소프트웨어 테스팅을 위한 테스트 데이터의 자동 생성†

한성대학교 정인상*

1. 서 론

소프트웨어가 실생활과 밀접한 관계를 유지함에 따라 소프트웨어의 신뢰성과 같은 품질 제고가 주요 관심사가 되고 있다. 최근에 정형적 방법을 이용하여 소프트웨어의 오류를 검증하는 연구가 국내외적으로 진행되고 있다. 정형적 방법에 바탕을 둔 프로그램 증명 방법은 "모형검사(model checking)" 방법과 "정리 증명 방법(theorem proving)"으로 분류할 수 있다. "모형검사" 방법은 시스템이 가질 수 있는 상태 공간을 유한한 모델로 표현하고 검증하고자 하는 시스템의 성질을 입력으로 시스템이 기술된 성질을 만족하는지 여부를 검사하는 기술을 말한다[1]. 만약 시스템이 주어진 성질을 만족하지 못한다면 모형검사기는 반례(counter example)를 생성하여 시스템이 왜 주어진 성질을 만족하지 못했는지에 관한 정보를 사용자에게 준다. 기본적으로 모델이 유한한 상태 공간을 다루기 때문에 모형검사 방법은 완전하게 자동화 할 수 있고 빠르다는 장점이 있다. 대표적인 모형검사 도구로는 SMV[2], Spin[3], UPPAAL[4] 등이 있다. 이에 반해 정리 증명 방법은 프로그램 명세와 구현을 두 개의 일차 논리식으로 표현하여 구현의 가능한 모든 행위가 프로그램 명세를 만족하는지를 검사하는 방법이다. 따라서, 유한한 모델만을 고려하는 모형검사 방법과는 달리 무한한 상태공간을 다룰 수 있다는 장점이 있지만 이로 인해 완전히 자동화 할 수 없고 사용자가 증명 과정 중에 관여해야 하는 단점이 있다. 대표적인 정리 증명 도구로는 Isabelle, PVS[5] 등을 들 수 있다.

† 이 연구는 2001년 한성대학교 교내 연구비 지원을 받아 수행되었음.

* 종신회원

정형적 방법을 이용한 소프트웨어 검증 방법 외에 실제 현장에서 주로 사용되고 있는 방법으로 "소프트웨어 테스팅"이 있다. 소프트웨어 테스팅은 구현된 소프트웨어가 원하는 대로 또는 명세에 기술된 대로 잘 구현되었는지를 검사하는 방법을 말한다. 이를 위해 명세나 프로그램 분석을 통하여 생성된 테스트 데이터를 실제 프로그램에 실행하여 프로그램의 오류를 찾는다. 정형적 방법을 이용한 프로그램 검증 방법은 구현된 프로그램을 직접 다루지 않고 추상화 된 표현 양식을 다루는데 반해 프로그램 테스팅은 생성된 테스트 데이터를 가지고 프로그램을 직접 실행하여 검증한다.

프로그램을 테스팅하는 절차는 보통 다음과 같은 절차들로 구성된다. ① 테스팅 적합성 기준(testing adequacy criterion) 선정: 현실적으로 프로그램의 모든 입력 데이터를 사용하여 프로그램을 테스트할 수 없기 때문에 테스트 데이터를 선정 또는 생성하는 기준을 정하여 이 기준을 만족하는 입력 데이터만을 테스트 데이터로 사용한다. ② 테스트 데이터를 생성: 이 단계가 프로그램 테스팅 단계 중에서 가장 노력이 많이 들고 자동화하기 어렵다[6]. 지금까지 프로그램 테스팅에 관한 연구는 효과적인 테스트 데이터를 선정하기 위한 선정 기준에 많은 노력을 기울여 왔지만 상대적으로 테스트 데이터를 선정 기준에 따라 자동으로 생성하는 기술에 관한 연구가 미진한 것이 사실이다. 테스트 데이터를 생성하기 위해 명세나 프로그램 코드를 분석하여 생성하는 방법들이 있다. ③ 테스트 데이터 실행: 생성된 테스트 데이터를 프로그램의 입력으로 실행하여 실행 결과를 분석한다. 만약 오류가 발견되었다면 오류의 원인을 찾아 제거한다.

