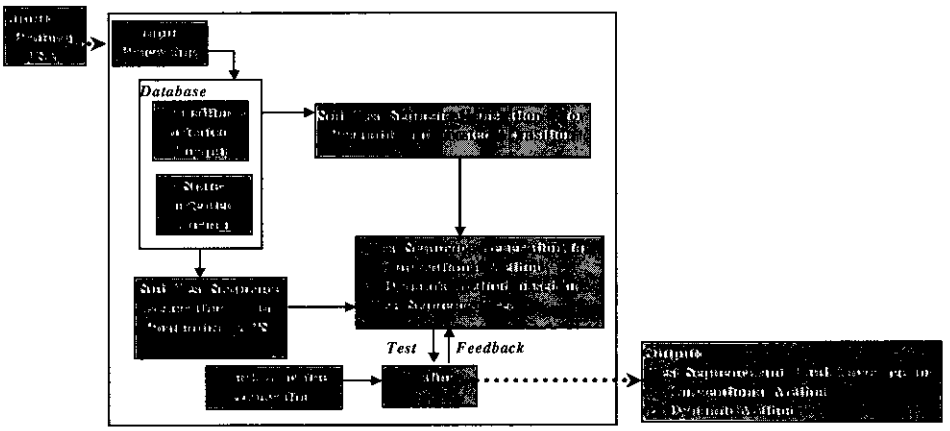



그림 9 자동화 도구의 구조

스를 이용하여 postamble을 구하게 된다. 이때도 기존 적합성 시험 방법을 위해 각 상태마다 하나의 postamble을 구하고, DCTM을 위해 MPS를 구하게 된다. 이렇게 만들어진 서브 시험 시퀀스들을 취합하여 Test Sequence Generation 모듈에서 완전한 시험 시퀀스가 만들어지게 된다. 이 모듈에서는 기존 적합성 시험 방법을 위한 시험 시퀀스 트리와 DCTM을 위한 시험 시퀀스 트리 (TST)를 만든다. 마지막으로 Testing 모듈을 거치게 되는데 이 모듈은 시험 도구의 내부에서 만들어진 임의의 fault machine을 입력으로 받아 기존의 적합성 시험 방법과 DCTM에서 생성된 시험 시퀀스를 동시에 시험한다. 마지막으로 두 방법에 의해 생성된 시험 시퀀스에 대한 fault coverage를 얻게 된다.

### 4.2 적용 결과

이 장에서는 4.1절에서 구현한 시험 도구를 이용하여 TCP 프로토콜을 입력으로 기존의 프로토콜 적합성 시험 방법과 DCTM을 시험하여 그 결과를 비교해보기로 한다. 이 시험 과정은 작업의 편리를 위한 전처리 작업과 내부적으로 사용되는 시험 시퀀스 생성, 그리고 각각의 fault machine의 경우에 있어 얼마나 올바른 진단을 내리는지 나타내는 fault coverage를 결과로 산출한다.

