기호실행을 통한 자동 생성된 불변식의 검정

Validating Inferred Invariants using Symbolic Execution

홍신, 김문주

한국과학기술원 전산학과
대전광역시 유성구 구성동 373-1
hongshin@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

Matt Staats

한국과학기술원 웹사이언스공학전공
대전광역시 유성구 구성동 373-1
staatsm@kaist.ac.kr

요약: 불변식(invariant) 형태의 요구사항 정형 명세 작성은 올바른 소프트웨어의 개발과 검증에 유용하다. 자동 불변식 추출은 프로그램 분석을 통해 개발자가 의도한 것으로 예상하는 불변식을 자동으로 유추하는 기법이다. 이러한 기술은 개발자의 소프트웨어의 요구사항 명세작성 과정을 지원하지만, 개발자가 추출된 불변식의 정확도를 직접 판별해야 한다는 문제점을 가지고 있다. 본 논문에서는 이러한 문제점을 보완하기 위하여 기호실행(symbolic execution)을 통하여 도구가 추출한 불변식을 자동으로 검정하는 프레임워크를 제안한다. 제안한 기법을 기호실행 도구인 Symbolic Java PathFinder 를 이용하여 구현하여 실험한 결과, Daikon 도구가 유추한 불변식에 대하여 효과적으로 불변식 검정을 수행함을 확인할 수 있었다.

핵심어: 자동 불변식 추출, 기호 실행, 자동 테스팅

1. 서론

정확하고 체계적인 소프트웨어 개발과 품질관리를 위해서 소프트웨어 명세 작성은 소프트웨어 개발에 필수적인 과정이다. 프로그램 불변식(program invariant)은 특정 코드의 동작을 정형적으로 기술하는 소프트웨어 명세의 한 종류이다. 불변식은 프로그램 코드와 동일한 형태로 작성되기 때문에 개발자에게 친숙하며 분명한 의미전달이 가능하다는 장점이 있다. 또한 불변식을 바탕으로 자동 테스트케이스 생성, 정형 검증 등 자동화된 프로그램 정확도 검사 수행을 가능하게 한다는 점에서 불변식은 높은 활용도를 가진다.

불변식 자동 추출(automatic invariant inference)은 프로그램 분석을 통하여 개발자가 의도한 것으로 예상되는 불변식을 유추하여 개발자의 소프트웨어 명세작성 과정을 돕는 기법이다. 최근 고신뢰도 소프트웨어의 품질관리 요구증가와 자동화된 기법을 통한 소프트웨어 테스팅 기법의 발달로 말미암아 불변식 작성에 대한 수요가 증가하고 있다 [1]. 하지만 프로그램의 동작을 총체적이고 정확하게 기술하는 불변식을 개발자가 직접 작성하는 데에는 큰 비용이 소요된다. 이러한 소프트웨어 명세과정을 자동화하기 위하여, 불변식 자동 추출 기법은 프로그램의 동작을 표현하는 조건식을 프로그램 코드 혹은 실행로부터 자동으로 발견함으로써 개발자가 불변식 작성하는 작업을 지원하는 자동화 기법이다. 최근에서는 프로그램 내의 다양한 변수 사이의 수식 관계를 자동으로 추출하는 기법과 도구들이 연구 개발되고 있다 [2, 3, 4, 5].

불변식 추출 기법을 통해 소프트웨어 명세작성의 자동화가 전망되고 있지만, 현재 개발된 기법은 프로그램 분석 기법의 한계로 잘못된 불변식을 유추할 수 있다. 틀린 불변식이 소프트웨어 명세로 사용되는 경우, 이를 이용하는 잘못된 테스트 수행 등을 일으킬 수 있으므로 실제 소프트웨어 명세작성 과정에서는 자동으로 추출된 불변식이 프로그램 동작을 올바르게 표현하는지 개발자가 직접 검정하는 작업이 불가피하다. 따라서 도구가 생성한 불변식을 개발자가 직접 검정해야 하는 비용이 필요하다는 한계점을 가진다.

