명세 변경 후 병행 프로그램의 순서 제약조건 기반 회귀 테스팅

(Sequencing Constraints-based Regression Testing of Concurrent Programs After Specification Changes)

김 현 수 † 정 인 상 †† 배 현 섭 †††
(Hyeon Soo Kim) (In Sang Chung) (Hyun Seop Bae)
권 용 래 †††† 이 동 길 †††††
(Yong Rae Kwon) (Dong-Gil Lee)

요 약 이 논문에서는 명세에 변경이 가해진 후에 병행 프로그램을 재검증하기 위해 사용하는 명세 기반 회귀 테스팅이라는 새로운 기법에 대해 논의한다. 이러한 형태의 회귀 테스팅은 이벤트에 대해 선후 관계를 기술하는 순서 제약조건을 필요로 한다. 순서 제약조건을 자동으로 추출하기 위해서 우리는 메시지 순차도(MSC)를 사용한다. 메시지 순차 도를 통해 부분적이고 비결정적인 명세를 작성할 수 있다. 회귀 테스팅의 비용을 줄이기 위하여 처음부터 새로운 순서 제약조건을 생성하기보다는 명세에 가해진 변경에 의해 영향받는 순서 제약조건을 어떻게 파악하는 가에 대해 기술한다. 또한 각각의 영향받는 순서 제약조건들이 테스팅 될 프로그램에 의해 만족되는 가를 결정하는 방법에 대해서도 기술한다.

Abstract This paper describes a new technique known as specification-based regression testing that is used for the revalidation of concurrent programs after changes are made to specifications. This type of regression testing requires sequencing constraint that specify precedence relations on the synchronization events. In order to extract sequencing constraint automatically, we use Message Sequence Charts(MSCs) that are considered partial and nondeterministic specifications. We show how to identify which sequencing constraint is affected by the modifications made to a specification rather than creating new sequencing constraint from scratch to reduce the cost of regression testing. We also describe how to determine that each affected sequencing constraint is satisfied by a program being tested.

1. 서 론

회귀 테스팅에서 주요 관심사는 프로그램에 도입된 변경 사항이 정확하고, 또한 그것이 변경되지 않은 프로그램 부분에 나쁜 영향을 주지 않음을 확인하는 것이다. 회귀 테스팅 과정 동안에 수정된 프로그램을 재검증하기 위해서는 앞서 사용된 모든 테스트 케이스들이 전개된다. 이러한 형태의 재검증 방법들은 많은 시간과 자원들을 낭비하게 된다. 회귀 테스팅의 비용을 줄이기 위해 효과적인 회귀 테스팅 기법의 설계에 관해 많은 연구가 진행되었다. 그러나 그런 기법들의 대부분이 순차 프로그램의 회귀 테스팅에 초점을 맞춰 이루어졌다.

게다가 회귀 테스팅에 관한 대부분의 선행 연구들이 원래 프로그램과 수정된 프로그램의 코드를 이용하여 테스트 케이스를 선택하는 코드 기반 방법들이다. 따라

† 정 회 원 : 금오공과대학교 컴퓨터공학부 교수
hskim@cespc1.kumoh.ac.kr
†† 종신회원 : 한성대학교 정보전산학부 교수
insang@hansung.ac.kr
††† 정 회 원 : 한국전자통신연구원 연구원
hsbae@etri.re.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
yrkwon@salmosa.kaist.ac.kr
††††† 비 회 원 : 한국전자통신연구원 개발환경연구실 연구원
dglee@etri.re.kr
논문접수 : 1999년 7월 22일
심사완료 : 2000년 2월 3일

서, 명세에 변경이 가해진 경우의 회귀 테스팅 기법에 대한 연구는 거의 없다. 실제적으로 명세의 오류를 정정하기 위해서 또는 프로그램의 기능을 확장하거나 변경하기 위해서 등의 다양한 이유로 인해 잦은 수정이 명세에 가해진다. 명세에 변경이 가해지면 따라서 해당 프로그램도 수정되어야 하고 그 수정된 프로그램은 반드시 재테스팅되어야 한다. 효과적으로 오류를 발견하기 위해서는 명세에 가해진 변경 정보가 테스트 케이스를 선택하는데 고려되어야 한다. 코드 기반 테스팅과 명세 기반 테스팅은 상호 보완적이다. 명세 기반 테스팅은 프로그램의 수행 중에 반드시 나타나야 할 행위들의 검증을 기초로 하고 있고, 코드 기반 테스팅은 프로그램 수행에 의해 실행 가능한 행위에 종속적이기 때문이다.

이 논문에서는 명세에 가해진 변경 정보를 바탕으로 병행 프로그램에 대한 명세 기반 회귀 테스팅 기법에 대하여 논의한다. 병행 프로그램에 대한 명세 기반 회귀 테스팅에서는 변경된 프로그램이 변경된 명세에 부합하는지를 검사하게 된다. 명세는 병행 프로그램이 수행 중에 만족해야 할 이벤트들 간의 순서 제약조건(sequencing constraints)을 포함하고 있다. 순서 제약조건을 이용한 병행 프로그램 테스팅에 대한 연구는 많이 진행되지 않았다. 예를 들어, Tai는 병행 프로그램의 이벤트 시퀀스들에 대해 제약을 명시하는 CSPE (Constraints on Succeeding and Preceding Events)의 사용을 기초로 한 테스팅 방법론을 제안하였다[1]. 이 테스팅 방법은 형식 명세로부터 자동으로 유도되거나 혹은 사용자로부터 주어진 순서 제약조건을 사용하여 이 순서 제약조건에 따르는 테스트 케이스로서 이벤트 시퀀스를 생성한다. 그러나, 이 방법은 명세에 변경이 가해진 후의 수정된 프로그램에 대한 재테스팅 문제에 대해서는 전혀 다루고 있지 않다.

본 논문에서는 메시지 순차도(Message Sequence Charts, MSCs)로 작성된 병행 프로그램의 명세로부터 명세 기반 회귀 테스팅을 위한 테스트 케이스를 자동으로 생성하고 생성된 테스트 케이스를 이용해서 병행 프로그램에 대한 재테스팅을 수행하는 방법에 대하여 논의한다. 명세에 변경이 가해지면 변경으로 인해 영향받는 이벤트들간의 순서 제약조건을 추출하는 것은 대단히 중요하다. 한 가지 방법으로 처음부터 순서 제약조건을 새롭게 생성하여야 한다. 그런데 이러한 방법은 많은 노력을 필요로 한다. 회귀 테스팅의 비용을 줄이기 위해서는 명세에 가해진 변경으로 인해 영향받는 순서 제약조건을 파악하고 단지 영향받은 순서 제약조건에 한해서만 수정된 프로그램에 대한 테스팅을 수행하는 것이다. 이러한 방법은 점진적 회귀 테스팅이라 불리며 이러한 방법을 통해서 우리는 새로운 순서 제약조건을 구축하기 위한 비용을 경감할 수 있다.

우리의 방법을 설명하기 위하여 부분적(partial)이고 비결정적(nondeterministic) 의미론을 갖는 MSCs를 사용한다. MSC는 병행/분산 프로그램의 선택된 수행 행위를 기술하기 위한 좋은 수단으로 널리 알려져 있다. 다른 형식 명세 기법과 비교하여 MSC는 태스크들 사이의 상호작용을 비교적 쉽게 기술할 수 있다. 이 논문에서 제시하는 테스팅 기법은 MSC 명세를 병행 프로그램의 수행 행위를 제약하기 위한 순서 제약조건으로 고려한다. 이러한 고려를 바탕으로 변경으로 인해 영향받는 순서 제약조건들을 파악하기 위한 변경 영향 분석 기법을 제시한다.

이 논문의 구성은 다음과 같다. 먼저 2절에서는 이 논문에서 사용하는 병행 프로그램에 대한 명세 언어인 MSC 명세 언어에 대해서 간략하게 설명하고 논문에서 제시한 기법을 설명하기 위한 예제로 전화 교환기 시스템의 동작 예제를 설명한다. 3절에서는 순서 제약조건 기반 회귀 테스팅 기법을 설명하고 4절에서는 관련연구를 설명하고 5절에서는 명세 기반 회귀 테스팅을 평가하기 위한 기준을 설정하고 이 논문에서 제시한 기법을 평가한다. 마지막으로 6절에서 결론 및 향후 연구 방향을 제시한다.

2. MSC 명세 언어

MSC는 통신 시스템과 같은 분산/병렬 시스템에서 통신 실체들 간에 이루어지는 메시지 교환을 직관적으로 기술할 수 있는 그래픽 언어로서 ITU의 Z.120에 의해서 표준화되어 있다[2,3]. MSC는 초기에 단지 SDL의 부가적인 다이어그램 형태로 사용되어 오다가 점차 통신 시스템의 요구 명세를 기술하기 위한 정형적인 언어로 발전하게 되었으며 최근의 MSC 버전은 구조적 언어의 구조 및 객체 지향 언어의 개념 등을 포함하기에 이르렀다.

MSC는 기본적으로 시스템에 포함된 병렬 컴포넌트(프로세스)들과 주변 환경간의 메시지 교환을 표현하는데 중점을 두고 있다. MSC에서 제공하는 구성요소는 기본요소와 구조요소로 나뉠 수 있다. 기본요소는 프로세스들과 각각의 내부 행동, 또한 그들 간의 메시지 전달 등을 표현하기 위해서 사용하는 요소들로서 프로세스 인스턴스, 메시지, 환경, 내부 행동, 타이머, 프로세스 생성 및 소멸, 조건(condition)과 병행영역(coregion)들로 구성된다. 구조요소는 다수의 메시지 순차도 간의 계

층구조를 표현하기 위해서 사용되는 요소들이다. 자세한 내용은 권고안 Z.120[2,3]을 참조 바란다.

