MSC 명세에 기반한 병렬 프로그램의 프로세스 간 테스팅

(Inter-Process Testing of Parallel Programs based on Message Sequence Charts Specifications)

배 현 섭 † 정 인 상 †† 김 현 수 ††† 권 용 래 ††††

(Hyun Seop Bae) (In Sang Chung) (Hyeon Soo Kim) (Yong Rae Kwon)

정 영 식 ††††† 이 병 선 †††††

(Young Sik Chung) (Byung Sun Lee)

요 약 병렬 프로그램 테스팅을 위한 기존의 연구는 대부분 프로그램 수행 중에 얻어진 이벤트 트레이스(event trace)를 바탕으로 재수행성을 보장하는데 중점을 두고 있다. 반면에 개발과정에서 만들어진 요구/설계 명세로부터 테스팅을 위한 이벤트 시퀀스를 생성하는 방법에 대한 연구는 빈약한 실정이다. 이 논문에서는 통신 소프트웨어 개발 분야에서 광범위하게 사용되는 메시지 순차도(MSC)로부터 병렬 프로그램의 모듈 테스팅을 위한 이벤트 시퀀스를 생성하는 방법을 제시한다. 명세로부터 이벤트 시퀀스를 생성하기 위해서는 명세 내에 묵시적으로 포함되어 있는 이벤트들과 그들 간의 선후관계를 파악해야 한다. 이를 위해서 이 연구에서는 프로그램 수행 중에 이벤트들의 발생 순서를 결정하기 위해 사용해오던 논리시간 벡터(logical time stamp)를 MSC 명세에 적용함으로써 이벤트 간의 선후관계를 추출한다. 또한 이를 바탕으로 이벤트 시퀀스를 자동 생성하는 방법을 제시하고 전화 통화 예제를 사용해서 제시한 방법의 효용성을 보인다.

Abstract Most of prior works on testing parallel programs have concentrated on how to guarantee the reproducibility by employing event traces exercised during executions of a program. Consequently, little work has been done to generate meaningful event sequences, especially, from specifications. This paper describes techniques for deriving event sequences from Message Sequence Charts(MSCs) which are widely used in telecommunication areas for its simplicity in specifying the behaviors of a program. For deriving event sequences from MSCs, we have to uncover the causality relations among events embedded implicitly in MSCs. In order to attain this goal, we adapt vector time stamping which has been previously used to determine the ordering of events taken place during an execution of interacting processes. Then, valid event sequences, satisfying the causality relations, are generated according to the interleaving rules suggested in this paper. The feasibility of our testing technique was investigated using the phone conversation example. In addition, we discussed on the experimental results gained from the example and how to combine various test criteria into our testing environment.

· 이 논문은 전자통신연구원 위탁 과제에 의해서 일부 지원되었음

† 정 회 원 : 한국전자통신연구원 연구원
hsbae@etri.re.kr
†† 종신회원 : 한성대학교 정보전산학부 교수
insang@hansung.ac.kr
††† 정 회 원 : 금오공과대학교 컴퓨터공학부 교수
hskim@cespc1.kumoh.ac.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
kwon@salmosa.kaist.ac.kr
††††† 종신회원 : 한국전자통신연구원 연구원
yschung@etri.re.kr
bslee@etri.re.kr

논문접수 : 1998년 5월 13일
심사완료 : 1999년 8월 25일

1. 서 론

일반적으로 병렬 프로그램은 **비결정성**을 내포하고 있기 때문에 동일한 입력 자료에 대해서 반복 수행시 서로 다른 결과를 유발할 수 있다[1]. 이와 같은 비결정적 특징 때문에 병렬 프로그램은 순차 수행 프로그램과 달리 테스팅 및 검증 단계에서 다음과 같은 두가지 어려움에 부딪힌다. 첫째, 한 입력 자료에 대해서 주어진 프로그램을 한 번 수행해보는 것만으로는 오류 여부를 판단하기 어렵다. 순차 수행 프로그램은 동일한 입력에 대해서 항상 동일한 출력을 발생하기 때문에 선택된 입력 자료에 대해서 주어진 프로그램을 한번씩만 수행해 보면 오류 여부를 판단할 수 있다. 그러나 비결정적 병렬 프로그램의 경우에는 주어진 입력 자료에 대해서 프로그램을 수행 했을때 옳은 결과를 출력했다 할지라도 다시 수행했을 때는 잘못된 결과를 낼 수 있다. 둘째, 오류가 발생했을 때 그 오류의 원인을 찾기가 쉽지 않다. 병렬 프로그램 수행시 한번 발생한 오류를 반복적으로 다시 관찰하기가 어렵기 때문에 오류의 원인을 찾기가 어려워진다.

이와 같은 문제점을 해결하기 위해서 기존의 병렬 프로그램 테스팅에 대한 연구에서는 테스트 케이스를 (입력 자료, 프로그램 수행 중에 발생하는 이벤트의 트레이스)로 정의하고, 프로그램의 **재수행성**을 보장하는데 중점을 두고 있다. 즉, 프로그램을 처음 수행 했을때 발생하는 이벤트 트레이스를 기록했다가 이를 이용해서 수행 결과를 이후에 재현해내는 방법에 대한 연구가 많이 진행되었다[1,2]. 이런 방법은 재수행을 위한 이벤트 트레이스를 명세보다는 실제 프로그램의 수행으로부터 얻으므로 **프로그램 기반 테스팅**으로 분류될 수 있다.

반면에 병렬 프로그램에 대한 **명세 기반 테스팅**에 대한 연구는 많이 진행되지 않았다. 명세는 병렬 프로그램이 수행 중에 만족해야 할 순서 제약조건(sequencing constraints)을 포함하고 있는데 이런 제약조건을 기반으로 이벤트 시퀀스[1]를 생성할 수 있다. 명세를 S라 하고 S로부터 생성된 이벤트 시퀀스들의 집합을 $ES(S)$라 하면, 병렬 프로그램 P는 명세 S에 대해서 다음과 같은 두가지 측면에서 검사될 수 있다[3]. 첫째, P의 **정당성(validity)**을 검사할 수 있다. P를 수행했을 때 발생하는 모든 이벤트 트레이스의 집합 R을 $ES(S)$와 비교해서 $R \subset ES(S)$이면 "P는 S에 대해서 정당하다"고 한다. 둘째, $ES(S)$의 **실현성(feasibility)**을 검사할 수 있다. $ES(S)$ 내에 있는 각 이벤트 시퀀스들을 따르도록 프로그램 P를 강제 수행함으로써 명세에서 허용한 시퀀스들이 실제 프로그램에 구현되었는지를 검사할 수 있다. 이상과 같은 고찰에서 나타나듯이 주어진 명세로부터 이벤트 시퀀스들을 생성하는 것은 프로그램의 정당성이나 명세의 실현성을 검사하는데 매우 중요한 역할을 한다. 따라서 이 논문에서는 주어진 명세로부터 병렬 프로그램의 테스팅을 위한 이벤트 시퀀스들을 생성하는 방법을 제시한다.

이벤트 시퀀스 생성을 위한 순서 제약조건(sequencing constraints)은 TSL(Task Sequencing Language)이나 CSPE(Constraints on Succeeding and Preceding Events) 등과 같이 제약조건 기술을 위해서 특별히 제안된 언어를 이용해서 기술될 수 있다[3,4]. 그러나 이런 방법은 테스팅을 위해서 별도의 명세를 작성해야 한다는 단점이 있다. 따라서 여기서는 일반적으로 널리 사용되는 명세로부터 제약조건을 추출하고 그 제약조건으로부터 이벤트 시퀀스를 생성하는 방법을 제시한다. 이 방법은 테스팅을 위한 별도의 명세가 필요하지 않으며 프로그램 개발 과정에서 산출된 명세를 테스팅 단계에 그대로 적용할 수 있다는 장점이 있다. 반면에 대다수의 명세 기법들이 이벤트 간의 순서 관계를 명시적으로 표현하지 않기 때문에 이 방법을 실제로 적용하기 위해서는 주어진 명세로부터 이벤트 간의 인과 관계나 순서 관계를 자동으로 도출해낼수 있는 기술이 필요하다.