4.2.1 전처리 작업

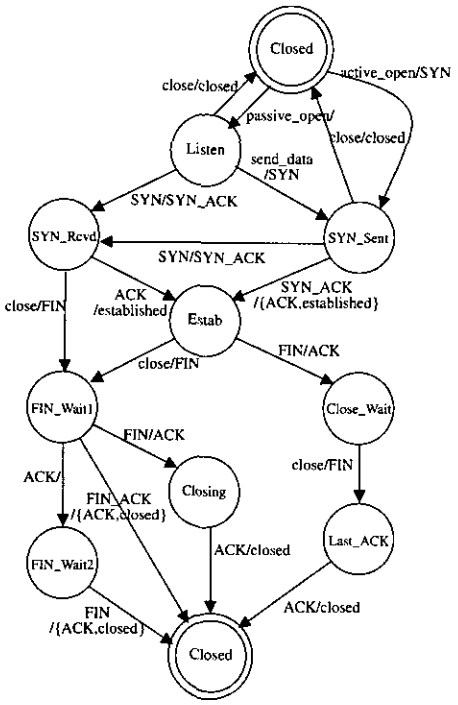
이 시험 도구의 입력으로 FSM 형태로 된 프로토콜을 받는데 이 FSM은 출발 상태, 입력, 출력, 도착 상태로 구성된 트랜지션의 집합으로 보며 실질적인 입력은 트랜지션들이다. 시험의 편리를 위하여 트랜지션의 스테이트와 입력, 출력의 문자열값을 임의의 자연수로 대체하여 시험하기로 한다. 이 논문에서 사용한 TCP의 FSM은 논문 [6]에서 사용한 FSM을 사용하였는데, 이 FSM은 fregmentation, duplication 등과 같은 데이타 전송 부분에 관한 데이타 전송 부분과 timer 관련 부분을 제외한 연결 설정, 해제등에 관한 전송 부분만을 다루고 있다. 여기에서 사용한 TCP FSM은 minimal, strong connected Mealy machine이다. 특히 UIO method를 사용하므로 completely specified 되어야 하기 때문에 TCP FSM을 보면 트랜지션 중 입력에 대해 출력을 갖지 않은 형태의 것은 pair에서 빠진 입력과 출력을 null로 추가하여 표현한다. 표 5와 표 6에 실제 프로토콜 문자열로 표현된 FSM을 자연수로 된 FSM으로 변환시킨 결과이다.

표 5 TCP FSM의 상태 매핑

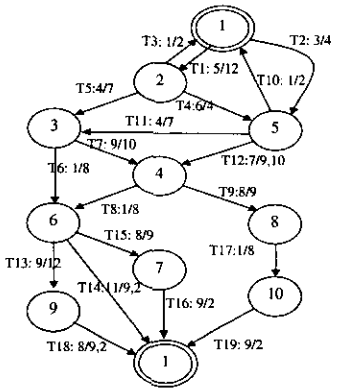
| State | Number | State | Number |
|---|---|---|---|
| Closed | 1 | FIN_Wait1 | 6 |
| Listen | 2 | Closing | 7 |
| SYN_Revd | 3 | Close_Wai | 8t |
| Estab | 4 | FIN_Wait2 | 9 |
| SYN_Sent | 5 | Last_ACK | 10 |

표 6 TCP FSM의 event 매핑

| State | Number | State | Number |
|---|---|---|---|
| close | 1 | SYN_ACK | 7 |
| closed | 2 | FIN | 8 |
| active_open | 3 | ACK | 9 |
| SYN | 4 | established | 10 |
| passive_open | 5 | FIN_ACK | 11 |
| send_data | 6 | null | 12 |




(a) TCP FSM




(b) 수정된 TCP FSM

그림 10 TCP FSM

표 5와 표 6의 변환 관계에 의해 그림 10 (a)의 TCP FSM이 그림 10 (b)의 FSM으로 수정됨을 볼 수 있다. 그림10과 같이 TCP FSM에는 총10개의 상태와 19개의 트랜지션이 존재한다. 여기에서 TCP FSM에 존재하는 19개의 트랜지션도 임의의 자연수로 이름을 대체하였다. FSM의 fault model은 다음 3가지의 클래스 [1]로 나누어 생각해 볼 수 있다.

(1) 주어진 입력에 대해 예상된 상태로 천이하지만 예상치 못한 출력을 내는 경우
(2) 주어진 입력에 대해 예상된 출력을 내지만 예상치 못한 상태로 천이하는 경우
(3) 주어진 입력에 대해 예상치 못한 출력을 내고 예상치 못한 상태로 천이하는 경우

이 논문에서는 시험의 단순화를 위하여 (1)의 경우에 해당하는 fault model만을 사용하여 fault machine을 생성하여 시험하도록 한다.

표 7에 TCP FSM의 각 상태에 대한 UIO 시퀀스를 볼 수 있으며 이는 표 8의 내용인 Multiple UIO의 서브 세트로서 기존 적합성 시험에서 사용할 UIO이다. 표 7와 표 8의 내용은 시험 도구의 Sub Test Sequence Generation II에서 만들어진다. 이때 표 8의 내용에서 더 많은 Multiple UIO 들이 존재할 수 있지만, 시험의 편의성과 Multiple UIO의 최소성을 고려하여 최대 길이가 2인 UIO들로 제한시켰다.

