Logical Clock을 이용한 분산 시험
(Distributed Test Method using Logical Clock)

최 영 준 † 김 명 철 †† 설 순 욱 †††
(Young Joon Choi) (Myungchul Kim) (Soonuk Seol)

요 약 분산 시스템 (distributed system)을 시험할 때, 병렬 이벤트들 (concurrent events)을 제어할 수 없다면 정확한 시험 결과를 얻어내기 어렵다. 기존의 분산 시험 방법은 정형적 방법으로 test sequence 생성 알고리즘을 제시하지 못하거나 동기화를 위한 부가적 메시지의 양이 많은 단점이 있었다. 본 논문에서는 병렬 이벤트 제어를 위해 logical clock을 이용한 정형적 test sequence 생성 알고리즘을 제시한다. 이 알고리즘은 제어-관찰 문제를 해결하고 시험 결과를 재현할 수 있다. 또한 어떠한 통신 패러다임에서도 적용될 수 있는 일반적 해결 방법을 제공한다. 분산 시험 방법에서는 분산 객체가 증가할수록 시험기 사이의 채널이 비선형 (non-linearly)으로 증가하는 단점이 있다. 이 단점을 극복하고자 원격 시험 방법 (remote test method)의 시험 구조를 제안한다. 제안된 알고리즘의 검증을 위해 SDL 도구를 사용하고, Q.2971 망 부분 (network side) 일 대 다 호/연결 설정을 위한 메시지 교환에 적용한다.

Abstract It is difficult to test a distributed system because of the task of controlling concurrent events. Existing works do not propose the test sequence generation algorithm in a formal way and the amount of message is large due to synchronization. In this paper, we propose a formal test sequence generation algorithm using logical clock to control concurrent events. It can solve the control-observation problem and makes the test results reproducible. It also provides a generic solution such that the algorithm can be used for any possible communication paradigm. In distributed test, the number of channels among the testers increases non-linearly with the number of distributed objects. We propose a new remote test architecture for solving this problem. SDL Tool is used to verify the correctness of the proposed algorithm, and it is applied to the message exchange for the establishment of Q.2971 point-to-multipoint call/connection as a case study.

1. 서 론

분산 시스템(distributed system)을 구성하는 분산 객체(distributed object)는 Open Distributed Processing (ODP) 등을 이용하여 구현된다. ODP는 분산 시스템을 개발하기 위하여 일반적인 프레임워크를 제공하는 ISO/ITU의 표준이다. 본 논문에서는 ODP를 이용하여 구현한 분산 객체로 이루어진 분산 시스템을 black box 형태로 하여 시험하고자 한다[1, 11].

† 정 회 원 : 금융결제원 전자금융연구소 연구원
cyjnice@kftc.or.kr
†† 종신회원 : 한국정보통신대학원대학교 공학부 교수
mckim@icu.ac.kr
††† 학생회원 : 한국정보통신대학원대학교 공학부
suscol@icu ac kr
논문접수 : 2000년 9월 22일
심사완료 : 2001년 7월 11일

논문[1]은 분산 시스템을 black box로 하여 분산 시험 방법으로 시험하였다. 이 과정에서 발생할 수 있는 제어-관찰 문제를 해결하기 위해 'C'와 'O' 시그널을 제시하였다. 이 시그널들은 협동 채널(coordination channel)이라는 멀티캐스트 채널을 통해 시험기 사이에서 주고 받아지며, 시험기들을 동기화 시키는 역할을 한다. 이 방법은 각 객체에서 발생하는 이벤트를 전체 순서화 시키는 장점이 있으나 출력 이동 문제(output-shifting fault)[8]가 발생할 수 있다.

논문[2]는 분산 시스템을 grey box Implementation Under Test(IUT)로 하여 시험할 때, IUT 내부의 관찰을 통해 병렬(concurrent) 이벤트에 대한 문제를 동기(synchronization) 메시지로 해결하는 방안을 제시하였다. 논문[2]의 방법은 IUT를 시험하는 고유한(unique) Test Sequence(TS)를 생성시키는 장점이 있으나 IUT 내부를 관찰하는 방법의 선택이 어렵고, 시험하려는 TS

이외에 부가적인 메시지가 많아지는 단점이 생긴다.

논문[3]은 분산 인터페이스를 고려해야 하는 분산 시험 방법에서 동기화가 가능한 TS를 생성하여 IUT를 시험하는 기존 모델을 분석하고, 이 모델의 fault coverage를 조사하였다. 그러나 정형적인 방법으로 TS를 생성하는 알고리즘을 제시하지 않았다.

본 논문은 분산 시스템의 시험에서 logical clock을 이용하여 TS를 생성하는 정형화된 병렬 이벤트 제어 알고리즘을 제시한다. 이 알고리즘은 이벤트의 발생 시간을 수치적으로 기록한 logical clock을 TS의 각 이벤트에 기록하고 비교함으로써 병렬 이벤트를 제어하기 위한 부가적인 시그널을 생성한다. 또한 제안한 알고리즘은 제어-관찰 문제를 해결할 수 있으며[1, 3], 전체 순서화 된 TS를 얻을 수 있으므로 시험 결과가 재현된다.

분산 시험 방법에는 여러 개의 시험기가 사용되며 시험기들은 협동(coordination) 메시지를 주고 받으며 서로 동기화 한다[1]. 분산 시험 방법은 분산 시스템의 시험과 같이 다중 포트(multiple ports)가 필요한 IUT의 시험에 효과적이지만 분산 객체가 증가할수록 시험기 사이의 채널이 비선형(non-linearly)으로 증가하는 단점이 있다. 이 단점을 극복하고자 원격 시험 방법(remote test method)의 시험 구조를 적용한다. 제안된 알고리즘의 검증을 위해 Specification and Description Language (SDL) 도구를 사용하고, Q.2971 망 부분(network side)일 때 다 호/연결 설정을 위한 메시지 교환에 적용한다.

이 논문의 구조는 다음과 같다. 2장에서는 분산 시스템의 시험에 대한 기존의 연구 및 배경을 기술하고, 3장에서는 logical clock을 이용한 TS 생성 알고리즘을 제시한다. 4장에서는 원격 시험 방법의 시험 구조에 LTS 생성 알고리즘을 적용하고, 5장에서는 결론과 향후 연구 계획을 기술한다.