또한 여기서 제시한 이벤트 시퀀스 생성 기법은 모듈 테스팅을 지원한다. 병렬 프로그램을 테스팅 할때 프로그램을 구성하는 모든 프로세스들을 동시에 테스팅 하기보다는 그 중에 일부분을 독립적으로 테스팅하는 경우가 자주 발생한다. 즉, 지정된 프로세스 집합 I가 나머지 프로세스들의 집합 X와 어떻게 상호작용 하는지를 테스팅 해야할 필요가 있다. 이 경우에 I와 X 간의 상호작용에 연관된 이벤트들에 대해서만 순서 제약조건을 추출함으로써 주어진 모듈 I에 대한 효과적인 테스팅이 가능하다.

이 논문에서는 메시지 순차도(Message Sequence Charts, MSC)로 작성된 명세로부터 **병렬 프로그램의 명세 기반 모듈 테스팅**을 위한 이벤트 시퀀스를 생성하는 방법을 제시한다. 메시지 순차도는 병렬/분산 프로그램의 (선택적인) 행동을 보여주기 위해서 사용되는 명세 언어로서 그래픽 표현을 지원하며 의미가 비교적 단순하고 명확하기 때문에 널리 사용된다[5]. 또한 많은

1) 용어의 혼동을 피하기 위해 프로그램 수행을 통해서 얻은 이벤트 리스트를 이벤트 트레이스라 하고 명세로부터 생성된 이벤트 리스트를 이벤트 시퀀스라 하도록 한다.

상업적인 지원도구들이 개발되어 있어서 통신 분야의 소프트웨어 개발에 실제적으로 사용되고 있다.

앞에서도 설명한 바와 같이 일반적인 명세로부터 이벤트 시퀀스를 생성하기 위해서는 이벤트들 간의 순서 제약조건을 추출하는 것이 매우 중요하다. 병렬 프로그램의 수행시 발생하는 이벤트들 간에는 부분 순서관계(partial order relation)가 존재한다. 이와 같은 부분 순서관계를 추출하기 위해서 우리는 Fidge가 제시한 논리시간 벡터(logical time stamp) 개념을 명세에 적용한다[6]. 원래 논리시간 벡터는 프로그램 수행 중에 이벤트들의 발생 순서를 판단하기 위해서 제안되었는데 이 논문에서는 명세 내의 이벤트 선후관계를 파악하는데 적용한다. 논리시간 벡터를 이용해서 MSC 명세내에 있는 이벤트들 간의 순서관계를 검출하고 나서 이를 바탕으로 이벤트 시퀀스를 생성하며 생성된 이벤트 시퀀스는 프로그램의 정당성과 명세의 실현성 검사를 위해서 사용된다.

이 논문의 구성은 다음과 같다. 먼저 2절에서는 MSC에 대해서 간략하게 기술하고 논문 전반에 걸쳐서 사용될 예제 명세를 설명한다. 3절에서는 논리시간 벡터를 MSC 명세에 적용해서 이벤트들의 순서관계를 추출하는 방법과 추출된 순서관계로부터 이벤트 시퀀스를 생성하는 방법을 제시한 후 예제를 통해서 그 과정을 보이고 4절에서는 실험적인 결과를 설명하고 기존의 테스트 케이스 선정 기준을 이 방법에 도입하는 방안에 관해서 토의한다. 5절에서는 관련 연구에 대해서 간략하게 검토하고 마지막으로 6절에서 결론을 맺고 향후 연구 방향을 설정한다.

2. 메시지 순차도(Message Sequence Charts)

MSC는 통신 시스템과 같은 분산/병렬 시스템의 부분적인 행동을 기술할 수 있는 그래픽 언어로서 ITU의 Z.120에 의해서 표준화되어 있다[5]. 현재 많은 분야에서 MSC 혹은 유사한 형태의 시나리오 명세 언어가 SDL 등과 함께 시스템의 요구 명세를 기술하기 위해서 사용되고 있다. MSC 명세 언어가 널리 사용되는 가장 중요한 원인은 그래픽 표기법이 직관적으로 이해하기 쉬우며 시스템의 부분적인 행동을 기술할 수 있기 때문이다.

MSC는 기본적으로 시스템에 포함된 병렬 컴포넌트(프로세스)들과 주변 환경 간의 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 요소들이고 구조요소는 다수의 메시지 순차도 간의 계층구조를 표현하기 위해서 사용되는 요소들이다.

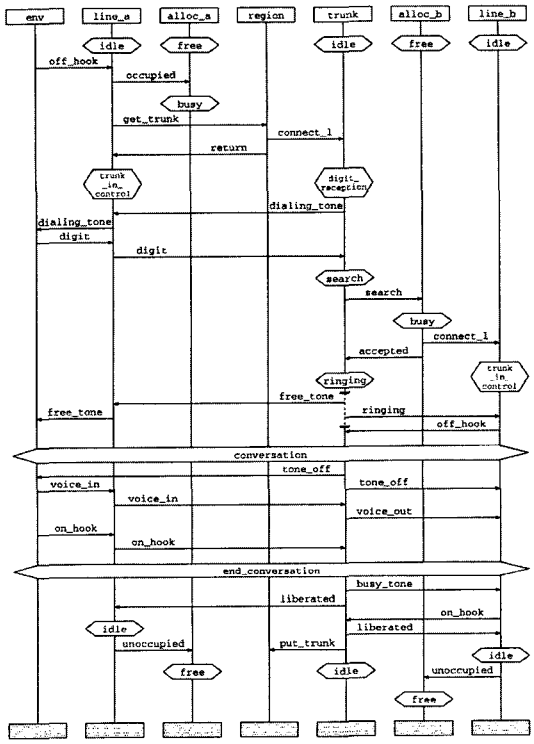

그림 1 전화 통화를 표현한 메시지 순차도의 예

그림 1은 전화 통화 과정을 보여주는 예제로서 표준 MSC가 제공하는 대부분의 기본요소들을 포함하고 있다. 그림에는 *env, line_a, alloc_a, region, trunk, alloc_b, line_b*의 일곱개 프로세스가 포함되어 있는데 각각은 상단에 프로세스의 이름이 적혀져 있는 수직선으로 나타나 있다. 프로세스를 가로지르는 육각형은 수행 중에 프로세스가 만족해야 하는 조건을 표현한다. 예를 들어서, 전화 통화 시나리오가 시작되는 시점에서 프로세스 *line_a*는 *idle* 상태에 있어야만 한다. *conversation*이나 *end_conversation*과 같이 두개 이상의 프로세스에 걸쳐있는 조건은 공유조건을 의미한다. 두 프로세스를 연결하는 수평 화살표들은 메시지 전달에 의한 통신을 의미한다. 그림 1은 전화 통화 시나리오가 *env* 프로세스로부터 *line_a* 프로세스로 *off_hook* 메시지가 전달됨으로써 시작됨을 보여준다. 마지막으로, 수직

점선으로 나타난 병행영역(coregion)이 존재한다. 한 프로세스 내에 포함된 이벤트들은 수직선 상의 위치에 따라 순서를 가지지만 병행영역에 포함된 이벤트들은 순서를 가지지 않는다. 예를 들어서, 프로세스 *line_a*는 순서 관계에 의해서 *off_hook*를 받기 전까지는 *occupied* 메시지를 보낼수 없지만, 프로세스 *trunk*는 *free_tone* 메시지와 *ringing* 메시지를 임의의 순서로 보낼 수 있다.

3. 순서 제약조건 추출 및 이벤트 시퀀스 생성

이 절에서는 MSC에 논리시간 벡터를 적용해서 순서 제약조건을 추출하는 방법과 제약조건으로부터 이벤트 시퀀스를 생성하는 방법을 설명한다.

3.1 이벤트 시퀀스 생성 절차

이벤트 시퀀스는 생성은 다음과 같은 네단계를 거친다.

(1) 주어진 MSC 명세에 포함된 프로세스들을 두개의 그룹으로 나눈다. 테스팅 대상이 되는 프로세스들의 그룹을 I라하고 그 외의 프로세스들의 그룹을 X라 한다.