본 논문에서는 동적 생성 기법을 통하여 추출된 불변식을 기호실행 기법을 통하여 자동으로 검정하는 프레임워크를 제안한다. 또한, 제안한 프레임워크가 실제로 현재 불변식 자동 추출 기법의 한계점을 효과적으로 보완하는지 확인하기 위한 실험 결과를 소개한다. DySy [3]는 동적 불변식 추출 기법의 개선을 위해 기호실행을 적용했다는 점에서 본 연구와 유사점이 있으나, DySy 는 불변식 종류의 확장을 목표이고 본 연구는 불변식의 정확도 검정에 목표라는 점에서 차이가 있다.

섹션 2 에서는 불변식 자동 추출 기법을 개관하고 개선점을 논의한다. 섹션 3 에서는 기호실행 기법을 활용한 불변식 자동 검정 프레임워크를 제시한다. 섹션 4 에서는 앞서 제시한 프레임워크를 이용하여 Daikon 도구가 추출한 불변식을 자동으로 검정한 결과를 보고하고 논의한다. 마지막으로, 섹션 5 에서는 연구 결과를 종합하고 향후 연구 방향을 논의한다.

2. 동적 자동 불변식 추출

불변식(invariant)은 프로그램의 모든 실행이 특정 지점에서 항상 만족하는 조건을 기술한 식이다. 불변식에는 명세하는 프로그램 조건의 형태에 따라 함수/메쏘드 실행전조건(pre-condition), 실행후조건(post-condition), 반복문불변식(loop invariant), 클래스불변식(class invariant) 등의 종류가 있다. 불변식 자동 추출(automatic invariant generation)은 입력된 코드 혹은 실행을 분석하여 프로그램의 모든 실행에서 만족하는 것으로 보이는 조건을 유사 불변식(likely invariant)으로 자동 생성하는 기법이다. 동적 불변식 자동 추출(dynamic invariant inference)은 프로그램의 실행에서 변수 값의 변화를 바탕으로 불변식을 귀납적으로 유추하는 기법을 뜻한다. 특히 이 기법은 프로그램의 실행 정보를 분석하는 동적 분석 기법을 기반으로 하므로 규모가 큰 실제 프로그램에 대해서도 효과적인 적용이 가능하다.

Daikon [2]은 대표적인 동적 불변식 자동 추출 도구로써 C, Java, Perl 로 작성된 프로그램을 Daikon 프레임워크 내에서 테스트 실행을 수행할 경우, 추가적인 수정 없이, 불변식 추출 기능을 수행한다. Daikon 은 실행 가능한 도구가 공개되어 있어서 소프트웨어 공학적 연구에서 널리 응용되고 있는 대표적인 도구이다.

Daikon 은 분석 대상 프로그램의 실행 정보를 바탕으로 함수(혹은 메쏘드)의 실행후조건 불변식과 클래스불변식을 유추한다. Daikon 에는 조건식의 종류별로 유추 작업을 수행하는 모듈이 존재한다. 조건식의 종류는 연관된 변수의 개수와 수식의 형태로 구별되며, 기본적으로 유추 기능을 제공하는 수식의 형태로는 (1) 메쏘드의 입출력 파라미터와 공역 변수 간의 선형 대수식(linear arithmetic) (2) 배열의 성질(크기, 순서)에 관한 대수식이 있다. 각 종류의 식에서 연관된 변수의 각각의 유추 모듈은 입력된 프로그램 실행에서 변수 값 변화를 분석한다. 유추 모듈은 관찰한 실행에 대해서 일관성을 가지고 있으며 (관찰한 실행은 불변식을 항상 만족함) 통계적인 방법을 통해 일정 수준 이상의 신뢰도를 가지는 불변식만을 생성한다. 하지만, Daikon 은 다음의 원인으로 인하여 추출하는 불변식의 정확도에 한계를 가진다.