2. 배 경

분산 시스템을 시험할 때 각 분산 객체에서 발생하는 이벤트들은 항상 인과(causal) 관계를 갖지 못하고 병렬 관계에 놓일 수 있다. 병렬 관계에 놓인 이벤트들이 존재하는 경우 다음과 같은 **제어-관찰 문제(control-observation problem)**가 발생할 수 있다[1, 3].

- 병렬 입력이 black box 형태인 IUT에 가해졌을 때, IUT에서 읽어 들이는 순서에 따라 시험 결과가 달라진다.
- Black box 형태인 IUT의 시험에서, 입력과 출력 매핑이 잘못되었어도 시험 결과는 정상이다.

같은 입력에 대하여 동일한 시험 결과의 재현을 위해서는 병렬 관계에 놓인 이벤트들이 인과 관계를 갖도록 하는 메커니즘이 필요하다. 인과 관계는 다른 말로 "happened before" 관계라고도 하며 '→'로 표시한다. 이벤트 'a'가 이벤트 'b'의 happened before 이벤트가 되려면 다음의 규칙 중 하나 이상을 만족해야 한다[5, 6].

규칙 1

(1) 'a'와 'b'는 동일 분산 객체의 이벤트이고, 이벤트 'a'가 이벤트 'b'보다 먼저 발생한다.

(2) 'a'는 보내는 이벤트이고 'b'는 'a'를 받는 이벤트이다. 단, 동기 통신에서는 보내는 이벤트와 받는 이벤트가 동시에 발생한다.

(3) 이벤트 'a'가 이벤트 'c'보다 먼저 발생되고 이벤트 'c'는 이벤트 'b'보다 먼저 발생한다.

Logical clock은 선형 시간(linear time), 벡터 시간(vector time), 행렬 시간(matrix time)[5] 등으로 표현될 수 있는 time-stamping 메커니즘으로서 규칙 1을 구체화할 수 있다. 본 논문에서는 이벤트 발생 시각을 쌍으로 하여 기록하는 벡터 시간을 사용한다[5]. 벡터 시간을 사용한 경우에 logical clock에 기록된 두 이벤트 발생이 인과 관계가 되기 위해서는 다음의 규칙을 만족시켜야 한다.

규칙 2

분산 객체 a, b, c로 구현된 IUT에서 발생하는 이벤트 'X'와 '이벤트 'Y'의 logical clock 사이에 다음 관계가 성립하면 두 이벤트는 인과 관계에 놓여있다고 한다 (단, ($X_a=Y_a$, $X_b=Y_b$, $X_c=Y_c$)는 제외). 이벤트 'X'와 'Y'의 logical clock 값은 각각 (X_a, X_b, X_c), (Y_a, Y_b, Y_c)이다.

$$(X_a \geq Y_a \ , \ X_b \geq Y_b \ , \ X_c \geq Y_c)$$

Stable state란 새로운 입력 없이는 상태 천이가 일어나지 않는 global state를 말한다[7]. Black box인 IUT를 시험할 경우 transient state는 관찰하기 어려우며 오직 stable state만을 관찰할 수 있다.

그림 1 stable state와 transient state

그림 1은 stable state 0에서 'a'를 입력받은 후 transient state인 1, 2를 지나서 stable state 3에 도달하는 Finite State Machine(FSM)을 나타낸다. 시험자는 입력 'a'에 대한 천이 전 stable state 0와 'x', 'z' 출력 후 stable state 3의 확인만으로 시험 수행 과정의

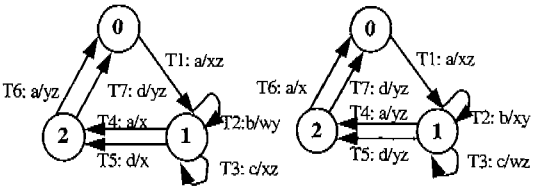
단계를 유추할 수 있다. 이렇게 함으로써 IUT의 시험에서 다루어야 하는 state 수를 줄일 수 있으며 IUT 내부의 transient state를 관찰해야 하는 어려움을 피할 수 있다.

3. 알고리즘 제안

이 장에서는 black box 형태인 IUT의 시험에서 발생하는 병렬 이벤트를 제어하는 기존 방법의 문제점을 분석하고, 이 문제의 해결책으로 logical clock을 이용한 LTS 생성 알고리즘을 제시한다.

3.1 기존 시험 방법의 문제점

분산 시스템을 black box로 하여 시험하는 논문[1]의 방법을 분석하여 보자. 본 논문에서 사용할 FSM M은 그림 2(a)와 같고 각 state는 stable state이다. 이 FSM M은 3개의 포트에 입출력이 있는 3-port FSM 이고, 각 포트별 입출력은 표 1과 같다. 그림 2(b)는 그림 2(a)의 faulty FSM M′이다. 각 포트는 동일한 id를 가진 시험기와 일 대 일(one-to-one)로 연결되어 있다.



(a) FSM M (b) Faulty FSM M′

그림 2 3-port FSM

표 1 포트별 입출력

	port 1	port 2	port 3
Input	a	b	c, d
output	x, w	y	z

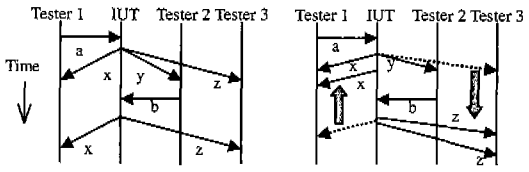
정의 1: $x \in X$, $y \in Y$, $\{s_i, s_j\} \subset S$, $s_j = \delta(s_i, x)$, $y = \lambda(s_i, x)$인 FSM에서 트랜지션 $T=(s_i, s_j, x/y)$이다. 이 때, X는 입력의 유한 집합, Y는 출력의 유한 집합, S는 state의 유한 집합, δ는 state transfer function, λ는 output function 이다.

