18-3-R05

술어 추상화를 이용한 자바 프로그램의 단순화 및 모델 체킹*

(Simplification and Model Checking of Java Programs using Predicate Abstraction)

이 태훈 · 권 기현
경기대학교 정보과학부
{taehoon, khkwon}@kyonggi.ac.kr

요 약

모델 체킹은 오류를 찾아내기 위한 검증 기술중의 하나로서 하드웨어 검증, 소프트웨어 검증, 프로토콜 검증등에 적용되어 왔다. 현재 많은 연구 그룹들이 모델 체킹을 이용해서 프로그램 소스 코드를 직접 검증하려는 시도를 진행중이다. 유한 상태만을 다루는 현재의 모델 체킹 기술로는 프로그램 소스 코드를 직접 검증하기 힘들다. 왜냐하면 프로그램은 무한 상태를 갖고 있기 때문이다. 이를 해결하기 위한 기술 중에 하나가 술어 추상화이다. 술어 추상화를 통해서 무한 상태 공간을 유한 상태 공간으로 축소할 수 있다. 현재까지 개발된 술어 추상화 기반의 모델 체킹 도구로는 SLAM, BLAST, MAGIC 등이 있다. 이들 도구는 C 언어와 같은 절차적 프로그램의 검증만을 지원하는 반면에, 자바 언어와 같은 객체지향 프로그램의 검증은 지원하지 않는다. 본 논문에서는 술어 추상화를 자바 프로그램에 적용하여 추상화된 이진 프로그램을 구한 후, SMV를 이용해서 이진 프로그램을 검증하는 방법을 제안한다.

1. 서 론

모델 체킹은 시스템이 가지고 있는 잠재적인 오류를 찾기 위한 기술이다. 모델 체킹은 유한 상태 모델 M과 시제 논리식ϕ를 받아서 만족성 관계 $M \models \phi$를 결정한다[1]. 모델이 갖는 상태 공간을 철저히 탐색하면서 만족성 여부를 결정하기 때문에, 테스팅이나 시뮬레이션으로는 찾을 수 없는 난해한 버그를 찾을 수 있다. 그리고 사용자의 개입없이 자동으로 검증 과정이 진행되기 때문에, 정리 증명에 비해서 빠르다. 또한 모델이 논리식을 만족하지 않는 경우, 그 이유를 담은 반례를 제공함으로서 모델의 디버그 작업을 돕는다[2]. 이와같은 이점들 때문에 모델 체킹은 지난 20년 동안 하드웨어 검증, 소프트웨어 검증, 프로토콜 검증등 다양한 분야에 적용되어 왔다.

그러나 모델 체킹이 소프트웨어 검증에

* 본 연구는 과학기술부 목적기초연구 (R05-2004-000-10612-0) 지원으로 수행되었음.

는 아직까지 널리 사용되지 못하고 있다. 대부분의 모델 체킹 도구는 유한개의 상태만을 다룰 수 있는데 비해서, 프로그램의 상태 공간은 일반적으로 무한하다. 그러므로 현재의 모델 체킹 기술을 적용해서 프로그램을 검증하기는 힘들다. 유한 상태 공간만을 취급하는 모델 체킹 도구로 프로그램을 검증하기 위해서는, 무한 상태 공간을 유한 상태 공간으로 변환해야 한다. 술어 추상화는(Predicate Abstraction)[3] 무한 상태 공간을 유한 상태 공간으로 축소하는 기술이다. 술어 추상화를 기반으로 한 프로그램 코드 검증 도구들이 많이 연구되어 왔다. 대표적인 연구는 Bandera[4], SLAM[5], BLAST[6], MAGIC[7] 등이 있다. Bandera 는 캔사스 대학에서 개발 중인 자바 검증 도구이다. SLAM 은 마이크로 소프트에서 개발 중인 C 검증 도구이다. MAGIC는 CMU 에서 개발 중인 C 검증 도구이다 (각각의 도구 마다 검증 방법과 특성이 다르다. 이들에 대한 비교는 [8]을 참조하기 바람).