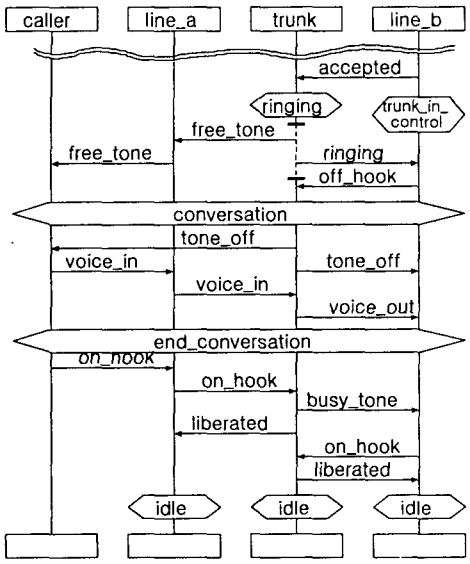

그림 1 전화 통화 예제에 대한 MSC

그림 1은 MSC 사용 예제로서 표준 MSC가 제공하는 대부분의 기본 요소들로 기술된 전화 통화 시나리오의 일부분을 보여주고 있다. 그림에는 caller, line_a, trunk, line_b의 네 개의 프로세스가 있는데 각각은 수직선으로 나타나있다. line_b 프로세스의 육각형 trunk_in_control과 같이 프로세스를 가로지르는 육각형은 수행 중에 프로세스가 만족해야 하는 조건을 표현한다. 물론, conversation과 end_conversation과 같이 두 개 이상의 프로세스를 가로지르는 육각형은 공유 조건을 의미한다. 두 프로세스를 연결하는 수평 화살표들은 메시지 전달에 의한 통신을 의미한다. 마지막으로, trunk 프로세스에는 수직 점선으로 표현된 부분이 나타나는데 이를 병행영역(coregion)이라 부른다. 한 프로세스 내에 포함된 이벤트들은 수직선상의 위치에 따라 순서를 가지지만 병행영역에 포함된 이벤트들은 순서를 가지지 않는다. 예를 들어, trunk 프로세스는 순서 관계에 의해서 caller 프로세스에게 메시지 tone_off를 보내기 전까지는 메시지 tone_off를 line_b 프로세스에게 보낼 순 없지만, 메시지 free_tone과 메시지 ringing은 병행영역에 포함되어 있으므로 임의의 순서로 보낼 수 있다.

3. 순서 제약조건 기반의 회귀 테스팅

이 절에서는 이벤트들 사이의 순서 제약조건을 사용하는 회귀 테스팅 기법에 대하여 다룬다. 이 기법은 제약조건 추출 단계, 변경 영향 분석 단계, 비결정성 테스팅 단계, 마지막으로 결정성 테스팅 단계로 구성된다.

- 제약조건 추출(constraints elicitation): MSC명세에 기술된 이벤트들은 이벤트 선후 그래프(event precedence graph)를 통해 순서화 된다. 두 이벤트들 간에 존재하는 순서 제약조건은 이벤트 선후 그래프에서 하나의 이벤트를 나타내는 한 노드에서 다른 이벤트를 나타내는 다른 노드까지 에지를 통해 도달 가능한지 여부를 통해 결정될 수 있다.
- 변경 영향 분석(impact analysis): 명세 기반 회귀 테스팅에서는 명세에 변경이 가해진 후 이 변경으로 인해 영향받는 순서 제약조건들을 파악하기 위하여 변경 영향 분석을 실시하여야 한다. 우리는 모든 제약조건들을 테스팅하기 보다는 단지 영향받는 제약조건들에 대해서만 병렬 프로그램을 재테스팅 함으로써 회귀 테스팅의 비용을 경감할 수 있다.
- 비결정성 테스팅(*nondeterministic testing*): 비결정성 테스팅은 가능한 한 많은 이벤트 시퀀스가 실행될 수 있도록 주어진 입력 X를 가지고 병렬 프로그램 P를 여러 번 수행하는 방법이다. P의 반복적인 수행 동안에 P에 의해 실행되어 지는 이벤트 시퀀스들은 수집되고 이것들은 계속해서 변경 영향 분석과정에서 얻어진 순서 제약조건들을 만족하는지 정밀하게 검사된다.
- 결정성 테스팅(deterministic testing): 결정성 테스팅 기법은 프로그램이 주어진 입력 데이타를 가지고 주어진 이벤트 시퀀스를 따르도록 강제하는 방법이다[4,5]. 이 기법은 비결정성 테스팅이 적용되지 않는 순차적인 제약조건을 테스팅하기에 유용한 방법이다.

3.1 이벤트 선후 그래프(Events Precedence Graph)

주어진 MSC M에 포함된 이벤트들간의 선후 관계는 부분 순서 집합(partially ordered set)인 $(E_M, \rightarrow_M)$으로 표현될 수 있다. 여기서 E_M은 M에 있는 이벤트들의 집합을 의미하고 $\rightarrow_M$은 이벤트들 사이의 선후 관계를 의미한다. E_M에 속하는 각 이벤트들은 프로세스 이름과 관련된 행위의 타입, 그리고 메시지의 이름을 통하여 구별되어 진다. 예를 들어, "프로세스 t1이 메시지 m1을 전송한다"는 이벤트와 "프로세스 t2가 메시지 m2를 수신한다"는 이벤트는 각각 "t1:send(m1)"과 "t2:recv(m2)"로 표현된다. (공유)조건문과 관련된 이벤트들은 단지 통신 이벤트들간의 선후 관계를 파악하는데 사

용될 뿐이지 E_M에는 포함되지 않는다.

부분 순서 집합 (E_M, $\rightarrow_M$)은 DAG(directed acyclic graph) G_M = (E_M, A_M)으로 표현될 수 있다. 여기서, A_M은 에지의 집합으로서 다음과 같이 정의된다: A_M = {(u, v)|u, v $\in E_M$, u $\rightarrow_M$ v}. 이 논문에서 앞으로 G_M을 이벤트 선후 그래프(Event Precedence Graph) 또는 간단히 EPG라 부른다. A_M의 에지 (u,v)는 두 이벤트 u와 v사이의 직접 선후 관계를 의미하며 이행(transitive) 선후 관계는 G_M에서 명시적으로 나타나지는 않는다. A_M에 속하는 각 에지 (u,v)에 대해 u는 v의 직접 선행자(predecessor)라 부르고, v는 u의 직접 후행자(successor)라 부른다. 또한, PRED(v)는 이벤트 v의 모든 직접 선행자들의 집합을, SUC(v)는 이벤트 v의 모든 직접 후행자들의 집합을 나타낸다.

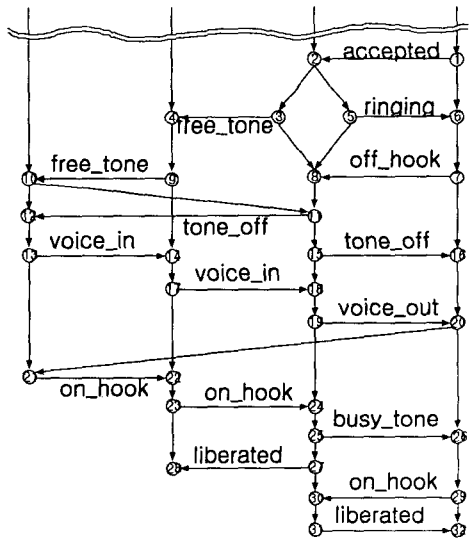

그림 2 그림 1의 MSC에 대한 EPG

그림 2는 그림 1의 MSC에 해당하는 EPG이다. 실제적으로 각각의 노드들은 "caller:recv(free_tone)"과 같이 이벤트 이름들로 명명되어야 하지만 간단히 표현하기 위해 메시지 이름을 에지의 이름으로 사용하고 각 노드들에 대해서는 번호를 할당하였다. 따라서 m이라는 이름의 에지의 시작 노드는 "ti:send(m)"이라는 이벤트를, 끝 노드는 "tj:recv(m)"이라는 이벤트를 의미한다. 예를 들어, 9번 노드는 "line_a:send(free_tone)" 이벤트를, 10번 노드는 "caller:recv(free_tone)" 이벤트를 나타낸다. 또한 이 EPG에서 PRED(8) = {3,5,7}이고 SUC(3) = {4,8}이다.

3.2 기본 변경에 대한 변경 영향 분석

이 절에서는 MSC M에 하나의 변경이 가해진 후, 이 변경으로 인해 영향받는 순서 제약조건을 G_M을 통해 파악하기 위한 변경 영향 분석 기법에 대해서 다룬다. 논문을 전개해 나가기 위해서 주어진 G_M에 대한 다음과 같은 표기법을 정의한다. 여기서 $\rightarrow_M^*$은 G_M에서 이행적 관계를 나타낸다.

- B(G_M/V) : 집합 V에 속하는 이벤트들의 모든 선행자들의 집합
  B(G_M/V)={w | for v $\in$V, w $\in E_M$ $\wedge$ w $\rightarrow_M^*$ v}
- F(G_M/V) : 집합 V에 속하는 이벤트들의 모든 후행자들의 집합
  F(G_M/V)={w | for v $\in$V, w $\in E_M$ $\wedge$ v $\rightarrow_M^*$ w}

이 논문에서 순서 제약조건은 이벤트들의 쌍 (a,b) 형태로 표현되며, 이는 프로그램 수행 중에 만족되어 할 이벤트 a와 b의 수행 순서에 대한 제약을 명시할 때 사용된다. 원래의 MSC를 M이라 하고 변경된 MSC를 M'이라 하자. 그러면 a, b $\in$ E_M $\cap$ $E_{M'}$에 대해 하나의 변경으로 인해 다음과 같은 결과가 초래된다면 이벤트들의 쌍 (a,b)는 영향받는 순서 제약조건이 된다.

(A1) a $\not\rightarrow_M$ b $\wedge$ b $\not\rightarrow_M$ a $\wedge$ a $\rightarrow_{M'}$ b
(A2) a $\rightarrow_{M'}$ b $\wedge$ a $\not\rightarrow_{M'}$ b $\wedge$ b $\not\rightarrow_{M'}$ a

(A1)은 병행관계에 있던 두 이벤트 a와 b가 변경에 의해 순차화 되었음을 의미하고 (A2)는 두 순차적 이벤트 a와 b가 변경 후에 병행 관계가 되었음을 의미한다. 통상적으로 MSC M에 메시지를 추가하는 연산과 메시지를 삭제하는 연산들을 기본 변경 연산으로 고려할 수 있다. 먼저, M에 메시지를 추가하는 경우를 고려하자.

