Bandera Toolset 을 이용한 객체지향형 소프트웨어 모델 체킹*

방 기 석°, 이 주 용, 최 진 영
고려대학교 컴퓨터학과

{kbang, jlee, choi}@formal.korea.ac.kr

Model Checking of Object Oriented Software using Bandera toolset

Ki-Seok Bang°, Joo-Yong Lee, Jin-Young Choi
Department of Computer Science and Engineering, Korea University

요 약

객체지향형 소프트웨어가 개발되고 분산 시스템에 적용되면서 소프트웨어 시스템의 분석 및 안전성의 보장이 매우 어려워지고 있다. 정형 기법을 이용해서 소프트웨어 시스템의 안전성을 증명하는 연구가 진행되고 있지만 소스코드 레벨에서의 보장은 아직 어려운 상태이다. 본 연구에서는 이러한 소프트웨어 시스템의 소스코드 레벨에서의 안전성 보장을 위한 연구로 Bandera toolset 을 이용한 정형검증에 대해 논한다.

1. 서 론*

최근에 객체 지향형 소프트웨어가 개발되고 발전함에 따라 동시성 프로그래밍의 수요가 증가하고 그 기법 역시 다양해 지고 있다. 그와 함께 동시수행 소프트웨어(concurrent software)의 재사용성과 안정성을 높이기 위한 기법이 개발되고있다. 이 결과 소프트웨어의 복잡도가 매우 높아지고, 특히 수행시간 복잡도(run-time complexity)가 매우 높아져서 소프트웨어의 설계와 안정성 검증 분야에 많은 문제가 발생되고 있다. 일반적인 시뮬레이션이나 테스팅 기법 만으로는 소프트웨어 시스템의 크기나 복잡성을 극복하여 시스템의 완전성을 보장하기가 매우 어렵다. 때문에 정형기법을 이용하여 소프트웨어 시스템의 안정성을 증명하려는 연구가 진행되고 있다. 현재 C 언어와 같은 범용 프로그래밍 언어로 개발된 소프트웨어의 경우는 소스 코드 레벨에서의 모델링 및 정형검증이 많이 발전되어 있다. 그러나 객체지향형 소프트웨어의 경우에는 객체가 갖는 동적인 특성을 극복하기 어렵기 때문에 소스코드 레벨에서의 정형검증이 매우 어렵다. 본 논문에서는 그러한 연구의 하나로 Bandera toolset[1]을 이용하여 객체지향형 소프트웨어 시스템을 정형 검증하는 방법에 대해 논한다. 본 논문은 다음과 같이 이뤄진다. 2 절에서는 소프트웨어 모델체킹의 방법론을 설명하고, 3 절에서는 Bandera toolset 을 이용한 객체지향형 소프트웨어 시스템을 모델체킹하는 방법에 대해 논한다. 그리고 마지막으로 결론 및 향후 과제를 설명한다.

* 본 연구는 2000 년 한국 과학 재단 특정기초 연구 지원사업(2000-1-30300-010-3)에 의해 연구되었습니다.

2. 관련 연구

2.1 소프트웨어 모델 체킹

객체 지향형 소프트웨어의 신뢰도를 높이기 위해 정형기법을 적용하는 방법이 많이 연구되고 있다. 특히 모델체킹[2]을 이용하여 복잡한 소프트웨어 시스템의 동작적인 특성을 증명하려는 연구가 많이 진행되고 있다. 모델체킹 방법은 시스템을 유한 상태 전이 시스템으로 모델링하고 그 모델이 만족해야 할 특성을 시제논리[3]로 명세하여 모델이 특성을 만족하는지 검증한다. 또한 모델이 특성을 만족하지 못할 경우 모델의 전이를 추적하여 왜 특성을 만족하지 못하는지 반례를 자동으로 생성하여 오류의 수정을 용이하도록 도와준다. 하지만 이런 장점에도 불구하고 현재 모델체킹이 소프트웨어 시스템의 검증에 사용되기는 매우 어렵다. 그 이유는 다음과 같이 요약할 수 있다[4].

