동시성 프로그램 테스트를 위한 구조 커버리지 기법 조사

안재민, 홍신, 김문주

한국과학기술원 전산학과
대전광역시 유성구 구성동 373-1
jjaeminn@kaist.ac.kr, hongshin@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

요약: 멀티코어 프로세서를 사용하는 하드웨어 플랫폼이 널리 보급됨에 따라 최근 많은 소프트웨어 시스템이 동시성 프로그램으로 개발됨에 따라 동시성 프로그램 테스트에 대한 수요가 증가하고 있다. 체계적인 동시성 프로그램 테스트를 위해서 다양한 동시 실행 특성을 표현하는 커버리지 기법들이 제안되어 왔다. 본 논문에서는 동시성 프로그램 테스트를 위하여 다양한 커버리지 기법들을 정형화된 형태로 기술하고, 상호 간의 관계를 연구한 결과를 소개한다.

핵심어: 동시성 프로그램, 구조 커버리지

1. 서 론

멀티코어 프로세서를 사용하는 하드웨어 플랫폼이 널리 보급됨에 따라 최근 많은 소프트웨어 시스템이 동시성 프로그램(concurrent program)으로 개발되고 있다. 근래에는 높은 정확도를 요구하는 내장형 시스템에서도 멀티코어 프로세서의 활용도를 높이기 위해, 동시성 프로그램을 소프트웨어를 사용하는 경우가 증가하고 있다.

동시성 프로그램은 쓰레드 스케쥴링 순서에 따라 프로그램 동작이 결정되므로 순차적 프로그램(sequential program)보다 높은 복잡도를 가진다. 쓰레드 동기화 결함으로 발생하는 데이터 경쟁상태(data race)나 교착상태(deadlock)와 같은 오류의 경우, 특정 쓰레드 스케쥴링에 해당하는 실행에서만 오류가 발생하기도 한다. 따라서, 동시성 프로그램의 정확도를 보장하기 위해서는 다양한 입력 값에 대한 테스트뿐만 아니라 다양한 스케쥴링 경우에 대한 테스트가 필수적이다.

테스트 커버리지 기법은 체계적인 테스트를 수행을 위하여 현재 테스트 실행의 진행 정도를 측정하는 기술이다. 순차 프로그램 테스트에서는 구문 커버리지(statement coverage)와 분기 커버리지(branch coverage) 기법이 테스트가 수행된 순차적 실행경로를 효과적으로 측정하는데 사용되어 왔다. 하지만 순차적 프로그램을 대상으로 한 커버리지 기법은 다양한 동시 실행 경우의 표현이 불가능하므로, 동시성 프로그램에 특화된 커버리지 기법이 필요하다.

본 논문에서는 동시성 프로그램 테스트를 위하여 현재 제시된 다양한 커버리지 기법들[2, 3, 5, 6, 7] 중 잠금(lock)과 관련된 커버리지 기법을 정형화된 형태로 기술하고, 상호 간의 관계를 연구한 결과를 소개한다. 순차적 프로그램을 위한 커버리지 기법의 경우 표준적인 정의가 존재하며, 커버리지 기법 간의 관계가 정립되어 있다. 하지만 동시성 프로그램 커버리지 기법의 경우, 여전히 연구자에 따라 별개의 정의와 용어를 사용하고 있는 실정이다. 본 연구에서는 동시 프로그램의 실행 모델과 커버리지 모델을 이용하여 (1) 여러 커버리지 정의를 일관되게 기술하였고, (2) 동일한 정의를 가지는 커버리지 기법 간의 관계를 파악하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 동시 실행 모델과 커버리지 모델을 소개하도록 한다. 3 장에서는 앞에서 정의한 모델을 이용하여 잠금에 초점을 맞춘 동시성 프로그램의 구문 커버리지를 3 개의 종류로 분류한다. 마지막으로 4 장에서는 연구 결과를 정리한다.

2. 동시 실행 모델과 커버리지 모델

본 장에서는 다양한 동시성 프로그램 커버리지 기법들을 일관된 형태로 기술하기 위한 기반으로 동시 실행 모델과 커버리지 모델을 정의한다.

동시성을 가지는 프로그램의 실행은 인터리브 실행(interleaved execution), 즉 여러 쓰레드가 번갈아 가며 명령을 실행한 순열로 표현 가능하다. 따라서, 하나의 동시 실행은 유한한 길이를 가지는 명령의 순열로 정의한다. 명령은 프로그램 실행의 최소 단위이다. 순열에서 명령 사이의 순서 관계는 인터리브 실행에서 실행 차례와 같다. 각각의 명령에는 명령을 수행한 쓰레드, 명령과 연관된 구조 요소 (혹은 코드 요소), 명령이 참조하는 메모리 주소를 속성으로 가진다.