위에 도구들중에 Bandera를 제외한 나머지 도구들은 술어 추상화를 기반으로 하고 있다. 하지만 Bandera의 경우 술어 추상화 기술을 지원하지 않고 자료 추상화라는 추상화 기법을 적용해서 프로그램의 크기를 축소한다. 현재 술어 추상화 기반 도구들은 대부분 C 언어를 대상으로 하고 있다. 현재의 프로그램은 C 와 같은 선언적 언어와 함께 C++ 와 자바와 같은 객체지향 언어가 많이 사용되고 있다. 기존의 기술만으로는 객체지향언어의 검증에는 한계가 있다. 객체의 생성과 객체 간들의 통신은 객체지향언어의 기본적으로 검사해야할 속성이지만 기존 방법으로는 검사할 수가 없었다. 따라서 객체지향언어를 검증할 수 있는 기술에 대한 연구가 필요하다.

본 논문에서는 술어 추상화를 이용해서 자바 프로그램을 추상화 한 후, 모델체킹 도구인 SMV[9]를 이용해서 검사하는 방법에 대해서 설명하고, 자바 언어의 검증에 술어 추상화와 모델 체킹 기술이 어떻게 사용될 수 있는지 설명한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 C 언어를 대상으로 한 술어 추상화 기술에 대해서 설명한다. 3장에서는 자바 프로그램을 술어 추상화하는 방법에 대해서 설명하고, 4장에서는 추상화된 자바 프로그램을 SMV 입력언어로 변환하는 과정을 살펴본다. 5장에서는 결론과 향후 연구 방향에 대해서 살펴본다.

2. C 프로그램의 술어 추상화

본 장에서는 술어 추상화를 이용한 일반적인 모델 체킹 방법에 대해서 설명한다. 본 장에서 설명하는 방법은 SLAM에서 사용되는 방법이다. SLAM 은 마이크로 소프트에서 개발 중인 C 언어 모델 체킹 도구이다. 반례를 이용한 추상화 개선과 술어 추상화를 이용해서 C 언어 모델 체킹 도구에 최초로 적용한 도구이다. C 언어 코드를 입력하면 이진 변수로 구성된 이진 프로그램으로 변환을 하고, 이 프로그램에 대해서 모델 체킹을 수행한다. 여기서는 SLAM에서 술어 추상화를 수행하는 C2BP의 작동 방법에 관해서 살펴본다.

2.1 술어 추상화의 간단한 예제

술어 추상화는 술어가 주어졌을 경우 각각의 소스 코드 구문이 술어를 만족하는지 하지 않는지에 따라서 술어에 대응되는 이진 변수에 참 혹은 거짓 값을 입

력한다. 그림 1에서는 기본적인 술어 추상화 에 대해서 보여준다. 그림 1 에서는 (z=0) 과 (x=y) 로 구성된 술어가 주어졌을 때, 하나의 함수에 4개의 변수로 구성된 C 프로그램이 어떻게 이진 프로그램으로 변환되는지를 보여준다.

예제 프로그램에서 왼쪽은 C 프로그램이다. 오른쪽은 술어 {(z=0),(x=y)}를 적용한 이진 프로그램이다.

SLAM에서 검사하려는 속성은 12 번째 줄의 assert(0) 구문에 도달할 수 있는지이다. 오른쪽 소스 코드에는 b1 과 b2 두 개의 이진 변수가 있다. 이것은 각각 (z=0) 과 (x=y) 두개의 술어가 만족하는 것을 의미하는 변수이다. (z=0) 이라는 술어는 C 코드의 4번째 줄에서 만족한다. 따라서 이진 프로그램에서는 5번째 줄에서 b1이라는 (z=0) 에 대응 되는 변수에 1 을 할당하게 된다.

또한 C 코드의 8번째 줄에서 z=1 이라는 문장이 있다 이 문장은 술어 (z=0)을 만족하지 못하게 한다. 따라서 대응되는 이진 프로그램에서는 b1 에 0을 대입하게 된다.

```
1:int x,y,z,w;
2:void foo(){
3:      do{
4:            z=0;
5:            x=y;
6:            if(w){
7:               x++;
8:               z=1;
9:            }
10:     }while(x!=y)
11:     if(z){
12:           assert(0);
13:     }
14:}
```

```
1:decl b1,b2;
2:void foo()
3:begin
4:      do
5:            b1:=1;
6:            b2:=1;
7:            if(*)
8:             begin
9:                 b2:=h(0,b2)
10:                 b1:=0;
11:            end
12:     while(b2)
13:     if(!b1)
14:           assert(0)
15:     end
16:boolean H(e1,e2)
17:begin
18:     if(e1)then
19:           return(1);
20:     elsief(e2) then
21:           return(0);
22:     else return (*);
23:     fi
24:end
```

