논•문 14-2-4

함수프로그램의 모델검사를 위한 변환 (A tranformation for model-checking functional programs)

신 승 철
동양대학교 컴퓨터공학부
scshin@phenix.dyu.ac.kr

요 약

일반 프로그래밍 언어로 작성된 소프트웨어를 이미 구현된 모델 검사 기법을 이용하여 검증하기 위해서는 모델 검사기의 입력 언어가 프로그래밍 언어의 모든 기능을 표현할 수 있을 만큼 확장되거나 프로그래밍 언어로 작성된 프로그램을 모델 검사기 입력 언어의 명세로 자동 변환될 수 있어야 한다. 본 논문은 함수 프로그램이 기존의 모델 검사기를 이용하여 검증될 수 있는지에 대한 고찰로서 함수 프로그램을 SPIN 모델 검사기의 입력 언어인 Promela의 명세로 자동 변환하는 방법을 제시하고 함수 프로그램의 검증을 위해 temporal 논리 명세를 주는 방법에 대하여 설명한다. 함수 프로그램을 모델 검사하고자 할 때 가장 큰 어려움은 함수 프로그램이 명시적인 프로그램 포인트를 갖지 않는다는 것인데 이것은 모델 검사기 입력 명세로 자동 변환될 때와 함수 프로그램에 검증하고자 하는 temporal 논리 특성을 부여할 때에 모두 해결되어야 한다. 우리는 함수를 프로세스로 변환하는 방법을 이용하여 자동 변환을 구현하고 함수 호출 포인트에 기반하여 temporal 논리식을 부여한다.

1. 서 론

모델 검사(model checking)와 같은 유한 상태 검증 기법(finite-state verification)은 순차 시스템이나 동시 시스템에서 발생하는 논리상의 매우 미묘한 결점까지도 발견해내는 능력 때문에 시스템 검증 분야에서 상당한 호응을 얻고 있다. 이러한 기법을 이용하는 도구의 지원은 현재 하드웨어와 정보통신 산업 분야에서 개발 단계의 중요한 요소로 결합될 정도의 성숙도를 보이는 수준까지 와 있다. 이러한 도구들을 소프트웨어 시스템 검증에 적용하는 것은 여러 가지 이유로 쉽지 않다[4,13].

모델 검사는 일반적으로 (1) 대상 시스템에 대하여 검사하고자 하는 특성을 시제 논리식(temporal logic formula)으로 표현하고 (2) 대상 시스템의 모든 동작을 나타내는 상태 전이 모델(state-transition model)을 구성

한 후에 (3) 그 모델이 시제 논리식으로 주어진 명세를 만족하는 지를 검사하는 과정으로 구성된다[11]. 일반 프로그래밍 언어로 작성된 프로그램을 기존의 SMV[9]나 SPIN[6]과 같은 모델 검사기를 이용하여 검증하기 위해서는 모델 검사기의 입력 언어가 일반 프로그래밍 언어의 모든 기능을 표현할 수 있도록 확장될 필요가 있는데 이 경우 이미 구현된 모델 검사기 자체를 다시 구현해야 하는 어려움이 있다[7].

또 다른 접근 방법은 일반 프로그램을 모델 검사기의 입력 언어로 자동 변환하는 것이데 프로그램 변환과 분석에 대한 연구 성과들을 최대한 이용하고 모델 검사기의 새로운 설계와 복잡도의 증가를 피하는 방법으로 이용될 수 있다. 이 방법의 대표적인 결과로는 Bandera 시스템[3,5]과 JPF[13] 등이 있으며 이들 시스템들은 일반 프로그래밍 언어가 가지는 다양한 타입구조와 제어구조를 다소 제한적인 모델 검사기 입력 언어로 표현하고 검사 시간을 좌우하는 상태 모델의 크기를 줄이기위해 요약해석(abstract interpretation)에 기반한 데이터의 요약법과 프로그램 분할(slicing) 및 부분 연산(partial evaluation)에 기반한 제어의 요약법을 이용한다.

본 논문은 지금까지 주로 절차적인 언어로 작성된 프로그램에 대하여 이루어지고 있는 프로그램 모델 검사에 대한 연구가 어떻게 하면 함수 프로그램에 적용될 수 있을까에 대한 고찰로서 SPIN 모델 검사기를 이용한 함수 프로그램의 검증에 대하여 설명하고자 한다. 함수 프로그램을 기존의 모델 검사기 SPIN을 그대로 이용하여 검증하고자 하면 먼저 주어진 함수 프로그램을 SPIN의 입력 언어인 Promela로 변환하여야 한다. 절차적인 프로그램에 나타나는 프로그램 포인트에 상응하는 개념이 함수 프로그램 상에는 명시적으로 나타나지 않기 때문에 적절한 변환 과정을 통하여 프로그램 포인트를 갖는 절차적인 형태로 표현할 필요가 있다.