커버리지(coverage)란 프로그램의 테스트 수행 정도를 나타내는 값으로, 주어진 테스트 요구사항 중 현

재 테스트 과정이 달성한 요구사항을 측정한 값을 뜻한다. 테스트 요구사항(test requirement)이란 테스트 과정에서 반드시 만족해야 하는 프로그램 실행 조건이다. 커버리지 조건(coverage criteria)은 테스트 대상 프로그램의 코드, 요구사항 명세를 바탕으로 테스트 요구사항 집합을 도출하는 방법을 뜻한다 [1,4].

구조 커버리지(structural coverage)란 프로그램 코드의 구조에 대한 조건의 형태로 테스트 요구사항을 생성하는 커버리지 기법이다. 예를 들어, 순차적 프로그램 테스트에서 표준적으로 사용되고 있는 구문 커버리지(statement coverage)는 구조 커버리지의 일종으로, 주어진 프로그램 코드에 존재하는 각각의 구문이 최소 한 번 이상의 실행이 되었는지 검사하는 조건을 테스트 요구사항으로 생성한다. 구조 커버리지는 테스트 진행 정도를 코드의 범위 형태로 표현하기 때문에, 개발자가 커버리지의 의미를 직관적으로 이해할 수 있다는 장점이 있다.

일반적인 동시성 프로그램을 위한 구조 커버리지 기법에서 테스트 요구사항을 생성할 때 쓰레드 스케쥴링에 연관된 정보를 반영할 수 있도록 순차적 프로그램의 테스트 요구사항을 다음과 같은 항목에서 확장한다.

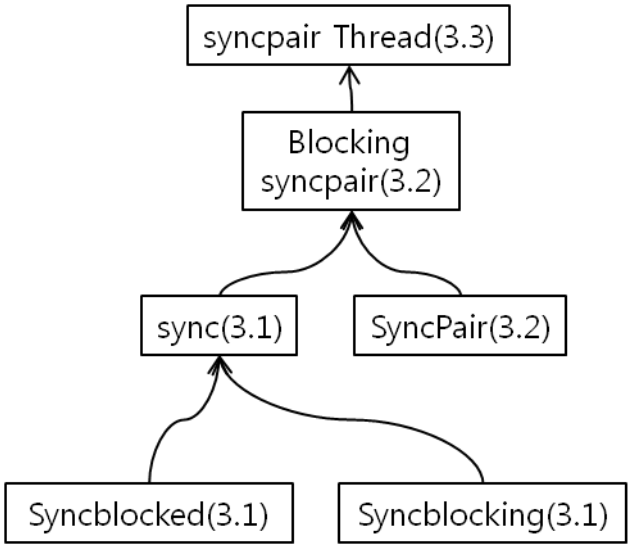
- 관련된 코드 요소: 순차적 프로그램에서는 분기, 읽기, 쓰기 등이 주요한 커버리지 측정 코드 요소이지만, 동기화 프로그램에서는 동기화 관련 구문 (예: 잠금(lock), 해제(unlock))이 테스트 요구사항에 사용되고 있다.
- 실행 문맥: 동시 실행에서는 특정 실행이 어떤 문맥을 가지는 상황에서 실행되는지가 중요한 요소가 된다.
- 순열: 동시 실행에서의 테스트 요구사항은 여러 쓰레드 간의 실행 순서를 표현하기 위해서 단일 구조요소에 대한 조건이 아니라 구조요소의 순열 형태를 가질 수 있다.

이러한 확장을 가지는 동시성 프로그램의 테스트 요구사항은 <명령, 문맥>^{N} 과 같이 기술할 수 있다. 하지만 본 논문에서는 여러 커버리지 기준을 기술하기 위해 테스트 요구사항을 <명령 ^{N}> 과 같은 형태로 기술하고 실행 문맥은 테스트 요구사항을 만족시키기 위한 실행의 조건을 설명할 때 명시하도록 한다.

각각의 커버리지 기준의 분류, 비교를 할 때에도 앞서 설명한 확장성에 초점을 맞추도록 하겠다.

3. Concurrent 커버리지 기준

본 장에서는 지금까지 여러 연구[2,3,4]에서 제시된 잠금 명령에 초점을 둔 동시성 구조 커버리지



[그림 1] 잠금에 관련된 커버리지 기준간의 상관관계

기준을 앞서 정의한 실행 모델을 이용하여 정형적으로 기술한다. 여러 가지 연구에서 제시된 커버리지 기준 중 같거나 유사한 정의를 사용하는 경우에는 한 절에서 소개하도록 한다.

