Assertion Based Verification을 이용한 Statechart 모델의 검증

황대연° 방기석 최진영
고려대학교 컴퓨터학과
{dyhwang, kbang, choi}@formal.korea.ac.kr

Statechart verification using Assertion Based Verification

Dae-Yon Hwang° Ki-Seok Bang Jin-Young Choi
Dept. of Computer Science & Engineering, Korea University

요 약

Statechart는 매우 널리 쓰이고 있는 명세 언어이다. 현재는 UML에 포함되어 많은 사람들이 구현하고자 하는 시스템의 명세에 Statechart를 이용하고 있다. Statechart 명세는 구현될 시스템의 설계도 역할을 하기 때문에 오류가 있을 경우 매우 치명적일 수 있으며, 반대로 시스템의 오류를 명세 단계인 Statechart 명세에서 찾아내게 되면 수정에 필요로 하는 비용과 시간의 손실을 최소화 할 수 있다. 본 논문에서는 하드웨어 검증 분야에서 사용되고 있는 Assertion Based Verification (ABV) 방법론을 Statechart 명세에 적용하여 시뮬레이션 등으로는 찾아내기 힘든 오류를 찾아낼 수 있음을 보였다.

1. 서 론

Statechart[3,4] 명세는 구현하고자 하는 시스템의 설계도와 같은 역할을 한다. 구현 요구 사항의 명세 및 시스템의 행위, 인터페이스, 각 구성 요소들 간의 관계 등등 시스템이 구현되기 위한 기초가 되어준다. 소프트웨어 시스템의 크기가 점차 커지고 그 구조가 복잡해짐에 따라 그 행위를 검증하는 일이 어려워지고 있다. 따라서 시스템의 오류를 최소화하고 또 오류 복구에 필요한 시간과 자본을 최소화하기 위해서는 설계도인 Statechart 명세가 올바로 명세 되는 것이 매우 중요하다.

시스템에 대한 명세를 검증하는 방법은 매우 다양하다. 시뮬레이션은 그 중에서 가장 보편적이고 널리 쓰이는 방법 중 하나이다. 정형 기법을 이용하여 시스템 모델을 수학적으로 검증하는 방법도 현재 많이 사용 되고 있다. 하지만 시뮬레이션은 검증 범위의 한계를 가지며 구석 상태에 숨어있는 오류를 찾는 것이 매우 힘들다. 정형 검증은 시스템 전체에 대한 검증 능력을 가지지만 시스템의 크기에 대한 제약 크다.

본 논문에서는 현재 하드웨어 검증 분야에서 널리 쓰이는 Assertion Based Verification [1,2]을 Statechart에 적용하여 정형 기법으로는 추상화하지 않고는 검증이 안 되며 시뮬레이션으로는 찾기 힘든 Statechart 명세의 오류를 찾아낼 수 있음을 보였다.

본 논문의 구성은 다음과 같다. 2장은 ABV에 대한 간략한 소개를 하고 3장에서는 Statechart 명세를 검증한 예제를 소개한다. 마지막으로 4장에서는 결론 및 향후 연구 방향을 이야기 한다.

2. Assertion Based Verification (ABV)

시스템의 크기가 커지고 복잡해짐에 따라 널리 사용되던 블랙박스 테스팅의 한계가 나타났다. 요구 명세에 의해 입력을 만들고 그에 맞는 출력이 나오는지를 검사하는 블랙 박스 테스팅 기법에는 큰 시스템을 검사하기에는 부족한 부분들이 몇 가지 있었다. 그 중 하나는 오류가 어디서 발생했는지를 알 수 없다는 것이었다.

화이트박스 테스팅은 블랙박스 테스팅의 단점을 보완한다. 검사할 시스템의 내부를 보지 않는 블랙박스와 반대로 화이트박스 테스팅은 검사할 시스템에 대한 필요한 정보들을 이용하여 오류가 생길 가능성을 분석하고 확인함으로써 오류가 일어났을 때 정확하게 어디서 일어나며, 어떤 방식으로 입력을 제어해야 할지를 알 수 있다. 이러한 정보들을 assertion이라 한다.

Assertion은 시스템이 지켜야 할 요구 사항들이라고 할 수 있다. 화이트박스 테스팅은 대상 시스템을 시뮬레이션을 하면서 모니터링 함으로써 입력한 assertion에 위배되는 상태가 되면 오류를 알리는 방법인 것이다.