(2) 프로세스 그룹 I 내에 속한 이벤트들의 순서관계를 파악하기 위해서 각 이벤트에 논리시간 벡터를 할당한다. 마찬가지로 프로세스 그룹 X 내에 속한 이벤트들에 대해서도 논리시간 벡터를 할당한다. 이 과정을 **순서화 과정**이라하며 3.2 절에서 설명된다. 각각의 그룹이 독립적으로 순서화 됨에 유의해야 한다.

(3) 각 프로세스 그룹의 내부 이벤트들을 제거한다. 모듈 테스팅의 견지에서 볼때 우리는 프로세스 그룹 I와 X 사이의 상호작용에 대해서만 테스팅하게 되므로 이벤트 시퀀스 생성에서도 각 프로세스 그룹의 외부 이벤트만 의미를 가진다. 따라서 내부 이벤트들은 제거한다. 이 과정을 **축약 과정**이라하며 3.3 절에서 설명된다.

(4) 논리시간 벡터가 할당된 후 축약된 MSC로부터 이벤트 시퀀스를 생성한다. 이 과정에서는 두 프로세스 그룹 I와 X 간에 발생 가능한 통신 패턴별로 이벤트 시퀀스들을 생성한다. 이벤트 시퀀스 생성을 위한 규칙은 3.4 절에서 제시된다.

3.2 순차화 과정: 논리시간 벡터 할당

순차화 과정은 주어진 MSC 내의 프로세스들을 서로 소인 두개의 프로세스 그룹 I와 X로 분할한 후에 수행된다. 주어진 MSC에 n개의 프로세스들이 존재하고 이 중에 k개의 프로세스 $p_1,\ldots,p_k$를 테스팅하는 경우라면 $I=\{p_1,\ldots,p_k\}$이고 $X=\{p_{k+1},\ldots,p_n\}$이다. 그림 2의 MSC는 그림 1에 나타난 MSC를 두개의 그룹 $I=\{alloc_a, region, trunk, alloc_b\}$와 $X=\{env, \text{line}_a, \text{line}_b\}$로 나누고 논리시간 벡터를 할당한 결과이다.

순차화 과정은 프로세스 그룹 내의 각 이벤트에 대해서 논리시간 벡터를 할당하고 이를 이용해서 이벤트 간의 선후관계를 파악하는 것을 목적으로 한다. 이벤트에 논리시간 벡터를 할당하는 방법은 MSC의 기본구성요소에 따라서 다섯 가지의 규칙으로 설명될 수 있다. 이 규칙들을 설명하기 전에 이벤트들을 구분하기 위해서 다음과 같이 각 이벤트에 이름을 붙이는 방법을 생각하자.

* $RECV(x, p_i)$는 프로세스 p_i에서 메시지 x를 받는 이벤트를 뜻한다.

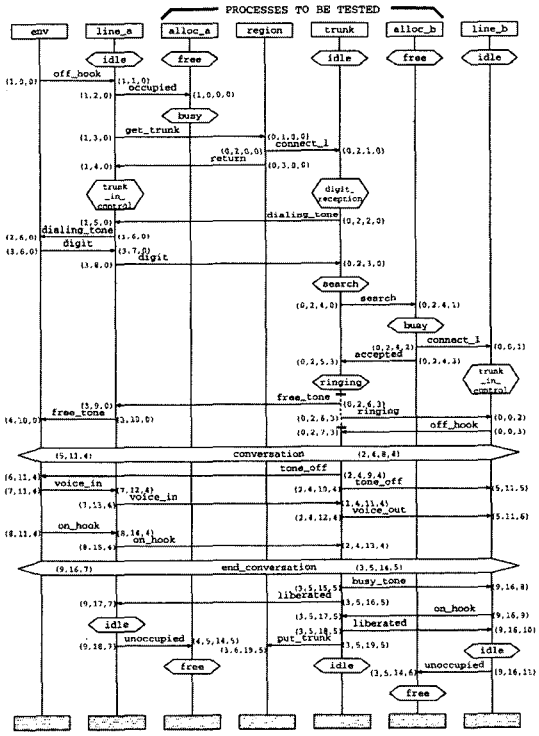

그림 2 전화 통화 예제에 논리시간 벡터를 할당한 결과

* $SEND(x, p_i)$는 프로세스 p_i에서 메시지 x를 보내는 이벤트를 뜻한다.

2절에서 설명한 것처럼 MSC는 프로세스와 그들 간의 메시지 전송 및 공유 조건, 그리고 병행 영역으로 구성된다. 따라서 MSC 내에 포함된 이벤트들 간의 순서관계를 파악하려면 MSC의 구성요소인 프로세스, 메시지 송/수신 이벤트, 공유 조건 등에 대해서 논리 시간 벡터를 할당해야 한다. 다음의 규칙 1과 5는 각 프로세

스에 대한 논리 시간 벡터 할당 규칙이며 규칙 2는 한 프로세스 내에 포함된 이벤트들 간의 선후 관계를 표현하기 위한 규칙이다. 또 규칙 3은 메시지 송/수신 이벤트 간의 선후 관계를 나타내고 규칙 4는 공유 조건에 의한 선후 관계를 표현한다.

***규칙 1 (초기화 규칙)** : 각 프로세스는 초기에 영 벡터를 논리시간 벡터로 가진다. 즉, 프로세스 p_i의 현재 논리시간 벡터를 T_i라 하면 초기에 $T_i=(0,\ldots,0)$이다. 벡터 T_i의 차원(dimension)은 p_i가 속한 프로세스 그룹 내의 프로세스 갯수와 같다. 예를 들어서, 그림 2에서 X는 세 개의 프로세스를 포함하고 I는 네개의 프로세스를 포함하므로 *env*와 *trunk*는 각각 3차원과 4차원의 시간 벡터를 가진다.

***규칙 2 (상속 규칙)** : 기본적으로 각 이벤트는 자신이 속한 프로세스 내의 선행 이벤트로부터 시간 벡터를 상속 받는다. 이 규칙은 이벤트가 병행영역에 있는지 여부에 의해서 영향 받는다. T_{ij}를 프로세스 p_i의 j번째 이벤트가 가지는 시간 벡터라 하자.

- **규칙 2-1 (정상 상속)** : 프로세스 p_i의 j번째 이벤트와 $j+1$번째 이벤트가 둘 다 병행영역에 있지 않은 경우에 $T_{ij+1}=T_{ij}+i$번째 단위벡터이다. 다시 말해서, 병행영역에 있지 않은 이벤트는 자신의 선행 이벤트로부터 시간 벡터를 상속받은 후 i번째 행의 값을 하나 증가시켜서 자신의 시간 벡터를 만든다.
- **규칙 2-2 (병행영역 상속)** : 프로세스 p_i의 $j+1$번째,...,m번째 이벤트가 병행영역에 있는 이벤트라고 가정하고 j번째 이벤트가 병행영역에 포함되지 않은 마지막 이벤트라고 하자. 그러면 병행 영역에 포함된 이벤트들의 시간벡터는 $\forall x\in\{j+1,\ldots,m\}$, $T_{ix}=T_{ij}+i$번째단위벡터이다. 즉, 병행영역 내의 모든 이벤트들은 동일한 논리시간 벡터를 가지는데 이 벡터는 병행영역 직전의 이벤트로부터 시간벡터를 상속받은 후 i번째 행의 값을 하나 증가시켜서 만든다. 그림 2의 *trunk* 프로세스 내에 있는 병행영역을 고려해보자. 두개의 메시지 송신 이벤트는 동일한 시간 벡터 (0,2,6,3)을 가지며 이는 *accepted* 이벤트로부터 상속받은 것이다.

