모델 체킹을 위한 자바프로그램의 술어추상화*

이정림° 이태훈 권기현
경기대학교 정보과학부
{jrim°, taehoon, khkwon}@kyonggi.ac.kr

Predicate Abstraction of Java Program for Model Checking

Jungrim Lee° Taehoo Lee Gihwon Kwon
Computer Science Department, Kyonggi University

요 약

모델 체킹은 시스템이 올바르게 동작하는 지를 자동으로 검증해주는 기법이다. 지금까지 모델 체킹은 방법의 특성상 상태폭발문제 때문에 작은 규모의 상태공간을 갖는 하드웨어나 프로토콜 검증에 주로 사용되어져 왔다. 그러나 최근에는 상태폭발문제를 다루기 위한 연구와 술어추상화 등과 같은 추상화기술의 발달로, 다룰 수 있는 상태공간의 크기가 증가하고 정리증명기의 성능이 향상됨에 따라 소프트웨어 자체의 논리적 오류를 검증하려는 움직임이 활발하다. 일반적으로 소프트웨어 검증을 위해 추상화-모델 체킹-추상화 개선의 3단계 프로세스를 사용하는데 주로 C프로그램에 대해서만 이루어져 왔다. 우리는 이 프로세스를 자바프로그램에 적용하고 자동으로 자바프로그램을 이진프로그램으로 변환하는 술어추상화 모듈을 개발하였다. 이 모듈은 우리가 개발한 자바 모델 체커의 front-end부분이다. 본 논문에서는 자바프로그램에 대한 주요 추상화 알고리즘을 보이고 특정 자바프로그램의 안전성과 궁극성을 검증한다.

1. 서 론

모델 체킹은 정형명세 언어를 사용하여 시스템을 유한상태모델로 표현하고 그 시스템이 만족해야하는 안전성 속성 등을 논리식으로 표현하여 시스템이 속성을 만족하는 지를 자동으로 검증해주는 기법이다[1]. 지금까지 모델 체킹은 모델이 커짐에 따라 검사해야 할 상태공간의 크기가 지수적으로 증가하는 상태폭발문제 때문에 작은 규모의 상태공간을 갖는 하드웨어나 프로토콜 검증에 주로 사용되어져 왔다. 그러나 최근에는 상태폭발문제를 다루기 위한 다양한 연구와 술어추상화 등과 같은 추상화 기술의 발전으로, 다룰 수 있는 상태공간의 크기가 증가하고 정리증명기의 성능이 향상됨에 따라 소프트웨어 자체의 논리적 오류를 검증하려는 움직임이 활발하다. 이런 움직임의 근본적인 이유는, 기존의 모델 체킹은 사용자의 실수로 모델이 제대로 표현되지 않았다면 속성 검증 시 잘못된 결과를 가져올 여지가 있다는 것이다. 또한 비전문가는 정형명세 언어로 모델을 표현하기가 쉽지 않지만 소프트웨어 자체를 검증하게 되면 별도로 정형명세를 할 필요가 없다. 그리고 잘못된 코드 작성은 소프트웨어의 논리적 오류 발생 가능성이 매우 높고 이것은 시스템의 신뢰도에 직접적인 영향을 미치게 된다. 따라서 소프트웨어 자체를 검증하면 개발자 입장에서 더 많은 논리적 오류들을 원천적으로 막을 수 있고 높은 신뢰도의 시스템을 구현할 수 있게 된다.