표 7 기존의 적합성 시험에서 사용할 각 상태에 대한 UIO 시퀀스

| State | UIO sequences | State | UIO sequences |
|---|---|---|---|
| 1 | [T1:5/12] | 6 | [T13:9/12] |
| 2 | [T3:1/2] | 7 | [T16:9/2] |
| 3 | [T6:1/8] | 8 | [T17:1/8, T19:9/2] |
| 4 | [T9:8/9] | 9 | [T18:8/9,2] |
| 5 | [T11:4/7] | 10 | [T19:9/2, T1:5/12] |

표 8 DCTM에서 사용할 Multiple UIO

| State | Multiple UIO sequences |
|---|---|
| 1 | { [T1:5/12], [T2:3/4] } |
| 2 | { [T4:6/4], [T5:4/7], [T4:6/4, T10:1/2], [T4:6/4, T11:4/7], [T4:6/4, T12:7/9,10] , [T5:4/7, T6:1/8], [T5:4/7, T7:9/10] } |
| 3 | { [T6:1/8], [T7:9/10], [T6:1/8, T13:9/12], [T6:1/8, T14:11/9,2], [T6:1/8, T15:8/9] } |
| 4 | { [T8:1/8], [T9:8/9], [T8:1/8, T13:9/12], [T8:1/8, T14:11/9,2], [T8:1/8, T15:8/9], [T9:8/9, T17:1/8] } |
| 5 | { [T10:1/2], [T11:4/7], [T12:7/9,10] , [T11:4/7, T6:1/8], [T11:4/7, T7:9/10] } |
| 6 | { [T13:9/2], [T14:11/9,2], [T15:8/9], [T15:8/9, T16:9/2] } |
| 7 | { [T16:9/2], [T16:9/2, T1:5/12], [T16:9/2, T2:3/4] } |
| 8 | { [T17:1/8], [T17:1/8, T19:9/2] } |
| 9 | { [T18:8/9,2] } |
| 10 | { [T19:9/2], [T19:9/2, T1:5/12], [T19:9/2, T2:3/4] } |

4.2.2 시험 결과

시험 결과는 기존 적합성 시험의 시험 시퀀스, DCTM의 TST, 그리고 fault machine을 시험한 fault coverage로 나누어 볼 수 있다. 기존 적합성 시험을 위한 시험 시퀀스는 표 9에, DCTM을 위한 서브 시험 시퀀스의 집합인 TST는 부록에 나타나 있다. 부록에 수록된 DCTM에 의한 서브 시험 시퀀스들은 표 6의 Multiple UIO중에 기존 적합성 시험에서 사용하고 있는 UIO를 갖는 부분만을 선택하여 나타내었다. 실제의 DCTM을 위한 시험 시퀀스들은 표 6의 Multiple UIO를 고려할 때 부록에 나타난 것보다 훨씬 많은 서브 시험 시퀀스가 존재한다는 것을 알 수 있다.

표 9 기존 적합성 시험의 서브 시험 시퀀스

| Position | Sub Test Sequences | Position | Sub Test Sequences |
|---|---|---|---|
| 1 | [T1,T3] | 11 | [T2,T11,T6] |
| 2 | [T2,T11] | 12 | [T2,T12,T9] |
| 3 | [T1,T3,T1] | 13 | [T1,T5,T6,T13,T18] |
| 4 | [T1,T4,T11] | 14 | [T1,T5,T6,T14,T1] |
| 5 | [T1,T5,T6] | 15 | [T1,T5,T6,T15,T16] |
| 6 | [T1,T5,T6,T13] | 16 | [T1,T5,T6,T15,T16,T1] |
| 7 | [T1,T5,T7,T9] | 17 | [T2,T12,T9,T17,T19,T1] |
| 8 | [T2,T12,T8,T13] | 18 | [T1,T5,T6,T13,T18,T1] |
| 9 | [T2,T12,T9,T17,T19] | 19 | [T2,T12,T9,T17,T19,T1] |
| 10 | [T2,T10,T1] | | |