메시지 m을 추가하는 연산은 EPG G_M에서 e = (ti:send(m), tj:recv(m))과 같은 새로운 에지 e를 생성하는 연산으로 해석할 수 있다. 이는 두 이벤트 "ti:send(m)"과 "tj:recv(m)" 사이에 다음과 같은 선후 관계를 제공한다: ti:send(m) $\rightarrow_M$ tj:recv(m). 이러한 관계가 이행적이라는 사실을 이용하여 새롭게 생성된 에지 e는 ti:send(m)의 선행자들의 집합 B(G_M/PRED(ti:send(m)))과 tj:recv(m)의 후행자들의 집합 F(G_M/SUC(tj:recv(m))) 사이에 선후 관계를 형성한다. 이 사실은 메시지 m을 추가함으로써 영향받는 모든 순서 제약조건들은 위의 (A1) 형태와 같고 아울러 이 순서 제약조건들은 집합 B(G_M/PRED(ti:send(m)))과 집합 F

(G_M/SUC(tj:recv(m)))의 카티션 곱(cartesian product)으로부터 얻을 수 있음을 의미한다.

집합 B(G_M/PRED(ti:send(m)))은 $\Delta_{B_m}[M]$으로 집합 F(G_M/SUC(tj:recv(m)))은 $\Delta_{F_m}[M]$으로 표현하자. 또한, 이들 두 집합의 카티션 곱은 $\Delta_m[M]$으로 표현하자. 즉 $\Delta_m[M] = \Delta_{B_m}[M] \times \Delta_{F_m}[M]$. 여기서 주목할 것은 Δ_m은 새롭게 추가된 이벤트인 "ti:send(m)"과 "tj:recv(m)"을 포함하지 않는다는 것이다. 메시지 m을 MSC M에 추가하더라도 새로운 선후 관계가 발생하지 않고 메시지를 추가하기 전에 구축되었던 관계가 계속 유지되는 경우가 있기 때문에, 영향받는 순서 제약조건을 파악하기 위해서 Δ_m의 모든 이벤트 쌍을 고려할 필요는 없다. 그러한 경우는 $\Delta_{B_m}[M]$에 속한 이벤트로부터 $\Delta_{F_m}[M]$에 속한 이벤트로 새로 추가된 에지 이외의 다른 경로가 G_M에 존재하는 경우에 발생한다. 이것은 (a,b) $\in \Delta_m[M]$에 대해 메시지 m을 추가하기 전에 이미 존재하던 관계 a $\rightarrow_M$ b가 메시지 추가 후에도 변함없이 유지됨을 뜻한다.

영향받는 순서 제약조건을 세밀히 파악하기 위해서 우리는 그래프 이론적인 기법을 사용한다. 이 문제는 다음과 같이 다시 기술될 수 있다.

$\Delta_{B_m}[M]$과 $\Delta_{F_m}[M]$으로부터 각각 유도된 두 서브 그래프 사이를 끊기 위해서는 얼마나 많은 에지가 EPG G_M으로부터 제거되어야 하는가?

cutset을 발견함으로써 이 문제에 답할 수 있다. cutset이란 에지의 삭제로 인해 두 서브그래프 사이의 연결이 끊어지게 되는 그런 에지들의 집합이다. 하나 이상의 cutset이 존재할 수 있지만 여기서는 단지 e=(ti:send(m), tj:recv(m))의 에지를 원소로 포함하는 cutset만을 고려한다. G_M에서 $\Delta_{B_m}[M]$과 $\Delta_{F_m}[M]$으로부터 유도된 두 서브 그래프들 사이의 연결을 끊는 cutset C를 파악한 후에, C가 단지 하나의 에지 e만을 포함하는지 검사해야만 한다. (C가 하나의 에지 e만을 포함할 때 e는 브리지라 불린다.) 만일 e가 브리지일 때 에지 e를 포함하는 경로를 제외한다면 $\Delta_{B_m}[M]$에 있는 이벤트로부터 $\Delta_{F_m}[M]$에 있는 이벤트로 도달할 수 있는 별도의 경로가 존재하지 않는다. 이 경우에, 병행적으로 다루어졌던 $\Delta_m[M]$의 모든 이벤트 쌍들은 변경 후 선후 관계에 의해 관련되어 진다. 그렇지 않다면 $\Delta_{B_m}[M]$에 있는 어떤 이벤트는 집합 C - {e} (즉, C_e)에 속하는 에지들을 통하여 $\Delta_{F_m}[M]$에 있는 다른 이벤트에 도달할 수 있다. 이것은 비록 메시지 m이 추가된 이후에도 Δ_m에는 영향받지 않는 순서 제약조건이 존재함을 의미한다.

영향받지 않는 순서 제약조건들은 집합 C_e에 속하는 에지들에 영향을 미치는 이벤트들의 이행 폐포(transitive closure)를 통해서 쉽게 파악될 수 있다. C_e가 $(e_{s_1}, e_{t_1}),\ldots,(e_{s_p}, e_{t_p})$의 에지들을 포함한다고 하면, $\bigcup_{i=1}^{p} B(G_M/\{e_{s_i}\}) \times F(G_M/\{e_{t_i}\})$를 계산함으로써 영향받지 않는 순서 제약조건들을 구할 수 있다. 이 식은 $\Delta_{U_m}[M]$으로 간략히 표현된다. $\Delta_{U_m}[M]$의 모든 이벤트 쌍들의 선후 관계는 비록 변경이 발생하였더라도 계속해서 보존된다. 즉, 변경 전에 a $\rightarrow_M$ b $\in \Delta_m[M]$의 관계가 성립하였고, 변경 후에도 이 관계가 계속해서 성립한다면, (a,b) $\in \Delta_{U_m}[M]$이다. 결과적으로 Δ_m과 Δ_{U_m}의 차집합은 $\Delta_{A_m}[M]$으로 표현되며, 이 집합은 MSC M에 메시지 m의 추가로 인해 영향받는 순서 제약조건들만을 포함한다. 즉, $\Delta_{A_m}[M] = \Delta_m[M] - \Delta_{U_m}[M]$

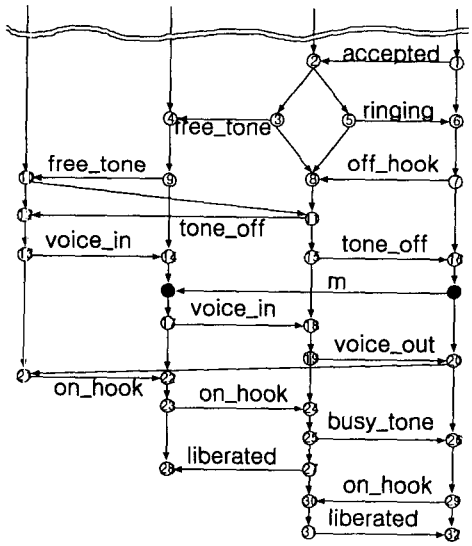

그림 3 메시지 m의 추가 후의 EPG

예를 들어, 그림 1의 MSC에서 프로세스 line_b로부터 프로세스 line_a로 메시지 tone_off 바로 아래와 voice_in 위의 위치에 메시지 m을 추가하는 경우를 고려하자. 메시지를 추가하고 나면 EPG는 line_b:send(m)으로 표현되는 노드 및 line_a:recv(m)으로 표현되는 노드와 함께 e = (line_b:send(m), line_a:recv(m)) 에지를 추가적으로 포함하게 된다. EPG에서 line_b:send(m) 노드는 노드 16과 노드 20 사이에 위치하게 되고,

line_a:recv(m) 노드는 노드 14와 노드 17 사이에 위치하게 된다. 이 경우에 $\Delta_m[M]$은 다음과 같이 계산된다. $\Delta_m[M]$ = {1,2,3,4,5,6,7,8,9,10,11,15,16} x {17,18,19,20,21, 22,23,24,25,26,27,28,29,30,31,32}. 또한 cutset C_e는 에지 (16,20), (15,18), (11,17)을 포함한다. 그러면 $\Delta_{U_m}[M]$ = $B(G_M/\{16\})$ x $F(G_M/\{20\})$ ∪ $B(G_M/\{15\})$ x $F(G_M/\{18\})$ ∪ $B(G_M/\{11\})$ x $F(G_M/\{17\})$으로 계산되고, 따라서 $\Delta_{A_m}[M]$ = $\Delta_m[M]$ - $\Delta_{U_m}[M]$ = {(15,17),(16,17),(16, 18),(16,19)} 이다.

위에서 본 것과 같이 집합 $\Delta_{A_m}[M]$은 새롭게 추가된 이벤트들인 ti:send(m)과 tj:recv(m)으로 인해 발생하는 선후 관계를 고려하지 않는다. $\Delta_{NA_m}[M]$을 MSC M에 메시지 m을 추가한 후에 새롭게 생성된 선후 관계들의 집합이라 하자. 그러면 $\Delta_{NA_m}[M]$은 다음과 같이 정의될 수 있다.

$$\begin{aligned}\Delta_{NA_m}[M] = {} & \{(\text{ti:send(m)}, \text{tj:recv(m)})\} \\ & \cup \{(a, \text{ti:send(m)}) \mid a \in \Delta_{B_m}[M]\} \\ & \cup \{(a, \text{tj:recv(m)}) \mid a \in \Delta_{B_m}[M]\} \\ & \cup \{(\text{tj:recv(m)}, a) \mid a \in \Delta_{F_m}[M]\} \\ & \cup \{(\text{ti:send(m)}, a) \mid a \in \Delta_{F_m}[M]\}\end{aligned}$$

예를 들어, 위 예의 경우에 대해서 $\Delta_{NA_m}[M]$은 {(line_b:send(m), line_a:recv(m)), (16, line_b:send(m)), (line_a:recv(m), 17), (line_b:send(m), 20), (14, line_a:recv(m))} 이다. 또한, 다음으로 설명하게 될 $\Delta_{ND_m}[M]$은 {(16,20), (14,17)} 이다.