소프트웨어는 무한한 상태공간을 갖는다. 예를 들어, 정수형 변수 하나만 하더라도 무수히 많은 상태(..., -2, -1, 0, 1, 2, ...)를 가질 수 있고 이런 정수형 변수들의 조합은 더욱 무한한 상태공간을 발생시킨다. 따라서 소프트웨어를 모델 체킹하기 위해서는 상태공간을 줄이기 위한 추상화가 필수적이고 매우 중요하다. 여러 추상화 기법들 중 술어 추상화는 소프트웨어 모델 체킹 분야에서 프로그램을 자동으로 추상화시키는데 많이 활용되어지고 있다. 대표적인 연구로는 마이크로 소프트사의 C 검증도구인 SLAM[2], 버클리 대학의 C검증도구인 BLAST[3], 카네기 멜론 대학의 C검증도구인 MAGIC[4] 등이 있다. 특히 SLAM은 실제 윈도우의 디바이스 드라이버의 오류를 검증한 가장 실용적인 도구로 평가된다. 하지만 SLAM은 C를 대상으로 하기 때문에 다른 언어에서 효율적인 적용방법이 필요하다. 따라서 우리는 SLAM의 추상화 알고리즘을 바탕으로 자바의 특징을 추가하여 자바프로그램 J 와 술어의 집합 P 가 주어졌을 때 각각의 자바 구문이 어떻게 이진프로그램 $\Omega(J,P)$으로 자동 변환되는 지를 보인다. 이는 우리가 개발한 자바 모델 체커의 front-end부분이다. 이를 이용한 검증 과정은 그림1과 같이 일반적으로 사용되는 추상화 - 모델 체킹 - 추상화개선으로 이루어진다. 변환된 $\Omega(J,P)$는 CTL식 (Computation Tree Logic, [5])으로 표현된 속성과 함께 모델 체커의 입력이 된다. 모델 체킹을 수행한 후 속성을 만족하면 $true$, 만족하지 않으면 반례를 생성하고 반례가 실제 코드에서 도달가능 한 경로이면 $false$, 그렇지 않으면 가짜 반례로 판단한다. 추상화된 이진프로그램은 상위 근사화 된 모델이기 때문에 원래 프로그램에는 없는 행위를 가질 수 있다. 이런 경우 술어를 추가하여 이진프로그램을 개선한 후 다시 프로세스를 반복 수행한다.

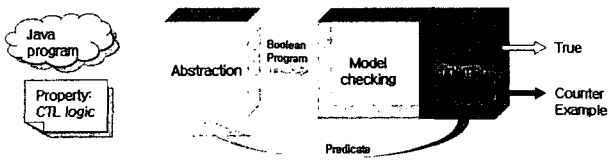

그림 1. 자바 모델 체커

본 논문의 2장에서는 술어추상화를 이용해 자바프로그램의 참조변수와 참조메소드를 어떻게 추상화했는지를 보이고 3장에서 자바언어로 작성된 임베디드 소프트웨어에 대해서 안전성과 궁극성을 검증한다. 그리고 마지막으로 4장에서 결론을 맺는다.

* 본 연구는 과학기술부 기초과학 연구
(R01-2005-000-11120-0) 지원으로 수행되었음.

2. 술어추상화

술어추상화는 Graf와 Saidi에 의해서 처음 제안되었다[6]. 이 기법은 술어 $p_i \in P$를 기준으로 매우 큰 데이터 범위를 갖는 변수를 관심 있는 부분만 술어 p_i에 대응하는 이진 변수 $v_i \in V$로 나타낼 수 있다. 예를 들어, 자바의 integer형 변수 x는 4byte내의 매우 큰 상태 공간을 갖는다. 하지만 x의 값이 음수인지, 양수인지, 0인지만을 알고 싶다면 $P=\{x<0, x>0, x=0\}$의 3개의 술어만 사용해서 겨우 6개의 상태공간으로 표현이 가능하다. 본 논문에서 $\Omega(J,P)$는 변수의 타입이 이진(boolean)형식만 사용이 가능한 프로그램이다. 이진 프로그램은 n개의 이진 변수와 m개의 프로그램 구문으로 구성된다고 하자. 이진 변수는 $v_1,...,v_n$ 으로 구성되고 모든 v_i에 대해서 $v_i \in \{true, false, *\}$이다. 여기서 * 은 참과 거짓인지 알 수 없는 경우이다. v_i의 값을 알기위해 $select(WP(s,p_i), WP(s,\neg p_i))$ 함수를 정의한다. 이 함수는 자바프로그램의 구문 s와 술어의 집합 P를 입력 받아서 그에 대응하는 이진 변수 v_i의 값을 되돌려 준다. 여기서 $WP(s,p_i)$는 *Weakest Precondition*[4]으로 다음 2.1절에서 설명한다. 만약 select()함수의 첫 번째 인자 $WP(s, p_i)$가 $true$이면 술어 p_i에 대응하는 v_i는 s구문 이후에 true이고 두 번째 인자 $WP(s,\neg p_i)$가 $true$이면 v_i는 $false$이다. 만일 두 인자 모두 $false$이면 v_i의 값은 알 수 없는 값(*)이 된다. 본 논문에서 각 구문의 술어 평가는 Simplify라는 정리 증명기를 사용한다. 그리고 추상화는 클래스 단위로 이루어지며 Inner클래스를 포함할 수 있다.