표 9와 표 10의 시험 시퀀스들을 사용하여 내부적으로 fault 기계를 만들어 기존의 적합성 시험의 절차와 DCTM의 절차를 적용하여 fault coverage를 구한 결과가 표 10에 나타나 있다. 여기서 fault 기계는 각 faulty transition 숫자에 대해 가능한 모든 faulty machine을 모두 생성한 것이다. 표 10에서 faulty transition number의 숫자가 1이면 총 19개의 트랜지션 중 하나의 트랜지션이 fault인 경우이므로 ${}_{19}C_1$이 되어 19, 만약 2이면 ${}_{19}C_2$가 되어 171개의 서로 다른 fault 기계가 생성된다. 또한 이 fault 기계들의 상태 수는 TCP FSM에 의한 기계의 상태수보다 작거나 같다.

표 10에서는 기존 적합성 시험 방법과 DCTM에 대한 fault coverage가 나타나 있다. 이때 DCTM을 시험하기 위해서 preamble 부분에 포함된 트랜지션과의 dependence를 고려한 DCTM과 preamble 부분과 postamble 부분에 포함된 트랜지션과의 dependence를 고려한 DCTM으로 나누어 결과를 도출하였다. 위의 표에서 각 method의 pass 개수는 잘 구현된 트랜지션의 개수를 말하는 것으로, local verdict을 나타낸다. 이 local verdict을 바탕으로 전체 시스템의 상태를 진단하는 global verdict을 내게 된다. 즉 local verdict의 결과의 향상이 global verdict의 향상을 가져오는 것이다. 이 local verdict의 향상된 결과는 fault coverage의 향상을 가져오게 된다. 즉 정확한 진단 결과를 내게 되는 것이다. 2개의 fault 트랜지션을 가지고 있는 fault machine가 있다고 할 때 이런 machine은 총 171가지

표 10 Fault Model을 사용하여 실험한 결과 비교

| Faulty transition number | Total # of different FSM implementions | The # of all possible test result | Conventional Test Method | | Dynamic Test Method | | | | Ideal Tester |
|---|---|---|---|---|---|---|---|---|---|
| | | | Total # of pass results | The # of average pass transitions | Dynamic Test for preamble | | Dynamic Test for preamble and postamble | | The # of pass transition |
| | | | | | Total # of pass results | The # of average pass transitions | Total # of pass results | The # of average pass transitions | |
| **1** | 19 | 171 | 287 | 15.105 | 314 | 16.526 | 323 | 17 | 18 |
| **2** | 171 | 3,249 | 2,039 | 11.924 | 2,418 | 14.140 | 2,578 | 15.076 | 17 |
| **3** | 969 | 18,411 | 9,052 | 9.342 | 11,484 | 11.851 | 12,667 | 13.072 | 16 |
| **4** | 3,876 | 73,644 | 28,135 | 7.259 | 37,582 | 9.696 | 42,959 | 11.083 | 15 |
| **5** | 11,628 | 90,972 | 65,000 | 5.590 | 89,800 | 7.723 | 106,613 | 9.169 | 14 |
| **6** | 27,132 | 515,508 | 115,635 | 4.262 | 162,158 | 5.977 | 200,520 | 7.391 | 13 |
| **7** | 50,388 | 957,372 | 161,876 | 3.213 | 226,187 | 4.489 | 292,336 | 5.802 | 12 |
| **8** | 75,582 | 1,436,058 | 180,609 | 2.390 | 247,204 | 3.271 | 335,306 | 4.436 | 11 |
| **9** | 92,378 | 1,755,182 | 161,590 | 1.749 | 213,609 | 2.312 | 305,458 | 3.307 | 10 |
| **10** | 92,378 | 1,755,182 | 115,973 | 1.255 | 146,622 | 1.587 | 222,102 | 2.404 | 9 |
| **11** | 75,582 | 1,436,058 | 66,404 | 0.879 | 79,946 | 1.058 | 128,902 | 1.705 | 8 |
| **12** | 50,388 | 957,372 | 29,965 | 0.595 | 34,402 | 0.683 | 59,327 | 1.177 | 7 |
| **13** | 27,132 | 515,508 | 10,432 | 0.384 | 11,490 | 0.423 | 21,303 | 0.785 | 6 |
| **14** | 11,628 | 90,972 | 2,705 | 0.233 | 2,882 | 0.248 | 5,780 | 0.497 | 5 |
| **15** | 3,876 | 73,641 | 492 | 0.127 | 511 | 0.132 | 1,117 | 0.288 | 4 |
| **16** | 969 | 8,411 | 56 | 0.058 | 57 | 0.059 | 137 | 0.141 | 3 |
| **17** | 171 | 3,249 | 3 | 0.018 | 3 | 0.018 | 8 | 0.047 | 2 |
| **18** | 19 | 171 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