한편 이와는 반대로 메시지 m이 MSC M으로부터 삭제될 때에는 새로운 선후 관계가 형성되지는 않고 메시지 m과 관련되었던 기존의 관계들이 삭제될 것이다. 에지 (ti:send(m), tj:recv(m))으로 표현되는 선후 관계가 그래프 G_M으로부터 삭제되는 경우를 고려하자. 결과적으로, 이 선후 관계와 관련된 어떤 이벤트들은 위 (A2)에서 묘사된 것처럼 병행적인 관계가 될 것이다. 메시지를 추가하는 경우와 마찬가지로 메시지 m을 삭제함으로써 영향받는 순서 제약조건들은 다음과 같이 계산될 수 있다:

$$\Delta_{D_m}[M] = \Delta_m[M] - \Delta_{U_m}[M]$$

비록 $\Delta_{A_m}[M]$과 $\Delta_{D_m}[M]$의 계산은 동일한 방법으로 이루어지지만 $\Delta_{A_m}[M]$은 각각이 선후 관계를 갖는 이벤트 쌍들로 구성된다. 그러나 $\Delta_{D_m}[M]$은 각각이 병행적인 이벤트 쌍들로 구성된다. 물론 이러한 병행적인 이벤트 쌍들도 이전에는 선후 관계를 가졌으나 메시지 m을 삭제함으로써 병행적인 관계로 바뀌게 된 것이다. 예를 들어, 그래프 G_M으로부터 에지 (trunk:send(busy_tone), line_b:recv(busy_tone))가 삭제되는 경우를 고려하자. 이 경우에 $\Delta_m[M]$과 $\Delta_{U_m}[M]$은 메시지 m을 추가하는 경우와 동일한 방법으로 계산될 수 있다. 즉, $\Delta_m[M]$ = {1, ... ,24} x {29, ... ,32}이고 여기서 cutset은 {(20,29), (24,30)}이며 $\Delta_{U_m}[M]$ = $B(G_M/\{20\})$ x $F(G_M/\{29\})$ ∪ $B(G_M/\{24\})$ x $F(G_M/\{30\})$이다. 따라서 $\Delta_{D_m}[M]$ = {(21,29), (22,29), (23,29), (24,29)} 이다.

우리는 또한 메시지 m을 삭제함으로써 더 이상 유효하지 않은 선후 관계를 갖는 이벤트 쌍들의 집합을 정의할 수 있다. 이 집합은 $\Delta_{ND_m}[M]$으로 표현된다. 메시지 m의 삭제는 그것과 관련된 ti:send(m)과 tj:recv(m) 이벤트들을 해당하는 그래프 G_M으로부터 제거하기 때문에 이들 두 이벤트들과 관련되어 삭제되는 선후 관계 집합은 $\Delta_{NA_m}[M]$의 경우와 정확하게 동일하다.

비록 두 집합 $\Delta_{NA_m}[M]$과 $\Delta_{ND_m}[M]$은 동일하지만 그들의 의미는 상당히 다르다. $\Delta_{NA_m}[M]$의 각 이벤트 쌍은 수정된 MSC에서만 존재하는 새롭게 생성된 선후 관계를 나타내지만 $\Delta_{ND_m}[M]$의 각 이벤트 쌍은 원래의 MSC에서는 존재했으나 수정된 MSC에서는 더 이상 존재하지 않는 선후 관계를 나타낸다. 따라서, 두 집합 $\Delta_{NA_m}[M]$과 $\Delta_{ND_m}[M]$은 정형적으로 다음과 같이 특징지어질 수 있다:

(A3) 만일 (a,b) ∈ $\Delta_{NA}[M']$이면 a ∉ E(M) 또는 b ∉ E(M)이지만 a $\rightarrow_{M'}$ b이다.

(A4) 만일 (a,b) ∈ $\Delta_{ND}[M']$이면 a $\rightarrow_M$ b 이거나 b $\rightarrow_M$ a이지만 a ∉ E(M') 또는 b ∉ E(M')이다.

3.3 복합 변경에 대한 변경 영향 분석

실제적이기 위해서는 MSC에 가해지는 복합적인 변경의 효과를 고려해야만 한다. 복합적인 변경이란 메시지 삽입 연산 혹은 삭제 연산 같은 기본 변경 연산들의 연속된 순서를 말한다. 우리는 앞 절에서 기본 변경 연산이 MSC 명세에 가해진 후에 반드시 재테스팅 되어야 할 순서 제약조건을 파악하기 위한 방법에 관하여 논의하였다. 비록 복합 변경 연산은 기본 변경 연산들의 연속된 순서로 변환될 수 있지만 기본 변경 연산이 적용될 때마다 테스팅을 수행하기보다는 차라리 복합 변경 연산에 대해서 한 번 수행하는 것이 낫다. 게다가 복

합 변경 중에 나중에 적용되는 기본 변경 연산은 앞서 적용된 기본 변경 연산의 효과를 상쇄시킬 수도 있기 때문이다. 따라서, 우리는 복합 변경 연산 후에 각 기본 변경 연산으로부터 영향받는 순서 제약조건을 구성하는 방법과 반드시 재테스팅 되어야 할 제약조건을 결정하기 위한 방법을 제안한다.

앞 절에서 기술한 것처럼 두 종류의 기본 변경 연산이 존재하는데 α_m으로 표현되는 메시지 m의 추가 연산과 δ_m으로 표현되는 메시지 m의 삭제 연산이 있다. 계속해서, n개의 기본 변경 연산을 통하여 원래의 MSC M을 최종적인 MSC M^*로 변환시켜 주는 다음과 같은 복합 변경 연산을 가정한다.

$$M = M_0 \rightarrow^{\gamma_{m_1}} M_1 \rightarrow^{\gamma_{m_2}} \ldots \rightarrow^{\gamma_{m_n}} M_n = M^*$$

변경 순서에서 M_i는$(0 < i \leq n)$ 하나의 기본 변경 연산 γ_{m_i}를 M_{i-1}에 적용한 후에 수정된 MSC를 나타낸다. 여기서 $\gamma_{m_i} \in \{\alpha_{m_i}, \delta_{m_i}\}$이다.

복합 변경의 효과를 알아보기 위하여 n = 2, $\gamma_{m_1} = \alpha_{m_1}$, $\gamma_{m_2} = \delta_{m_2}$ 인 복합 변경을 고려하자. 먼저 $\Delta_{A_{m_1}}[M_0]$와 $\Delta_{D_{m_2}}[M_1]$ 사이의 관계를 고려하자. 첫 번째 기본 변경 연산 α_{m_1}은 변경 전에 병행적이던 M_0의 어떤 이벤트들 사이의 관계를 선후 관계로 바꾸어 준다. 한편, 나중에 적용되는 기본 변경 연산 δ_{m_2}는 α_{m_1}에 의해 도입된 선후 관계 중 일부를 무효화시키는 역할을 한다. 동일한 이유로 $\Delta_{A_{m_1}}[M_1]$은 두 개의 병행적인 이벤트를 나타내는 $\Delta_{D_{m_2}}[M_0]$의 이벤트 쌍 중 일부에 대해 선후 관계로 다시 관련지음으로써 앞서 파악된 순서 제약조건인 $\Delta_{D_{m_2}}[M_0]$의 일부를 무효화 할 수 있다. 또한 동일한 상황은 $\Delta_{NA_{m_1}}[M_0]$(또는 $\Delta_{ND_{m_1}}[M_0]$)와 $\Delta_{ND_{m_2}}[M_1]$(또는 $\Delta_{NA_{m_2}}[M_1]$) 사이에도 존재함을 알 수 있다.

이러한 고려를 바탕으로 복합 변경에 의해 영향받는 순서 제약조건들의 집합을 귀납적인 방법을 통해 정의할 수 있다. 여기서, $\Delta_A[M_i]$와 $\Delta_D[M_i]$는 i번째 기본 변경 연산을 적용한 후에 각각 (A1)과 (A2)를 만족하는 영향받는 순서 제약조건들을 포함한다. 다음의 공식에서 $\lfloor S \rfloor_M = \{(a,b) \mid (a,b) \in S, a,b \in E(M)\}$이다.

(P1) $\Delta_A[M_i] = \Delta_D[M_i] = \emptyset$, if i = 0

(P2) $\Delta_A[M_i] = \lfloor (\Delta_A[M_{i-1}] - \Delta_{D_{m_i}}[M_{i-1}]) \cup (\Delta_{A_{m_i}}[M_{i-1}] - \Delta_D[M_{i-1}]) \rfloor_{M_i}$ $\Delta_D[M_i] = \lfloor \lfloor (\Delta_D[M_{i-1}] - \Delta_{A_{m_i}}[M_{i-1}]) \cup (\Delta_{D_{m_i}}[M_{i-1}] - \Delta_A[M_{i-1}]) \rfloor_{M_i} \rfloor_{M_0}$

if $1 \leq i \leq n$

우리는 $\Delta_A[M_i]$와 $\Delta_D[M_i]$의 계산이 순서에 독립적이라는 것을 쉽게 관찰할 수 있다. 순서 독립이라는 것은 M_0로부터 M_n으로 변환시키는 임의의 변경 순서에 대해서도 항상 동일한 계산 결과를 산출함을 의미한다. 이 사실을 바탕으로 원래의 변경 순서를 동일한 MSC를 생성하는 보다 작은 순서로 변환함으로써 계산 과정을 줄일 수 있다.