- 잘못된 불변식이 유추되는 경우

  Daikon 은 사용자가 테스트 수행을 통해 입력하는 프로그램 실행 정보만을 이용해 귀납적 추론을 수행하므로 추출하는 불변식의 정확도는 입력되는 프로그램 정보에 종속된다. 사용자가 프로그램의 다양한 동작을 입력하지 않은 경우, Daikon 은 관찰하지 못한 프로그램의 동작을

```
00  int max(int a, int b) {
01      int r ;
02      if (a > b)
03          r = a ;
04      else
05          r = b ;
06      return r ;
07  }
```

그림 1: 예제 프로그램

  올바르게 기술하지 못하는 잘못된 불변식을 유추할 수 있다.

- 프로그램에 존재하는 불변식을 빠뜨리는 경우

  Daikon 이 제공하는 불변식 유형의 한계로 말미암아 개발자가 의도한 다양한 요구사항의 추출이 불가능하다. 예를 들어서, Daikon 은 복잡한 자료 구조를 가지는 프로그램에서 중요한 불변식의 일종인 메모리 참조 간의 별칭 관계(aliasing)는 일반적으로 추출하지 못한다.

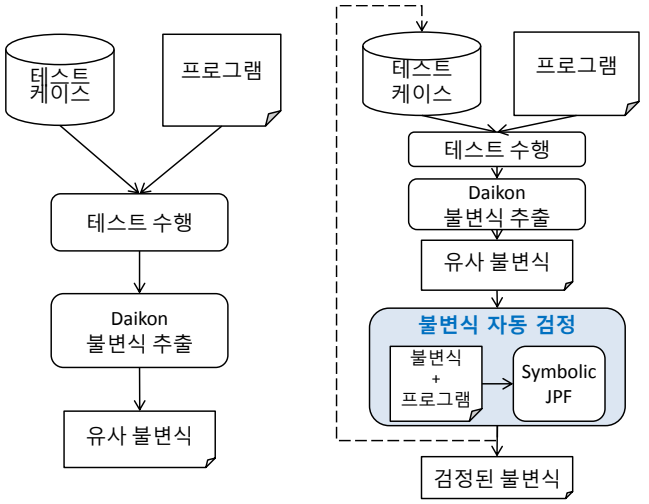
두 가지 중 첫 번째로 설명한 경우와 같이, 잘못된 불변식이 생성되는 경우에는, 불변식 생성에서 발생한 오류가 테스팅이나 정확도 검증의 오경보(false positive)로 이어져 추가적인 비용의 손실을 일으키는 위험성을 가지고 있다. 따라서 개발자가 자동으로 유추되는 불변식이 올바른 조건인지 잘못된 조건인지 직접 분별하는 검정(validation) 단계가 필요하다. 이러한 작업을 정확하게 수행하기 위해서는 개발자의 많은 노력과 시간이 필요하므로, 자동화 불변식 추출 기법을 유용하고 효율적으로 소프트웨어 개발 과정에 적용하는데 주요한 문제점이 될 수 있다.

본 연구에서는 첫 번째 경우(잘못된 불변식)에 집중하여, 자동 불변식 추출 기법의 정확도를 보완하는 기법을 제시한다. 예제를 통해서 잘못된 불변식이 유추되는 경우를 구체적으로 살펴보자.

그림 1 은 두 변수를 비교하여 최곳값을 반환하는 간단한 함수의 코드다. 이 때 프로그램 실행 입력을 위한 테스팅에서 `max()`의 두 인자(`a, b`)의 값이 항상 양수고 같은 값을 갖는 경우가 없었다고 생각해보자. Daikon 은 변수 값을 바탕으로 귀납적인 추론을 수행하므로 "함수의 반환 값에 2 를 곱하면 인자 `a` 와 `b` 를 더한 값보다 크다"라는 식을 불변식을 결론적으로 생성할 수 있다. 하지만 이러한 불변식은 식은 `a` 와 `b` 가 같은 값을 가지는 실행에서 위반이기 때문에 틀린 불변식이다. 만약 이와 같이 틀린 식이 테스팅이나 검증에 사용될 경우 올바른 실행을 오류 실행으로 오경보 하는 문제를 발생시킬 수 있다.