***규칙 3 (통신 규칙)** : 두개의 프로세스가 메시지를 주고 받는 경우에 그들 간에는 묵시적인 선후관계가 형성되며 이 선후관계는 다음과 같은 규칙에 의해서 표현될 수 있다.

- **규칙 3-1 (송신 규칙)** : 프로세스 p_i의 j번째 이벤트가 송신 이벤트인 경우에 이 이벤트의 시간 벡터는 규칙 2를 이용해서 계산된후 수신 프로세스 쪽으로 전달된다.
- **규칙 3-2 (수신 규칙)** : 프로세스 p_m의 n번째 이벤트가 수신 이벤트인 경우에 T_{mn}은 규칙 2를 통해서 계산된 자기 자신의 시간 벡터와 규칙 3-1을 통해서 송신 프로세스로부터 전달받은 시간 벡터를 행별로 최대값을 취함으로써 만들어진다. 예를 들어서, 그림 2의 프로세스 *env*에서 *dialing_tone* 메시지를 수신하는 이벤트를 고려해보자. 이 이벤트의 시간 벡터 (2,6,0)은 *PAIRWISE_MAX {(2,0,0),(1,6,0)}*을 통해서 만들어진다. 이때 (2,0,0)은 *SEND(off_hook, env)* 이벤트로부터 상속받은 시간 벡터이고 (1,6,0)은 *SEND(dialing_tone, line_a)* 이벤트로부터 전달받은 시간 벡터이다.

이상과 같은 기본요소 외에 시스템의 상태를 표현하기 위해서 사용되는 조건문과 프로세스 생성 및 소멸에 대한 고려가 필요하다. MSC에서 사용되는 조건문은 한 프로세스의 상태를 표현하기 위한 지역 조건문과 두개 이상의 프로세스들이 동시에 만족하는 상태를 표현하기 위한 공유 조건문으로 나뉜다. 특히 공유 조건문은 프로세스들 간의 묵시적 동기화를 표현하게 되므로 이를 논리시간 벡터에 반영해야 한다. 또한 MSC에서는 프로세스의 생성과 소멸을 표현할 수 있다. 프로세스의 동적 생성이나 소멸이 허용되는 경우에 프로세스의 수가 가변적이므로 시간 벡터의 차원을 결정하기가 어렵게된다. 이 문제는 "순서쌍의 집합"과 같은 자료 구조를 통해서 해결할 수 있으며 여기서는 문제를 용이하게 하기 위해서 프로세스의 개수는 미리 알려져 있다고 가정한다. 이 경우에 프로세스 소멸에 대한 별도의 규칙은 필요치 않으며 다만 프로세스 생성을 처리할 수 있는 규칙이 필요하다. 다음의 규칙 4와 규칙 5는 각각 공유 조건문과 프로세스 생성을 처리하기 위한 규칙들이다.

***규칙 4 (공유 조건 규칙)** : 하나의 조건문을 공유하는 프로세스 $(p_i,\ldots,p_l)$들은 각 시간 벡터들의 최대값을 각자의 시간 벡터로 가진다. 즉, $\forall x\in\{j,\ldots,l\}$, $T_{xc}=PAIRWISE_MAX\{T_j,\ldots,T_l\}$이며 여기서 T_{xc}는 공유 조건문에 대한 시간 벡터이다. 예를 들어서, 공유 조건 *conversation*은 두 개의 시간 벡터 (5,11,4)와 (2,4,8,4)를 가지는데 전자는 X 그룹 내에 있는 프로세스들에 대한 시간 벡터이고 후자는 I 그룹 내의 프로세스들에 대한 것이다. 여기서 시간 벡터 (5,11,4)는

PAIRWISE_MAX{(5,10,0),(3,11,0),(0,0,4)}로부터 계산되어진다.

***규칙 5 (프로세스 생성 규칙)** : 프로세스 p_i가 프로세스 p_j를 생성하는 경우에 p_j는 p_i의 현재 시간 벡터를 취한다. 즉, $T_j = T_i$이다.

이상과 같은 시간벡터 할당 규칙에 덧붙여서 이벤트들 간의 선후관계를 비교하기 위한 비교 규칙이 필요하다. 이벤트들 간의 선후관계는 '→'로 표현되며 다음의 규칙을 이용해서 판단될 수 있다.

***비교 규칙** : 프로세스 p_i내의 이벤트 a와 프로세스 p_j내의 이벤트 b 간의 선후 관계는 다음과 같이 결정된다. $a \rightarrow b \Leftrightarrow (T_{ia}(i) \le T_{jb}(i)) \wedge (T_{ia}(j) < T_{jb}(j))$ 여기서 T(k)는 시간 벡터 T의 k번째 행의 값을 의미한다.

3.3 축약 과정: 내부 이벤트 제거

순차화 과정을 거친후 수행되는 축약과정에서는 각 프로세스 그룹의 내부 이벤트들을 제거한다. 모듈 테스팅은 선택된 서브 시스템(혹은 모듈) S_1이 명세에 기술된 요구사항을 만족하는지를 검사하는 과정이다. 따라서 병렬 프로그램의 모듈 테스팅에서는 테스팅하고자 하는 프로세스 그룹 I를 구현하고 있는 서브 시스템 S_1의 외부 행동(external behavior)이 명세에 나타난 I와 X 간의 상호작용 순서와 합치하는지 여부만을 검사하게 된다. 따라서 테스트 시퀀스에는 I와 X 간의 상호작용에 관한 이벤트만 포함되고 상호 작용에 무관한 이벤트들은 축약 과정에서 모두 제거된다. 프로세스 그룹 I와 X 간의 상호 작용을 테스팅할 때 축약 과정을 거친 후에 남게 되는 이벤트들은 다음과 같다.

$$\begin{aligned} ExternalEvent(I,X) = \; & \{RECV(x,p_i) \mid p_i \in I \wedge x \text{ is sent by } p_j \in X\} \\ & \cup \{RECV(x,p_i) \mid p_i \in X \wedge x \text{ is sent by } p_j \in I\} \\ & \cup \{SEND(x,p_i) \mid p_i \in I \wedge x \text{ is received by } p_j \in X\} \\ & \cup \{SEND(x,p_i) \mid p_i \in X \wedge x \text{ is received by } p_j \in I\} \end{aligned}$$

그림 3은 전화 통화 예제 MSC에 대해서 축약 과정을 거친 후의 결과를 보여준다. 그림에서 나타난 바와 같이 각 프로세스 그룹의 내부 이벤트들은 모두 제거되지만 제거되지 않은 이벤트들의 시간 벡터는 그대로 보존된다. 따라서 내부 이벤트를 제거한 후에도 제거되지 않은 이벤트들 간의 선후관계는 명확하게 남아있다.

시스템 전체를 테스팅하고자 할 경우, 즉 I = {모든 프로세스}인 경우를 고려해보자. 이 경우에 모든 이벤트가 내부 이벤트가 되므로 모든 이벤트가 제거되고 결국 테스트 시퀀스는 하나도 생성되지 않는다. 이와 같은 결과는 이 논문에서 가정하는 MSC가 닫혀있는 시스템(closed system)을 모델링하고 있기 때문이다. 닫혀있는 시스템은 MSC에 나타나 있지 않은 외부 환경이나 시스템과의 상호작용이 없는 시스템을 의미한다. 그러나 실제로 많은 병렬 소프트웨어들 특히 통신 소프트웨어들은 외부 환경과의 상호 작용을 가지기 때문에 닫혀있지 않은 경우가 많다. 이와 같이 닫혀 있지 않은 소프트웨어 시스템 전체를 테스팅하기 위해서는 그림 1의 *env* 프로세스처럼 MSC 내에 외부 환경의 역할을 하는 프로세스를 하나 포함시키고 I = {모든 프로세스}/{외부 환경 프로세스}, X = {외부 환경 프로세스}로 지정하면 된다. 즉, 외부 환경 자체를 하나의 프로세스로 모델링하는 방법을 사용한다.

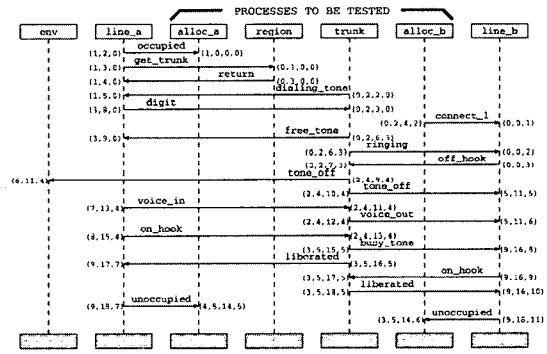

그림 3 전화 통화 예제의 축약 결과

3.4 이벤트 시퀀스 생성 규칙

축약이 끝난 MSC로부터 이벤트 시퀀스를 생성하는 기본 규칙은 논리시간 벡터에 나타난 부분 순서관계를 만족하는 모든 교차 시퀀스(interleaving sequence)를 생성하는 것이다. 교차 규칙은 프로세스 그룹 I와 X 간의 통신 패턴에 따라서 달라진다. 교차 규칙을 설명하기에 앞서 다음과 같은 기호들을 정의하자.