Faulty FSM M′으로 표현된 IUT를 TS(=T1→T3 T2→T5→T6)로 시험하는 경우에 sub sequence T3→T2와 T5→T6에서 제어-관찰 문제가 발생한다. Sub sequence T3→T2에서 트랜지션 T3의 입력 'c'와 트랜지션 T2의 입력 'b'가 state 1에서 IUT로 병렬 입력되었을 때, IUT가 입력 'b'를 먼저 읽게 되면 시험기는 fault detection을 하지 못한다. 왜냐하면 각 시험기에서 관찰되는 출력 순서가 FSM M을 시험하는 것과 동일하기 때문이다. 그리고 sub sequence T5→T6에서는 입출력 매핑이 잘못되었어도 시험기에서 관찰되는 출력 순서가 정상이므로 각 시험기에서 fault detection을 하지 못한다[1, 3].

분산 객체의 TS는 일반적으로 입출력, 병렬 이벤트를 제어하는 시그널 'C'와 'O'로 구성된다. 시그널 'C'는 IUT에 가해지는 병렬 입력이 인과 관계를 갖도록, 시그널 'O'는 병렬 관계에 놓인 입력과 출력 사이에 인과 관계를 맺도록 해주는 시그널이다. 시그널 'C'와 'O'는 다음 단계로 생성된다. 시그널 'C'와 'O'를 포함하는 각 분산 객체의 TS를 Local Test Sequence(LTS)라고 한다.

1. 시험하려는 TS의 모든 이벤트에 logical clock 값 기록
2. 이벤트들의 logical clock 값의 비교로 병렬 혹은 인과 관계에 놓인 이벤트로 분류
3. 병렬 관계에 놓인 이벤트가 존재한다면, 병렬 이벤트를 제어하는 시그널 'C'와 'O'를 생성하고 각 분산 객체의 TS에 삽입

제어-관찰 문제를 해결하는 방법으로 논문[1]은 'C'와 'O' 시그널을 시험기들이 주고 받을 수 있는 협동 채널을 구현하였다. 그러나 논문[1]의 방법에서는 제어-관찰 문제의 subset인 출력 이동 문제를 제어하지 못하는 경우가 발생할 수 있다.



(a) (b)

그림 3 출력 이동 문제의 예

논문[1]의 방법을 이용하여 black box 형태인 IUT에 TS(=a/xyz→b/xz)로 시험하는 경우는 그림 3(a)와 같이 'C'와 'O' 시그널이 생성되지 않는다. 그러나 그림 3(b)와 같이 "a/xyz"의 출력 'z'가 전진 이동(forward shifting), "b/xz"의 출력 'x'가 후진 이동(backward shifting) 되어도 시험기들은 fault detection을 하지 못한다. 왜냐하면 각 시험기에서 관찰되는 입출력의 순서가 그림 3(a)와 동일하기 때문이다. 이것을 **출력 이동**

문제(output-shifting faults)[8]라고 한다. 논문[1]의 방법을 적용했을 때 출력 이동 문제가 발생하는 이유는 새로운 입력이 IUT에 가해지기 전에 이 입력과 병렬 관계에 놓인 직전 트랜지션의 출력을 수신하는 시험기들을 모두 제어하지 못했기 때문이다.

3.2 Local Test Sequence 생성 알고리즘

출력 이동 문제를 포함하는 제어-관찰 문제를 해결할 수 있는 LTS 생성 알고리즘을 간단한 예로써 설명하고자 한다. 본 논문은 다음의 가정으로 logical clock을 기록한다.

- IUT는 black box이다.
- Logical clock의 벡터는 (tester 1, IUT, tester 2, tester 3, ...) 형식으로 표현되며, 각 구성 요소에서 이벤트가 발생하면 해당 필드의 값이 증가한다.
- 트랜지션의 출력 수신 지연 시간은 0이다.
- Logical clock은 단위 시간(unit time)으로 1씩 증가한다.

그림 4는 FSM M으로 표현되는 IUT를 시험하는 TS1(=T1→T3→T2→T4→T7)의 각 이벤트에 logical clock을 기록한 것이다.

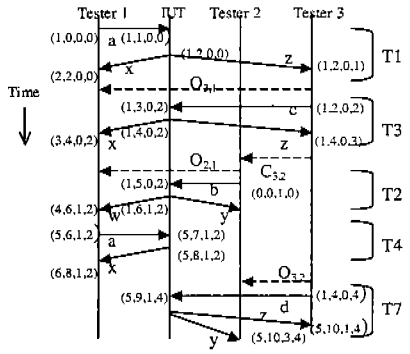


그림 4 TS1의 이벤트에 logical clock을 기록한 예

규칙 2에 의해서 분류된 병렬 이벤트들을 제어하려면 부가적인 시그널이 필요하다. 표 2는 전진 출력 이동 문제를 해결하기 위하여 sub sequence(T1→T3와 T3→T2)의 이벤트에 기록된 logical clock을 비교한 것이다. Compare_LC(x, y)는 이벤트 'x'와 'y'가 인과 관계(규칙 2)를 만족하면 true, 그렇지 않으면 false를 return하는 함수이다.

표 2에 의하면 sub sequence T1→T3에서 트랜지션 T3의 입력 'c'는 트랜지션 T1의 출력 'x'와만 병렬 관계이다. 이 경우 입력 'c'와 출력 'x'가 인과 관계를 갖을

표 2 Logical clock 값의 비교

T1(0, 1, a/xz)→T3(1, 1, c/xz)	T3(1, 1, c/xz)→T2(1, 1, b/wy)
Compare_LC(a, c)=true	Compare_LC(x, c)=false
Compare_LC(z, c)=true	Compare_LC(c, b)=false
Compare_LC(x, b)=false	Compare_LC(z, b)=false

수 있도록 제어하는 시그널이 필요하다. 이 시그널을 'O' 라고 하며 트랜지션 T3의 입력 'c'를 IUT에 가하는 Tester 3이 T1의 출력 'x'를 수신하는 Tester 1로 '$O_{3,1}$' 시그널을 전송한다.

그리고 sub sequence T3→T2에서 트랜지션 T2의 입력 'b'는 트랜지션 T3의 모든 입출력과 병렬 관계이다. 따라서 입력 'b'가 트랜지션 T3가 수행된 후 IUT에 가해질 수 있도록 제어할 수 있는 시그널이 필요한데, 이 시그널을 'C'라고 한다. 이 경우에는 트랜지션 T3의 출력을 수신한 시험기 중 하나인 Tester 3에서 트랜지션 T2의 입력 'b'를 IUT에 가하는 Tester 2로 '$C_{3,2}$' 시그널을 전송한다. 이 '$C_{3,2}$' 시그널은 Tester 3에서 출력 'z'를 수신했음을 알려준다. 그러나 Tester 1에서 수신하는 출력 'x'와 입력 'b'는 여전히 병렬 관계에 놓이므로 Tester 2에서 Tester 1로 '$O_{2,1}$' 시그널을 전송한다.