3. 기호실행을 통한 불변식 자동 검정

본 섹션에서는 기호실행(symbolic execution)으로



(a) 일반적인 동적 불변식 자동 추출 프레임워크 (b) 불변식 자동 검정을 통한 개선

그림 2: 불변식 자동 추출 및 검정 프레임워크

Daikon 도구가 추출한 불변식을 자동으로 검정하는 프레임워크를 제안한다. 자바 프로그램을 대상으로 기호실행을 수행하기 위해서 본 프레임워크에서는 Symbolic Java PathFinder(SPF)를 사용하였다 [4]. 그림 2 는 기존의 동적 불변식 추출 기법 사용 과정(왼쪽)과 본 논문에서 제시한 프레임워크를 통해 개선한 과정(오른쪽)을 묘사한 개념도이다. 왼쪽의 그림에서 보는 바와 같이, Daikon 은 분석하는 프로그램의 동작 정보를 사용자가 입력하는 제한된 실행에서만 추출한다는 한계점을 가지고 있다. 오른쪽 그림에 나온 프레임워크는 프로그램 코드를 이용한 기호실행을 통해 특정 조건을 만족하는 테스팅을 능동적인 방법으로 추가로 수행함으로써 동적 불변식 생성 기법의 정확도를 보완한다. 이때 유용한 실행을 생성하기 위하여 기호실행 기법을 활용하고 있다. 특정 불변식 검증을 목적으로 생성된 테스트케이스는 이후 테스팅 과정과 불변식 추출과정에 계속 활용되도록 한다.

SPF 는 모델 검정 도구인 Java PathFinder 에 기호실행 기능을 확장한 도구로써, 자바 프로그램을 대상으로 기호실행 기능을 제공하는 안정적인 공개 소프트웨어 도구이다. SPF 는 Java PathFinder 프레임워크 상에서 실행되는 자바 프로그램을 추적하여 같은 경로의 기호실행을 수행한다. SPF 는 메쏘드 호출에서 인자로 사용된 변수 중 사용자가 명시한 변수들의 값을 기호값으로 추적하고 이를 이용해 프로그램 실행 경로식(path condition)을 기호식(symbolic formula)의 형태로 구한다. SPF 는 각각의 프로그램 실행 경로에 대해 기호식을 구하고 이를 SMT-solver 와 자동 계산기로 해를 구하여, 특정 경로 조건을 만족하는 테스트 입력값을 생성하고 이를 이용해 테스팅을 수행한다. 분석 대상 프로그램에 단언구문(assert statement)가 있는 경우에는 해당 구문에 도달하는 경로식을

```
00  int max(int a, int b) {
01      int r ;
02      if (a > b)
03          r = a ;
04      else
05          r = b ;
06      assert(2 * r > a + b) ;
07      return r ;
08  }
```

그림 3: 잘못된 불변식을 추가한 예제 프로그램

생성하고 단언 조건을 만족하는 실행과 위반하는 경우를 모두 탐색하여 가능한 실행을 생성한다.

그림 3 은 그림 1 의 코드에 잘못 추출된 불변식(6 번 라인)을 추가한 코드이다. 이 코드에는 분기문에 따라 두 개의 실행이 가능하며 각각 실행에서 단언이 만족하는 경우와 만족하지 않는 경우가 가능한지 확인하기 위하여 다음의 네 가지 기호식이 생성된다.

1: $(a > b) \wedge (2 \times a > a + \mathrm{b})$
2: $(a > b) \wedge \neg(2 \times a > a + \mathrm{b})$
3: $\neg(a > b) \wedge (2 \times a > a + \mathrm{b})$
4: $\neg(a > b) \wedge \neg(2 \times a > a + \mathrm{b})$

SMT-solver 를 통해 위의 네 가지 기호식의 해를 구하면, 4 번째 기호식으로부터 단언이 위반되는 경우로 `<a:1, b:1>`와 같은 테스트 입력값을 구할 수 있다. 이렇게 구한 테스트 입력값을 이용해 추가적인 실행을 수행함으로써 검정하는 불변식이 올바른 실행을 위반하는 잘못된 불변식으로 판별할 수 있다.