프로세스 그룹 G 내의 모든 이벤트들의 집합을 $E(G)$라 하자. 여기서 $G \in \{I, X\}$이다. 내부 이벤트가 축약된 상태이기 때문에 $RECV(x,p_i) \in E(X)$이면 $SEND(x,p_j) \in E(I)$이고 그 역도 성립한다. 임의의 이벤트 $e \in E(G)$에 대해서 대응 이벤트는 $e^{'}$로 표현한다. 이벤트 집합 $E(G)$는 송신 이벤트들 만을 포함하는 집합 '*![E(G)]*'와 수신 이벤트들 만을 포함하는 집합 '*?[E(G)]*'의 두 개의 부분집합으로 분할될 수 있다. 그림 3에서 !$[E(X)]$는 집합 *{SEND(occupied, line_a), SEND(get_trunk, line_a), SEND(digit, line_a), SEND(voice_in, line_a), SEND(on_hook, line_a), SEND(uno*

ccupied, line_a), SEND(off_hook, line_b), SEND(on_hook, line_b), SEND(unoccupied, line_b)} 이다. 이벤트 시퀀스를 기술하기 위해서 우리는 이벤트들을 ';' 기호로 연결하며 이 기호는 이벤트들이 순차적으로 연결됨을 의미한다. 이상과 같은 기호를 이용해서 다음의 규칙들은 프로세스 그룹 X가 프로세스 그룹 I를 시뮬레이션 한다는 관점에서 교차 규칙을 기술한다.

***생성 규칙 1 (병행 송신)** : 두 이벤트 $e_1, e_2 \in ![E(X)]$에 대해서 $e_1 \not\to e_2$이고 $e_2 \not\to e_1$이면 e_1과 e_2는 두 개의 이벤트 시퀀스로 교차 된다.

***생성 규칙 2 (병행 수신)** : 두 이벤트 $e_1, e_2 \in ?[E(X)]$에 대해서 $e_1 \not\to e_2$이고 $e_2 \not\to e_1$이면 e_1과 e_2는 두 개의 이벤트 시퀀스로 교차 된다.

***생성 규칙 3 (순차 송신)** : 두 이벤트 $e_1, e_2 \in ![E(X)]$에 대해서 $e_1 \to e_2$이면 e_1과 e_2는 하나의 이벤트 시퀀스로 순차화 된다.

***생성 규칙 4 (병행 송신으로부터의 순차 수신)** : 두 이벤트 $e_1, e_2 \in ?[E(X)]$에 대해서 $e_1 \to e_2$이고 두 이벤트 e_1'와 e_2'간에 순서관계가 없으면 (즉, $e_1' \not\to e_2'$이고 $e_2' \not\to e_1'$), e_1과 e_2는 두개의 이벤트 시퀀스로 교차 된다.

***생성 규칙 5 (순차 송신으로부터의 순차 수신)** : 두 이벤트 $e_1, e_2 \in ?[E(X)]$에 대해서 $e_1 \to e_2$이고 $e_1' \to e_2'$이면 e_1과 e_2는 하나의 이벤트 시퀀스로 순차화 된다.

***생성 규칙 6 (병행 송/수신)** : 두 이벤트 $e_1 \in ![E(X)]$와 $e_2 \in ?[E(X)]$에 대해서 e_1과 e_2 사이에 순서관계가 없으면 e_1과 e_2는 두개의 이벤트 시퀀스로 교차 된다.

***생성 규칙 7 (순차 송/수신)** : 두 이벤트 $e_1 \in ![E(X)]$과 $e_2 \in ?[E(X)]$에 대해서 $e_1 \to e_2$이면 e_1과 e_2는 하나의 이벤트 시퀀스로 순차화 된다. 마찬가지로 $e_2 \in ![E(X)]$과 $e_1 \in ?[E(X)]$에 대해서 $e_2 \to e_1$이면 e_2와 e_1은 하나의 이벤트 시퀀스로 순차화 된다.

전화 통화 예제에는 지금까지 설명한 일곱개의 생성 규칙에 해당하는 경우가 모두 나타나 있다. 다음은 각 생성 규칙이 전화 통화 예제에서 나타난 경우를 보여준다.

규칙 1의 예 : SEND(on_hook, line_b) 와 SEND (unoccupied, line_a)

규칙 2의 예 : RECV(free_tone, line_a) 와 RECV (ringing, line_b)

규칙 3의 예 : SEND(occupied, line_a) 와 SEND (get_trunk, line_a)

규칙 4의 예 : RECV(return, line_a) 와 RECV (dialing_tone, line_a)

규칙 5의 예 : RECV(dialing_tone, line_b) 와 RECV(free_tone, line_a)

규칙 6의 예 : SEND(on_hook, line_a) 와 RECV (voice_out, line_b)

규칙 7의 예 : SEND(on_hook, line_a) 와 RECV (liberated, line_a)

그림 4는 전화 통화 예제에 대한 두개의 예시적인 이벤트 시퀀스를 보여준다. 이 이벤트 시퀀스에는 규칙 1, 2, 4, 6에 해당하는 이벤트들이 교차 되었음을 알 수 있다.

이벤트 시퀀스 1

```
1:  line_a ! occupied to     alloc_a;
2:  line_a ! get_trunk       to     region;
3:  line_a ? return   from   region;
4:  line_a ? dialing_tone    from   trunk;
5:  line_a ! digit           to     trunk;
6:  line_b ? connect_l       from   alloc_b;
7:  line_a ? free_tone       from   trunk;
8:  line_b ? ringing  from   trunk;
9:  line_b ! off_hook to     trunk;
10: line_b ? tone_off from   trunk;
11: env    ? tone_off from   trunk;
12: line_a ! voice_in to     trunk;
13: line_b ? voice_out       from   trunk;
14: line_a ! on_hook  to     trunk;
15: line_b ? busy_tone       from   trunk;
16: line_b ? liberated from  trunk;
17: line_b ! on_hook  to     trunk;
18: line_a ? liberated from  trunk;
19: line_a ! unoccupied      to     alloc_a;
20: line_b ! unoccupied      to     alloc_b;
```

이벤트 시퀀스 2

```
line_a ! occupied  to     alloc_a;
line_a ! get_trunk to     region;
line_a ? dialing_tone     from   trunk;
line_a ? return   from    region;
line_a ! digit            to     trunk;
line_b ? connect_l from   alloc_b;
line_b ? ringing  from    trunk;
line_a ? free_tone from   trunk;
line_b ! off_hook to      trunk;
line_b ? tone_off from    trunk;
env    ? tone_off from    trunk;
line_a ! voice_in to      trunk;
line_a ! on_hook  to      trunk;
line_b ? voice_out from   trunk;
line_b ? busy_tone        from   trunk;
line_b ? liberated from   trunk;
line_a ! unoccupied       to     alloc_a;
line_b ! on_hook  to      trunk;
line_a ? liberated from   trunk;
line_b ! unoccupied       to     alloc_b;
```

그림 4 전화 통화 예제에 대한 두개의 이벤트 시퀀스

4. 구현 및 토의

이 논문에서 제시한 이벤트 시퀀스 생성 방법을 사용하는 병렬 프로그램 테스팅 환경이 구축되었으며 이 테스팅 환경은 MSC를 기반으로 통신 소프트웨어 구현에 많이 사용되는 CHILL 프로그램의 테스팅을 지원한다. 이 절에서는 구현된 테스팅 환경의 효율성과 정확성을 보이기 위해서 수행된 실험의 결과를 기술하고 실험을 통해서 얻은 몇가지 문제에 대해서 고찰한다.