의 서로 다른 구현물이 존재할 수 있고, 기존의 적합성 시험 방법에서는 17개의 올바로 구현된 트랜지션이 있음에도 불구하고, 평균 11.924개의 올바른 트랜지션을 진단해 낼 수 있고, preamble 부분에 포함된 트랜지션과의 dependence를 고려한 DCTM은 평균 14.140개의 올바른 트랜지션을 진단해 낼 수 있으며 preamble과 postamble 부분에 포함된 트랜지션들과의 dependence를 고려한 DCTM은 평균 15.076개의 올바른 트랜지션을 진단해 낼 수 있다. 즉 DCTM이 기존 방법보다 ideal tester의 결과에 가까운 결과를 나타내고 이 DCTM 중에서도 단순히 preamble 부분에 포함된 트랜지션과의 dependence만을 고려한 방법보다 preamble과 postamble 부분에 포함된 트랜지션들 모두 고려하는 방법이 ideal tester의 결과에 근사한 결과를 도출함을 알 수 있다. 이러한 결과는 이 논문에서 적용한 TCP 프로토콜 뿐만 아니라, ITU-T Q.2931 signalling protocol에 대해 수작업으로 구한 경우 [7]에서도 유사한 결과가 나타난다. 이런 현상은 fault 트랜지션의 개수가 12인 경우까지 나타나다가 그 이후로는 기존의 적합성 시험과 DCTM의 결과의 차이가 크게 나타나지 않는다. 하나의 트랜지션만을 제외하고 모두 fault 트랜지션일 경우, 즉 fault 트랜지션의 개수가 18인 경우는 두 방법 모두 올바른 트랜지션을 하나도 찾아내지 못한다. 대개 프로토콜을 구현하는 벤더들이 적은 실수를 하는 경우를 생각 할 때 DCTM은 기존의 적합성 시험보다 정확한 진단을 할 수 있다는 결론이 나온다. 그림 11은 표 10의 내용을 그래프로 나타낸 것이다. ideal tester의 결과에 근접한 순서로는 preamble과 postamble에 대한 DCTM, preamble에 대한 DCTM, 그리고 기존 시험 방법으로 나타났다. 여기에서 preamble과 postamble에 대한 DCTM이 ideal tester에 가장 근접한 결과를 나타냄을 알 수 있다.

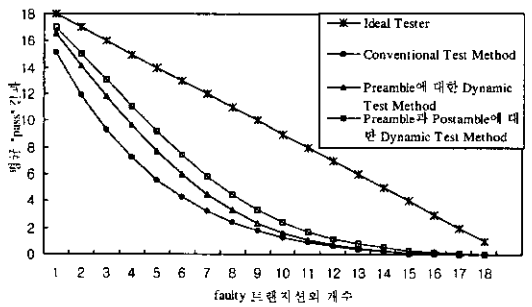



그림 11 Ideal Tester와의 비교

이 논문에서 제안한 DCTM은 fault coverage를 향상시키기 위하여 서브 시험 시퀀스 트리의 관리 작업을 요구하는데 이 작업으로 인해 overhead가 생기지만 이것은 단순히 시험비용의 증가를 의미하는 것은 아니다. 만약 시험 중 fail 판정을 받으면 alternative 경로를 탐색해서 시험 시퀀스를 재설정 하므로 기존 시험 방법에서보다 더 긴 시험 시퀀스를 사용하게 되어 시험비용이 더 커지게 되는 경우가 있으나 fault 트랜지션을 포함하고 있는 PTS들은 시험하지 않고 fail 판정을 내기 때문에 시험비용을 줄일 수 있는 경우가 있을 수 있기 때문이다.