4. 실험

본 논문에서 제시한 프레임워크가 실제 Daikon 이 추출하는 불변식의 정확도를 효과적으로 판별하는지 확인하기 위하여 실제 자바 프로그램을 대상으로 실험을 수행하였다. 실험에서 불변식 추출 대상으로 사용한 자바 프로그램은 배열을 통해 스택 자료 구조를 구현한 StackAr 이다. StackAr 은 하나의 자바 클래스로 총 151 LOC 이며 2 개의 생성자 메쏘드와 7 개의 공개 메쏘드, 그리고 1 개의 간단한 테스트 메쏘드를 포함하고 있다. 이 프로그램은 이전에도 Daikon 기법의 연구에 벤치마크로 사용되어왔다. 실험에서 프레임워크의 각 단계를 수행한 구체적인 설명은 다음과 같다.

(1) Daikon 을 통한 불변식 추출

Daikon 을 일반적인 방법으로 이용하여 StackAr 프로그램의 불변식을 추출했다. 일반적인 프로그램의 경우, 소프트웨어 개발 과정에서는 만들어진 테스트 드라이버를 활용하여 Daikon 의 입력 실행으로 사용할 수 있다. 본 실험에서는 개발과정에서 생성된 테스트 드라이버를 대신하기 위하여 자동 테스트 생성 도구인 Randoop 을 사용 했다 [5]. Randoop 도구는 입력된 객체 지향 프로그램을 분석하여,

다양한 메쏘드 순열 조합을 수행하는 테스트 프로그램을 자동으로 생성한다. 다양한 메쏘드 호출 순서를 반영하기 위해 Randoop 을 통해 총 1000 개의 독립적인 경우를 테스트하였다. 이를 Daikon 에 입력한 결과 총 85 개의 불변식을 추출하였고 그 중 4 개는 클래스 불변식(class invariant) 형태이며, 나머지 81 개는 메쏘드 실행후 조건식(method post-condition)이었다.

(2) SPF 를 통한 기호실행

첫 단계에서 생성한 불변식을 프로그램에 추가하여 기호실행을 수행함으로써 단언이 위반되는 프로그램 실행을 탐색하였다. Daikon 이 생성한 불변식은 JML [6]의 형태인데 이를 같은 의미의 자바 단언구문으로 번역하였다. 현재 SPF 구현은 클래스의 생성자에 대한 기호실행을 지원하지 않는다. 이를 고려하여 StackAr 생성자 메쏘드에 관련된 불변식을 시험 대상에서 제외하였고, 생성자 내부의 선언을 생성자 밖으로 이동하여 기호실행이 가능하도록 변경하였다. SPF 에서 기호실행을 수행하기 위해서는 프로그램에 기호값으로 추적할 변수와 기호실행 경로를 정하여야 한다. 본 실험에서는 StackAr 클래스 내에 정의되어 있는 테스트 메쏘드(27 LOC)를 활용하여 클래스의 모든 공개 메쏘드를 한 번 이상 호출하는 경로에 대해 기호실행이 이루어지도록 하였으며, 테스트 메쏘드에 나오는 모든 상수를 기호 값으로 기호실행을 수행하였다.

프레임워크를 적용하여 Daikon 이 자동으로 생성한 불변식을 검정한 결과는 표 1 과 같다. Daikon 이 생성한 모든 불변식을 사람이 직접 수작업으로 검정한 결과, 추출된 85 개의 중 29 개의 불변식이 틀린 것을 확인하였다. 프레임워크는 총 29 개의 틀린 불변식 중 12 개의 틀린 불변식(41%)에 대하여 각각의 불변식 조건을 위반하는 실행을 생성함으로써 자동으로 검정하였다.

SPF 가 검정에 실패한 17 개의 불변식에 대하여 기호실행을 통한 자동 검정이 실패한 원인을 분석하였다. 그 결과 다음의 두 가지 원인을 판별할 수 있었고, 이를 통해 제시한 프레임워크의 효용성의 한계점으로 생각된다.