그림 1 예제 C 프로그램

2.2 술어 추상화 의 수행 방법

C 프로그램 P와 술어 집합 $E=\{\phi_1,...,\phi_n\}$ 가 주어졌을 때 C2BP는 P와 E를 이용해서 이진 프로그램을 생성한다. 생성된 이진 프로그램은 원래 프로그램 P 의 제어 구조를 가지고 있고 이진 변수만을 가지고 있는 프로그램이다. 이진 프로그램이 가지고 있는 변수 $V=\{b_1,...b_n\}$는 술어 의 개수와 동일하고 각각의 이진 변수 b_i 는 ϕ_i $(1\le i\le n)$ 에 대응된다. C 프로그램을 이진 프로그램으로 변환하기 위해 다음과 같은 중간 형태로 변환된다. (1) 모든 제어 구조는 if-else 와 goto 로 변환된다. (2) 모든 표현은 부작용(side-effect)이 없으며, 다중 포인터와 같은 표현은 사용하지 않는다. (3) 함수 호출은 최 상위 단계 에서만 행해진다.(z=x+f(y) 는 t=f(y); z=x+t 로 변환된다.)

술어 추상화를 수행하기 위해 우선 약한 사전조건(weakest precondition)에 대해서 살펴본다. WP(s,φ)는 문장 s에서 φ의 약한 사전조건을 의미한다. 사전조건이란 s 만족하기 전의 참 거짓 값이 s 가 끝날 때까지 만족되도록 하는 술어를 되돌려준다. WP(x=e,φ)는 φ 안에 있는 x 에 대해서 e 로 대치시킨다. 예를 들어

$$WP(x = x+1, x < 5) = (x+1) < 5 = x < 4$$

가 된다. 만일 예제에서 E={(x<5),(x=2)} 가 있고, 위의 예제로서 (x<4) 라는 술어가 나왔다고 하자. 하지만 (x<4)는 술어의 집합 E 안에 속해 있지 않다. 따라서 C2BP는 결정 절차에 따라서 E 안에 있는 술어 중에 어떤 것이 만족하는지 살펴보게 된다. 이 예제에서는 (x=2) => (x<4) 가 만족하게 된다. 따라서 (x=2) 가 x=x+1 이전에 만족했다면 (x<5) 는 이후에 만족하게 된다.

이진 변수들의 집합 V는 cube를 통해 표현할 수 있다. cube 는 $c_1 \wedge c_2 ... \wedge c_k$ 로 표현할 수 있고, 하나의 cube는 $c_i \in \{b_i, \neg b_i\}$ 이다. $\varepsilon(b_i)$는 대응되는 ϕ_i를 의미한다. FV(ϕ)는 ϕ=>ε(c)인 c를 찾아준다. 위의 예제에서 ε(FV(WP(x=x+1,x<5)))=ε(FV(x<4))=(x=2)가 된다. 다른 예제를 살펴보면 (p=p+x) 라는 구문에서 술어 E={(p<=0),(x==0),(r==0)} 가 있다고 할때, WP(p=p+x , p<=0)=(p+x<=0) 이 된다. 이 예제에서 ε(FV(WP(p=p+x,p<=0)))를 구하게 되면 (p<=0)∧(x==0)이 된다. 왜냐하면 (p<=0) => (p+x<=0) 타당한 식이 아니고 (x==0) => (p+x<=0)도 역시 타당한 식이 아니다. 하지만 (p<=0)∧(x==0)=>(p+x<=0)는 타당한 식이 된다. 따라서 결과로 (p<=0)∧(x==0) 가 나오게 된다.

구문 s 에서의 변수들의 값은 다음과 같이 할당되게 된다.

$$
\begin{aligned}
&b_1, ...b_n = \\
&choose(FV(WP(s,\phi_1)), FV(WP(s,\neg\phi_1))), \\
&, ..., \\
&choose(FV(WP(s,\phi_n)), FV(WP(s,\neg\phi_n)))
\end{aligned}
$$

가 된다. 위의 예제를 살펴보게 되면,

$$
\begin{aligned}
&\{p <= 0\}, \{x == 0\}, \{r == 0\} = \\
&\quad choose(\{p <= 0\} \wedge \{x == 0\}, \neg\{P <= 0\} \wedge \{x == 0\}), \\
&\quad choose(\{x == 0\}, \neg\{x == 0\}) \\
&\quad choose(\{r == 0\}, \neg\{r == 0\})
\end{aligned}
$$

이 된다.