앞 절에서 우리는 기본 변경 연산에 의해 도입된 새로운 이벤트와 관련된 순서 제약조건을 계산하기 위하여 $\Delta_{NA_m}[M]$을 정의하였다. 또한 기본 변경 연산에 의해 MSC로부터 어떤 이벤트를 제거함으로써 무효화되는 순서 제약조건을 파악하기 위하여 $\Delta_{ND_m}[M]$을 정의하였다. 비슷하게, 복합 변경에 의해 새롭게 추가되는 이벤트나 삭제되는 이벤트와 관련된 순서 제약조건에 대한 두 집합 $\Delta_{NA}[M_i]$와 $\Delta_{ND}[M_i]$가 필요하다. 이들은 다음과 같이 계산된다:

(P3) $\Delta_{NA}[M_i] = \Delta_{ND}[M_i] = \emptyset$, if i = 0

(P4) $\Delta_{NA}[M_i] = (\Delta_{NA}[M_{i-1}] - \Delta_{ND_m}[M_{i-1}]) \cup (\Delta_{NA_m}[M_{i-1}] - \Delta_{ND}[M_{i-1}])$

$\Delta_D[M_i] = (\Delta_{ND}[M_{i-1}] - \Delta_{NA_m}[M_{i-1}]) \cup (\Delta_{ND_m}[M_{i-1}] - \Delta_{NA}[M_{i-1}])$

if $1 \leq i \leq n$

끝으로 동일한 메시지가 동일한 위치에 차례로 삽입되었다가 삭제되는 또는 그 역의 상황을 고려해야 한다. 이 상황에서는 두 개의 기본 변경 연산으로 인한 어떠한 영향도 없어야 한다. 그러나 다른 기본 변경 연산이 그들 사이에 끼여들게 되는 경우에는 약간의 잘못된 이벤트 쌍이 도입될 수 있다. 따라서 이들은 $\Delta_A[M^*]$에서 제거해야만 한다. $\Delta_D[M^*]$의 경우에 있어서는 플로어링(flooring) 연산 때문에 그러한 문제가 발생하지 않는다. 따라서 $\Delta_A[M^*]$와 $\Delta_D[M^*]$의 최종적인 결과는 각각 다음과 같이 정의된다:

$\Delta_A[M^*] = \Delta_A[M_i] - X$

$\Delta_D[M^*] = \Delta_D[M_i]$

여기서 $X = \{(a,b) \mid (a,b) \in \Delta_A[M_i] \wedge a \rightarrow^*_{M_i} x \wedge y \rightarrow^*_{M_i} b$ for $(x,y) \in S\}$이고 $S = \bigcup_{k=1}^{n} \Delta_{NA}[M_k] \cap \bigcup_{k=1}^{n} \Delta_{ND}[M_k]$이다.

위에서 언급한 바와 같이 하나의 MSC 명세에 가해

지는 변경은 하나 이상의 기본 변경 연산으로 이루어진

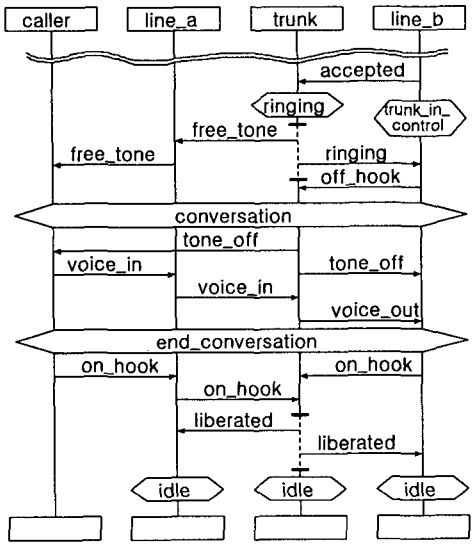

그림 4 복합 변경 연산 후의 MSC

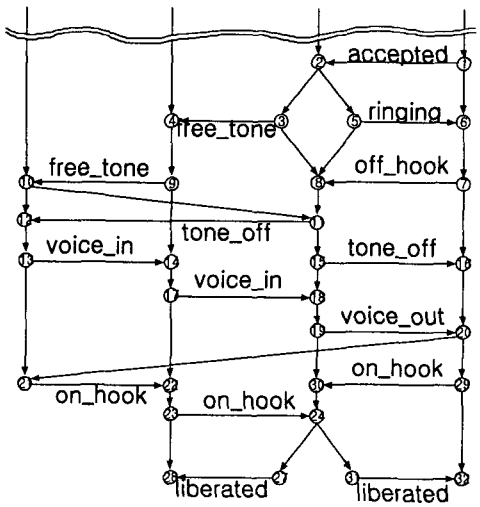

그림 5 그림 4의 MSC에 대한 EPG

다. 예를 들어, 전화 통화 시나리오를 보여주는 그림1과 4의 두 MSC를 고려해 보자. 그림 1의 MSC에서는 호출자(caller)가 먼저 전화를 끊고 나면 피호출자(line_b)가 비지톤(busy_tone)을 듣고 난 후에 전화를 끊는 상황을 묘사하고 있다. 그러나 실제적으로 그들 중 누구라도 먼저 통화를 끊을 수 있기 때문에 호출자가 피호출자에 앞서서 통화를 끊을 필요가 없다. 이러한 시나리오는 그림 4의 MSC에 반영되어 표현되었다. 그림 4에서는 호출자와 피호출자가 동시에 통화를 끊을 수 있다. 이러한 작은 변경은 여러 개의 기본 변경 연산을 필요로 한다.

그림 1에서 4로의 변경은 다음과 같은 5개의 기본 변경 연산을 통해 이루어진다.

1) line_a로부터 trunk로의 on_hook 메시지 삭제
2) line_a로부터 trunk로의 on_hook 메시지 추가
3) trunk로부터 line_a로의 liberated 메시지 삭제
4) trunk로부터 line_a로의 liberated 메시지 추가
5) trunk로부터 line_b로의 busy_tone 메시지 삭제

물론 이들은 임의의 순서로 그림 1의 MSC에 가해질 수 있다. 그렇지만 비록 어떠한 변경 순서가 적용될 지라도 모든 변경이 가해지고 난 후에 수정된 MSC는 항상 동일한 형태와 동일한 선후 관계를 갖는다. 다섯 개의 기본 변경 연산이 적용된 후의 결과는 다음 표와 같다.

i	$\Delta_A[M_i]$	$\Delta_B[M_i]$
1	∅	{(21,29),(22,29),(23,29),(24,29)}
2	∅	{(16,27),(20,27),(21,27),(22,27),(21,29),(22,29),(21,30),(22,30),(21,31),(21,32),(22,31),(22,32)}
3	{(28,31),(28,32)}	{(16,27),(20,27),(21,27),(22,27),(21,29),(22,29),(21,30),(22,30)}
4	{(28,31),(28,32)}	{(21,29),(22,29),(21,30),(22,30)}
5	{(28,31),(28,32)}	{(21,29),(22,29),(21,30),(22,30)}
M^*	∅	{(21,29),(22,29),(21,30),(22,30)}

i	$\Delta_{NA}[M_i]$	$\Delta_{ND}[M_i]$
1	{(20,29),(24,27)}	{(24,25),(25,26),(26,29),(20,26),(25,27)}
2	{(22,28),(19,27)}	{(22,23),(23,24),(24,27),(23,28),(19,24)}
3	{(28,23),(23,24),(24,31),(30,24)}	{(30,31)}
4	{(19,30),(22,23)}	{(19,27),(27,28),(28,23),(22,28),(27,30)}
5	{(24,27),(27,28),(23,28)}	∅
M^*	{(24,27),(19,30),(20,29),(24,31),(30,24)}	{(19,24),(20,26),(24,25),(25,26),(26,29),(25,27),(27,30),(30,31)}

3.4 영향받는 제약조건을 기반으로 한 테스팅

영향받는 순서 제약조건 $\Delta_A[M]$과 $\Delta_D[M]$, 추가된 제약조건 $\Delta_{NA}[M]$과 무효화된 제약조건 $\Delta_{ND}[M]$을 계산한 후에, 수정된 병행 프로그램이 수정된 MSC 명세에 순응하는지를 이들 제약조건을 통해 테스트 할 필요가 있다. 비결정적 테스팅(nondeterministic testing)과 결정적 테스팅의 조합 또는 결정적 테스팅(deterministic testing)을 통해 각 제약조건들을 어떻게 커버(cover)할 것인지에 대한 자세한 사항은 아래에서 다루게 될 것이다.

테스팅 기법을 자세히 기술하기에 앞서서 테스팅 단계에서 $\Delta_A[M]$, $\Delta_D[M]$, $\Delta_{NA}[M]$ 그리고 $\Delta_{ND}[M]$의 의미에 관하여 논의할 필요가 있다. 집합 $\Delta_A[M]$과 $\Delta_{NA}[M]$은 a $\rightarrow_M$ b 형태의 제약조건으로 구성된다. 이 제약조건은 두 이벤트가 나타날 때는 항상 이벤트 a가 이벤트 b에 선행함을 의미한다. 한편, $\Delta_D[M]$의 각 이벤트 쌍은 두 이벤트가 병행적이 됨을 의미한다. (c,d)를 $\Delta_D[M]$의 제약조건이라 할 때 이 제약조건은 프로그램 P의 어떤 수행에서는 이벤트 c가 d를 선행할 수 있고 다른 수행에서는 d가 c를 선행할 수 있음을 나타낸다. 끝으로, $\Delta_{ND}[M]$은 무효화된 제약조건들을 포함한다. 이 제약조건들은 P의 수행 중에는 결코 나타나지 말아야 한다.

명세로부터 추출된 순서 제약조건을 갖는 병행 프로그램의 테스팅에는 제약조건 중심의(constraints-oriented) 접근 방법과 트레이스 중심의(trace-oriented) 접근 방법이 있다. 제약조건 중심의 접근 방법에서 테스트 기준(criteria)은 개별적인 제약조건의 커버리지(coverage) 위에서 정의된다[1]. 만일 a와 b 두 이벤트 모두가 수행 중에 발생하고 a가 b보다 먼저 발생한다면 개별적인 제약조건 (a,b)는 P의 수행에 의해 커버되었다고 말하여 진다. 비슷하게 만일 수행 중에 b가 a를 선행한다면 P의 수행은 제약조건 (a,b)를 위반한다. 이러한 틀 구조에서 네 개의 집합 $\Delta_A[M]$, $\Delta_D[M]$, $\Delta_{NA}[M]$, 그리고 $\Delta_{ND}[M]$의 커버리지는 다음과 같이 정의될 수 있다.