Sub sequence T4→T7에서 출력 'y'가 후진 이동되어도 Tester 2의 시험 시나리오는 정상인 FSM을 시험하는 것과 동일하므로 fault detection을 하지 못한다. 이 문제를 해결하기 위하여 Tester 3은 입력 'd'를 IUT에 가하기 전에 '$O_{3,2}$' 시그널을 Tester 2에 전송한다.

언급한 바대로 제안한 알고리즘은 출력 수신 지연 시간이 0이고, 트랜지션의 출력들은 동시에 각 시험기에서 수신된다. 따라서 병렬 출력을 제어하는 'O' 시그널은 IUT에 입력을 가하는 시험기에서 생성된다. 이 가정은 논문[1]에서도 적용된 것이다. 그러나 제안한 TS 생성 알고리즘은 실제 분산 시스템을 시험하는데 제약이 따른다.

가령, IUT가 정확히 구현되었을지라도 Tester 1이

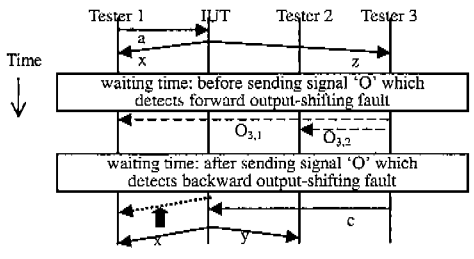


그림 5 대기 시간과 대기 시간 초과 문제

'$O_{3,1}$' 시그널을 수신한 후 a/xz의 출력 'x'를 수신하면 fault로 판정한다. 그리고 Tester 2는 '$O_{3,2}$' 시그널을 수신한 후 c/xy의 후진 이동된 출력 'y'를 수신한 경우에 fault detection을 할 수 없다. 그 이유는 시험기 사이의 동기화는 동기 시그널을 통해 이루어졌지만, IUT는 시험기와 직접적으로 동기화 되지 못했기 때문이다. 따라서, 시험기들은 IUT와 동기화 하기 위하여 그림 5와 같은 충분한 대기 시간이 필요하다. 그러나 Tester 1이 그림 5와 같이 대기 시간 후에 c/xy 후진 이동 된 출력 'x'를 수신한다면 fault detection이 불가능하다. 이 문제를 대기 시간 초과 문제(over-waiting time problem)라고 하자. 이 문제를 발생시키지 않기 위해 대기 시간과 출력 수신 지연 시간을 0으로 가정한다. 제안된 알고리즘과 관련 정의는 다음과 같다.

정의 2

- 트랜지션 T=(s_i, s_j, x/y)에서 "x/y"는 "!x?y"로 표현될 수 있다.
- '−'는 다른 시험기로의 시그널 전송, '+'는 다른 시험기로부터의 시그널 수신을 의미한다.
- Port(x)는 이벤트 'x'가 출입하는 포트 id를 얻어내는 함수이다.
- PortS(X)={Port(x_1), Port(x_2), ..., Port(x_n)}. 단, X={x_1, x_2, ..., x_n}, x_i는 이벤트.
- Compare_LCS(x, Y)는 이벤트 집합 Y의 모든 원소에 (∀a∈Y) 대하여, Compare_LC(x, a)가 false이면 false, 그렇지 않으면 true를 return하는

Algorithm 1 /* LTS Generating Algorithm */

```
 1: INPUT: TS=!s1?Y1!sm?Ym /*The elements of Yi can be more than zero and is represented with an array */
 2: OUTPUT: local test sequences=(LTS1,...,LTSn)  /* n is the number of distributed objects */
 3: Send_Set=null /* Send_Set is a set of testers receiving signal 'O' */
 4: for(j=1; j  m; j++)     /* m is the number of transitions */
 5:  Label_LC(sij, Port(sj ));
 6:  Label_LCS(Yi, PortS(Yi));          /*Logical colock values labeling at input sj and outputs Yi*/
 7: end for
 8: for(i=1; i  m; i++)
 9:  k=Port(si); h=Port(si+1);
10:  Append_In(Port(si), si );
11:  if i < m
12:   case |Yi| ≠ 0:                    /*Comparison of logical colock values for generating signal 'C'*/
13:    Append_Out(PortS(Yi), Yi);
14:    if (Compare_LCS(si+1,Yi)=false) and (Compare_LC(si+1, si)=false)
15:     select a sender (tester) which generates signal 'C' where sender  PortS(Yi);
16:     Append(LTSsender, Append("−", "Csender,k")); Append(LTSk, Append("+", "Csender,k"));
17:    end if                          /*Comparison of logical clock values for generating signal
18:    for(j=0; j < |Yi|; j++)          'O' to protect forward output-shifting faults*/
19:     if (Compare_LC(si+1, Yi[j])=false) and (Port(Yi[j]) ≠ sender)
20:      Send_Set←Send_Set ∪ Port(Yi[j]);
21:     end if
22:    end for                         /*Comparison of logical clock values for generating signal
23:    for(j=0; j < |Yi+1|; j++)        'O' to protect backward output-shifting faults*/
24:     if[(Compare_LCS(Yi+1[j], Yi)=true) and (Port(Yi+1[j])∈PortS(Yi)) and (Port(Yi+1[j])≠h)]
        or[(Compare_LCS(Yi+1[j], Yi)=false) and (Port(Yi+1[j])≠h)]
25:      Send_Set←Send_Set∪Port(Yi+1[j]);
26:     end if
27:    end for                         /*Comparison of logical clock value for generating signal 'C'*/
28:   case |Yi|=0:
29:    if Compare_LC( si, si+1)=false
30:     Append(LTSk, Append("−", "Ck,h"); Append(LTSh, Append("+", "Ck,h"));
31:    end if
32:    for(j=0; j < |Yi+1|; j++)
33:     if Port(Yi+1[j])≠h              /*Port comparison for generating signal 'O' to protect
                                         backward output-shifting faults*/
34:      Send_Set←Send_Set∪Port(Yi+1[j]);
35:     end if
36:    end for
37:   end case
38:   for ∀p∈Send_Set
39:    Append(LTSh, Append("−", "Oh,p"); Append(LTSp, Append("+", Oh,p));
40:   end for
41:  else if (i=m) and (|Yi |≠0)
42:   Append_Out(Port(Yi), Yi);
43:  end if
44: end for
end Algorithm 1
```

함수이다.