이를 수행하기 위해 C2BP는 OCAML 로 구현이 되었고 Simplify 와 Vampyre 라는 외부 정리증명기의 도움을 받아서 추상화를 수행한다. C2BP에서 수행 시간은 외부 정리증명기의 호출 회수에 비례해서 수행시간이 오래 걸리게 된다. 따라서 정리증명기의 호출 횟수를 줄이는 것이 성능 향상에 많은 영향을 미친다.

3. 자바 프로그램의 술어 추상화

기존 SLAM 에서는 C 언어를 추상화한 이진 프로그램을 제시하고 이진 프로그램(Boolean Program) 상에서 검증을 수행했다. 본 논문에선 이진 프로그램을 확장한 객체 이진 프로그램(Object Boolean Program)을 제안하고 객체 이진 프로그램을 모델 체킹 도구인 SMV 로 변환 한다.

객체 이진 프로그램은 자바 프로그램의 기본적인 형태를 지니고 있다. 클래스와 클래스 안에 변수 와 메소드를 가지고 있다. 또한 모든 for 문과 while 등의 구문은 if 와 goto 문으로 변환 된다. 그리고 프로그램에 있는 모든 기본 변수는 이진 변수이다. 여기에서 기본 변수는 int , long 과 같은 기본형과 String 과 같이 일반적으로 많이 사용되는 클래스를 의미한다. 이런 기본 변수가 아닐 경우에는 클래스에 대한 참조(Reference)로서 사용된다.

자바 프로그램에 대한 술어추상화를 수행할 때 술어는 클래스 마다 주어진다고

가정한다. 현재까지는 외부 라이브러리를 사용하는 상황에 대한 고려는 하지 않는다.

자바 프로그램과 객체 이진 프로그램과의 가장 큰 차이는 객체의 생성에 있다. 자바 프로그램에서 객체를 생성하는 명령어인 new 는 객체 이진 프로그램에서는 없다. 대신에 객체 이진 프로그램 에서는 미리 객체가 생성이 된 상황을 설정한다. 단지 동작중인지 아닌지 만을 설정한다. 이런 방법을 채택한 이유는 대부분의 모델 체킹 도구에서는 동적 객체 생성을 지원하기 힘들다. 따라서 추상화를 수행 시 미리 객체의 생성된 상태를 객체 이진 프로그램으로 만들어 놓는다. 그림 2 와 그림 3 에서는 이 상황 설정에 대한 예제를 보여준다.

```
class OBP
{
	int i=0;
	public static void main(String[] args)
	{
		OBP a=new OBP();
		OBP b=new OBP();
		a.setI(5);
		b.setI(3);
	}
	 private void setI(int i){
		this.i=i;
	 }
}
```

그림 2 예제 자바 프로그램

```
class OBP{
	bool i;
	void main(){
		OBP a;
		OBP b;
		a.setI(true);
		b.setI(false);
	}
	void setI(bool i){
		this.i=i;
	}
}
```

그림 3 예제 객체 이진 프로그램

그림 2는 예제 자바 프로그램을 나타낸다. 그림 3에서는 OBP 라는 클래스에 (i==5)라는 술어를 적용시켰을 경우에 나오게 되는 객체 이진 프로그램을 보여준다. 자바 프로그램에서는 OBP 클래스와 클래스 내에서 a 와 b 라는 인스턴스를 생성한다. 이를 추상화한 객체 이진 프로그램에서는 OBP 클래스와 a,b 인스턴스가 있다. 그리고 (i==0)이라는 술어가 참인지 아닌지를 나타내는 i라는 이진 변수가 있다. 이를 통해서 객체가 생성된 상태를 이용해서 모델 체킹 도구로의 변환을 쉽게 하였다.

술어는 일반적인 자바 표현이다. 변수에 대한 값의 비교가 일반적인 술어의 형태이다. 객체 이진 프로그램에서 술어가 적용되는 범위는 개별 클래스가 될 수 있다.