- $\Delta_A[M]$과 $\Delta_{NA}[M]$: (a,b) $\in$ $\Delta_A[M] \cup \Delta_{NA}[M]$를 만족하는 각 순서 제약조건은 P의 수행 중 적어도 한 번 커버된다. 게다가 v가 a를 포함하지 않는 유효하고 가능한 P의 순서 제약조건일 때는 언제나 v.b는 불가능하다.
- $\Delta_D[M]$: (a,b) $\in$ $\Delta_D[M]$인 각 순서 제약조건에 대해 P의 어떤 수행에서 a는 b에 앞서 발생한다. 또한 그 역도 가능하다.
- $\Delta_{ND}[M]$: $\Delta_{ND}[M]$의 이벤트 쌍은 P의 수행 중에 결코 발생하지 않는다.

a $\rightarrow_M$ b로 기술된 순서를 보존하는 시퀀스 s가 있을 때, 만일 수집된 이벤트 시퀀스 중 적어도 하나가 s를 부 시퀀스로서 갖는다면 P의 비결정적 테스팅은 제약조건 a $\rightarrow_M$ b를 커버할 수 있다. 예를 들어, 이벤트 시퀀스 (e_1,e_2,e_3,e_4,e_5)는 그것의 부 시퀀스 중 하나인 (e_3,e_4,e_5)가 e_3와 e_5 사이의 선후 관계를 보존하기 때문에 $e_3 \rightarrow_M e_5$를 커버한다. 그러나 비결정적 테스팅은 모든 가능한 실행 시퀀스를 시행하지는 못한다. 더욱이 우리는 비결정적 테스팅을 통해 주어진 이벤트 시퀀스의 비실현성을 보여 줄 수 없다[1]. 결정적 테스팅을 통해 비결정적 테스팅의 약점을 보완할 수 있다[1]. 사실, 우리는 주어진 이벤트 시퀀스의 비실현성뿐만 아니라 a $\rightarrow_M$ b의 커버리지를 보이기 위해서 결정적 테스팅을 수행할 수 있다.

비록 제약조건 중심의 접근 방법이 영향받는 시퀀스를 하나씩 조사하는데는 유용하지만 명세와 프로그램간의 일치성을 보장하지는 못한다. 게다가 제약조건 중심의 접근 방법은 모든 유효한 시퀀스는 주어진 명세로부터 유도될 수 있다는 관점에서 명세의 완벽성을 가정한다. 이러한 가정을 기반으로 한 테스팅 기법은 모든 유효한 시퀀스는 가능할 뿐만 아니라 모든 가능한 시퀀스도 또한 유효하다라는 것을 보장함으로써 명세와 프로그램간의 동치 관계를 보이려 한다.

트레이스 중심의 접근 방법은 개별적인 제약조건보다는 차라리 실행 트레이스의 관점에서 명세와 프로그램간의 일치성을 보장하려 한다. 이 접근 방법은 명세와 프로그램간의 다양한 순응 관계(conformance relations)를 수용할 것이다[6].

비록 트레이스 중심의 테스팅 방법과 관련된 선행 연구들의 대부분은 동치 순응 관계에 집중되었지만[4,7,8,9], 실제적으로 명세와 프로그램간의 동치 관계에 바탕을 둔 티스팅 기법을 적용하기 어려운 상황이 많이 있다. 첫째, 병행 프로그램 테스팅을 위해 부분 명세(partial specifications)가 사용될 때, 우리는 가능한 시퀀스를 유효하지 않은 것으로 표시할 수 없다. 왜냐하면 부분 명세로부터 모든 유효한 시퀀스를 유도하는 것이 불가능하기 때문이다. 거기에 덧붙여 병행 프로그램에 대한 명세 언어는 이벤트들 사이의 순서 제약조건을 기술하기 위해 대개 부분 순서(partial order) 의미론을 갖는다. 동치 관계를 기반으로 한 앞선 테스팅 프레임워크는

부분 순서 집합으로부터 유도된 모든 전 순서(totally ordered) 시퀀스가 프로그램 수행 중에 관찰되어야 함을 요구한다. 이런 접근 방법에서 하나의 중요한 문제점은 설계 결정 과정에 따라 전 순서 시퀀스의 단지 부분 집합만이 대응되는 구현에서 실현될 수 있다는 것이다. 이 경우에 비록 모든 유효한 시퀀스의 실현성은 만족되지 않을지라도 타겟 프로그램은 여전히 명세를 만족한다.

MSC의 부분적이고 비결정적인 특성을 극복하기 위해 우리는 의도에 따라 명세에서 두 타입의 비결정성을 고려한다[6]. 하나는 명세에 기술된 대로 정확히 프로그램으로 구현될 것을 위해 의도된 것이고 다른 하나는 명세의 간편함을 위해 단지 도입된 것이다. 예를 들어, 그림 1의 두 병행 이벤트 line_a:send(on_hook)과 line_b:send(on_hook)을 고려하자. 이 경우에 호출자가 피호출자에 앞서서 전화를 끊을 수 있고 또한 그 역도 가능하기 때문에 프로그램은 반드시 비결정성을 구현해야 한다. 이런 종류의 비결정성을 우리는 필수적 비결정성(obligatory nondeterminacy)이라 부른다. 이와 비교하여 그림 1의 두 비결정적 이벤트 line_a:recv(free_tone)과 line_b:recv(ringing)은 다른 의도를 갖고 있다. 피호출자가 ringing 시그널을 듣기에 앞서서 호출자가 free_tone 시그널을 듣거나 혹은 그 역의 경우에도 문제가 되지 않는다. 이런 종류의 비결정성은 선택적 비결정성(optional nondeterminacy)이라 불리며 설계 결정 과정 중에 순차화 된다.

이러한 고려 사항을 바탕으로 우리는 부분적이고 비결정적인 명세를 사용하는 병행 프로그램을 테스팅하기 위해 고안된 두 가지 순응 관계를 다음과 같이 정의할 수 있다.

정의 3.1] 병행 프로그램 P의 실행에 의해 수행되는 가능한 이벤트 시퀀스가 MSC M에 대해 유효하기만 하다면 P는 행위적 순응(behavioral conformance) 관계를 갖는다.

정의 3.2] 필수적 비결정성 관계인 병행적인 이벤트 e_1과 e_2가 존재하고, 병행 프로그램 P의 두 가능한 이벤트 시퀀스 s_1과 s_2에 대해서 s_1에서는 이벤트 e_1이 e_2에 앞서고 s_2에서는 이벤트 e_2가 e_1에 앞선다고 할 때 이러한 s_1, s_2가 MSC M에 대해 유효하기만 하다면 P는 M에서 이벤트 쌍 (e_1,e_2)에 대해 비결정적 순응(nondeterminacy conformance) 관계를 갖는다. 비슷하게, P가 M의 모든 필수적 비결정성 관계인 이벤트 쌍에 대해 비결정적 순응 관계를 갖기만 한다면 P는 M에 대해 비결정적 순응 관계를 갖는다.

행위적 관계를 기반으로 한 테스팅은 먼저 $\Delta_A[M] \cup \Delta_{NA}[M]$에 속하는 제약조건 중 일부를 커버하게 될 P의 비결정적 테스팅을 통해 수행될 수 있다. 커버되지 않은 제약조건 (a,b)에 대해서는 유효한 시퀀스 $v.a.\beta.b$를 생성하고 이 시퀀스를 이용하여 P의 결정적 테스팅을 수행한다.

비슷하게 P의 비결정적 실행은 $\Delta_D[M]$에 속하는 제약조건 중 일부를 커버하게 될 것이다. 커버되지 않은 제약조건 (c,d)에 대해서는 두 개의 유효한 이벤트 시퀀스 $v.c.\beta.d$와 $v.d.\beta.c$를 생성하고 이들 시퀀스를 이용하여 P의 결정적 테스팅을 수행한다. 만일 P가 그 시퀀스 중 하나에 대해서 성공한다면, P는 순서 제약조건 (c,d)에 대해서 행위적 순응 관계를 만족한다. 행위적 순응 관계는 가능할 것 같은 두 시퀀스 중 단지 하나만 만족하면 된다. 우리는 이러한 과정을 커버되지 않은 모든 제약조건이 P에 의해 만족될 때까지 반복한다.

$\Delta_A[M]$이나 $\Delta_{NA}[M]$에 속하는 이벤트 쌍들은 순차화되기 때문에 비결정적 순응 테스팅은 단지 $\Delta_D[M]$에 대해서만 적용된다. 행위적 순응 테스팅에서는 비결정적 테스팅에 의해 커버되지 않은 제약조건들은 결정적 테스팅을 통해 강제로 수행된다. 예를 들어, 만일 $\Delta_D[M]$의 이벤트 쌍을 구성하는 e1과 e2 사이의 비결정성이 필수적 관계이고 또한 단지 하나의 순서 제약조건 $e_1 \rightarrow_M e_2$가 P의 비결정성 테스팅에 의해 커버되었다면, (e_1,e_2)에 대해 비결정성 순응 관계를 만족하기 위하여 $v.e_2.\beta.e_1$을 이용하여 P의 결정적 테스팅이 수행될 것이다.

4. 관련 연구

병행 프로그램 테스팅에서 명세 기반 테스팅에 관한 연구는 어떤 수학적 모델을 사용하는지에 따라서 나눌 수 있는데 크게 유한 상태 기계를 이용하는 방법[8]과 도달성 그래프를 이용한 방법[5,10,11] 그리고 순서제약조건을 이용하는 방법[1,6,12,13]으로 나눌 수 있다.

유한 상태 기계를 이용하는 방법은 프로토콜 테스팅 분야에서 많이 연구가 진행되었으며 프로토콜 테스팅에 적합하도록 몇 가지 가정을 내포한다. 지금까지 제시된 테스팅 기법들은 주로 유한 상태 기계와 프로그램간의 동치 관계(equivalence relation)를 검사하는 것을 목표로 하고 있으며 테스팅 대상 컴포넌트의 즉각적인 반작용성(reactive nature)을 가정하고 있다. 또한 테스팅을

통해서 검출하고자 하는 오류의 종류도 출력 오류와 전이 오류(transition errors)로 한정된다. 따라서 일반적인 병행 프로그램에 적용하기 위한 추가적인 연구가 필요하다[8].