- Label_LC(x, k)는 이벤트 'x'가 포트 k에서 발생할 때, 포트 k의 벡터 값을 증가시키고 이벤트 'x'의 logical clock을 기록하는 함수이다.
- Label_LCS(X, PortS(X))={Label_LC(x_1, Port(x_1)), Label_LC(x_2, Port(x_2)), ..., Label_LC(x_n, Port(X_n))}. 단, X={x_1, x_2, , x_n}, x_i는 이벤트.
- Append(X, Y)는 인자 순서대로 배열 X와 Y를 결합(concatenation)시키는 함수이다.
- Append_In(Port(x), x)=Append($LTS_{Port(x)}$, Append("!", x)). 단, $LTS_{Port(x)}$는 Port(x)와 연결된 시험기의 LTS.
- Append_Out(PortS(X), X)={Append($LTS_{Port(X1)}$, Append("?", "x_1")), Append($LTS_{Port(X2)}$, Append("?", "x_2")), ..., Append($LTS_{Port(Xn)}$, Append("?", "x_n"))}. 단, X={x_1, x_2, ..., x_n}, x_i는 이벤트, $LTS_{Port(x)}$는 Port(x)와 연결된 시험기의 LTS.

Line 4~7은 TS의 이벤트에 logical clock을 기록하는 부분이다. IUT에서 입출력이 발생될 때마다 Label_LC(또는 Label_LCS) 함수를 이용하여 입출력이 발생된 포트의 벡터 값을 갱신한다. Line 14~40은 최종 트랜지션을 제외한 트랜지션에서 'C'와 'O' 시그널을 생성하는 부분이다. 트랜지션의 출력이 null이 아닌 경우는 Line 19와 같이 현재 트랜지션의 모든 이벤트와 다음 트랜지션의 입력에 기록된 logical clock을 비교하고, 모두 병렬 관계라면 'C' 시그널을 생성한다.

Line 19와 24은 각각 전진 출력 이동 문제와 후진 출력 이동 문제의 제어를 위한 logical clock 비교문이다. 트랜지션의 출력이 null인 경우는 Line 29와 같이 현재 및 다음 트랜지션의 입력에 기록된 logical clock을 비교하고, 병렬 관계라면 'C' 시그널을 생성한다. Line 33은 출력이 null인 경우에 후진 출력 이동 문제를 제어하기 위한 비교문이다. 그림 6은 TS1(=T1→T3→T2→T4→T7)을 Algorithm 1에 적용하여 얻어진 시험 시나리오이다.

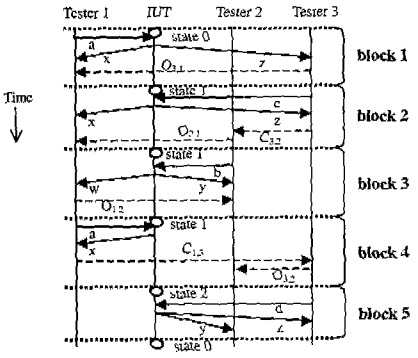


그림 6 TS1의 Algorithm 1 적용

그림 6에서 black box 형태인 IUT를 TS1로 시험하는 경우에 관찰되는 state 0, 1, 2는 새로운 입력이 IUT로 가해지기 전에는 다른 state로 천이하지 않으므로 stable state 이다. 이 stable state 사이를 block으로 표시하면 각 block의 내부 이벤트는 'C'와 'O' 시그널을 이용하여 전체 순서화된 상태이므로 block 사이에도 happened before관계가 성립된다. 위의 시험 시나리오는 각 트랜지션을 분리하여 시험한 것과 동일한 결과를 도출한다.

3.3 Local Test Sequence 생성 알고리즘의 확장

논문[1]과 Algorithm 1의 LTS 생성 알고리즘은 출력 수신 지연 시간이 0이었다. 이 가정이 성립되지 않는다면 시험 결과는 재현되지 않을 수 있다. 그림 6에서 보인 문제의 해결을 위해서 입력 'c'와 병렬 관계인 'x'를 수신하는 Tester 1이 Tester 3으로 '$O_{3,1}$' 시그널을 전송하고, Tester 3은 이벤트 수신 순서에 관계없이 '$O_{3,1}$' 시그널과 'z'를 모두 수신한 후 입력 'c'를 IUT에 가하도록 Tester 3의 LTS를 생성해야 한다.

출력 수신 지연 시간이 0이 아닌 경우의 LTS 생성 알고리즘을 Algorithm 2라 하자. Append_RD(X, Y)는 X와 Y가 배열일 때, X의 배열 순서는 변화 시키지 않으면서 Y가 X의 원소 사이에 삽입되는 것을 의미한다. 예를 들어 Append_RD(xy, z)는 xyz, xzy 혹은 zxy를 의미한다.

Line 13~34는 Algorithm 1을 수정한 부분이다. Line 13(Case 1), 19(Case 2), 22(Case 3)는 출력이 null이 아닐 경우에 전진 출력 이동 문제의 제어를 위한 logical clock 비교문이다. Case 1은 현재 트랜지션의 모든 이벤트가 다음 트랜지션의 입력과 병렬 관계인 경우, Case 2는 현재 트랜지션의 출력과 다음 트랜지션의 입력 사이에만 병렬 관계인 경우이다. 그리고 Case 3은 현재 트랜지션의 출력의 일부와 다음 트랜지션의 입력이 인과 관계인 경우이다. 각 경우별로 생성되는 'O' 시그널은 트랜지션의 출력을 수신한 시험기(Temp_Set)에서 다음 트랜지션의 입력을 IUT에 가하는 시험기로 전송된다. 그러므로 입력측 시험기의 LTS는 입력 이전에 수신해야 하는 이벤트의 모든 수신 순서 조합을 포괄할 수 있어야 한다. 후진 출력 이동 문제를 해결하기 위한 'O' 시그널 생성은 Algorithm 1과 동일하다.