첫째, 상태 폭발의 문제(state explosion problem)가 야기될 수 있다. 유한 상태 전이 시스템 모델은 원래 시스템의 요소들이 늘어날 경우 지수적으로 그 상태가 증가하게 된다. 이런 상태 폭발 문제를 해결하기 위해 많은 방법이 제기되고 있지만 대부분이 하드웨어 요소들에 대한 기법이며 소프트 웨어 시스템의 경우는 하드웨어보다 복잡하여 그 추상화의 문제가 더 심각하다.

두번째 문제는 모델의 구현에 있다. 대부분의 소프트웨어 개발자는 C, Java 와 같은 범용 프로그래밍 언어로 구현을 한다. 하지만 정형 검증을 위한 명세언어는 그런

범용 언어와는 매우 다른 문법적인 특성 및 의미론을 바탕으로 하고 있다. 따라서, 실제 프로그램을 정형검증 도구에 적용하기 위해서는 일단 범용 프로그래밍 언어로 구현된 모델을 다시 정형검증 도구용 입력 언어로 변환시키는 작업이 필수적이다. 이런 변환 작업 도중에 많은 오류가 발생할 수 있고 시간의 낭비가 일어나게 된다.

세번째 문제는 요구 명세의 문제이다. 소프트웨어의 요구사항을 현재 사용되는 시제논리를 이용하여 완벽히 표현하는 것은 매우 어렵다. 비록 모델 체커의 명세 언어가 시제논리에 기반하여 매우 논리적으로 정의되어 있지만, 복잡한 이벤트의 특성을 표현하기에는 그렇게 쉽지 않다. 또한 모델 체커의 명세언어는 상태적인 특성을 수학적 모델에 적용하기 위해 개발된 것이기 때문에 실제 프로그램 코드를 설계하기에는 적합하진 않다. 정형 검증 도구의 입력언어들은 대부분 정적인 상태의 변화나 지역적 변수의 변화 같은 특성을 검사하기에 적합하다. 그러나 실제 프로그램 코드에서는 이런 특성 외에도 많은 동적인 특성을 지니고 있기 때문에 정형 검증을 하려는 사용자는 반드시 이런 차이점을 극복할 수 있도록 코드를 추상화해야 한다. 특히, 객체 지향형 소프트웨어의 경우 수행중에 동적으로 생성되는 객체나 쓰레드의 생성과 소멸 같은 특성은 모델링 언어로 표현하기에는 매우 어렵다.

마지막으로 출력 결과의 해석에서 문제가 발생할 수 있다. 모델이 특성을 만족하지 못했을 경우 모델체킹 도구는 오류에 대한 역추적을 가능하도록 하는 반례를 만들어 준다. 그러나 모델의 크기가 매우 클 경우 그 반례 역시 수백 또는 수천 라인으로 증가하게 된다. 이런 큰 추적코드를 수동으로 실제 코드에 적용해서 오류의 위치를 찾아내는 것은 매우 어렵다. 또한 오류 추적 코드는 추상화된 모델에서 매우 낮은 수준으로 생성되기 때문에 실제 코드를 추적하는 데엔 많은 차이가 존재한다. 즉, 실제 복잡한 소스 코드의 한 단계의 전이가 추상화된 오류의 수십 줄의 단계에 해당하는 경우가 매우 많다.

이런 많은 문제점 때문에 현재 소프트웨어 시스템의 소스코드 단계에서의 모델 체킹은 큰 효과를 거두지 못하고 있다. 그러나, 많은 연구 기관과 학계에서 최적의 결과는 아니지만 매우 좋은 검증 도구를 개발하여 현재 적용중이며, 좋은 결과를 많이 가져오고 있다. 대표적인 도구로는 dSPIN, JPF, 그리고 Bandera toolset 등을 들 수 있다.

2.2 소프트웨어 시스템 검증 도구들

위에서 언급한 바와 같이 소프트웨어 시스템의 정형 검증은 소프트웨어가 갖는 특성과 모델 체킹 기법의 한계 때문에 매우 어렵다. 그러나, 현재 많은 연구기관에서 소프트웨어를 검증하기 위한 도구를 개발하여 발표하고 좋은 연구 결과가 도출되고 있다. 대표적으로 SPIN[6], dSPIN[5] 그리고 여러 도구를 위한 통합 환경인 Bandera toolset 등이 있다.