4. SMV로의 변환

주어진 프로그램과 술어를 이용해서 객체 이진 프로그램을 생성하면 이 객체 이진 프로그램에서 다시 SMV 로 변환을 하여 검증을 수행한다. 그리고 프로그램에

서 만족해야할 속성은 변경된 프로그램에서 만족해야할 속성을 SMV 에 알맞게 변경하면 된다.

객체 이진 프로그램의 각각의 클래스는 SMV 의 모듈이 된다. 클래스 안에 있는 각각의 함수는 상태 전의 로 표현을 하게 된다. 이진 변수들은 바로 SMV에서 이진 변수와 대응되다.

프로그램은 제어 흐름과 데이터로 구성된다. 객체 이진 프로그램 역시 원래 프로그램의 제어흐름과 데이터를 포함하고 있다. 제어흐름을 SMV 로 변경하기 위해서 우리는 프로그램 카운터라는 변수를 새로 생성한다. 프로그램 카운터는 현재 어떤 함수에 어떤 구문을 실행중인지 나타내는 변수이다. 이를 통해서 현재 프로그램의 제어가 어느 부분에 있는지를 나타낸다. 그리고 데이터는 객체 이진 프로그램의 변수를 그대로 사용한다. 객체 이진 프로그램 에서는 변수의 종류가 두 가지 종류가 있다. 하나는 이진 변수와 하나는 다른 클래스에 대한 참조 이다. 이진 변수는 SMV 로 바로 변환이 가능하다. 그리고 다른 클래스에 대한 참조 역시 모듈로 선언된 클래스에 대해서 변수처럼 사용할 수 있다. 그리고 현재 모듈이 활성화된 상태인지 아닌지를 나타내는 이진 변수를 추가한다. 이 프로그램 카운터와 데이터 변수는 각각의 모듈마다 생성이 되게 된다.

프로그램 카운터의 구현 방법은 간단하다 객체 이진 프로그램 에서는 if 문과 goto 문과 함수 호출을 제외하면 항상 다음 구문으로 넘어가게 되어있다. 또한 if 구문을 만났을 경우 해당 구문이 참인지 아닌지에 따라서 프로그램 카운터를 분기하게 된다. 각각의 프로그램 카운터의 값에 따라서 데이터의 값이 변경이 된다. 메소드 호출의 경우 데이터변경이 메소드 호출을 한 클래스와 호출을 당한 클래스 두군데서 발생하게 된다. 호출을 당하게 되는 메소드에서는 어떤 메소드가 호출을 했는지에 대해 저장을 하고 있어야 한다. 메소드의 호출은 단순한 프로그램 카운터의 변경으로 진행된다. 그리고 메소드의 끝까지 다다랐을 경우 호출했던 함수로 되돌아가게 된다. 이 방법은 재귀 호출과 같은 경우를 다루기는 힘들지만, 일반적인 함수 호출 방법을 쉽게 표현할 수 있다.

```
Class a{
   bool a=false;
   bool b=false;
   public a(bool a){
      if(i){
         a=true;
      }else {
         b=true;
      }
   }
}
```

```
MODULE A(i)
VAR
  a:boolean;
  b:boolean;
  PC:{1,2,3,4};
ASSIGN
init(PC):=1;
next(PC):=case
    PC=1&i:2;
    PC=1&!i:4;
    1:PC;
init(a):=false;
next(a):=case;
    PC=2:true;
    1:a;
init(b):=false;
next(b):=case;
    PC=4:true;
    1:b;
```

그림 4 SMV로의 변환 예

그림 4와 그림 5에서는 간단한 객체 이진 프로그램에서 SMV 코드로 변환하는 과정

에 대해서 설명한다.