2.1 할당문의 술어추상화

기본적으로 먼저, 프로그램 내에 할당문 $x=e$와 이와 관련된 술어 p가 있을 때 *Weakest Precondition*을 계산한다. $WP(x=e,p)$는 $x=e$라는 할당문이 참이 되기 위한 가장 약한 선 조건을 말한다. 술어 p내에 x와 같은 변수가 존재한다면 p내의 변수 x를 e로 대체한다($p[e/x]$) . 예를 들어, 할당문 $x=x+2$와 술어의 집합 $P=\{x<3, x<0, x\equiv -1\}$을 가정하면,

$$WP(x=x+2, x<3) = x+2<3 = x<1$$

이 된다. 이것은 구문 $x=x+2$ 이후에 $x<3$이 만족되기 위해서는 선 조건으로 $x<1$이 참이 되어야 한다는 의미이다. 하지만 $x<1$은 위의 술어의 집합에 없기 때문에 $x<1$을 만족할 수 있는 가장 약한 다른 술어를 p에서 찾는다. $(x\equiv -1)\Rightarrow(x<0)$이 참이므로, 가장 약한 술어는 오른쪽에 있는 $x<0$이 된다. 만일 P 내에 만족하는 술어가 없다면 $x<3$에 대응하는 이진 변수 v는 알 수 없는 값(*)을 갖는다. 또한 P와 관련이 없는 할당문은 속성 검증에서 관심의 대상이 되지 않기 때문에 skip()한다.

이제 우리는 자바프로그램에서 객체에 의해 참조되는 참조변수를 다룬다. 예를 들어 $WP(a.x=e,p)$의 경우 a는 객체명, x는 a객체가 가지고 있는 변수명이 된다. 이때 술어 p내에 $b.x$라는 참조변수가 있을 때, 객체 a와 b가 같은 주소 값을 가리킨다면 $b.x$는 e로 대체된다($p[e/b.x]$). 다시 정의하면,

$$p[a.x,e,b.x] = (\ \&a.x = \&b.x \wedge p[e/b.x]) \vee (\ \&a.x\ != \&b.x \wedge p)$$

이다. 위의 정의로부터,

$$WP(a.x=c.y+1, b.x<c.y) = (\&a.x=\&b.x \wedge c.y+1<c.y) \vee (\ \&a.x\ != \&b.x \wedge b.x<c.y)$$

이다.

다음은 참조 변수를 갖는 할당문에 대한 술어추상화를 수행한 예이다.

```
public static class A{
    int value = 3;
}
public static void main( String[] args ) {

    Predicate O: a.value = 5;

    A a = new A();
    A b = new A();
    A c = new A();

    a = b;
    c = a;

    c.value = 5;
```

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

    Predicate O: a.value = 5;

    A a = new A();
    A b = new A();
    A c = new A();

    skip();
    skip();

    O = true;
```

위 예에서 O는 main()메소드 내의 지역술어 $a.value=5$에 대응하는 이진변수이다. 객체 c는 객체 a를 참조하기 때문에 $O[\&c.value=\&a.value \wedge 5=5]=true$가 된다. 그리고 객체 할당문 $a=b$와 $c=a$는 이진변수 O와 관련이 없기 때문에 skip()된다. 이 외에,

- $x=y++;$와 같은 후위연산자 구문은 $x=y;\ x=x+1;$로 대체 처리한다. 그리고
- $path[position]=LEFT$ 와 같이 배열과 상수도 술어추상화가 가능하다.

2.2 조건문의 술어추상화

조건문은 if(p)문과 while(p)문을 처리한다. 조건문내에 있는 조건식 p는 술어가 되고 각 불럭 내에 assum()문을 둬서 만족해야 할 술어들의 조합을 표현한다.