그림 7은 TS1을 Algorithm 2에 적용한 시험 시나리오의 예이며 Algorithm 2를 검증하기 위하여 SDL 도

Algorithm 2

```
1: INPUT: TS=!s1?Y1...!sm?Ym /*The elements for Yi can be more than zero and is represented by an array */
2: OUTPUT: local test sequences=(LTS1,...,LTSn)  /* n is the number of distributed objects */
3: Send_Set=null /* Send_Set is a set of testers which should receive signal 'O' */
4: for(j=1; j  m; j++)  /* m is the number of transitions */
5:  Label_LC(si, Port(si));
6:  Label_LCS(Yi, PortS(Yi));        <-- /*Logical clock values labeling at input si and outputs Yi*/
7: end for
8: for(i=1; i  m; i++)
9:  k=Port(si); h=Port(si+1);
10:  Append_In(k, si);
11:  if i < m                        <-- /*Case 1:comparison of logical clock values for generating signals 'C' and 'O'*/
12:   case |Yi|≠0:
13:    if (Compare_LCS(si+1, Yi)=false) and (Compare_LC(si+1, si)=false)
14:     Append_Out(PortS(Yi), Yi);
15:     select a sender (tester) which generates signal 'C' where sender∈PortS(Yi);
16:     Append(LTSsender, Append("−", "Csender,h"));
17:     Append_RD(temp, Append("+", "Csender,h"));
18:     Temp_Set←PortS(Yi) \ sender;  /* / is a difference set */
19:    else if (Compare_LCS(si+1, Yi)=false) and (Compare_LC(si+1, si)=true)
20:     Append_Out(PortS(Yi), Yi);
21:     Temp_Set←PortS(Yi);          <-- /*Case 2:comparison of logical clock values for generating signals 'O'*/
22:    else if Compare_LCS(si+1, Yi)=true
23:     for(j=0; j < |Yi|; j++)      <-- /*Case 3:comparison of logical clock values for generating signals 'O'*/
24:      if Compare_LC(si+1, Yi[j])=false
25:       Temp_Set←Temp_Set∪Port(Yi[j]);
26:       Append(LTSa, Append("?", Yi[j]));
27:      else Append(temp, Append("?", Yi[j]));
28:      end if
29:     end for
30:    end if
31:    for ∀p∈Temp_Set
32:     Append(LTSp, Append("−", "Op,h")); Append_RD(temp, Append("+", "Op,h"));
33:    end for
34:    Append(LTSh, temp);
35:    for(j=0; j < |Yi+1|; j++)
36:     if[(Compare_LCS(Yi+1[j], Yi)=true) and (Port(Yi+1[j])∉PortS(Yi)) and (Port(Yi+1[j])≠h)]
       or[(Compare_LCS(Yi+1[j], Yi)=false) and (Port(Yi+1[j])≠h)]
37:      Send_Set←Send_Set∪Port(Yi+1[j]);
38:     end if                       <-- /*comparison of logical clock values for generating signals 'O'
39:    end for                            to protect backward output-shifting faults*/
40:   case |Yi|=0:
41:    if Compare_LC(si, si+1)=false  <-- /*Comparison of logical clock value for generating signal 'C'*/
42:     Append(LTSk, Append("−", "Ck,h"); Append(LTSh, Append("+", "Ck,h"));
43:    end if
44:    for(j=0; j < |Yi+1|; j++)      <-- /*Port Comparison for generating signal 'O' to protect backward output-shifting faults*/
45:     if Port(Yi+1[j])≠h
46:      Send_Set←Send_Set∪Port(Yi+1[j]);
47:     end if
48:    end for
49:   end case
50:   for ∀p∈Send_Set
51:    Append(LTSk, Append("−", "Ok,p"); Append(LTSp, Append("+", "Ok,p"));
52:   end for
53:  else if (i=m) and (|Yi |≠0)
54:   Append_Out(Port(Yi), Yi);
55:  end if
56: end for
end Algorithm 2
```

구(SDT)[14]를 사용한다. 검증을 위하여 FSM을 IUT 프로세스로 표현하고, TS1을 Algorithm 2에 적용하여 얻은 LTS를 프로세스로 기술한다. 여기서 트랜지션의 출력 수신 지연 시간은 0이 아니다.

그림 7의 시나리오를 Message Sequence Chart (MSC)를 사용하여 Reachability Tree로 분석하여 보자. 각 점에 표현된 state는 Tester 1, IUT, Tester 2, Tester 3의 각 상태를 나타내는 global state이다.

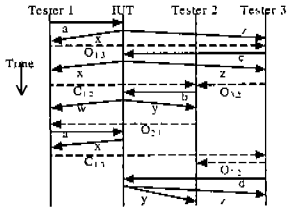


그림 7 TS1의 Algorithm 2 적용으로 생성된 LTS

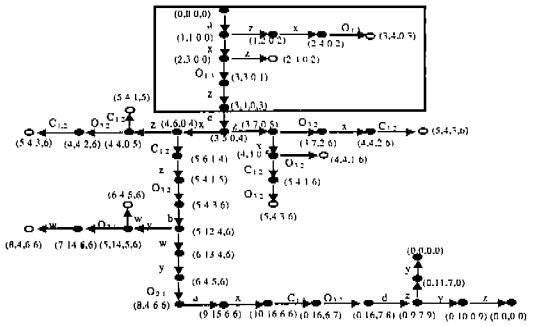


그림 8 Reachability Tree

위의 그림에서 박스는 TS1에서 트랜지션 T1의 Reachability Tree를 나타낸다. 트랜지션 T3의 입력 'c'를 IUT에 가하기 위해서는 logical clock 값이(3, 4, 0, 3) 이어야 한다. 즉, stable state(3, 4, 0, 3)에서만 입력 'c'가 IUT에 가해진다. 따라서 Algorithm 2는 병렬 이벤트의 제어를 통해 시험 결과를 재현하는 LTS 생성 알고리즘임을 알 수 있다.

Algorithm 1과 Algorithm 2를 Q.2971 망 부분 일대 다 호/연결 설정을 위한 메시지 교환에 적용하여 보자. 그림 9는 호 발생자(call initiator)가 S_A이고 호 수신자(call receiver)가 R_A와 R_B인 경우의 메시지 교환이다[12, 13]. IUT는 S_A, R_A, R_B와 연결된 3개의 포트를 가진다. 그림 9를 각각 Algorithm 1과 Algorithm 2에 적용한 결과는 그림 10과 같다.