- null 을 참조하는 경우를 고려하지 못함 (16 개)
  16 개의 틀린 불변식은 특정 메쏘드의 입력으로 null 이 사용되는 실행을 고려하지 못하여 틀린 불변식이다. 현재 SPF 가 동적 메모리 참조에 대한 기호 실행이 제한적으로 구현되어 null 을 가능한 기호값으로 사용하고 있기 때문에 이를 올바르게 판별하지 못한 것으로 보인다.
- 특정 메쏘드 호출 순서를 고려하지 못함 (1 개)
  1 개의 불변식은 특정한 메쏘드 호출 순서 경우를 올바르게 포함하지 못하여 잘못된 불변식이다. 하지만 경로실행에 사용한 테스트 메쏘드에서 해당 메쏘드 호출 순서를 테스트 메쏘드의 실행을 생성할 수 없어서 올바르게 검정에 실패했다.

Daikon 이 생성한 틀린 불변식으로 제시한 프레임워크를 통해 자동 검정에 성공한 경우	12
Daikon 이 생성한 틀린 불변식으로 제시한 프레임워크를 통한 자동 검정에 실패한 경우	17
합	29

표 1: 프레임워크를 통한 자동 검정 결과

검정에 실패한 불변식을 분석한 결과, 현재 프레임워크를 보완하기 위해서는 (1) 메모리 참조를 올바르게 지원하는 기호실행 기법이 필요하며 (2) 기호실행은 다양한 입력값의 조합은 가능하나 다양한 메쏘드 호출 순서의 조합이 불가능하므로 다양한 메쏘드 호출 순서가 가능하게 작성한 테스트 프로그램이 필요함을 알 수 있다.

5. 결론

본 연구에서는 동적 기법을 통해 추출된 유사 불변식을 기호실행을 통해 자동으로 검정하는 프레임워크를 제안하였다. 제시한 프레임워크는 기호실행을 통해 추출한 불변식이 만족하지 않을 것으로 보이는 실행 경로를 효과적으로 검사함으로 Daikon 과 같은 기존의 동적 불변식 생성 기법의 제약점을 효과적으로 보완한다. SPF 기호실행 도구를 통해 구현한 프레임워크의 실험함으로써 제안한 기법의 유효성과 한계점을 확인 하였다.

향후 연구 주제로는, 보다 효과적이며 효율적인 불변식 자동 추출을 위해서 불변식 추출을 결합한 기호실행 기반 테스팅 기법의 개발이 있다. 최근에는 동적 기호실행을 통해 자동으로 테스트케이스를 생성하는 기법이 활발히 개발되고 있다. 본 논문에서 제시한 프레임워크에 동적 기호실행 기반 자동 테스팅 기법을 적용하여 테스트 초기 단계에서부터 효과적인 불변식 추출과 개선을 고려할 경우, 더욱 효율적인 불변식 추출이 가능할 것으로 예상한다.

참고문헌

[1] Y. Kim, M. Kim, Y. Kim, Y. Jang. Industrial Application of Concolic Testing Approach: A Case Study on libexif by Using CREST and KLEE. *Submitted to ICSE 2012 SEiP*

[2] M. D. Ernst, J. Cockrell, W. G. Griswold, D. Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Transactions on Software Engineering*, vol. 27, no. 2, 2001.

[3] C. Csallner, N. Tillmann, Y. Smaragdakis. DySy: Dynamic Symbolic Execution for Invariant Inference. *In ICSE* 2008

[4] C. S. Pasareanu, S. Person, M. Pape. Combining Unit-level Symbolic Execution and System-level Concrete Execution for Testing NASA Software. *In ISSTA* 2009.

[5] C. Pacheco and M. D. Ernst. Randoop: Feedback-directed Random Testing for Java. *In OOPSLA 2007 Companion*

[6] The Java Modeling Language (JML) `http://www.eecs.ucf.edu/~leavens/JML`