## 5. 결 론

이 논문에서는 프로토콜 적합성 시험의 fault coverage향상을 위한 DCTM을 제안하고 적용하였다. 시험 시퀀스의 preamble부분과 postamble 부분에 포함된 잘못 구현된 트랜지션들이 시험대상 트랜지션의 진단 결과에 영향을 주는 현상을 해결하기 위해 DCTM을 제안하게 되었는데, 이 방법은 시험 시퀀스 생성시 각 트랜지션들의 dependence 문제를 피하기 위한 해결책을 제시한다. 이를 증명하기 위해 DCTM 알고리즘을 구현하여 기존의 적합성 시험 방법과 DCTM을 TCP 프로토콜에 적용하여 그 결과를 비교해 보았다. 그 결과 DCTM이 기존의 적합성 시험 방법보다 fault coverage 면에서 우수함을 알수 있었다. DCTM은 preamble 부분의 alternative 경로를 포함하여 만든 TST를 이용하여 dependence 문제를 해결하고 있어 기존 적합성 시험 방법보다 ideal tester에 가까운 진단 결과를 도출하였다. 즉 시험하려는 트랜지션에 대한 더욱 정확한 진단을 내리고 있다. 특히 프로토콜 구현물에 많지 않는 fault가 있을 때 DCTM 절차가 기존의 적합성 시험 절차에서보다 정확한 진단율을 보이고 있으므로 DCTM이 실제 프로토콜을 구현하는 벤더들에게 많은 도움을 줄 것으로 보인다.

향후 연구로는 non-deterministic FSM에도 이 방법론을 적용시켜 보는 것과 제안한 DCTM에서의 fault coverage와 시험비용사이의 관계를 연구하여 이를 실제 프로토콜에 적용하는 것이다.

## 참 고 문 헌

[1] D. P. Sidhu and T. K. Leung, "Formal methods for protocol testing: A detailed study," IEEE Trans. Sofrware Engineering, Vol. 15, No. 4, pp. 413-426, Apr. 1989.

[2] A. Podgurski and L. A. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," IEEE Trans. Software Engineering. Vol. 16, No. 9, pp. 965-979, Sep. 1990.

[3] S. Horwitz and T. Peps, "The use of program dependence graphs in software engineering," ACM Proceedings of the 14th international conference on Software engineering, pp.392-411, 1992.

[4] S. T. Chanson and Q. Li, "On static and dynamic test case selections in protocol conformance testing," The 5th Int'l Workshop on Protocol Test Systems, pp.225-267, 1992.

[5] ISO 9646, "Information technology OSI conformance testing methodology and framework," 1992.

[6] S. Seol, M. Kim, S. Kang, Y. Park and Y. Choe, "Interoperability test suite derivation for the TCP," IFIP Joint International Conference on Formal Description Techniques (FORTE XII) and Protocol Specification Testing and Verification(PSTV XIX), pp.357-376, Oct. 1999.

[7] S. Yoo, M. Kim and D. Kang, "An approach to dynamic protocol testing," IFIP TC6 10th International Workshop on Testing of Communicating Systems, Vol. 10, No. 12, pp.183-199, 1997.

[8] B. Yang and H. Ural, "Protocol conformance test generation using multiple UIO sequences with overlapping," Pro. of the ACM Symposium on Communications architectures & protocol, pp.118-125, 1990.

[9] R. S. Pressman, "Software engineering A practitioner's approach" Third Edition, Mcgraw-hill international editions, pp. 595 630, 1992.