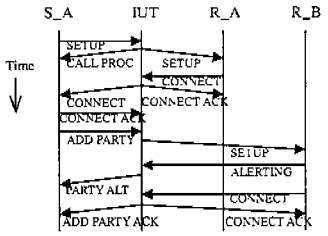
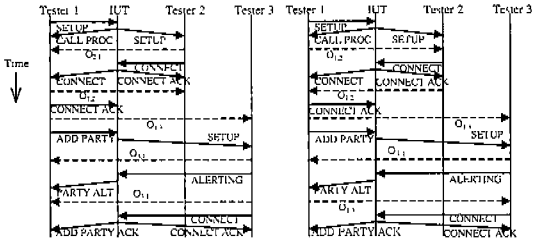


그림 9 Q.2971 망 부분 일 대 다 호/연결 설정을 위한 메시지 교환



(a) Algorithm 1 적용 (b) Algorithm 2 적용

그림 10 Algorithm 1및 Algorithm 2를 적용한 Q.2971의 LTS

그림 10(a)에서 시험기들은 거의 동시에 출력을 수신해야 한다. 그러나 Q.2971을 사용하는 통신망에서의 메시지 전송은 그와 같은 가정이 성립될 수 없다. 따라서 그림 10(b)의 시험 시나리오가 보다 일반적이라 하겠다. Algorithm 1, Algorithm 2와 논문[1]의 LTS 생성 알고리즘을 비교하면 아래 표 3과 같다.

표 3 LTS 생성 알고리즘 비교

	Paper[1]	Algorithm 1	Algorithm 2
출력 수신 지연 허용 여부	No	No	Yes
대기 시간 초과 문제를 제외한 후진 출력 이동 문제 발생 여부	Yes	No	No
전진 출력 이동 문제 발생 여부	Yes	No	No
대기 시간 초과 문제 발생 여부	-	Yes	No
채널 복잡도	$O(n^4)$	$O(n^2)$	$O(n^4)$
'C'와 'O' 시그널 개수	Paper[1] < Algorithm 1 =Algorithm 2		

표 3에서 볼 수 있듯이 Algorithm 1은 대기 시간 초과 문제를 제외한 출력 이동 문제는 발생하지 않는다. Algorithm 2는 분산 시스템의 시험 시, 출력 지연을 허용함으로써 대기 시간 초과 문제를 해결한다. 다시 말하면, Algorithm 2는 출력 수신 지연 시간이 0인 경우에서도 LTS를 생성할 수 있다. 또한 각 알고리즘은 시험기 사이의 채널 연결이 그물형(mesh)인 분산 시험 방법의 시험 구조에 적용되므로 채널 복잡도는 모두 $O(n^2)$이다.

4. 시험 구조

이 장에서는 시험기 사이의 협동 채널이 없는 원격 시험 방법의 시험 구조에 Algorithm 2를 적용하여 분산 시스템을 시험할 수 있는 방안을 강구한다.

4.1 제안된 시험 구조

분산 시험 방법의 시험 구조에서는 시험기 사이의 채널 연결은 그물형이기 때문에 협동 채널이 비선형으로 증가한다. 따라서 IUT의 포트가 증가할 때, 시험기가 통신해야 하는 객체가 증가하여 시험기의 진단력이 감소한다. 이와 같은 문제점을 해결하고자 채널 복잡도가 O(n)인 그림 11의 시험 구조를 제안한다. Management and Control Entity(MCE)는 시험기로부터 동기 시그널을 수신하고 이 시그널을 시험기로 전송하는 역할을 한다.

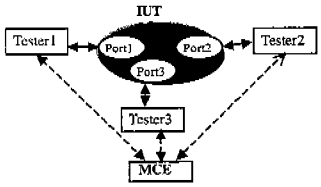


그림 11 제안한 시험 구조

연속된 두 트랜지션 (state a, state b, x, y)와 (state b, state c, x, y)에서 PortS(y)=null이고 Port(x)=Port(x)인 경우와 Port(x)=PortS(y)인 경우를 제외하고는 Port(x)와 연결된 시험기는 MCE로부터 항상 동기 시그널을 수신해야 한다. 왜냐하면 오직 MCE를 통해서만 다른 시험기와 동기화 할 수 있기 때문이다. 따라서 출력 사이 및 출력과 입력 사이의 병렬 문제는 동기 시그널 만으로 제어 가능하므로 'O' 시그널을 동기 시그널로 사용하기로 한다. 그림 12는 Q.2971 망 부분 일 대 다 호/연결 설정을 위한 메시지 교환을 제안한 알고리즘에 적용한 것이다.

제안한 알고리즘의 적용에 있어서 입력에 대한 출력이 없고 병렬 이벤트의 제어를 위한 'O' 시그널이 필요한 트랜지션의 경우에는 시험기와 MCE 사이에 부가적인 시그널이 필요하다. 그림 12의 박스에서 볼 수 있듯이 "ADD PARTY/SETUP"에서 출력 'SETUP'의 후진 출력 이동 문제를 제어하는 '$O_{M,3}$' 시그널의 생성을 위해서는 부가적으로 '$O_{1,M}$' 시그널이 필요하다. 왜냐하면 'O' 시그널의 전송은 오직 MCE에서 가능하기 때문이다. 여기에서 '$O_{M,1}$' 시그널은 입력 'ADD PARTY'를 IUT에 가하는 Tester 1의 동기 시그널이다.

Q.2971의 시험을 위해서는 분산 시험 방법의 시험 구조 보다 그림 11에서 제안한 시험 구조가 더 효율적이다. 'O' 시그널의 증가로 인해 MCE에서 병목 현상이 발생할 수 있는 단점이 있으나 호 수신자 수를 증가시켜도 시험 구조의 변경이 최소화되고, 채널 fault에 의한 시험 오류가 감소하기 때문이다.