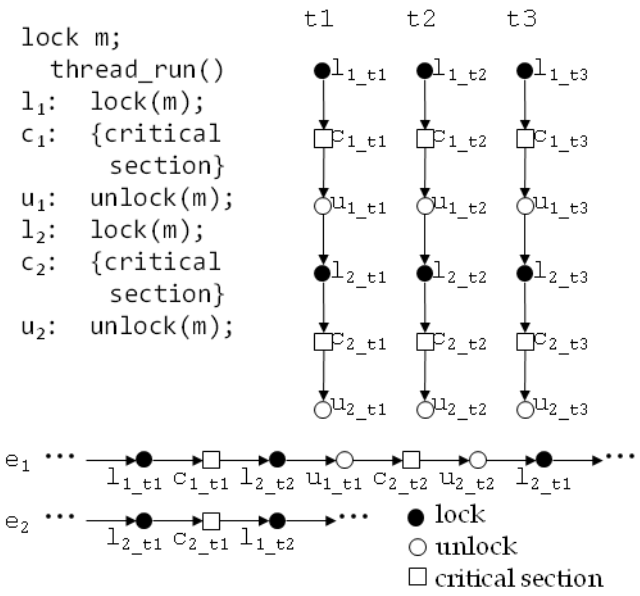
[그림 1]은 본 장에서 소개할 커버리지 기준간의 포함관계를 나타낸다. 상위에 있는 커버리지 기준을 만족하는 테스트 집합은 하위에 있는 기준 또한 만족한다.

3.1 Syncblocked, Syncblocking, sync

Syncblocked[3,5], Syncblocking[3,5], sync[2,5]는 다음과 같이 정의된다.

정의1: (Syncblocked) 테스트 요구사항 집합은 실행 가능한 모든 잠금(lock) 명령을 포함한다. 각각의 테스트 요구사항은 하나의 명령 op로 구성되며, 다음의 조건을 만족시키는 실행 e=…p…q…을 통해 만족된다.
- 공유메모리 m은 잠금 변수
- 명령 q는 op와 코드 요소가 같고, 잠금(m)
- 명령 p는 잠금(m)
- p의 잠금은 성공
- p와 q사이에 해제(m) (unlock(m))은 존재하지 않음

정의2: (Syncblocking) 테스트 요구사항 집합은 실행 가능한 모든 잠금(lock) 명령을 포함한다. 각각의 테스트 요구사항은 하나의 명령 op로 구성되며, 다음의 조건을 만족시키는 실행 e=…p…q…을 통해 만족된다.
- 공유메모리 m은 잠금 변수
- 명령 p는 op와 코드 요소가 같고, 잠금(m)
- 명령 q는 잠금(m)
- p의 잠금은 성공
- p와 q사이에 해제(m) 은 존재하지 않음



[그림 2] 잠금에 관련된 커버리지 기준을 위한 예제 프로그램, t_1, t_2, t_3 3개의 쓰레드가 같은 함수 thread_run()을 실행한다. e_1, e_2는 각각 서로 다른 인터리브 실행의 일부분이다.

정의3: (sync) 테스트 요구사항 집합은 실행 가능한 모든 잠금(lock) 명령을 포함한다. 각각의 테스트 요구사항은 하나의 명령 op로 구성되며, 다음의 조건을 각각 만족시키는 실행 e=…p…q…, e′=…p′…q′…을 통해 만족된다.

e(=…p…q…)의 조건
- 공유메모리 m은 잠금 변수
- 명령 q는 op와 코드 요소가 같고, 잠금(m)
- 명령 p는 잠금(m)
- p의 잠금은 성공
- p와 q사이에 해제(m)은 존재하지 않음

e′(=…p′…q′…)의 조건
- 공유메모리 m은 잠금 변수
- 명령 p는 op와 코드 요소가 같고, 잠금(m)
- 명령 q는 잠금(m)
- p의 잠금은 성공
- p와 q사이에 해제(m)은 존재하지 않음

[그림 2]의 프로그램에서 Syncblocked커버리지 기준의 테스트 요구사항 집합은 {l_1, l_2}가 된다. 실행 e1에서 $l_{1_t1} \rightarrow l_{2_t2}$를 통해 테스트 요구사항 l_2가 만족된다.