## 부록 DCTM을 사용한 TCP의 Test Sequence Tree(TST)

| TSST | PTS | Sub Test Sequences |
|---|---|---|
| 1 | $PTS_1^1$ | [T1,T3] |
| 2 | $PTS_2^1$ | [T2,T11] |
| 3 | $PTS_3^1$ | [T1,T3,T1] |
| 4 | $PTS_4^1$ | [T1,T4,T11] |
| 5 | $PTS_5^1$ | [T1,T5,T6] |
| 6 | $PTS_6^1$ | [T1,T5,T6,T13] |
| | $PTS_6^2$ | [T2,T11,T6, T13] |
| | $PTS_6^3$ | [T1,T4,T11,T6, T13] |
| 7 | $PTS_7^1$ | [T1,T5,T7,T9] |
| | $PTS_7^2$ | [T2,T11,T7,T9] |
| | $PTS_7^3$ | [T1,T4,T11,T7,T9] |
| 8 | $PTS_8^1$ | [T2,T12,T8,T13] |
| | $PTS_8^2$ | [T1,T4,T12,T8,T13] |
| | $PTS_8^3$ | [T1,T5,T7,T8,T13] |
| | $PTS_8^4$ | [T2,T11,T7,T8,T13] |
| | $PTS_8^5$ | [T1,T4,T11,T7,T8,T13] |
| 9 | $PTS_9^1$ | [T2,T12,T9,T17,T19] |
| | $PTS_9^2$ | [T1,T4,T12,T9, T17,T19] |
| | $PTS_9^3$ | [T1,T5,T7,T9,T17,T19] |
| | $PTS_9^4$ | [T2,T11,T7,T9,T17,T19] |
| | $PTS_9^5$ | [T1,T4,T11,T7,T9,T17,T19] |
| 10 | $PTS_{10}^1$ | [T2,T10,T1] |
| | $PTS_{10}^2$ | [T1,T4,T10,T1] |
| 11 | $PTS_{11}^1$ | [T2,T11,T6] |
| | $PTS_{11}^2$ | [T1,T4,T11,T6] |
| 12 | $PTS_{12}^1$ | [T2,T12,T9] |
| | $PTS_{12}^2$ | [T1,T4,T12,T9] |
| 13 | $PTS_{13}^1$ | [T1,T5,T6,T13,T18] |
| | $PTS_{13}^2$ | [T2,T11,T6,T13,T18] |
| | $PTS_{13}^3$ | [T2,T12,T 8,T13,T18] |
| | $PTS_{13}^4$ | [T1,T4,T11,T6,T13,T18] |
| | $PTS_{13}^5$ | [T1,T4,T12,T8,T13,T18] |
| | $PTS_{13}^6$ | [T1,T5,T7,T8,T13,T18] |
| | $PTS_{13}^7$ | [T2,T11,T7,T8,T13,T18] |
| | $PTS_{13}^8$ | [T1,T4,T11,T7,T8,T13,T18] |
| 14 | $PTS_{14}^1$ | [T1,T5,T6,T14,T1] |
| | $PTS_{14}^2$ | [T2,T11,T6,T14,T1] |
| | $PTS_{14}^3$ | [T2,T12,T8,T14,T1] |
| | $PTS_{14}^4$ | [T1,T4,T11,T6,T14,T1] |
| | $PTS_{14}^5$ | [T1,T4,T12,T8,T14,T1] |
| | $PTS_{14}^6$ | [T1,T5,T7,T8,T14,T1] |
| | $PTS_{14}^7$ | [T2,T11,T7,T8,T14,T1] |
| | $PTS_{14}^8$ | [T1,T4,T11,T7,T8,T14,T1] |
| 15 | $PTS_{15}^1$ | [T1,T5,T6,T15,T16] |
| | $PTS_{15}^2$ | [T2,T11,T6,T15,T16] |
| | $PTS_{15}^3$ | [T2,T12,T8,T15,T16] |
| | $PTS_{15}^4$ | [T1,T4,T11,T6,T15,T16] |
| | $PTS_{15}^5$ | [T1,T4,T12,T8,T15,T16] |
| | $PTS_{15}^6$ | [T1,T5,T7,T8,T15,T16] |
| | $PTS_{15}^7$ | [T2,T11,T7,T8,T15,T16] |
| | $PTS_{15}^8$ | [T1,T4,T11,T7,T8,T15,T16] |
| 16 | $PTS_{16}^1$ | [T1,T5,T6,T15,T16,T1] |
| | $PTS_{16}^2$ | [T2,T11,T6,T15,T16,T1] |
| | $PTS_{16}^3$ | [T2,T12,T8,T15,T16,T1] |
| | $PTS_{16}^4$ | [T1,T4,T11,T6,T15,T16,T1] |
| | $PTS_{16}^5$ | [T1,T4,T12,T8,T15,T16,T1] |
| | $PTS_{16}^6$ | [T1,T5,T7,T8,T15,T16,T1] |
| | $PTS_{16}^7$ | [T2,T11,T7,T8,T15,T16,T1] |
| | $PTS_{16}^8$ | [T1,T4,T11,T7,T8,T15,T16,T1] |
| 17 | $PTS_{17}^1$ | [T2,T12,T9,T17,T19,T1] |
| | $PTS_{17}^2$ | [T1,T4,T12,T9,T17, T19,T1] |
| | $PTS_{17}^3$ | [T1,T5,T7,T9,T17, T19,T1] |
| | $PTS_{17}^4$ | [T2,T11,T7,T9,T17, T19,T1] |
| | $PTS_{17}^5$ | [T1,T4,T11,T7,T9,T17,T19,T1] |
| 18 | $PTS_{18}^1$ | [T1,T5,T6,T13,T18,T1] |
| | $PTS_{18}^2$ | [T2,T11,T6,T13,T18, T1] |
| | $PTS_{18}^3$ | [T2,T12,T8,T13,T18, T1] |
| | $PTS_{18}^4$ | [T1,T4,T11,T6,T13,T18, T1] |