본 논문에서는 CPS 변환[1]과 람다 리프팅[8]을 이용하여 각 함수의 절차적인 표현을 얻은 후에 각 함수를 Promela의 프로세스로 변환한다. 또한 주어진 함수 프로그램에 대하여 검사하고자 하는 특성들을 시제 논리식으로 표현하고자 할 때에 기준이 되는 프로그램 포인트를 함수 호출 포인트를 이용하여 표현한다.

본 논문의 나머지 부문은 2장에서 모델 검사기 SPIN의 입력 언어 Promela를 간단히 소개하고 함수 프로그램에서 검사하고자 하는 특성들을 표현하는 방법을 설명한 후에 3장에서는 함수 프로그램을 Promela 프로세스로 변환하는 과정을 설명하고 4장에서 결론 및 향후 연구 방향으로 끝을 맺고자 한다.

2. 모델 검사와 함수 프로그램

2.1 Promela와 LTL

모델 검사기 SPIN은 주어진 시스템의 정확성을 정형적으로 서술된 성질(property)에 대하여 분석, 검증하는 도구이다. 주어진 대상 시스템은 C 언어를 닮은 Promela 언어로 표현되고 검증하고자 하는 성질들은 시제 논리 LTL(Linear Temporal Logic)에 의해 표현된다.

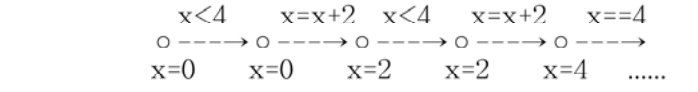
Promela 프로그램은 매우 제한적인 언어

이지만 소프트웨어나 하드웨어, 물리적인 객체 등을 포함하는 시스템들의 상태 변화를 정형화할 수 있다. 하나의 Promela 프로그램은 여러개의 Promela 프로세스로 구성되는데 각 프로세스는 채널이나 공유 변수를 이용하여 서로 통신하면서 동시에 수행될 수 있다. Promela의 구문과 시멘틱스에 관한 자세한 정보는 여기선 생략하기로 한다.

SPIN은 시스템의 실행 경로에 대한 성질을 서술할 수 있는 시제 논리 LTL을 제공한다. LTL은 기본적으로 두 가지 논리식을 갖는다: []P 는 주어진 실행 경로의 모든 상태에서 P가 참임을 나타내고 <>P 는 실행 경로에는 P가 참인 상태가 존재한다는 것을 나타낸다. 예를 들어 다음과 같은 Promela 프로그램을 생각해 보자.

```
int x;
proctype P()
{do
:: x == 4 -> x = 0
:: x < 4 -> x = x + 2
od}
```

이 프로그램은 다음과 같은 실행 경로를 갖는다.



여기에 'x는 항상 4보다 크지 않다' 또는 'x가 0보다 크면 반드시 x는 4가 된다'와 같은 성질을 다음과 같은 논리식으로 표현할 수 있다.

```
[]!(x>4)
[](x > 0 -> <> x == 4)
```

2.2 함수 프로그램의 LTL 명세

함수 프로그램은 앞서 보여준 Promela 프로그램에서 나타난 것과 같은 프로그램 상태를 표현할 수 있는 프로그램 포인트가 존재하지 않는다. 따라서 프로그램 포인트에 기반하여 프로그램 상태를 표현하고 이에 대한 특성을 말하는 것이 불가능하다. 게다가 우리는 Promela 프로그램 상에서의 특성이 아닌 주어진 함수 프로그램 상의 특성을 검사할 특성으로 표현하여야 한다.

하나의 함수 프로그램은 여러개의 작은 함수들로 이루어지며 우리는 각 함수에 대한 특성을 검사하고자 한다. 좀 더 정확하게는 함수의 정의에 대해서가 아니라 호출된 함수의 각 인스턴스에 대하여 그 특성을 검사하고자 한다. 왜냐하면 모델 검사에서 검사되는 대상들은 실행 경로상의 상태들이기 때문에 여기에 착안하여 함수 호출들의 시퀀스를 함수 프로그램의 실행 경로로 간주하고 각 함수 인스턴스들의 인수와 결과값에 대하여 검사할 특성을 만들 수 있다. SML 구문으로 된 다음의 프로그램을 생각해 보자.