[그림 2]의 프로그램에서 Syncblocking커버리지 기준의 테스트 요구사항 집합은 마찬가지로 {l_1, l_2}가 된다. 실행 e_1에서 $l_{1_t1} \rightarrow l_{2_t2}$를 통해 테스트 요구사항 l_1이 만족된다 하지만 $l_{2_t2} \rightarrow l_{2_t3}$에서 테스트 요구사항 l_2는 만족되지 않는다. $l_{2\_t2} \rightarrow l_{2\_t3}$ 사이에 u_{2_t2}가 있어서 실제로 l_{2_t3}가 잠금을 획득하기 위해 실행이 중지되지 않기 때문이다.

[그림 2]의 프로그램에서 sync커버리지 기준의 테스트 요구사항 집합은 {l_1, l_2}가 된다. 하지만 실행 e_1을 통해서는 어떠한 테스트 요구사항도 만족되지 않는다. l_1의 blocking조건, l_2의 blocked조건만 만족되기 때문이다. 추가적인 실행 e_2에서 $l_{2_t1} \rightarrow l_{1_t2}$를 통해 l_1의 blocked, l_2의 blocking조건을 만족시켜 테스트 요구사항 l_1, l_2를 만족시킬 수 있다.

Syncblocked, Syncblocking, sync커버리지 기준은 동시성 프로그램에서 제한 영역(critical section) 실행의 보장 등을 위해 사용하는 잠금의 순서에 따라 쓰레드간 인터리브 실행이 제어될 때의 상황을 검증하기 위해 고안되었다. 순차적 프로그램에서는 항상 같은 순서로 명령이 실행되지만, 동시성 프로그램에서는 인터리브 실행에 따라 다른 동작을 보일 수 있기 때문에 이를 검증해야 한다.

Trainin 등은 Syncblocked, Syncblocking커버리지 기법을 통한 테스팅으로 동시성 프로그램 고유의 버그를 발견하는 연구를 수행하였다[3].

3.2 syncpair, Blockingsyncpair

syncpair[2], BlockingSyncPair[3]는 다음과 같이 정의된다.

정의4:(syncpair) 테스트 요구사항 집합은 실행 가능한 모든 (잠금, 잠금) 명령 쌍을 포함한다. 각각의 테스트 요구사항은 명령쌍 (op_1, op_2)로 구성되며, 다음의 조건을 만족시키는 실행 e=…p…q…을 통해 만족된다.
- 공유메모리 m은 잠금 변수
- 명령 p는 op_1와 같고, 코드 요소는 잠금(m)
- 명령 q는 op_2와 같고, 코드 요소는 잠금(m)
- p의 잠금은 성공
- q의 잠금은 성공
- p와 q사이에 잠금(m)은 존재하지 않음

정의5:(Blockingsyncpair) 테스트 요구사항 집합은 실행 가능한 모든 (잠금, 잠금) 명령 쌍을 포함한다. 각각의 테스트 요구사항은 명령쌍 (op_1, op_2)로 구성되며, 다음의 조건을 만족시키는 실행 e=…p…q…을 통해 만족된다.

- 공유메모리 m은 잠금 변수
- 명령 p는 op_1와 코드 요소가 같고, 잠금(m)
- 명령 q는 op_2와 코드 요소가 같고, 잠금(m)
- p의 잠금은 성공
- p와 q사이에 잠금(m), 해제(m)은 존재하지 않음
- q의 잠금은 성공

[그림 2]의 프로그램에서 syncpair커버리지 기준의 테스트 요구사항 집합은 {(l_1, l_1), (l_1, l_2), (l_2, l_1), (l_2, l_2)}가 된다. 실행 e_1에서 $l_{1_t1} \rightarrow l_{2_t2}$를 통해 테스트 요구사항 ($l_1$, l_2)가 만족되고, $l_{2_t2} \rightarrow l_{1_t1}$을 통해 테스트 요구사항 ($l_2$, l_1)이 만족된다.

[그림 2]의 프로그램에서 SyncPair커버리지 기준의 테스트 요구사항 집합은 마찬가지로 {(l_1, l_1), (l_1, l_2), (l_2, l_1), (l_2, l_2)}가 된다. 실행 e_1에서 $l_{1_t1} \rightarrow l_{2_t2}$를 통해 테스트 요구사항 ($l_1$, l_2)가 만족된다. 하지만 실행 e_1에서 $l_{2_t2} \rightarrow l_{1_t1}$을 통해 테스트 요구사항 ($l_2$, l_1)이 만족되지는 않는다. u_{2_t2}가 사이에 있어 l_{1_t1}의 실행 쓰레드가 block되지 않기 때문이다. 테스트 요구사항 (l_2, l_1)은 실행 e_2에서 만족된다.

syncpair와 Blockingsyncpair커버리지 기준은 모두 동시성 프로그램의 제한 영역 순서를 검증하기 위한 커버리지 기준이다. 3.1절에 소개된 커버리지 기준과 다른 점은 3.1절에 소개된 커버리지 기준은 하나의 잠금 구문이 쓰레드 인터리브 실행에 영향을 주는지 여부만 검증했지만 syncpair, Blockingsync pair커버리지 기준은 가능한 모든 잠금 명령쌍의 검증을 통해 모든 제한 영역 실행 순서에 따른 동시성 프로그램의 동작을 검증할 수 있다. 따라서 3.1절에서 소개한 커버리지 기준보다 더욱 다양한 테스팅을 가능하게 해 준다.