이 연구의 가장 큰 특징 중에 하나는 MSC 명세 내에 포함된 이벤트들간의 순서 관계를 논리 시간 벡터를 이용해서 추출하는 방법이다. 원래 Fidge가 제시한 논리 시간 벡터는 수행 중에 발생하는 이벤트들간의 순서를 정하기 위해서 개발되었기 때문에 MSC의 구성 요소중 일부만을 지원한다. 즉, 병행 프로세스 간의 메시지 송/수신 이벤트와 같은 동적 이벤트에는 잘 적용되지만 공유 조건과 같은 정적 이벤트나 병행 영역과 같은 특수 구조는 고려하지 않고 있다. 이런 문제점을 해결하기 위해서 이 연구에서는 논리 시간 벡터를 생성하기 위한 다섯가지 규칙을 제시했으며 이 규칙들은 메시지 송/수신 이벤트뿐만 아니라 공유 조건, 병행 영역, 프로세스 생성 등 MSC의 모든 구성 요소들을 다루고 있다. 특히, 규칙 4는 공유 조건을 묵시적인 동기화로 해석하고 있는데 현재 MSC의 공유 조건에 대한 정형적 의미가 명확하지 않기 때문에 이러한 해석에 대한 검증이 필요하다.

MSC 내에 포함된 공유 조건을 동기화로 해석하는 것이 자연스러운지를 조사하기 위해서 이 연구에서는 구현된 테스팅 환경을 이용해서 전화 통화 예제를 대상으로 실험을 수행하였다. 실험에서는 공유 조건의 해석 방법이 테스팅에 어떤 영향을 끼치는지를 조사하기 위해서 그림 2에 포함된 두개의 공유 조건 *conversation*과 *end_conversation*을 서로 다르게 해석해서 테스팅을 수행하였다. 첫번째 실험에서는 두 공유 조건을 모두 동기화로 해석하지 않은 경우이고 두번째 실험은 둘 중의 하나만 동기화로 해석한 경우이며 마지막 실험에서는 두 공유 조건을 모두 동기화로 해석했다. 각각의 경우에 대해서 테스트 시퀀스를 생성한 결과가 표 1에 나타나 있다. 이 표에서 보는 바와 같이 공유 조건을 동기화로 해석하면 이벤트 시퀀스의 갯수가 현저하게 줄어듬을 알 수 있다. 이는 프로세스 간의 동기화가 이벤트의 교차 수행을 방지하기 때문이며 동기화가 많을수록 병행성이 저하된다는 일반적인 예측에 부합된다.

표 1 공유 조건의 해석에 따른 전화 통화 예제 테스팅 결과

공유 조건 수	0개	1개	2개
이벤트 시퀀스 수	1160	900	448
수행된 시퀀스 수	224	224	224

이와 같이 생성된 테스트 시퀀스를 가지고 전화 통화 예제를 구현한 CHILL 프로그램을 테스팅 하면 흥미로운 결과를 얻을 수 있다. 이 수행 결과는 각각의 경우에 대해서 CHILL 프로그램을 1000회씩 자유 수행해서 얻은 결과이다. 즉, 수행 중에 CHILL 프로그램의 수행 경로를 전혀 제어하지 않았다. 테스팅 결과는 크게 두가지 의미를 내포하는데 첫째, 표에 나타난 것처럼 세가지 경우에 모두 224개의 시퀀스만이 수행된다. 이때 224개의 시퀀스는 두 공유 조건을 모두 동기화로 해석한 세번째 경우에서 만들어진 448개의 시퀀스에 포함된다. 다시 말해서, 공유 조건을 동기화로 해석하지 않음으로써 추가로 생성된 시퀀스들(첫번째 실험에서는 712개, 두번째 실험에서는 452개)은 프로그램 수행 중에 발생하지 않았다. 이는 MSC 내에 나타난 공유 조건이 실제 프로그램에서 동기화로 구현되고 있음을 보여주는 단적이 예로서 이 논문에서 제시한 논리 시간 벡터의 정당성을 부분적으로 입증한다.

반면에 생성된 시퀀스 중에 수행되지 않는 시퀀스가 있다는 사실은 또 다른 의미를 가진다. 세번째 실험의 경우에 정확하게 절반의 테스트 시퀀스만 수행되었는데 실제 수행된 테스트 시퀀스를 분석해보면 그림 2에서는 두 이벤트 *RECV(tone_off, env)*와 *RECV(tone_off, line_b)*가 병행 관계에 있지만 실제 수행시에는 항상 *RECV(tone_off, env)*가 먼저 수행되었다. 지금까지 이 논문에서는 이벤트 시퀀스를 선택하기 위한 특별한 기준을 사용하지 않고 "가능한 모든 이벤트 시퀀스"를 생성하는 방법만을 설명했다. 이는 하나의 MSC가 시스템의 전체 동작이 아닌 부분적인 동작만을 보여주기 때문에 모든 이벤트 시퀀스를 생성하는 것이 가능하기 때문이다. 그러나 하나의 MSC가 아닌 다수의 MSC가 존재하는 경우에는 가능한 모든 이벤트 시퀀스를 생성하는 것이 현실적으로 불가능해진다. 게다가 실험 결과에서 보듯이 모든 이벤트 시퀀스가 수행가능하지는 않다. 따라서 테스트 커버리지 개념을 도입해서 이벤트 시퀀스의 개수를 줄일 필요가 있다.

병렬 프로그램에 대한 테스트 커버리지는 Taylor 등에 의해서 구조적 커버리지(structural coverage)가 제시된 이후에 다양한 연구가 진행되었다[7,8]. Taylor 등

은 도달성 그래프를 기반으로 순차 수행 프로그램의 구조적 커버리지 기준을 병렬 프로그램으로 확장하였다[8].

순차 수행 프로그램의 제어흐름 그래프에 적용하던 노드 커버리지, 에지 커버리지 등을 도달성 그래프에 적용함으로써 병렬 프로그램 테스팅에서 이벤트 시퀀스 선택을 위한 기준을 제시했다. 한편 순차화 시퀀스 테스팅(ordered sequence testing) 기법에서는 이벤트 시퀀스의 길이를 기준으로 하는 OSC_k 커버리지를 제시하였다[7]. OSC_k 커버리지는 전체 이벤트 시퀀스의 부분 시퀀스에 대한 조건으로서 길이가 k인 모든 부분 이벤트 시퀀스가 교차중에 적어도 한번은 나타나야 함을 의미한다. 이상과 같은 기준들을 프로그램의 구조와 관련된 기준으로서 프로그램 기반 테스팅을 위한 커버리지이다.

현재까지 제시된 다양한 테스트 커버리지를 이 논문에서 제시한 이벤트 시퀀스 생성 과정에 적용하기 위한 하나의 방법으로 여기서는 DAG(Directed Acyclic Graph)를 이용하는 방법을 설명한다. 앞절에서 설명한 바와 같이 MSC 내의 이벤트들은 부분 순서관계를 가지며 이 순서관계는 논리시간 벡터를 통해서 검출된다. 일반적으로 부분 순서관계는 DAG를 이용해서 표현할 수 있으며 이벤트의 순서관계를 그래프로 표현하기만 하면 기존의 테스트 커버리지들을 쉽게 도입할 수 있다. 그림 5는 전화 통화 에제에 대한 DAG를 보여준다. 이 그림에서 노드와 에지는 각각 상태와 트랜지션을 의미하고 에지위에 씌인 숫자는 이벤트를 뜻한다. 이벤트와 숫자 간의 대응관계는 그림 4의 이벤트 시퀀스 1에 나타나 있다.

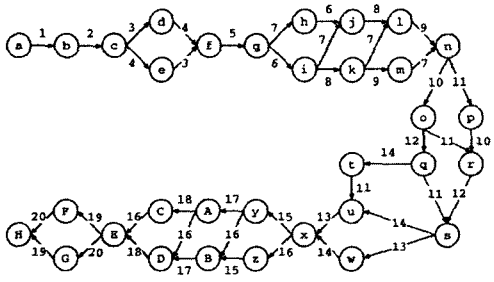

그림 5 전화 통화 예제에 대한 DAG 표현

DAG는 *Enabled_Events(State s)* 함수를 이용해서 자동적으로 구축될 수 있는데 이 함수는 주어진 상태 s에서 수행 가능한 모든 이벤트들의 집합을 구해준다. 예를 들어서, 그림 5에서 *Enabled_Events(c)*는 *{RECV (return, line_a), RECV(dialing_tone, line_a)}*를 리턴한다. 물론 이 함수는 주어진 상태에서 수행 가능한 이벤트들을 구하기 위해서 논리시간 벡터를 사용한다.