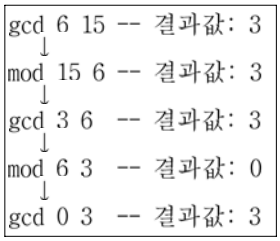
```
let fun mod x y = x - (x/y)*y
    fun gcd a b = if a = 0 then b
                  else gcd (mod b a) a
in gcd 6 15
end
```

이 프로그램의 실행에 대하여 프로그래머가 생각하는 값호출 실행 경로는 다음과 같다.

```
gcd 6 15
|
mod 15 6
|
3(mod 15 6의 결과값 산출)
|
gcd 3 6
|
mod 6 3
|
0(mod 6 3의 결과값 산출)
|
gcd 0 3
|
3(gcd 0 3의 결과값 산출)
|
3(gcd 3 6의 결과값 산출)
|
3(gcd 6 15의 결과값 산출)
```

이러한 실행 경로상에서 각 함수 인스턴스에 대하여 검사할 수 있는 특성은 함수 호출 인수와 그 결과값에 대한 성질이다. 따라서 각 호출 인스턴스에 대하여 그 인

수들과 결과값을 대응시켜야 하므로 함수 호출 인스턴스들만의 실행 경로를 만들고 각 인스턴스의 결과값을 따로 연결시키면 다음과 같은 실행 경로를 나타낼 수 있다.



각 함수 f에 대하여 첫 번째 인수는 f.1, 두 번째 인수는 f.2와 같이 나타내고 그 결과값은 f.r로 표기하기로 하자. 이제 각 인스턴스에 대한 호출 인수들의 특성, 인스턴스의 결과값에 대한 특성, 인스턴스의 인수와 결과값의 관계에 대한 특성을 말할 수 있다. 즉,

gcd.1 == 0

mod.1 > mod.2

와 같은 단순식(atomic formula)을 만들 수 있다. 또한 각 함수 호출 인스턴스에 대하여 또는 함수의 재귀적인 호출로 인한 인스턴스의 시퀀스에 대하여 LTL 시제 논리 연산자를 이용하여 다음과 같은 LTL 명세를 만들 수 있다.

<>gcd.1 == 0

[](gcd.1 == 0) -> (gcd.2 == gcd.r)

여기서 함수 gcd는 하나의 호출 인스턴스를 나타낸다. 즉, 첫 번째 LTL 명세는 gcd 함수의 호출 인스턴스중에 첫 번째 인수가 0이 되는 것이 반드시 존재함을 나타내고 두 번째는 gcd의 모든 인스턴스에 대하여 첫 번째 인수가 0이면 그 인스턴스의 결과값과 두 번째 인수가 같다는 것을 의미한다.

3. 함수를 위한 Promela 프로세스

주어진 함수 프로그램을 Promela 프로그램으로 변환하는 과정은 세가지 단계로 나누어 설명할 수 있다. 먼저 함수 프로그램에 실행 단계가 명시적으로 나타나게 하기 위하여 CPS(Continuation-Passing Style)로 변환하고 모든 함수들이 내포되지 않고 최상위에서 정의되도록 내포된 함수들을 최상위로 끌어올리기 위해 람다리프팅(λ-lifting)을 수행한 후에 각 함수들을 Promela 프로세스로 변환한다.

3.1 함수의 변환

CPS 변환과 람다리프팅은 각각 Appel의 알고리즘[1]과 Johnsson의 알고리즘[8]을 이용한다. 여기서는 잘 알려져 있는 알고리즘들을 설명하기 보다 그 다음 단계에서 Promeal 프로세스로의 변환을 설명할 때 필요한 예를 가지고 CPS 변환과 람다리프팅을 설명한다.

먼저 다음과 같이 간단한 SML 함수를 생각해 보자.

```
fun fac n = if n = 0 then 1
            else n * fac (n-1)
```

이 함수에 명시적인 실행 단계를 주기 위해서 다음과 같은 CPS 함수로 변환할 수 있다.

```
fun fac n k = if n = 0 then k 1
              else let fun k' m = k (n*m)
                   in fac (n-1) k'
                   end
```

여기서 k는 함수 fac의 continuation이고 k'은 새로운 함수 호출 인스턴스의 새로운 continuation이다.