3.3 syncpair Thread

syncpair Thread[2]은 다음과 같이 정의된다.

정의 6: (syncpair Thread) 테스트 요구사항 집합은 실행 가능한 모든 ((잠금, 쓰레드 id), (잠금, 쓰레드 id)) 명령 X 쓰레드 쌍을 포함한다. 각각의 테스트 요구사항은 명령 X 쓰레드쌍 ((op_1, th_1), (op_2, th_2))로 구성되며, 다음의 조건을 만족시키는 실행 e=…p…q…을 통해 만족된다.

- 공유메모리 m은 잠금 변수
- 명령 p 는 op_1과 같고, 코드 요소는 잠금(m)
- 명령 q 는 op_2와 같고, 코드 요소는 잠금(m)
- p 의 실행 쓰레드는 th_1
- q 의 실행 쓰레드는 th_2
- th_1과 th_2는 다름
- p 와 q 의 잠금(m)은 성공

[그림 2]의 프로그램에서 syncpair Thread 커버리지 기준의 테스트 요구사항 집합은 {((l_1, t_1), (l_1, t_2)), ((l_1, t_1), (l_1, t_3)), ((l_1, t_1), (l_2, t_2)), …… }가 된다. 실행 e1 에서는 $l_{1\_t1} \rightarrow l_{2\_t2}$ 를 통해 테스트 요구사항 ((l_1, t_1), (l_2, t_2))가 만족되고, $l_{2\_t2} \rightarrow l_{1\_t1}$ 을 통해 테스트 요구사항 ((l_2, t_2), (l_1, t_1))이 만족된다.

Sync Pair Thread 커버리지 기준은 동시성 프로그램의 제한 영역 실행을 Sync Pair 커버리지 기준보다 엄밀하게 검증한다. Sync Pair Thread 는 같은 잠금 명령이라도 다른 쓰레드에서 실행되면 다른 명령으로 간주하기 때문에 실행 문맥에 따라 제한 영역 내부에서 쓰레드의 비공유 데이터 값을 사용하는 경우에 대한 엄밀한 검증이 가능하다.

4. 결 론

본 논문은 concurrent 소프트웨어를 대상으로 하는 구조 커버리지 중 잠금 명령에 기초해 제안된 6 개의 커버리지 기준을 3 가지 카테고리로 분류해 제시하고, 비교, 분석했다.

이를 위해 다양한 커버리지 기준을 표현할 수 있는 통일된 concurrent 실행 모델을 만들고, 이에 따라 지금까지 제안된 커버리지 기준들을 재정의했다. 재정의한 커버리지 기준을 바탕으로 커버리지 기준간의 상관관계를 규명했다.

참고문헌

[1] P. Amman, J. Offutt, Introduction to Software Testing.

[2] Elena Sherman, Matthew B. Dwyer, Sebastian G. Elbaum. Saturation-based testing of concurrent programs. In Proceedings of ESEC/SIGSOFT FSE'2009. pp.53~62

[3] Ehud Trainin, Yarden Nir-Buchbinder, Rachel Tzoref-Brill, Aviad Zlotnick, Shmuel Ur, Eitan Farchi. Forcing small models of conditions on program interleaving for detection of concurrent bugs. In Proceedings of PADTAD'2009.

[4] B. Marick. The Craft of Software Testing, Subsystem testing Including Object-Based and Object-Oriented Testing. Prentice-Hall, 1985

[5] Arkady Bron, Eitan Farchi, Yonit Magid, Yarden Nir, Shmuel Ur. Applications of synchronization coverage. In Proceedings of PPOPP'2005. pp.206~212

[6] Cheer-Sun D. Yang, Lori L. Pollock. All-uses testing of shared memory parallel programs. Softw. Test., Verif. Reliab., 2003: 3~24

[7] Chao Wang, Mahmoud Said, Aarti Gupta. Coverage guided systematic concurrency testing. In Proceedings of ICSE'2011. pp.221~230