도달성 그래프를 이용한 방법에서는 도달성 그래프를 기반으로 다양한 테스트 커버리지를 제공한다. 이 연구들은 도달성 그래프를 순차 수행 프로그램 테스팅에서 사용되던 제어/자료 흐름 그래프의 확장 모형으로 생각하고 테스팅을 수행한다. 따라서 기존의 테스팅 개념을 쉽게 지원한다는 장점이 있지만 병행 프로그램의 도달성 그래프는 잘 알려진 바와 같이 상태수 급증(state explosion) 문제를 안고 있기 때문에 대규모 소프트웨어의 테스팅에 적용하기가 어렵다.

제약조건 기반 테스팅 기법에서는 이벤트들 간의 선후관계를 제약조건으로 표현하고 이를 바탕으로 테스팅을 위한 이벤트 시퀀스를 생성한다. 이 때 이벤트 시퀀스 생성을 위한 여러 가지 테스트 커버리지가 제공된다. 기존의 연구들은 병행 프로그램 테스팅을 위한 순서 제약조건을 기술하기 위해 다른 언어가 정의되었다. 그러한 예 중 하나는 CSPE(Constraints on Succeeding and Preceding Events)라 불리는 제약조건 표기법이 있다[1]. 우리의 방법과 유사하게, CSPE 제약조건 기반 테스팅은 비결정적 및 결정적 테스팅 방법의 조합을 사용한다. 게다가, 비록 가능한 모든 이벤트 쌍을 포함한다면 제약조건은 완전할 수 있지만 이 방법은 제약조건이 완전하지 않은 상황에서도 적용될 수 있다. 그러나, 이 방법은 좀더 효율적인 테스팅을 위해서는 기존의 명세로부터 선후관계를 자동으로 도출해 낼 수 있는 방법이 필요하다. 우리의 방법은 보편적으로 사용되는 명세인 MSC로부터 선후관계를 자동으로 도출하기 때문에 테스팅을 위한 별도의 노력이 극소화되는 장점이 있다.

위에서 본 것과 같이 병행 프로그램 테스팅을 위해 다양한 기법들이 제시되었지만 그러한 기법 대부분은 수정된 명세를 바탕으로 수정된 프로그램을 테스팅하는 방법에 대해서는 언급하고 있지 않다. 우리의 접근 방법과 앞선 테스팅 기법들과의 중요한 차이점은 명세에 가해진 변경을 고려한다는 것이다. 회귀 테스팅의 비용을 줄이기 위해서 명세에 가해진 변경으로 인해 영향받는 순서 제약조건들을 얻기 위한 변경 영향 분석이 시행된다. 추가적으로 커버될 순서 제약조건들은 변경 영향 분석을 통해 자동적으로 유도될 수 있기 때문에 우리의 테스팅 기법[6,12]은 CSPE와 같이 순서 제약조건을 기술하기 위해 고안된 언어를 필요로 하지 않는다.

아울러, 고려 대상이 되는 명세가 부분적이고 비결정적이라 가정되기 때문에 우리는 두 순응 관계, 즉 행위적 순응 관계와 비결정적 순응 관계를 제안하였다. 행위적 순응 관계는 MSC 명세에서 기술된 각각의 유일한 행위에 대해 적어도 하나의 동기화 시퀀스를 구현하는 병행 프로그램을 요구한다. 비결정적 순응 관계는 필수적 비결정성이 대응되는 구현에서 반드시 보존되기를 요구한다.

5. 평가 및 토의

서론에서 설명한 바와 같이 기존의 회귀 테스팅 기법은 대부분 코드를 기반으로 하고 있다. 코드 기반 회귀 테스팅은 코드에 발생한 변경 부분을 빠짐없이 테스팅할 수 있다는 장점이 있는 반면에 명세의 변경이 코드에 모두 반영되었는지를 평가하기에는 부족하다. 따라서 일반적인 테스팅에서와 마찬가지로 코드 기반 회귀 테스팅과 명세 기반 회귀 테스팅은 상호 보완적이다. 기존의 연구가 주로 코드 기반 회귀 테스팅에 집중되었기 때문에 회귀 테스팅에 대한 평가 기준도 코드 기반 회귀 테스팅 기법을 대상으로 제시되고 있다.

Harrold와 Rothermal은 코드 기반 회귀 테스팅 기법이 만족해야 할 네 가지 성질을 제시하고 이를 기반으로 기존의 코드 기반 회귀 테스팅 기법들에 대한 광범위한 비교 평가를 수행하였다[14]. 그들은 회귀 테스팅 기법을 평가하기 위한 기준으로 다음과 같은 네 가지 기준을 제시하였다.

- 포함도(inclusiveness) : 변경 후 프로그램과 변경 전 프로그램에 적용했을 때 서로 다른 결과를 산출하게 만드는 테스트 케이스(소위, 변경검출 테스트 케이스)를 회귀 테스팅 기법이 얼마나 선택하는지에 대한 척도
- 정확도(precision) : 회귀 테스팅 기법이 변경검출 테스트 케이스 이외의 불필요한 테스트 케이스를 선택하지 않는가에 대한 척도
- 효율성(efficiency) : 회귀 테스팅 기법의 계산 비용 및 실제 프로그램에 대한 적용 가능성
- 일반성(generality) : 회귀 테스팅 기법이 실제 프로그래밍 언어에서 나타나는 다양한 문장들을 다룰 수 있는 정도

효율성과 일반성 개념은 각각 테스팅 기법의 성능과 다양한 언어에 대한 적용성을 평가하는 항목이므로 명세 기반 회귀 테스팅 기법에도 동일하게 적용 가능하다. 반면에 포함도와 정확도는 *변경검출 테스트 케이스(modification revealing test cases)*를 검출하는 능력

에 대한 기준이므로 명세 기반 회귀 테스팅 기법에 이 개념들을 적용하려면 "변경검출 테스트 케이스"의 의미를 정확하게 정의해야 한다. 위에서 정의한 포함도와 정확도는 코드 기반 회귀 테스팅 기법을 대상으로 하므로 변경검출 테스트 케이스는 변경 전 프로그램과 변경 후 프로그램에 적용했을 때 서로 다른 결과를 유발하는 테스트 케이스를 의미하게 된다. 즉, 변경검출 테스트 케이스는 *코드의 변경*을 검출할 수 있는 테스트 케이스를 의미한다. 반면에 명세기반 회귀 테스팅에서의 변경검출 테스트 케이스는 *명세의 변경*을 검출하는 테스트 케이스를 의미하게 된다. 명세 변경을 검출할 수 있는 테스트 케이스를 정의하기 위해서 다음과 같은 개념이 필요하다.

순차 수행 프로그램에 대한 테스트 케이스는 하나의 입력 값과 그에 해당하는 기대되는 출력 값의 순서쌍인 (I, O)로 이루어진다. 반면에 병행 프로그램은 일반적으로 외부 환경과 상호 작용하는 특성을 가지기 때문에 테스트 케이스는 외부로부터의 이벤트 시퀀스인 EIS(External Event Sequence)와 그에 해당하는 반응 시퀀스인 ERS(Expected Response Sequence)로 구성된다. 또한 병행 프로그램은 비결정성을 가지기 때문에 하나의 EIS에 대해서 하나 이상의 ERS를 가질 수 있다. 즉, 외부에서 동일한 이벤트 시퀀스가 가해졌을 때 두 가지 이상의 반응 시퀀스가 발생할 수 있다. 예를 들어서, 그림 1의 전화 교환기 시스템에서 가입자 A가 가입자 B의 전화 번호를 눌렀을 때, 가입자 A와 가입자 B는 각각 free_tone과 ringing 시그널을 받게되는데 두 시그널은 병행 관계에 있기 때문에 free_tone 시그널이 ringing 시그널에 비해서 먼저 도착할 수도 있고 반대의 경우가 생길 수도 있다. 병행 프로그램의 테스트 케이스를 (EIS,ERS)의 순서쌍으로 생각하고 비결정성을 고려할 때 명세 변경이 테스트 케이스에 미치는 영향은 다음과 같이 네 가지 경우로 분류할 수 있다.

- **새로운 EIS가 가능해지는 경우** : 그림 1의 명세는 전화를 건 caller가 받는 쪽보다 항상 먼저 전화를 끊는 경우를 보여준다. 반면에 그림 4의 명세에서는 caller와 callee가 전화를 끊는 순서가 정해져 있지 않다. 따라서 그림 1의 명세로부터 그림 4의 명세로 변경이 발생하면 callee가 먼저 전화를 끊는 새로운 이벤트 시퀀스가 가능해진다.
- **기존의 EIS가 불가능해지는 경우** : 역으로 만약에 그림 4의 명세가 원래 명세이고 그림 1의 명세가 변경된 명세라면 callee가 먼저 전화를 끊는 EIS는 이 변경에 의해서 불가능해진다. 이와 같이 명세에 대한 변경은 새로운 EIS를 추가하기도 하고 기존의 EIS를 불가능하게 만들기도 한다.
- **기존의 EIS에 대해서 기존의 ERS가 불가능해지는 경우** : EIS는 변경이 없지만 ERS가 변경되는 경우도 발생할 수 있다. 그림 1의 명세는 free_tone 시그널과 ringing 시그널이 동시에 발생하도록 작성되어 있다. 실제의 경우에는 구현의 편의를 위해서 항상 free_tone 시그널이 먼저 보내지도록 설계 과정에서 결정될 수 있다. 이와 같은 결정에 의해서 명세가 변경되면 EIS 자체는 변화가 없지만 ringing 시그널이 먼저 발생하는 ERS 시퀀스는 더 이상 발생할 수 없게 된다.
- **기존의 EIS에 대해서 새로운 ERS가 가능해지는 경우** : 역의 경우를 생각해 보면 기존의 EIS에 대해서 새로운 ERS가 가능해지는 경우도 발생할 수 있다.

따라서 병행 프로그램에 대한 명세 기반 회귀 테스팅에서 변경검출 테스트 케이스는 위의 네 가지 경우에 해당하는 테스트 케이스들의 집합으로 정의될 수 있다.