```
if( p1 ){
...
} else if ( p2 ) {
...
} else {
...
}
```

```
if( * ){
    assume( p1 ∧ !p2 ... ∧ ... )
...
} else if ( * ) {
    assume( !p1 ∧ p2 ... ∧ ... )
...
} else {
    assume( !p1 ∧ !p2 ... ∧ ... )
...
}
```

*는 조건식 p가 참일 수도 있고 거짓일 수도 있음을 나타낸다. 그리고 각 블록 내 구문들은 assume()문내의 술어들의 조합을 따른다.

2.3 메소드의 술어추상화

C프로그램에 함수가 있다면 자바프로그램에는 메소드가 있다. 물론 인수의 전달이나 리턴 값을 얻는 방식은 함수와 비슷하나 메소드는 C프로그램의 서브루틴이나 함수와 달리 특정 객체를 대상으로 한다. 따라서 우리는 동일 클래스내의 리턴 값을 갖는 메소드인 v=method(f_1, f_2,...) 뿐만 아니라 v=obj.method(f_1, f_2,...)형태의 참조 메소드도 다룬다. 자바프로그램의 추상화는 메소드 호출의 추상화를 위해 크게 2단계로 구성된다. 1단계는 각 메소드 정보를 Method_Info()에 저장한다. 2단계는 모든 구문과 함께 Method_Info()를 사용하여 메소드 호출을 추상화한다. 이 외에,

- v = obj.method(f_1, f_2, ...) + e; 는
  v = obj.method(f_1, f_2, ...); v = v+e;로 대체 처리한다.
- v = e + obj.method(f_1, f_2, ...); 도
  v = obj.method(f_1, f_2, ...); v = v+e;로 대체 처리한다.

3. 검증

위의 술어추상화 기법을 통해 얻은 이진프로그램 $\Omega(J,P)$와 CTL식을 자체 모델체커에 입력해서 특정 자바프로그램의 속성을 검증한다. $\Omega(J,P)$는 모델 체커 내에서 제어흐름 그래프로 변환된다. 이 변환된 그래프의 상태공간을 탐색하며 주어진 속성이 만족하는 지를 검사한다. 검증해야 할 속성에는 크게 안전성(Safety)과 궁극성(Liveness)이 있다. 안전성은 시스템 실행 도중, 나쁜 상태에 도달할 수 없음을 의미한다. 안전성의 대표적인 예는 데드락 부재이다. 궁극성은 시스템 도중 언젠가는 좋은 상태에 도달할 수 있음을 의미한다. 궁극성의 대표적인 예는 자원 사용 허가이다. 시스템이 정확하다는 것을 보장하기 위해서는 두 가지 속성이 만족됨을 보여야 한다. SLAM에서는 검증할 수 있는 속성이 안전성에 국한되어 있지만 우리는 CTL식을 통해 궁극성도 검증한다.

3.1 문제

먼저, 검증 대상으로 아래 그림 2의 핸드폰에 탑재되는 푸시푸시 게임 50판을 선정하였다.

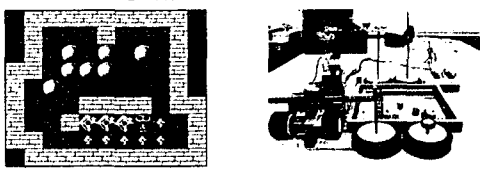

그림2. 푸시푸시 50판과 이를 구현한 크레인 시스템

푸시푸시 게임은 플레이어가 보드위에 흩어져있는 공들을 목적지에 모두 옮겨놓으면 클리어된다. 이 게임을 교육용 로봇제작 툴킷인 LegoMindstorm을 사용해 실제 구현하였다. 모터와 센서, 레고 블럭을 이용해 크레인을 만들고 RCX라는 임베디드 시스템을 연결하였다. 이 크레인은 Arm을 이용해 공을 상, 하, 좌, 우로 움직이며 터치센서를 통해 Arm이 움직이는 거리를 측정하고 다음 움직임을 결정한다. 또한 RCX 내부에는 LeJos로 만들어진 자바프로그램을 탑재하였다. LeJos는 레고로봇을 위한 자바언어로 JDK1.1 문법의 일부분만을 지원하기 때문에 검증이 수월했다.(본 논문의 술어추상화모듈은 JDK 1.2를 지원한다) 그리고 이 게임의 해답인 경로(338 step)는 모델체커 SMV에서 반례생성을 통해 얻어내었다[7]. 따라서 우리는 크레인이 경로를 따라 Arm을 움직여서 모든 공을 목적지로 옮기는 자바프로그램을 작성하였다.

3.2 속성

작성한 자바프로그램에서 검증하려는 안전성 속성과 궁극성 속성을 아래와 같이 CTL식으로 기술한다.

1) $AG\neg$(*game* == "Clear")