지금까지 제안된 테스트 데이터 방법은 여러 관점에서 분류할 수 있다. 우선 테스트 데이터를 생성할

때 사용하는 정보의 종류에 따라 명세 기반 방법 및 프로그램 구현 기반 방법으로 분류할 수 있다. 본 고에서는 테스트 데이터를 자동 생성하는 기술들 중에서 프로그램 구현 내역을 사용하여 테스트 데이터를 생성하는 방법들을 중심으로 기술한다.

2. 구현 기반 테스트 데이터 자동 생성

2.1 테스트 데이터 적합성 기준

테스팅 적합성 기준이란 프로그램이 얼마나 적합하게 테스팅 되었느냐를 판별하는 기준이다. 테스트 데이터를 생성하는 목적은 주어진 테스팅 적합성 기준을 만족하는 테스트 데이터를 생성하는 것이다. 테스팅 적합성 기준을 때로는 테스팅 완료 기준 또는 테스트 데이터 선정 기준이라고도 한다. 그 이유는 적합성 기준을 기반으로 테스트 데이터를 선정하고 이렇게 선정된 테스트 데이터 집합에 대해 프로그램의 오류가 발견되지 않을 때까지 테스팅을 수행하기 때문이다. 보통 적합성 기준들은 프로그램의 특정 부분들을 수행하는 입력 데이터들을 테스트 데이터 집합으로 간주한다. 이러한 적합성 기준의 예로 문장(statement) 적합성 기준, 분기(branch) 적합성 기준, 자료 흐름(dataflow) 적합성 기준 등을 들 수 있다.

```
int mid(int x, int y, int z) {
        int mid
    1: mid = z;
    2: if (y<z) {
    3:      if (x<y)
    4:              mid = y;
    5:      else if (x<z)
    6:              mid = x;
    } else {
    7:      if (x>y)
    8:              mid = y;
    9:      else if (x>z)
    10:             mid = x;
    }
    11: return mid;
}
```

그림 1 예제 프로그램[7]

예를 들어, 분기 완료 기준은 프로그램에 나타난 모든 분기들을 최소한 한 번 실행하는 입력 데이터들을 테스트 데이터로 한다. 이를 위해 보통 프로그램을 제어 흐름 그래프 또는 자료 흐름 정보를 지니고 있는 그래프로 표현하여 어떤 노드 또는 에지들의 집합을 수행하는 입력 데이터들의 집합을 테스트 데이터 집합으로 할 것을 기술한다. 그림 1의 프로그램에 분기 적합성 기준을 적용한 테스트 데이터 집합 T={(7, 5, 3), (1, 2, 3), (10, 15, 8), (8, 10, 7), (10, 9, 14), (6, 9, 15)}은 제어 흐름 그래프상의 분기를 나타내는 모든 에지들을 수행하므로 분기 적합성 기준을 만족한다. 여기에서 테스트 데이터를 이루는 튜플의 각 원소는 입력 변수 x, y, z에 대응된다.

이와 같은 적합성 기준에 따라 프로그램으로부터 테스트 데이터를 생성하는 문제는 주어진 프로그램의 부분(제어 흐름 그래프 상에서 노드 또는 에지)을 수행하는 입력 데이터를 구하는 과정으로 생각할 수 있다. 위에서 구한 테스트 데이터 집합 T에서 (7, 5, 3)은 프로그램 경로 <1, 2, 7, 8>을 실행하는 입력 데이터이며 이는 분기 적합성 기준 관점에서 볼 때 "y>=z"와 "x>y"가 참이 되게 하는 테스트 데이터이다.

2.2 테스트 데이터 생성 과정

프로그램의 구현 내역을 기반으로 테스트 데이터를 생성하는 일반적인 과정은 다음과 같은 단계로 구성된다: ① 테스팅 적합성 기준에 따라 검증하고자 하는 프로그램의 어떤 부분을 지나가는 프로그램 경로(들)를 선정하는 단계, ② 프로그램으로부터 제약식을 추출하는 단계와 ③ 추출된 제약식의 해를 구하는 단계로 구성된다.