ABV는 한 단계 더 나아가서 assertion들을 시뮬레이션 모니터링뿐만이 아니라 다양한 방법에 이용하여 시스템을 검증하는 방법론이다. 그 예로는 테스트 벤치의 자동적인 추출에 이용하거나, 정적 모델 체킹에서 검증할 속성으로 이용, 문서 작업의 기초 등으로 사용하는 방법 등이 있다.

정적 모델 체킹 또는 모델 체킹[5,6]은 정형 검증의 한 종류로 모델을 수학적으로 기술하고 시스템의 assertion들을 검사 속성으로 변환하여 검증하는 방법이다.

정적 모델 체킹은 시스템의 모델이 가질 수 있는 모든 행위에 대해 속성들을 검사를 하기 때문에 모델 체킹의 결과로 안전하다고 나왔을 경우 모든 상태에서 안전하다고 할 수 있다. 단, 모델 체킹은 검증하고자 하는 시스템의 크기가 너무 크면 메모리가 부족하게 되는 상태 폭발을 일으켜 검증을 할 수가 없게 된다.

ABV에 포함되는 또 다른 방법론 중 하나는 Dynamic Bounded Model Checking(DBMC)[1]이다. 시뮬레이션과 모델 체킹 두 방법의 장점들을 모아 시뮬레이션을 보완하는 방법이다.

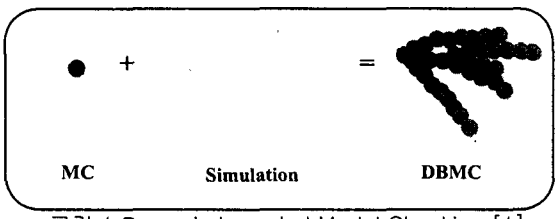

그림 1 Dynamic bounded Model Checking [1]

그림 1 은 DBMC에서 어떤 방법으로 시뮬레이션에 Bounded Model Checking (BMC)을 적용했는지를 보여준다. BMC는 전체 시스템이 아닌 제한된 깊이까지의 행위를 검사하는 모델 체킹 방법이다. BMC는 모델 체킹처럼 시스템을 완벽하게 검증하지는 못하지만 시작 상태에서부터 주어진 제한까지는 모든 행위를 검증하기 때문에 시뮬레이션으로는 찾기 힘든 구석 상태를 검사하는 것이 용이하다. 또, 시스템 전체의 상태를 검사하는 것이 아니기 때문에 제한 값의 크기만 조절을 하면 상태 폭발이 일어지 않는다. DBMC에서는 정해진 제한을 설정하고 대신 시뮬레이션을 진행하면서 추출하는 시스템 행위의 중간 상태를 매번 새로운 시작 상태로 해서 계속적으로 검증을 하게 된다. 이러한 중간 상태를 seed 상태라 한다. Dynamic이라는 단어가 붙은 것은 바로 동적으로 계속 seed 상태를 이용해 초기 상태를 지정하기 때문이다.

3. Statechart 명세의 검증

3.1 Statechart 명세의 변환

Statechart 명세에 DBMC을 적용하여 검증하기 위해서는 다음과 같은 단계가 필요하다.

먼저 검증할 시스템의 Statechart 명세와 모델에서 검증하고자 하는 속성, 또는 assertion을 정해야 한다. Assertion들은 요구 사항에서 추출하거나 Statechart 명세를 만드는 과정에서 만든다. Statechart 명세와 검증하고자 하는 assertion들이 결정되면 검증 도구를 결정 한다. 본 논문에서는 cadence SMV를 이용하여 검증을 했다. 검증 도구가 정해지면 Statechart 명세와 속성을 도구의 입력 언어에 맞게 변환한 하는 과정을 거친다. 이때 Statechart의 모델이 의미와 변환된 모델의 의미가 같도록 주위를 해야 한다. 예로 Statechart의 상태 기계는 일리 기계를 기반으로 하는 것에 반해 SMV의 모델은 무어 기계를 기반으로 하고 있음 등을 주의해야 한다.

다음은 Statechart 명세를 SMV 모델로 변환하는데 사용한 규칙 중 몇 가지 이다.