이 안전성 속성의 의미는 "모든 가능한 경우에 *game*이 Clear 되지 않는다" 이다. 다시 말해서, 크레인이 모든 공을 목적위치에 옮겨놓고 게임이 클리어 되는 경우는 존재하지 않는다.

2) AG(*sensing* == *true* ⇒ AF (*lr_cycle_cnt* ++))

이 궁극성 속성의 의미는 "모든 가능한 경우에 *sensing*이 일어나면 언젠가는 *lr_cycle_cnt* 가 1증가 한다" 이다. 다시 말해서, 언제든지 크레인의 터치센서가 센싱되면 움직인 거리가 측정되고 다음 움직임을 결정한다. 만약 터치센서가 센싱되어도 *lr_cycle_cnt* 가 증가하지 않으면 다음 움직임 없이 Arm은 한 방향으로만 무한히 진행된다.

위 두 가지 속성을 검증한 결과, 크레인 프로그램에는 논리적 오류가 없었다. 또한 실제 시스템을 실행한 결과도, 시스템이 올바르게 동작하여 모든 공을 목적지까지 옮기고 게임이 종료되었다.

4. 결론

오늘날, 소프트웨어 자체의 논리적 오류를 찾아내기 위해 소프트웨어 모델체커에 관한 연구가 매우 활발하다. 일반적으로 소프트웨어 검증 프로세스는 추상화-모델 체킹-추상화 개선이 사용된다. 하지만 이 프로세스는 C프로그램만을 대상으로 적용되어져 왔다. 따라서 우리는 이 프로세스를 바탕으로 자바 프로그램을 위한 모델체커를 개발했다. 본 논문에서는 이 모델체커의 front-end부분인 술어추상화 모듈을 다룬다. 먼저 자바프로그램을 추상화하기 위해 참조 변수와 참조 메소드 호출 등의 주요 술어추상화 알고리즘을 보였다. 그리고 실제 크레인 시스템을 구현하여 그 시스템에 탑재된 자바프로그램이 속성을 만족하는지 검증하였다. 속성은 CTL식을 이용하기 때문에 안전성 속성과 궁극성 속성 모두 표현이 가능하다. 따라서 두 가지 속성을 검증한 결과, 크레인 프로그램에 논리적 오류는 없었고 실제 시스템을 실행한 결과도, 시스템이 올바르게 동작하여 모든 공을 목적지까지 옮기고 게임이 종료된다는 것을 확인할 수 있었다. 하지만 현재는 자바프로그램의 정적인 부분에 대해서만 술어추상화가 적용되고 있다. 따라서 자바의 멀티 쓰레드나 런타임 시 동적 객체 생성에 대한 술어추상화 연구가 필요하다.

참 고 문 헌

[1] E. M. Clarke, O. Grumberg and D. Peled, Model Checking, MIT Press, 1999.

[2] T. Ball, R. Majumdar, T. Millstein and S.K. Rajamani, "Automatic Predicate Abstraction of C programs," SIGPLAN Notices, Vol. 36, No.5, pp.203-213, 2001.

[3] T.A. Henzinger, R. Jhala, R. Majumdar and G. Sutre, "Lazy Abstraction," in Proceedings of Principles of Programming Languages, pp.58-70, 2002.

[4] S. Charki, E.M. Clarke, A. Groce, S. Jha and H. Veith, "Modular Verification of Software Components in C," IEEE Transactions on Software Engineering, Vol.30, No.6, pp.388-402, 2004.

[5] E. A. Emerson, Temporal and modal logic, in the Handbook of Theoretical Computer Science : Formal Models and Semantics, J. van Leeuwen, editor, Elsevier, pp.995-1072, 1990.

[6] S. Graf and H. Saidi, "Construction of Abstraction State Graphs with PVS," in Proceedings of Computer Aided Verification, pp.72-83, 1997.

[7] http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/