사용하는 적합성 기준에 따라 프로그램의 경로들은 달라질 수 있다. 예를 들면 사용자가 분기 적합성에 기준에 따라 테스트 데이터를 선정 또는 생성한다고 하면 프로그램의 모든 분기를 최소한 한번은 실행하게 하는 경로들을 생성한다. 적합성 기준에 따라 생성된 각 프로그램 경로로부터 제약식을 추출한다.

주어진 프로그램 경로에 대한 제약식은 보통 입력 변수들만으로 구성된 일련의 등식 또는 부등식들로 표현되며 해당하는 프로그램 경로를 실행하게 하는 입력 데이터에 대한 조건을 기술한다. 예를 들면 그림 1의 프로그램에서 경로 <1, 2, 7, 8>에 대해서 "x>y and y>=z"라는 제약식을 구할 수 있다. 이러한 제약식을 보통 경로 조건이라고도 한다.

일단 제약식이 추출되면 여러 제약식 해결 알고리즘을 이용하여 제약식의 해를 구한다. 3장에서는 지

금까지 제안된 테스트 데이터 생성 방법들은 사용된 기술에 따라 분류하고 논의한다.

3. 테스트 데이터 생성 기술 및 분류

이 장에서는 실제 프로그램으로부터 테스트 데이터를 생성하는 여러 기술들에 대해 상세하게 소개한다. 이를 위하여 테스트 데이터 생성 방법을 심볼릭 실행(symbolic execution)을 이용한 기법, 함수 최소화(function minimization)에 기반을 둔 방법, 유전자 알고리즘(genetic algorithm)을 이용하는 방법, 제약 논리 프로그래밍(Constraint Logic Programming)에 기반을 둔 방법 및 수치 해석(numerical analysis) 기법을 이용한 방법 등으로 분류하여 기술한다.

3.1 심볼릭 실행을 이용한 테스트 데이터 생성

2.2절에서 언급하였듯이 프로그램으로부터 테스트 데이터를 생성하기 위해서는 우선 검증하고자 하는 프로그램 경로에 대한 제약식을 추출해야 한다. 이를 위해서 실제 프로그램을 어떤 특정한 값으로 실행하기보다는 모든 입력 도메인에 있는 값들을 대표할 수 있는 값으로 실행해야 한다. 심볼릭 실행은 프로그램의 특정한 실제 입력 값 대신에 심볼릭 값을 사용하여 프로그램을 실행하는 방식이다. 과거의 많은 테스트 데이터 생성 방법은 주어진 적합성 기준을 만족하는 경로(들)에 대한 경로 조건(path condition)들을 추출하기 위해 심볼릭 실행을 이용하였다[8,9,10]. 예를 들어 그림 2의 프로그램을 생각해보자.

```
1:x = y*2;
2:if (x >= 10) {
3:      x = x+10;
4:      y = y+1;
}
5:if (x+y < 110)
6:      printf("OK\n");
7:...
```

그림 2 심볼릭 실행 예제 프로그램

만약에 프로그램 경로 <1, 2, 3, 4, 5, 7, ...>가 주어지고 입력 변수 y에 대해 심볼 Y가 할당되었다면 1번과 2번 문장 수행 후의 경로 조건은 "Y*2>=1"이다. 이와 같은 방식으로 4번 문장을 수행한 후의 경로 조건은 "(Y+1)*2>= 10"이 된다. 또한, 분기 문장 5번이 거짓이 되어야 하므로 "(Y+1)*2>=10 and (Y*2+10)+(Y+1)>=110"이 생성된다. 이를 단순화 하면 'Y>= 33'이다. 이를 만족하는 입력 값 y가 경로 <1, 2, 3, 4, 5, 7, ...>을 실행하는 테스트 데이터이다. 그러나 이 방법은 배열이나 포인터 연산을 고려할 때 문제가 발생한다.

예를 들어 "x=TestGen[i+j]" 와 같은 문장에서 배열(TestGen)의 원소가 간접적으로 참조될 때 변수 'i'와 'j'가 특정한 값으로 바운드 되지 않기 때문에 실제 어느 배열의 원소를 참조하는지 알 수 있는 방법이 없다. 또한 다음 프로그램 블록에서처럼 포인터를 사용하여 동일한 기억 장소를 다른 변수 이름을 이용하여 접근하는 경우(aliasing problem)에도 문제가 발생한다.

```
*x = 20;
*y = 50;
 c = *x;
```

이 경우에 만약 포인터 변수 x, y가 동일한 기억 장소를 가르키지 않는다면 변수 c의 값은 20이 되지만 동일한 장소라면 c는 50이 저장되어 있을 것이다. 그러나 심볼릭 실행은 변수 x, y에 대해 실제 특정 값을 주지 않기 때문에 변수 c에 최종적으로 저장된 값을 알 수가 없다.