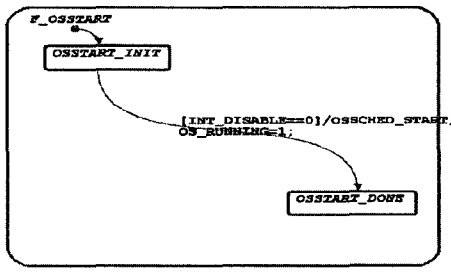

그림 2 Statechart 명세 예제

- 각 Statechart의 이름은 SMV의 변수가 되고, 그 안의 상태들은 변수가 가질 수 있는 값이 된다.
  예) VAR f_osstart : {o_init, o_done}
- 상태로 들어오는 전이에 출력이 있을 경우 그 출력들에 의해 상태가 나눈다.
  예) {o_init, o_done, o_done_out}
- 전이를 위한 이벤트들은 해당 전이의 가드들로 전환한다.
  예) next (f_osstart) :=
  f_osstart=o_init & int_disable = 0 :
  o_done_out
- 변수들 역시 SMV 변수로 전환 한다. 타입은 변수의 범위로 한다. 사용되는 값들이 정확할 경우는 그 값들로 한다.
  예) VAR int_disable : {0, 1}
- 하위 오토마타의 경우 상위의 오토마타에서 해당 상태에서 전이하는 조건들을 자신의 모든 전이들의 가드에 추가한다.

시스템 모델의 변환이 끝나면 속성들을 SMV의 입력인 CTL 형식으로 변환하다. DBMC는 전이에 제한을 두기 때문에 모델의 모든 행위를 검사하지 않는다. 때문에 BMC를 할 경우에는 반례를 들어 속성이 만족하지 않는 경우를 찾을 수는 있어도 항상 이러한 속성을 만족한다는 검증은 하지 못한다는 것을 염두에 두고 속성을 변환해야 한다.

모델과 속성의 변환이 끝나면 시뮬레이션과 그에 의한 seed 상태의 추출이 필요하다. Seed 상태의 경우 두 가지 방법으로 추출할 수 있다.

하나는 시뮬레이션 중 일정한 수의 전이를 할 때마다 도달하는 상태를 seed 상태로 잡는 방법과 검사하고자 하는 assertion을 분석하여 일정한 조건을 만족시켜주는 상태를 seed 상태로 추출하는 방법이다. 예로 ' 어떤 변수 x가 절대로 5를 초과하지 않는다' 라는 속성을 검사하고자 한다면 변수의 값이 5가 되는 상태를 seed 상태로 선택해서 5를 초과하는 상태로 전이가 가능한지를 살피는 방식이다.

마지막으로 SMV의 경우 정적 모델 체킹 도구이기 때문에 제한을 가해 주어야 한다. 이러한 제한은 모델에

전이 수를 제한함으로써 가능하다.

3.2 검증 예제

그림 3는 Statechart로 명세한 작은 OS의 전체적 구조를 보여준다.

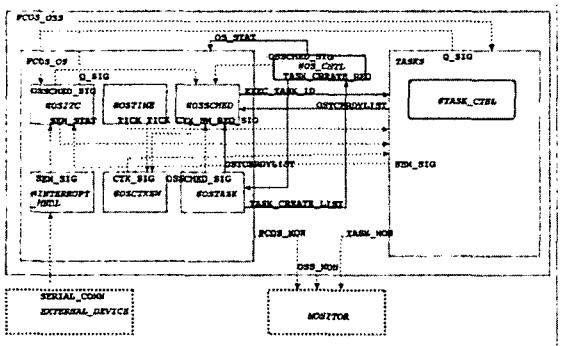

그림 3 PCOS 명세

PCOS[7]는 임베디드 시스템에 사용될 목적을 가진 실시간 OS이다. PCOS는 크게 OS_CNTL과 TASK_CNTL로 나뉘어져 구성되어 있다.

OS_CNTL은 OS 초기화 과정 동안의 행위를 담당하고, TASK_CNTL은 그 과정에서 생성된 태스크들의 행위들을 나타낸다. 예제에서 태스크는 4개로 0,1,2번의 태스크와 idle 태스크이다. 그림 4은 태스크 0의 명세 예제이다.