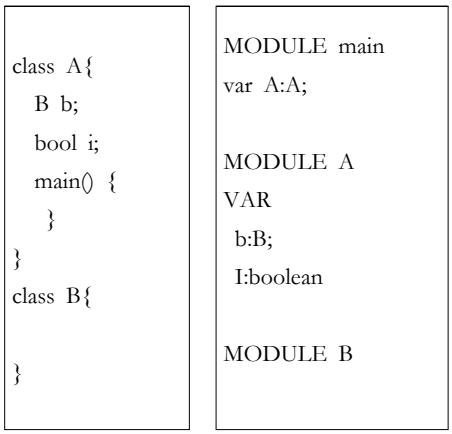

그림 5 SMV로의 변환 예

그림 4에서 if 문이 들어가 있는 메소드에 대해서 어떻게 프로그램 카운터가 사용되는지를 볼 수 있다. 현재 실행되고 있는 구문의 줄 번호를 프로그램 카운터에서 값으로 넣어놓고, 분기 나 함수 호출이 있을경우에 해당되는 프로그램 카운터로 이동하는 방식으로 변환했다. 그리고 해당되는 프로그램 카운터에서 데이터의 변경이 있을 경우 변경을 하도록 했다. 그림 5에서는 여러 개의 클래스가 어떻게 SMV 로 변경이 되는지 살펴본다. 하나의 클래스는 하나의 모듈로 대응이 되고 클래스에 대한 레퍼런스는 모듈 안에 변수로서 사용이 가능하다.

5. 결론 및 향후 연구 과제

본 논문에서는 C 언어를 대상으로 연구되었던 기존의 프로그램 모델 체킹 기술을 객체지향 언어로 확장했다. 그 결과 기존 연구에서는 제어 흐름 그래프 또는 자료 흐름 분석등을 사용하여 프로그램을 검증하였지만, 본 논문에서는 SMV를 이용하여 프로그램을 검증할 수 있게 되었다. 또한 기존 방법의 경우 오류 상태까지 도달 하는지에 대해서만 검사 가능했지만, 모델 체킹을 이용했기 때문에 CTL로 속성을 표현하고 검증할 수 있다.

하지만 기존 방법은 반례 기반 추상화 개선에 기반을 하고 있다[10,11]. 따라서 그 부분에 대한 연구와 구현이 필요하다. 또한 프로그램의 크기가 커질수록 검사해야될 상태의 크기는 커진다. 그리고 SMV의 경우 프로그램 검증을 위해 개발된 모델 체킹 도구가 아니기 때문에 비효율적인 부분이 있다. 따라서 프로그램 검증에 최적화된 모델 체킹 기술을 연구해야만 한다. 또한 현재는 자바 언어의 모든 특징을 검증하지 못한다. 멀티 스레드 시스템에 대한 검증과 외부 시스템에 대한 행위를 모델링해서 프로그램 자체만이 아닌 전체 시스템을 검증할 수 있는 방법이 연구되어야 한다.

참 고 문 헌

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

[2] E.M. Clarke, O. Grumberg, K.L. McMillan and X. Zhao, "Efficient Generation of Counterexamples and Witness in Symbolic Model Checking," in Proceedings of Design Automation Conference, pp.427-432, 1995.

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

[4] J. Corbett, et.al., "Bandera: Extracting Finite-state Models from Java Source

Code," in Proceedings of International Conference Software Engineering, 2000.
[5] T. Bal, R. Majumdar, T. Millstein and S.K. Rajamani, "Automatic Predicate Abstraction of C programs," SIGPLAN Notices, Vol. 36, No.5, pp.203-213, 2001.
[6] T.A. Henzinger, R. Jhala, R. Majumdar and G. Sutre, "Lazy Abstraction," in Proceedings of Principles of Programming Languages, pp.58-70, 2002.
[7] 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.
[8] 이태훈, 권기현, "프로그램 모델 체킹을 위한 술어 추상화 기법의 비교", 2004년 소프트웨어 공학 합동 워크샵 발표자료, pp.193-199, 2004.
[9] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking,"In the Proceedings of CAV'02, 2002.
[10] T. Ball and S.K. Rajamani, "Generating Abstract Explanations of Spurious Counterexamples in C programs," Technical Report MSR-TR-2002-09, 2002.
[11] E.M. Clarke, O.grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-Guided Abstraction Refinement," in Proceedings of Computer Aided Verification, pp154-169. 2000.

이 태 훈

1997년~2003년 경기대학교 전자계산학과(학사)
2003년~ 현재 경기대학교 전자계산학과 석사과정
관심분야 : 모델 검사, 소프트웨어 모델링, 정형기법

권 기 현

1985년 경기대학교 전자계산학과(학사)
1987년 중앙대학교 전자계산학과(이학 석사)
1991년 중앙대학교 전자계산학과(공학 박사)
1998년~1999년 독일 드레스덴 대학 전자계산학과 방문교수
1999년~2000년 미국 카네기 멜론 대학 전자계산학과 방문교수
1991년~현재 경기대학교 정보과학부 교수
관심분야 : 소프트웨어 모델링, 소프트웨어 분석, 정형 기법 등