DAG는 제어 흐름 그래프의 단순형이라고 볼 수 있으므로 분기 커버리지나 노드 커버리지와 같은 기존의 제어 흐름 그래프 기반 테스트 커버리지를 직접 적용할 수 있다. 마찬가지로 DAG는 도달성 그래프의 일부로 해석할 수도 있으므로 Taylor 등이 제시한 커버리지나 OSC_k 커버리지도 적용 가능하다. 다음의 네가지 이벤트 시퀀스는 그림 5에 나타난 DAG으로부터 분기 커버리지와 노드 커버리지를 만족하는 시퀀스를 추출한 것이다. 동시에 이 이벤트 시퀀스들은 Taylor가 제시한 구조적 커버리지 중에서 *all-edges-between-cc-states*를 만족하며 OSC_2도 만족한다.

1. 1;2;3;4;5;7;6;8;9;11;10;12;13;14;16;15;17;18;20;19
2. 1;2;4;3;5;6;8;9;7;10;12;14;11;13;15;17;18;16;19;20
3. 1;2;3;4;5;6;7;8;9;10;11;12;14;13;15;16;17;18;19;20
4. 1;2;4;3;5;6;8;7;9;10;12;11;14;13;15;17;16;18;20;19

5. 관련연구

병렬 프로그램 테스팅에 대한 현재까지의 연구는 그 목적에 따라서 크게 다음과 같은 세방향으로 나뉠 수 있다. 첫째, 비결정성을 가진 프로그램의 재수행성 보장 기법에 대한 연구가 수행되었다[1,2,9]. 비결정성을 가진 프로그램은 반복 수행이 어려우므로 테스팅 결과에 대한 신뢰도가 떨어지고 오류의 원인을 찾기가 어려워진다. 그러므로 테스팅이나 디버깅 단계의 핵심기술 중에 하나로 재수행성 보장 기술이 필요하다. 이 연구들은 대부분 프로그램 수행을 통해서 얻은 이벤트 트레이스를 강제 수행시켜서 프로그램의 재수행을 보장하므로 테스팅 기법의 관점에서 볼 때 프로그램 기반 테스팅 기법들로 분류될 수 있다.

둘째, 테스트 커버리지에 대한 연구가 수행되었다[3,7,8,10]. 이 연구들은 도달성 그래프나 TSL, CSPE 등과 같이 이벤트들 간의 선후관계가 명시적으로 주어져 있는 경우에 테스팅을 위해서 어떤 이벤트 시퀀스를 선택할지에 대한 기준을 제시한다. 앞에서 지적한 바와 같이 이 방법들은 테스팅을 위한 별도의 명세 과정이 필요하다는 단점을 내포하고 있다. 또한 병렬 프로그램 테스팅을 위해서는 입력 자료와 이벤트 시퀀스가 필요한데 현재까지는 입력 자료와 이벤트 시퀀스를 함께 고려하는 방법이 제시되지 못하고 있는 실정이다. 따라서 독립적인 테스팅 방법으로 사용되기 보다는 다른 테스

팅 방법에 연계되어서 사용되는 것이 바람직하다.

마지막으로, 실제 테스트 케이스를 생성하려는 노력이 있다[11,12,13]. 이 방법은 수행시에 얻어진 이벤트 트레이스를 변형시킴으로써 새로운 이벤트 시퀀스를 생성하는 방법[11,13]과 명세로부터 직접 이벤트 시퀀스를 생성하는 방법[12]으로 나뉘는데 전자의 경우에는 프로그램의 정당성(validity)은 검사할 수 있으나 명세의 실현성(feasibility)을 검사할 수 없다는 문제점이 있다. 또한, 후자의 경우에 명세에 묵시적으로 나타난 이벤트들 간의 선후관계를 파악하는 방법이 없었기 때문에 지금까지는 일부분의 이벤트 시퀀스 만을 생성해 왔다.

MSC는 병렬 프로그램 개발 과정에서 매우 빈번하게 사용되는 명세 언어임에도 불구하고 현재까지 MSC를 기반으로 하는 병렬 프로그램 테스팅 기법은 별로 제시되지 않고 있다. 이는 병렬 프로그램을 테스팅하려면 이벤트들 간의 순서관계를 파악해야 하는데 MSC 내에 포함된 이벤트들 간의 순서관계에 대한 정형적 정의가 없기 때문이다. Grabowski 등이 제시한 테스팅 방법은 MSC로부터 병렬 프로그램 테스팅을 위한 테스트 케이스를 생성한다는 점에서 이 논문에서 제시한 방법과 밀접한 관련이 있다. 그러나 이 방법은 MSC에 포함된 이벤트들 간의 선후관계를 파악하지 않고 테스트 시퀀스를 생성하기 때문에 가능한 모든 시퀀스를 생성하지 못하고 단지 MSC로부터 생성 가능한 일부분의 테스트 시퀀스만을 생성한다. 따라서 명세의 실현성이나 프로그램의 정당성을 검사하는데 한계가 있다. 반면에 이 연구에서 제시한 방법은 확장된 논리 시간 벡터를 이용해서 이벤트들 간의 선후 관계를 추출하기 때문에 테스팅을 위해서 별도의 명세를 작성해야 하는 부담도 없을 뿐만 아니라 MSC로부터 생성 가능한 모든 시퀀스를 만들수 있고 명세의 실현성과 프로그램의 정당성을 검사하는데 사용할 수 있다.

6. 결 론

이 연구는 통신 소프트웨어 구현에 많이 사용되는 CHILL 프로그램에 대한 모듈 단위의 블랙박스 테스팅에 대한 필요성에서 시작되었다. 병렬 프로그램에 대한 모듈 단위의 블랙박스 테스팅에서는 주어진 명세로부터 이벤트들과 그들 간의 선후 관계에 대한 순차 제약조건을 추출하고 이를 기반으로 이벤트 시퀀스를 생성해서 프로그램의 정당성(validity)과 명세의 실현성(feasibility)을 검사한다. 이 논문에서는 논리시간 벡터를 이용해서 MSC로부터 순차 제약조건을 검출하고 이벤트 시퀀스를 자동 생성하는 방법을 제시하였다. 이 방법을 채택한 테스팅 환경이 현재 구현 중에 있으며 이 테스팅 환경에서는 아래와 같은 과정을 통해서 병렬 프로그램 테스팅을 지원한다.

(1) 주어진 병렬 프로그램 P에 대해서 프로세스들을 두개의 그룹 I와 X로 나눈다. 이 때 테스팅 하고자 하는 프로세스들은 I에 속하고 그 외의 프로세스들은 X에 속하게 된다. 여기서는 블랙박스 테스팅을 지원하므로 I의 외부 행동(external behavior)을 검사하기 위한 테스트 케이스가 아래의 과정을 통해서 생성된다.

(2) 프로그램 P에 대응하는 MSC를 (1) 단계에서 나눈것과 동일하게 두부분으로 나누고 3절에서 제시한 방법으로 이벤트 시퀀스를 생성한다.

(3) 생성된 이벤트 시퀀스를 이용해서 테스트 드라이버(test driver)와 테스트 스텁(test stub)을 만든다. 만들어진 드라이버와 스텁은 프로그램 P의 I 부분과 함께 수행되면서 I의 외부 행동에 대한 로그를 기록한다. 이 단계에서는 실제 프로그램을 드라이버 및 스텁과 연동하여 수행해야 하므로 이벤트 시퀀스 외에 테스팅을 위한 입력자료가 추가로 필요하게 된다. 테스트 드라이버는 주어진 이벤트 시퀀스를 이용해서 I 부분에 대한 정당성을 검사하는 역할을 수행한다. 하나의 이벤트 시퀀스 v와 입력자료 i에 대해서 I와 드라이버가 성공적으로 연동 수행을 마치면 I는 v와 i에 대해서 정당(valid)하다. 연동 수행에 실패하면 (4) 단계의 분석을 실시한다.