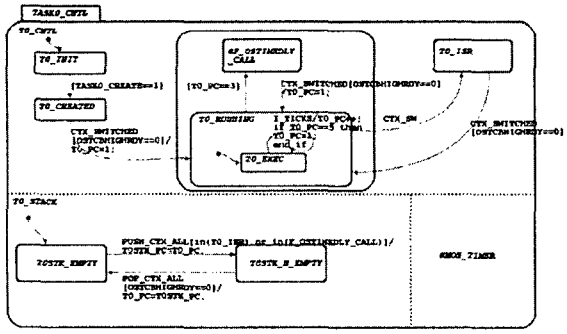

그림 4 태스크 0의 명세

다음은 예제에서 검증한 assertion들 중 일부이다.

1) SPEC AG !(T0_CNTL = t0_running
& T2_CODE = t2_running)
2) SPEC AG !(T0_PC = 5)
3) SPEC AG !(sem1_locked & sem1_unlocked)
4) SPEC AG !((ser_com = 1)&((OSSCHED_CNTL =
scheduling) & (SCHEDULING = sched_init) &
(OSTCBPRIOTBL [0] = 1)))

1)은 2개의 태스크가 동시에 돌아가는 일이 없음 검사한다. 2)는 태스크의 주기를 나타내는 수치로 그 이상 올라가는 일이 없음을 검사한다. 3)은 서로 반대되는 개념의 이벤트들은 동시에 일어날 수 없음 검사하는 예이다. 마지막으로 4)는 같은 변수의 값을 동시에 변화시키는 두 개의 전이가 동시에 일어나는지를 검사한다.

그림 5는 DBMC를 이용하여 제한을 두어 검증한 결과이다. Seed 상태는 매 15 상태 마다 추출했으며, 속성 4번에 대한 오류가 있음을 찾아낼 수 있었다.

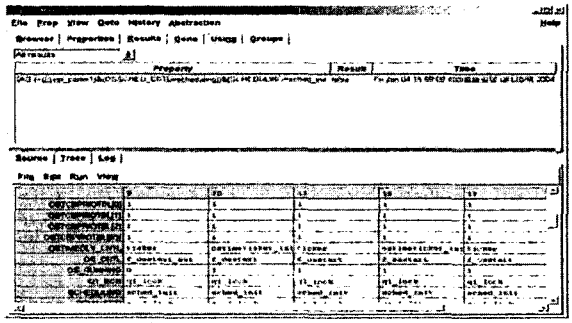

그림 5 PCOS에 대한 SMV 검증 결과

오류는 외부 인터럽트를 위해 만들어 놓은 부분에서 외부 인터럽트가 내부에서 태스크 변환 시기와 겹치면 다음 수행될 태스크가 비 결정적이 되는 문제였다.

4. 결론 및 향후 연구

ABV를 Statechart에 적용하여 검증을 하면 시뮬레이션을 보완하며 오류를 찾는데 도움이 될 수 있음을 보였다. DBMC는 시스템이 가질 수 있는 행위의 완전한 검증이 아니라 일부분이라는 약점이 있으나 모델 체킹처럼 상태 폭발을 일으키지 않으며 시뮬레이션으로는 찾기 힘든 오류를 찾을 수 있다. 다만 이 방법론은 완전한 검증이 아니기에 시뮬레이션의 보완용으로 사용될 수 있어도 시스템의 완벽한 검증에는 적합 하지가 않다.

향후 연구 과제로는 중복되지 않은 시뮬레이션 경로를 고르는 일, 검증 도구 입력으로의 자동 코드 변환, 검증 범위 계산 등에 관한 연구가 필요하다.

5. 참고문헌

[1] 0-In Design Automation, Inc. *Assertion-Based Verification for Complex Designs*, 2003

[2] Synopsys, Inc. *Assertion-Based Verification*, 2003

[3] D. Harel, A Naamad, " *The STATEMATE Semantics of statecharts*" ACM Transactions on Software Engineering and Methodology, vol. 5 No. 4, pages 293-333 1996

[4] D. Harel, " *Statecharts: A visual Formalism for Complex Systems*" , Sci. Comput. Prog. 1987

[5] K. L. McMillan, *Symbolic Model Checking* Kluwer Acadecmic Publisher, 1993

[6] E. M. Clarke, O. Grumberg, D. A. Peled, *Model checking* MIT Press, 1999

[7] J. J. Labrosse, *MicroC/OS-II*, CMPBooks 2002.