다시 함수 k'을 최상위로 끌어올리기 위해 람다리프팅을 수행하면 다음과 같은 두 개의 독립된 함수를 얻을 수 있다.

```
fun fac n k = if n = 0 then k 1
                  else fac (n-1) (k' n k)
fun k' n' k'' m = k'' (n'*m)
```

3.2 함수를 프로세스로 변환하기

동시 프로세스 이론의 기조로서 연구된 파이계산법(π-calculus)에서 Milner는 함수가 프로세스로 표현가능하다는 것을 보여주기 위하여 람다계산법(λ-calculus)을 파이계산법으로 표현하는 해석 방법[10]을 제시하였는데 이것에 기반하여 함수 프로그램의 함수들을 파이계산법의 프로세스로 변환하는 기법이 제시되었다[12]. 우리는 이 변환 기법을 수정하여 함수 프로그램을 Promela 프로그램으로 변환하는 방법을 구현하였다.

기본적으로 CPS 변환과 람다리프팅을 거친 함수 프로그램의 각 함수는 Promela 프로그램의 각 프로세스로 변환된다. if나 기본 연산자들은 Promela의 해당 구문으로 일대일 대응을 시킬 수 있고 다양한 데이터 타입에 대해서는 그렇지 못하지만 본 논문에서는 그 부분의 논의는 제외하기로 한다. 그러면 여기서는 함수를 Promela 프로세스로 변환하는 데에 결정적인 부분인 함수 정의와 함수 호출에 대한 변환 방법을 설명한다.

변환에 대한 기본적인 생각은 각 프로세스는 함수의 시뮬레이션을 보여주는데 각 인수를 하나씩 채널을 통해 전달하고 이때 다음 인수의 전달을 맡을 채널을 부가적인 채널을 통해 전달한다는 것이다. 함수의 인스턴스에 대응되는 프로세스가 run 구문을 통해 생성될 때 첫 번째 인수을 전달받을 채널도 함께 생성된다.

다음과 같은 함수 정의에 대하여

fun F x_0 x_1 ... x_{n-1} = e

우리는 다음과 같은 Promela 프로세스를 생성한다.

```
proctype F (chan f0) {
f0?x0,f1;
f1!f2 -> f2?x1,f3;
...
f2i-1!f2i -> f2i?xi,f2i+1;
...
f2n-5!f2n-4 -> f2n-4?xn-2,f2n-3;
f2n-3!f2n-2 -> f2n-2?xn-1;
statements for e
}
```

여기서 x_0, ..., x_{n-1}, f_0, ..., f_{2n-2}는 모두 바운드임을 알 수 있고 모든 f_i는 새로 선언된 채널 이름이다. 함수의 인수 전달은 프로세스의 비동기식 통신 방법에 의해 표현되고 있으며 이것은 장점은 함수의 부분 적용을 구현하기가 수월하다는 것이다.

또한 다음의 함수 호출에 대하여

F a_0 a_1 ... a_{n-1}

다음과 같은 Promela 문장들로 변환된다. 여기서 c_0, ..., c_{2n-2}는 모두 새로 선언된 채널

```
run F(c0) -> c0!a0,c1;
c1?c2 -> c2!a1,c3;
...
c2i-1?c2i -> c2i!ai,c2i+1;
...
c2n-5?c2n-4 -> c2n-4!an-2,c2n-3;
c2n-3?c2n-2 -> c2n-2!an-1;
```

이름들이다. 이 변환에서 유의해야 할 것은 알려진 함수들 즉, 위의 예에서 함수 fac나 k'에 대한 호출은 이 변환 방법으로 변환되지만 알려지지 않은 함수의 호출은 예를 들어 k'' (n' * m)와 같은 함수 호출은 그대로 k'' 채널로 (n' * m)의 결과를 전달하는 것

으로 변환된다. 이것은 다음의 예에서 분명하게 보여질 것이다.

이제 앞에서 CPS 변환과 람다리프팅을 거친 fac 함수에 대한 Promela 프로세스를 본여준다. 두 함수 fac와 k′은 각각 Promela

```
proctype Fac(chan fac0) {
fac0?n,fac1;
fac1!fac2 -> fac2?k;
if
:: (n == 0) -> k!1
:: !(n == 0) -> run k′(p4);
                p4!n,p5;
                p5?p6;
                p6!k,p7 -> p7?q;
                run fac(p8, r1);
                p8!(n-1),p9;
                p9?p10 -> p10!q
fi
}

proctype K′(chan k′0) {
k′0?n′,k′1;
k′1!k′2 -> k′2?k′′,k′3;
k′3!k′4 -> k′4?m;
k′′!(n′*m)
}
```

프로세스 Fac와 K′으로 변환된다. 함수 프로그램 상의 변수들의 이름은 그대로 유지되고 다만 인수 전달을 위해 필요한 채널들만이 새롭게 선언되어 사용된다. 함수 fac와 k′의 호출과 함수 k′′의 호출이 서로 다르게 변환되었음에 유의하자.