이 논문에서 제시한 테스팅 기법은 포함도, 정확도, 효율성, 일반성 측면에서 다음과 같은 특징을 가진다. 먼저, 일반성 측면에서 이 논문에서는 병행 프로그램에 대한 명세로 MSC를 사용하고 있다. 이는 크게 두 가지 요인에 기인하는데 첫째, MSC 명세는 표기법이 쉽고 직관적인 의미와 잘 부합되며 많은 도구들이 지원되기 때문에 병행 프로그램이나 객체 지향 프로그램의 명세에 광범위하게 사용되고 있다[2,3]. 또한 MSC 명세는 이벤트들 간의 순서 제약조건을 표현하기에 매우 적합하다. Tai 등의 연구에 의하면 병행 프로그램에 대한 대부분의 명세는 이벤트간의 순서 제약조건을 포함하고 있다[1,6,12,13]. 따라서 이 논문에서 제시한 기법은 비단 MSC 명세뿐 만 아니라 이벤트간의 순서 제약조건을 포함하고 있는 많은 명세에 적용 가능하다. 즉, 임의의 명세로부터 이벤트간의 순서 제약조건을 추출하고 이를 이용해서 3절에서 제시한 이벤트 선후 그래프를 구축함으로써 논문에서 제시한 회귀 테스팅 기법을 적용할 수 있다.

효율성 측면에서, 이 논문에서는 명세 변경에 의해서 영향받는 순서 제약조건의 집합을 한번에 구성하기보다는 점진적으로(incremental method) 구성하는 방식을 채택하고 있다. 점진적 기법을 사용할 수 있는 근거는 두 가지인데 먼저 앞 절에서 설명한 바와 같이 복합 변경을 기본 변경의 조합으로 생각할 수 있다는 점과 이

때 하나의 복합 변경은 여러 방식으로 조합될 수 있으나 어떤 조합을 택하는지에 상관없이 최종적으로 구해진 순서 제약조건들의 집합은 동일하다는 점이다. 실제 대부분의 명세 변경은 복합 변경이라는 점을 고려해 볼 때 이 논문에서 제시한 알고리즘의 효율성은 복합 변경을 기본 변경의 조합으로 나누는 방법에 크게 영향을 받는다. 복합 변경을 기본 변경의 조합으로 나누는데 있어서 우리는 greedy 관점을 채택했다. 즉, 하나의 복합 변경을 표현할 수 있는 여러 가지 조합들이 있는 경우에 가장 짧은 조합을 채택하는 방식이다. 이 방식은 점진적 알고리즘의 반복 적용 횟수를 줄이는 장점이 있지만 점진적 알고리즘 수행 중에 발생하는 중간 결과물의 크기를 최소화하지는 못한다. 중간 결과물의 크기를 최소화할 수 있는 기본 변경의 조합을 택하는 문제는 향후 이슈로 남아있다. 이 논문에서 제시한 알고리즘의 복잡도를 생각해 보면, 먼저 하나의 기본 변경 연산을 처리하려면 변경과 관련된 각 이벤트의 선행자들의 집합과 후행자들의 집합을 계산해야 하므로 이벤트 선후 그래프에 있는 노드의 개수를 n이라 할 때 기본 변경 연산 하나를 처리하는데 $O(n^2)$의 시간 복잡도가 발생한다. 또한 공간 복잡도 역시 $O(n^2)$이 된다. 따라서 복합 변경 연산의 경우에 하나의 복합 변경 연산이 k개의 기본 변경 연산의 조합으로 이루어진다면 시간 복잡도는 $O(k \times n^2)$이며 공간 복잡도는 중간 결과물을 유지할 필요가 없으므로 $O(n^2)$가 된다.

마지막으로 포함도와 정확도 측면에서 볼 때 이 논문에서는 직접 테스트 케이스를 구하기보다는 변경에 의해서 영향받은 순서 제약조건을 구한 후 이 제약조건들을 커버하는 테스팅 기법을 제시하고 있다. 이 논문에서 구한 네 가지 제약조건의 집합인 $\Delta_{NA}[M]$, $\Delta_{ND}[M]$, $\Delta_{A}[M]$, 그리고 $\Delta_{D}[M]$는 각각 앞에서 얘기한 네 가지 형태의 변경검출 테스트 케이스에 해당한다. 즉, $\Delta_{NA}[M]$는 새로운 EIS를 유발하고, $\Delta_{ND}[M]$는 기존의 EIS를 불가능하게 하며, $\Delta_{A}[M]$는 기존 EIS에 대해서 기존의 ERS를 불가능하게 하고, $\Delta_{D}[M]$는 기존의 EIS에 대해서 새로운 ERS를 가능하게 한다. 따라서 이 연구에서 제시한 회귀 테스팅 기법은 명세 변경에 따른 모든 변경검출 테스트 케이스를 추출하며 그 외의 테스트 케이스는 추출하지 않는 효과적인 방법이다.

6. 결 론

이 논문에서는 메시지 순차도로 작성된 병행 프로그램의 명세로부터 명세 기반 회귀 테스팅을 위한 테스트 케이스를 자동으로 생성하고 생성된 테스트 케이스를 이용해서 병행 프로그램에 대한 재테스팅을 수행하는 방법에 대하여 논의하였다. 명세에 변경이 가해지면 변경으로 인해 영향받는 이벤트들간의 순서 제약조건을 추출하는 것은 대단히 중요하다. 본 연구에서는 회귀 테스팅의 비용을 줄이기 위해서 명세에 가해진 변경으로 인해 영향받는 순서 제약조건을 파악하고 단지 영향받은 순서 제약조건에 한해서만 수정된 프로그램에 대한 테스팅을 수행하는 점진적 회귀 테스팅 방법을 제안하였다.

또한 이 논문에서는 기존의 코드 기반 회귀 테스팅을 비교하기 위해서 제시되었던 네 가지 기준들을 명세 기반 회귀 테스팅에 적용하는 방법을 설명하였고 이 기준들에 의거해서 여기서 제시한 기법을 평가하였다. 이 논문에서 제시한 병행 프로그램에 대한 명세 기반 회귀 테스팅 기법은 크게 다음과 같은 독창성과 장점을 가진다. 첫째, 기존의 기법들은 대부분 회귀 테스팅을 위한 정보를 코드로부터 추출하였지만 여기서는 명세로부터 추출하였다. 코드 기반 회귀 테스팅 기법과 명세 기반 회귀 테스팅 기법은 상호 보완적인 특성을 가지므로 이 논문에서 제시한 기법을 사용함으로써 기존의 회귀 테스팅 기법들로 찾아내기 어려운 변경된 명세와 (변경된) 프로그램간의 합치성을 효과적으로 검사할 수 있다. 둘째, 앞 절에서 평가한 바와 같이 이 논문에서 제시한 회귀 테스팅 기법은 네 가지 기준에서 볼 때 우수하다. 변경에 의해서 영향받은 순서 제약조건만을 찾아내서 재테스팅하기 때문에 포함도와 정확도가 뛰어나고 복합 변경의 경우에 기본 변경으로 나누어서 점진적인 기법을 사용하기 때문에 효율성이 뛰어나다. 그리고 순서 제약조건에 기반을 둔 테스팅 기법을 사용하였기 때문에 MSC 이외의 많은 명세에도 적용 가능하다.

참 고 문 헌

[1] R. H. Carver and K. C. Tai, "Use of Sequencing Constraints for Specification-Based Testing of Concurrent Programs," *IEEE Trans. on Software Eng.*, Vol. 24, No. 6, pp. 471-490, June 1998.

[2] E. Rudolph, J. Grabowski, and P. Graubmann, "Tutorial on Message Sequence Charts (MSC'96)," *Tutorial of the FORTE/PSTV'96 conf.*, October 1996.

[3] ITU-T Recommendation Z.120: Message Sequence Chart (MSC), April 1996.

[4] G. H. Hwang, K. C. Tai, and T. L. Huang, "Reachability Testing : An Approach to Testing Concurrent Software," *Proc. of Asia Pacific*

Software Eng. Conf. 1994, pp. 246-255, 1994.
[5] E. Itoh, Y. Kawaguchi, Z. Furukawa, and K. Ushijima, "Ordered Sequence Testing Criteria for Concurrent Programs and the Support Tool," *Proc. of Asia Pacific Software Eng. Conf. 1994*, pp. 236-245, 1994.
[6] I. S. Chung, H. S. Kim, H. S. Bae, and Y. R. Kwon, "Testing of Concurrent Programs based on Message Sequence Charts," *Proc. of Int'l Symp. on Soft. Eng. for Parallel and Distributed Systems*, pp. 72-82, 1999.
[7] H. AboElFotoh, O. Abou-Rabia, and H. Ural, "A Test Generation Algorithm for Systems Modelled as Nondeterministic FSMs," *IEE Software Eng. Journal*, pp. 184-188, July 1993.
[8] G. v. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing," *Proc. of Int'l Symp. on Software Testing and Analysis*, pp. 109-124, 1994.
[9] S. K. Damodaran-Kamal and J. M. Francioni, "Testing Races in Parallel Programs with an OtOt Strategy," *Proc. of Int'l Symp. on Software Testing and Analysis*, pp. 216-227, 1994.
[10] P. V. Koppol and K. C. Tai, "An Incremental Approach to Structural Testing of Concurrent Software," *Proc. of Int'l Symp. on Software Testing and Analysis*, pp. 14-23, 1996.
[11] R. N. Taylor, D. L. Levine, and C. D. Kelly, "Structural Testing of Concurrent Programs," *IEEE Trans. on Soft. Eng.*, Vol. 8, No. 3, pp. 206-215, March 1992.
[12] I. S. Chung, H. S. Kim, H. S. Bae, and Y. R. Kwon, "Testing of Concurrent Programs After Specification Changes," *Proc. of Int'l Conf. on Software Maintenance*, pp. 199-208, 1999.
[13] K. C. Tai and R. H. Carver, "A Specification-Based Methodology for Testing Concurrent Programs," *Proc. of European Software Eng. Conf.*, pp. 154-172, 1995.
[14] G. Rothermel and M. J. Harrold, "Analyzing Regression Test Selection Techniques," *IEEE Trans. on Software Eng.*, Vol. 22, No. 8, pp. 529-551, August 1996.

김 현 수
정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조

정 인 상
정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조

배 현 섭
정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조

권 용 래
정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조

이 동 길
정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조