(4) 연동 수행이 실패하면 프로그램 수행 중에 기록된 로그, 즉 이벤트 트레이스(event trace)를 명세로부터 생성된 다른 이벤트 시퀀스들과 비교한다. 테스트 드라이버는 한번에 하나의 이벤트 시퀀스만을 검사하지만 프로그램은 비결정성 때문에 여러가지 이벤트 트레이스를 산출할 수 있다. 따라서 프로그램이 산출한 이벤트 트레이스가 드라이버에서 기대하고 있던 이벤트 시퀀스와 일치하지 않는 경우에는 비록 프로그램이 정당한 작업을 수행했다 하더라도 연동 수행이 실패할 수 있다. 이 경우에 수행 중에 기록된 이벤트 트레이스를 다른 이벤트 시퀀스들과 비교해 보는 수행후 분석작업이 필요하다.

이 방법은 크게 두가지 특징을 가진다. 첫째, 기존의 방법들과는 달리 광범위하게 사용되는 명세로부터 직접 테스트 시퀀스를 생성한다. 기존의 연구들은 대부분 테스팅 목적으로 특별하게 기술된 명세인 TSL이나 CSPE 명세로부터 테스트 시퀀스를 생성했다. 이러한 명세들은 이벤트들 간의 순서 관계를 직접적으로 표현하기 때문에 테스트 케이스 생성이 쉽지만 개발자가 테스팅을 위해서 별도의 명세를 작성해야 한다는 부담이

있다. 둘째, 이 방법은 모듈 단위의 테스팅을 지원한다. 테스팅 대상 프로세스들을 선정하고 그 프로세스의 외부 행동을 검사하기 위한 이벤트 시퀀스를 생성하는 방법을 제공하기 때문에 시스템의 임의의 모듈을 테스팅할 수 있다.

한가지 주의할 점은 여기서 설명한 테스팅 환경은 병렬 프로그램의 수행을 강제로 조절하는 방법을 쓰지 않는다는 점이다. 프로그램 기반 테스팅 환경에서는 비결정성을 가진 문장을 제어함으로써 프로그램의 수행 방향을 조절하는 기능을 제공하지만 여기서 설명한 테스팅 환경은 명세 기반 블랙박스 테스팅 환경이므로 수행 조절 기능보다는 수행후 분석방법을 채택하고 있다.

병렬 프로그램의 이벤트 시퀀스에 대한 테스팅은 크게 두가지 방향에서 접근하는데, 첫째는 "프로그램 수행중에 발생한 이벤트 트레이스가 정당한가 ?"이고 다른 하나는 "명세로부터 생성된 이벤트 시퀀스가 실제 프로그램에서 실현가능한가 ?"이다. 이 연구에서는 현재 명세로부터 병렬 프로그램 테스팅을 위한 이벤트 시퀀스를 생성하는 방법을 제시하고 있다. 생성된 이벤트 시퀀스는 앞에서 언급한 두가지 문제를 해결하는 데 모두 응용될 수 있는데 먼저, 첫번째 문제를 사용하기 위해서는 수행중에 발생 가능한 모든 이벤트 시퀀스를 얻어내는 방법이 필요하다. 수행 중에 발생 가능한 이벤트 시퀀스를 명세로부터 생성된 이벤트 시퀀스와 비교함으로써 첫번째 문제를 해결할 수 있다. 반면에 두번째 문제 해결을 위해서는 프로그램 수행 조절 방법이 필요하다. 이 연구에서 제시한 방법을 통해서 얻어진 이벤트 시퀀스를 프로그램에 강제로 적용해봄으로써 실현 가능성을 검사할 수 있다. 따라서 향후에 병렬 프로그램 디버깅 기법들에서 제시되었던 수행 조절 방법을 우리가 구현중인 테스팅 환경에 포함하는 방법에 대한 연구가 계획되어 있으며 다양한 테스트 커버리지를 도입하는 방법을 대한 연구도 진행하고자 한다.

참 고 문 헌

[1] K. C. Tai, R. H. Carver and E. E. Obaid, "Debugging Concurrent Ada Programs by Deterministic Execution," *IEEE Trans. on Soft. Eng.*, **vol. 17, no. 1,** pp. 45-63, January 1991

[2] T. J. LeBlanc and J. M. Mellor-Crummey, "Debugging Parallel Programs with Instant Replay," *IEEE Trans. on Computers,* **vol. C-36, no. 4,** pp. 471-482, 1987

[3] 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

[4] D. Rosenblum, "Specifying Concurrent Systems with TSL," *IEEE Software,* pp. 52-61, May 1991

[5] ITU-T Recommendation Z.120: Message Sequence Chart (MSC), September 1994

[6] C. Fidge, "Logical Time in Distributed Computing Systems," *IEEE Computer,* **24(8),** pp. 28-33, August 1991

[7] 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, Tokyo, 1994

[8] 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

[9] H. S. Bae, H. S. Kim, Y. R. Kwon, and H. K. Kim, "Debugging Message-based Parallel Programs using Detect and Reproduce Method," *Journal of KISS(B): Software and Applicaions,* **23(2),** pp. 146-157, 1996

[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] 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

[12] J. Grabowski, D. Hogrefe, I. Nussbaumer, and A. Spichiger, "Test Case Specification Based on MSCs and ASN.1," *Proc. of the Seventh SDL Forum 1995,* pp. 307-322, 1995

[13] 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, Tokyo, 1994

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

[15] S. Weiss, "A Formal Framework for the Study of Concurrent Program Testing," *Proc. of 2nd Workshop on Software Testing, Analysis, and Verification,* pp. 106-113, July 1988

배 현 섭

1993년 한국과학기술원 전산학과 학사. 1995년 한국과학기술원 전산학과 석사. 1999년 한국과학기술원 전산학과 박사. 현재 한국전자통신연구원 선임연구원. 관심분야는 소프트웨어 테스팅 및 디버깅, 실시간 병렬 프로그램 개발 및 검증, 정형 명세

정 인 상

1987년 서울대학교 컴퓨터공학과 학사. 1989년 한국과학기술원 전산학과 석사. 1993년 한국과학기술원 전산학과 박사. 1993년 9월 ~ 1994년 2월 한국전자통신연구원 박사후연수연구원. 1995년 7월 ~ 1995년 5월 8일 영국 Durham 대학 Centre for Software Maintenance 방문연구원. 1994년 3월 ~ 1999년 2월 한림대학교 교수. 1997년 8월 ~ 1998년 7월 미국 Purdue대학 SERC 방문교수. 1999년 3월 ~ 현재 한성대학교 정보전산학부 교수. 관심분야는 소프트웨어 테스팅, CBSE, Formal Specification Techniques

김 현 수

1988년 서울대학교 계산통계학과(학사). 1991년 한국과학기술원 전산학과(석사). 1995년 한국과학기술원 전산학과(박사). 1995년 ~ 1995년 한국전자통신연구원 박사후연수연구원. 1996년 ~ 현재 금오공과대학교 컴퓨터공학부 조교수. 1999년 ~ 현재 Colorado State Univ. 연구교수

권 용 래

1969년 서울대학교 문리대학 이학사. 1971년 서울대학교 대학원 이학석사. 1971년 ~ 1974년 육군사관학교 전임강사. 1978년 미국 피츠버그대학 이학박사. 1978년 ~ 1983년 미국 Computer Science Corporation 연구원. 1983년 ~ 현재 한국과학기술원 전산학과 교수. 관심분야는 실시간 병렬 소프트웨어 검증, 실시간 시스템의 객체지향 기술, 고신뢰도 소프트웨어의 품질 보증

정 영 식

1983년 홍익대학교 전자계산학과 학사. 1993년 한남대학교 전자계산공학과 석사. 1983년 ~ 현재 한국전자통신연구원 선임연구원. 관심분야는 시스템 프로그래밍, 차세대 라우터 기술

이 병 선

1980년 성균관대학교 수학과 졸업. 1982년 동국대학교 전산학과 석사. 1999년 ~ 현재 한국과학기술원 전산학과 박사과정중. 1982년 ~ 현재 한국전자통신연구원 책임연구원. 관심분야는 Software Fault Tolerance, Software Reliability, Object-oriented software testing, Component-based software engineering