SPIN 은 원래 통신 프로토콜을 설계하고 검증하기 위해 개발된 도구이다. 그러나, 기능이 점차 확장되면서 소프트웨어 시스템을 모델링하고 검증하기 위해 연구가 진행되고 있다. 특히 C 와 같은 언어로 개발된 시스템의 경우 소스코드를 Promela 로 변환하여 검증하기에 매우 수월한 구조를 갖고 있다.

dSPIN 은 기존의 SPIN 을 객체지향의 개념을 추가하여 확장한 것이다. dSPIN 은 기존의 SPIN 이 검증할 수 있는 것 뿐 아니라 몇 가지 새로운 명세 및 정형 검증이 가능하다. dSPIN 에서는 동적인 메모리의 조작과 객체의 생성을 명세하기 위해 Pointer 의 개념을 추가하여 동적인 메모리 참조 모델을 가능하도록 하였다. 그러나 이 도구 역시 소스코드를 그대로 검증하는 것이 아니라, 다시 사용자가 모델링을 해야 한다.

위에서 설명한 도구들은 일단 개발된 시스템의 소스코드를 명세자가 다시 검증 도구의 입력 언어로 변환하는 작업이 필요하다. 그러나 앞에서 언급한 것처럼 그 과정에서 오류가 개입될 수 있다. 이런 문제를 해결하기 위해 개발되고 있는 것이 Bandera toolset 이다.

Bandera toolset 은 캔사스 주립대에서 주관하여 개발하고 있는 객체 지향형 소프트웨어 시스템의 검증도구이다. 이 도구는 자바 소스코드를 분석하고 검증언어로 변환하며 모델체킹을 하기 위한 통합환경을 제공하고 있다. 입력 언어로는 자바 소스코드를 그대로 입력하며, 적용하고자 하는 검증 방법에 맞는 명세 언어를 자동으로 출력해 준다. 이 도구에서는 현재 SPIN, dSPIN, SMV[7] 그리고 JPF II(Java Path Finder II)의 입력언어로의 변환이 가능하다. 이 도구는 소스 코드의 변환과 검증이 자동으로 이뤄지기 때문에 사용자의 추가적인 개입이 없다. 결국 코드의 변환과정에서 생길 수 있는 오류를 배제할 수 있다. 또한, 오류의 발생 시 역추적을 명세단계가 아닌 자바 소스코드의 레벨에서 보여주기 때문에 실제 코드의 수정을 통한 오류 수정이 가능하다. Bandera toolset 은 자바로 구현되어 있기 때문에 PC 및 UNIX/Linux 의 어떤 환경에서도 동작이 가능하다.

3. Bandera toolset 과 SPIN 을 이용한 객체 지향형 소프트웨어의 검증

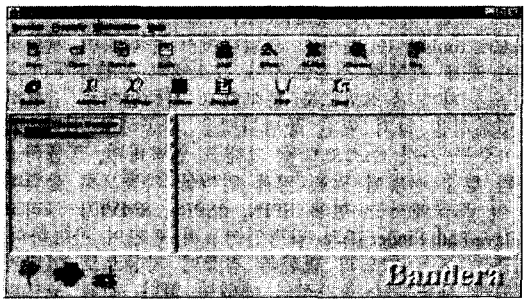
본 절에서는 Bandera toolset 을 이용하여 간단한 객체 지향형 프로그램을 검증해 본다. 다음은 Java 로 구현된 간단한 Dining philosopher 의 프로그램이다.

```
class Phil extends Thread
{
    Object left, right;
    int proc;

    Phil(Object l, Object r, int p) {
      left = l;
      right = r;
      proc = p;}
    ...
}}}

public class philosopher
{
    public static void main(String [] args)
```

```
{
    Object o1, o2, o3;

    o1 = new Object();
    o2 = new Object();
    o3 = new Object();

    Phil p1 = new Phil(o1, o2, 1);
    Phil p2 = new Phil(o2, o3, 2);
    Phil p3 = new Phil(o3, o2, 3);

    p1.start();
    p2.start();
    p3.start();
}}}
```

[그림 1] Dining Philosopher 의 예제

위 프로그램은 철학자 세 명이 동작하는 방식을 보여주고 있다. 위의 프로그램에서 데드락이 존재하는가를 검증하기 위해 Bandera toolset 을 적용하였다.