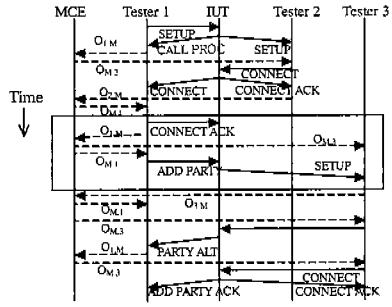


그림 12 Q.2971 망 부분 일 대 다 호/연결 설정을 위한 메시지 교환의 Algorithm 3 적용

5. 결론과 향후 연구

본 논문에서는 분산 시스템의 시험을 위해 logical clock을 이용한 LTS 생성 알고리즘을 제안하였다. 제안된 알고리즘은 우선 TS의 이벤트에 logical clock을 기록하고 비교하여 이벤트를 병렬 또는 인과 관계로 분류한다. 그리고 병렬 관계인 이벤트를 제어하기 위한 'C'와 'O' 시그널을 생성하고 각 시험기의 TS에 삽입한다. Logical clock은 수치적인 계산이 가능하므로 이벤트를 인과 또는 병렬 관계로 분류하기 용이하다. 또한 동기 및 비동기 통신에서 모두 적용 될 수 있다.

본 논문에서는 험기 사이의 채널이 존재하는 분산 시험 방법의 시험 구조에서 출력 수신 지연이 없는 경우와 그렇지 않은 경우에서 LTS 생성 알고리즘을 제시하고 이를 구현하였다. 그 결과 기 생성된 TS에 대한 LTS를 자동 생성할 수 있게 되었다. 제안된 알고리즘에 의해 생성된 시험 시나리오를 Reachability Tree로 분석한 결과 stable state에서만 입력이 IUT에 가해짐을 알 수 있었다. 따라서 제안된 알고리즘은 병렬 이벤트에 의해 발생되는 출력 이동 문제를 포함하는 제어-관찰 문제를 정형적인 방법으로 해결하는 것이라 할 수 있다.

또한 분산 시험 방법의 시험 구조는 시험기 사이의 채널 연결이 그물형이기 때문에 시험기의 수가 증가할 때 채널이 비선형으로 증가하는 단점이 있다. 본 논문에서는 이 단점을 극복하고자 채널 복잡도가 O(n)인 시험 구조를 제안하였다. 그리고 제안한 LTS 생성 알고리즘을 제안한 시험 구조에 적용하기 위하여 수정하였다. 그 결과 단순한 시험 구조로 인하여 시험기의 진단력이 증가하였다.

향후, 대기 시간 초과 문제와 동적 시험 방법을 이용한 분산 시스템의 시험 방법에 대하여 연구할 것이다.

참 고 문 헌

[1] M. Benattou, L. Cacciari, R. Pasini and O. Rafiq, "Principles and Tools for Testing Open Distributed Systems," Int'l Workshop on Testing of Communicating Systems, pp.77-92, Budapest, Hungary, September 1999.

[2] A. Ulrich and H. Konig, "Architectures for Testing Distributed Systems," Int'l Workshop on Testing of Communicating Systems, pp.93-107, Budapest, Hungary, September 1999.

[3] G. Luo, R. Dssouli, G.v. Bochmann, P. Venkataram and A. Ghedamsi, "Test Generation With Respect To Distributed Interfaces," Computer Standards and Interfaces, pp.119-132, 1994.

[4] K. Tai, R. Carver and E. Obaid, "Debugging Concurrent Ada Programs by Deterministic Execution," IEEE Trans. Software Engineering, Vol 17, No.1, pp.45-63, January 1991.

[5] M. Kim, S. T. Chanson, S. Kang and J. Shin, "An Enhanced Model for Testing Asynchronous Communicating Systems," FORTE/PSTV'99, June 1999.

[6] C. Fidge, "Logical Time in Distributed Computing Systems," IEEE Computer, pp.28-33, August 1991.

[7] G. Luo, G. v. Bochman and A. Petrenko, "Test Selection Based on Communicating Nondeterministic Finite-State Machines using a Generalized Wp-Method," IEEE Trans. Software Engineering, Vol 20, No. 2, pp.149-162, February 1994.

[8] Y. C. Young and K. C. Tai, "Observational Inaccuracy in Conformance Testing with Multiple testers," IEEE 1st Workshop on Application-specific Software Engineering and Technology, pp.80-85, 1998.

[9] T. V. Gioles, I. Schieferdecker, M. Born, M. Winkler and M. Li, "Configuration and Execution Support for Distributed Tests," Int'l Workshop on Testing of Communicating Systems, pp.61-76, Budapest, Hungary, September 1999.

[10] G. Coulouris, J. Dollimore and T. Kindberg, "Distributed Systems, Concepts and Design," Second Edition, Addison-Wesley, 1994.

[11] H. Herzog, K. Sunderhaft, "General Framework for fault tolerance from ISO/ITU Reference Model for Open Distributed Processing(RM-ODP)," Object-Oriented Real-Time Dependable Systems, pp.111-118, 1999.

[12] Y. Jung and J. Lee, "Experiences with Generation of Conformance Test Suite for Q.2971 Network-side Testing," Information Networking, pp.286-289, 1998.

[13] ITU-T Draft Recommendation, Q.2971 B-ISDN Digital Subscriber Signalling No. 2(-DSS2) User Network Interface layer 3 Specification for Point-to-Multipoint Call/Connection Control, 1995.

[14] Telelogic SDT3.2: Getting Started, Part1: Tutorials on SDT Tools, Telelogic, Setptmber 1997.

최 영 준

1995년 3월 ~ 1999년 2월 홍익대학교 컴퓨터공학과 졸업(학사). 1999년 3월 ~ 2001년 2월 한국정보통신대학원대학교(공학석사). 1999년 3월 ~ 1999년 12월 한국통신 통신망 연구소 망연동 연구실 위촉연구원. 2001년 2월 ~ 현재 금융결제원 전자금융연구소. 관심분야는 프로토콜 시험, 차세대인터넷, 이동 컴퓨팅, 전자 인증

김 명 철

1982년 아주대학교 전자공학과 졸업(학사). 1984년 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 통신망 및 멀티미디어 프로토콜 공학

설 순 욱

1998년 2월 한국기술교육대학교 정보통신공학과 졸업(학사). 2000년 2월 한국정보통신대학원대학교(공학석사). 1998년 ~ 1999년 한국전자통신연구원 네트워크 장비시험센터 위촉연구원. 2000년 3월 ~ 현재 한국정보통신대학원대학교 박사과정. 관심분야는 프로토콜 시험, 이동 컴퓨팅, 차세대인터넷.