4. 결 론

본 논문은 함수 프로그램이 모델 검사기 SPIN을 이용하여 검증되기 위해 어떻게 SPIN의 입력 언어인 Promela 프로세스로 변환될 수 있는가를 보여주었다. 또한 프로그램 포인트를 명시적으로 가지지 않은 함수 프로그램에 대하여 검증가능한 시제 논리식을 작성하는 방법에 대하여 설명하였다. 이것들은 함수 프로그램의 고유한 특성상 다른 패러다임(예를 들어 Java와 같은)의 프로그램을 모델 검사하는데에 요구되는 과정들에 부가적으로 필요한 전처리 과정으로 보여질 수 있다. 함수 프로그램의 완전한 모델 검사를 위해서는 본 논문에서 보여준 방법들의 개선과 함께 데이터 요약법과 제어 요약법을 이용한 상태 공간의 축소, 함수 프로그램과 Promela 프로세스간의 프로그램 포인트 대응 관계 등의 문제에 접근해야 한다. 이것들은 요약 해석과 프로그램 분할, 부분 연산, 흐름 분석 등의 기법들이 요구되고 어떤 패러다임의 프로그램을 모델 검사하더라도 공통적으로 필요한 부분이기도 하다[13]. 또한 기존의 유한 상태 모델 검사가 가지는 한계인 루프나 재귀 호출의 문제를 완전하게 해결하는 방법으로 무한 상태 모델 검사[2]가 한가지 대안으로 떠오르고 있지만 아직은 무한 상태 모델 검사 알고리즘이 일반 프로그램의 복잡한 구조를 수용할 수 있다고 알려져 있지 않다. 제한된 형태의 프로그램에 대하여 무한 상태 모델 검사가 가능하다고 하더라도 현재 본 논문과 이와 유사한 형태의 프로그램 모델 검사를 위한 변환과 분석 등의 과정이 마찬가지로 요구될 것이다.

참고문헌

[1] A.Appel, Compiling with Continuations, Cambridge University Press, 1992.

[2] O. Burkart and B. Steffen, Model checking the full modal mu-calculus for infinite sequential processes, ICALP'97, LNCS 1256, pp. 419--429. 1997.

[3] Matthew B. Dwyer and John Hatcliff, Slicing Software for Model Construction, Proc. ACM SIGPLAN Partial Evaluation

and Program Manipulation, January, 1999.
[4] C. Gunter, J. Mitchell, and D. Notkin, Strategic directions in software engineering and programming languages, ACM Computing Surveys, vol. 28, no. 4, pp.727-737, 1996.
[5] John Hatcliff, Matthew B. Dwyer, Shawn Laubach, and David Schmidt, Staging Static Analyses Using Abstraction -based Program Specialization, LNCS 1490, 1998.
[6] G.J. Holzmann, The model checker spin, IEEE Transactions on Software Engineering, 23(5):279-294, 1997.
[7] F. Huch, Verification of Erlang programs using abstraction interpretation and model checking, ICFP'99, pp.261-272, 1999.
[8] T.Johnsson, Lambda Lifting: transforming programs to recursive equations, Proc. Conference on Functional Programming and Computer Archtecture, pp.190-205, 1985.
[9] K.McMillan, Symbolic Model Checking-an approach to the state explosion problem, PhD thesis, CMU, 1992.
[10] Robin Milner, Functions as Processes, Automata, Languages and Programming, LNCS#443, pp.167-180, 1990.
[11] Markus M"uller-Olm, David Schmidt and Bernhard Steffen, Model Checking: a tutorial introduction, Proc. 6th Static Analysis Symposium, pp.330-354, 1999.
[12] S.C.Shin and W.H.Yoo, CHORE: A Process Network with Graph Reduction, Proc. Conference on Applied Modelling, Simulation and Optimization, pp.25-30, 1995.
[13] W.Visser, K.Havelund, G.Brat, and S.Park, Model Checking Program, Proceedings of Conference on Automated Software Engineering, 2000.

신승철
1983년~1987년 인하대학교 전자계산학과(이학사).
1987년~1989년 인하대학교 전자계산학과(이학석사).
1992년~1996년 인하대학교 전자계산공학과(공학박사).
1999년 ~ 2000년 Kansas State Univ. 박사후과정 연구원
1996년~현재 동양대학교 컴퓨터공학부 조교수
관심분야는 프로그래밍 언어, 함수 언어, 동시성 이론, 프로그램 분석, 프로그램 검증, 정형 기법.