[그림 2] Bandera 의 수행 모습

Bandera 를 이용한 결과 위의 코드는 자동으로 다음과 같은 Promela 코드로 변환이 되었다. 원시 코드는 매우 짧고 간단하지만 쓰레드가 표현해야 하는 많은 기능들을 모두 추가해 줘야 하기 때문에 변환된 코드는 다음과 같이 매우 길고 복잡하게 구현된다.

```
typedef lock_ {
  chan lock = [1] of { bit };
  byte owner;
};
...

init {
        atomic {
            Phil_col.instance[0].BIRLock.lock! 0 ;
            Phil_col.instance[1].BIRLock.lock! 0 ;
            Phil_col.instance[2].BIRLock.lock! 0 ;
            Phil_col_0.instance[0].BIRLock.lock! 0 ;
            Phil_col_0.instance[1].BIRLock.lock! 0 ;
            Phil_col_0.instance[2].BIRLock.lock! 0 ;
            Phil_col_1.instance[0].BIRLock.lock! 0 ;
            Phil_col_1.instance[1].BIRLock.lock! 0 ;
            Phil_col_1.instance[2].BIRLock.lock! 0 ;
            assert(run philosopher() == 1);
            assert(run Phil() == 2);
        }
}
```

[그림 3] 변환된 Dining Philosopher 코드

검증 결과 위의 프로그램에선 데드락이 발생하는 것을 알 수 있었다. 원래 Bandera 를 이용하면 오류가 발생할 경우 그 추적을 위해 별도의 창을 열고 소스코드를 추적하여 오류 발생을 보여준다.

```
State-vector 292 byte, depth reached 58, errors: 1
        23 states, stored
         0 states, matched
        23 transitions (= stored+matched)
        36 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)
```

[그림 4] Bandera 를 이용한 검증 결과

4. 결 론

Java 와 같은 객체 지향형 언어로 구현된 소프트웨어의 안정성을 증명하기 위해 정형기법을 적용하는 연구가 활발히 진행되고 있다. 특히 모델 체킹 방법론을 이용하여 복잡한 소프트웨어 시스템의 동작적인 특성을 검증한다. 그러나, 프로그램 소스코드를 모델 체킹 도구의 입력언어의 형태로 변환하고 그 결과를 분석하기 위해서는 사용자의 개입이 필요하고, 이 때 오류가 발생할 수 있다. 따라서, 프로그램의 소스코드를 그대로 입력으로 하고, 사용자의 개입없이 정형 검증 도구의 입력언어로 변환하여 모델 체킹을 수행하도록 하는 연구를 진행하였다. 그 결과 Bandera 라는 도구가 현재 개발중이며, 이 도구를 사용하면 SPIN, SMV 등 여러 가지 정형 검증 도구를 이용한 모델체킹을 적용하고, 그 결과를 소스코드 레벨에서 추적하여 오류를 수정할 수 있다. 따라서, 소스코드를 별다른 변환이나 모델링 없이 정형 검증의 입력 언어로 사용가능하여 보다 빠르고 정확한 정형 검증이 가능하다.

참고문헌

[1] *James Corbett, Matthew Dwyer, John Hatcliff, Corina Pasareanu, Robby, Shawn Laubach, Hongjun Zheng* Bandera : Extracting Finite-state Models from Java Source Code, Proc. of the 22nd ICSE, Toronto, 2001

[2] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model Checking*, The MIT Press, 1999.

[3] Zohar Manna, Amir Pnueli, "The Temporal Logic of Reactive and Concurrent Systems - Specification", Springer-Verlag, 1996.

[4] http://www.cis.ksu.edu/santos/bandera

[5] C. Demartini, R. Iosif, R. Sisto, "dSPIN: A Dynamic Extension of SPIN", in *Proc. 6th International SPIN Workshop, Theoretical and Practical Aspects of SPIN Model Checking*, Toulouse, September 1999.

[6] Gerard J. Holzmann, "The Model Checker SPIN," IEEE Transactions on Software Engineering, May 1997.

[7] Kenneth L. McMillan, "SYMBOLIC MODEL CHECKING:, Kluwer Academic Publisher 1993.