| TSST | PTS | Sub Test Sequences |
|---|---|---|
| 18 | $PTS_{18}^{5}$ | [T1,T4,T12,T8,T13,T18,T1] |
| | $PTS_{18}^{6}$ | [T1,T5,T7,T8,T13,T18, T1] |
| | $PTS_{18}^{7}$ | [T2,T11,T7,T8,T13,T18,T1] |
| | $PTS_{18}^{8}$ | [T1,T4,T11,T7,T8,T13,T18,T1] |
| 19 | $PTS_{19}^{1}$ | [T2,T12,T9,T17,T19,T1] |
| | $PTS_{19}^{2}$ | [T1,T4,T12,T9,T17,T19,T1] |
| | $PTS_{19}^{3}$ | [T1,T5,T7,T9,T17,T19, T1] |
| | $PTS_{19}^{4}$ | [T2,T11,T7,T9,T17,T19,T1] |
| | $PTS_{19}^{5}$ | [T1,T4,T11,T7,T9,T17,T19,T1] |

박 진 희

1999년 2월 동국대학교 컴퓨터공학과 졸업(공학사). 2001년 2월 한국정보통신대학원대학교(ICU) 컴퓨터/정보시스템 공학부 졸업(공학석사). 2001년 1월 ~ 현재 전자부품연구원(KETI) 인터넷미디어연구센터 연구원. 관심분야는 VoIP, Cable modem, 네트워크 프로토콜, 차세대 인터넷

김 명 철

1982년 2월 아주대학교 전자공학과 졸업(학사). 1984년 2월 KAIST 전산학과 졸업(석사). 1992년 Univ. of Brithish Columbia(박사). 1984년 ~ 1997년 한국통신 연구개발본부 표준연구단 실장. Co-chair of 10th International Workshop on Testing of Communication Systems. 1997년 PTS-SIG chair of Asia Oceania Workshop. 1997년 ~ 현재 한국정보통신대학원 교수. 관심분야는 이동인터넷, 프로토콜공학, 차세대인터넷

최 지 영

1989년 2월 숙명여대 전자계산학과 졸업(학사). 2000년 2월 한국정보통신대학원대학교 통신공학과 졸업(공학석사). 2000년 1월 ~ 현재 아이디 주식회사 선임연구원. 관심분야는 소프트웨어 호환성 시험 및 광전송 장치

유 상 조

1988년 2월 한양대학교 전자통신공학과 졸업(공학사). 1990년 2월 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 2000년 8월 한국과학기술원 전기 및 전자공학과 졸업(공학박사). 1990년 3월 ~ 2001년 2월 한국통신 연구개발본부 전임연구원. 2001년 3월 ~ 현재 인하대학교 정보통신전문대학원 조교수. 관심분야는 멀티미디어 네트워킹, 통신망 트래픽 제어, 통신 프로토콜 설계