이러한 문제들 때문에 심볼릭 실행에 바탕을 둔 많은 테스트 데이터 생성 방법들은 FORTRAN과 같이 포인터 연산이 없는 프로그래밍 언어에 한정되거나 배열의 각 원소를 서로 독립적인 변수들로 간주하지 않고 전체적으로 하나의 변수로 간주한다. 따라서 생성된 테스트 데이터는 주어진 프로그램 경로를 반드시 실행한다는 보장이 없다.

기존의 심볼릭 실행의 단점을 해결하기 위해서 입력 변수들에 심볼을 할당하는 대신에 실제 입력 값들을 사용하여 프로그램을 실행하는 방식이 제안되었다. 따라서 실제로 배열의 어떤 원소가 참조되는지를 알 수 있다. 마찬가지로 포인터에 의한 문제도 해결할 수 있다.

3.2 함수 최소화 기법을 이용한 테스트 데이터 생성

최근에 제안된 많은 테스트 데이터 생성 방법들은

함수 최소화 기법에 기반을 두고 있다[11,12,13]. 함수 최소화 기법은 실제 입력 값을 사용하여 프로그램을 실행한다. 따라서 기존의 심볼릭 실행에서 발생하는 배열을 처리할 때 발생하는 문제나 "이명 문제"(aliasing problem)을 해결할 수 있다. 예를 들면 TESTGEN[11]이나 ADTEST[13]와 같은 테스팅 시스템은 특정한 입력 값을 사용하여 주어진 프로그램 경로에 따라 실제로 프로그램을 실행하는 방식을 취한다.

이러한 시스템에서는 프로그램의 경로 상에 있는 분기 조건문을 함수로 간주한다. 예를 들면, 3.1절의 프로그램에서 분기 조건(문장 2번)이 "x>=10" 참이 되게 하는 테스트 데이터를 구하는 문제를 생각해보자. 이와 같은 분기 조건문은 "F(y) = 10-x if x<10, 0 otherwise"와 같은 함수로 생각할 수 있다. 이때 x를 문장 2번에 도달할 때의 변수 x에 저장되어 있는 값이라고 가정하면 함수 F(y)를 최소화하는 입력 데이터 y는 분기 조건 "x>=10"을 참이 되게 하는 입력 데이터 값이 된다.

이러한 입력 값을 찾기 위해 TESTGEN과 같은 시스템에서는 랜덤하게 생성된 초기 입력 데이터로 주어진 경로를 따라 프로그램을 실행한다. 만약 프로그램 실행 도중에 분기 조건문을 만난 경우에 현재의 입력 값이 주어진 프로그램 경로를 따라서 적절한 분기가 일어난다면 입력 값을 변경하지 않는다. 그러나, 만약 주어진 프로그램 경로와는 달리 분기가 일어난다면 (즉, 실제 프로그램 실행 경로와 주어진 프로그램 경로가 다른 경우) 실행의 흐름을 바꾸도록 함수 최소화 기법을 이용하여 현재의 입력 데이터를 수정한다. 만약 이러한 과정을 거치고도 적절한 입력 데이터를 찾지 못한다면 프로그램 경로 상에서 현재 분기 조건문에 바로 앞서 실행된 조건문으로 되돌아가 다른 입력 값을 찾는 과정을 되풀이한다.

예를 들어 변수 y의 초기 값이 3이라고 가정하면 분기 조건 변수 x의 값이 6으로 설정되기 때문에 분기조건 "x>=10"은 거짓이다. 따라서 함수 최소화 기법을 사용하여 실행 흐름을 변경할 입력 값을 찾을 필요가 있다. 이를 위해 Korel[11]은 다음과 같은 방법을 제안하였다. 우선 입력 변수 y에 저장되어 있는 값을 1만큼 증가한다. y의 값이 1만큼 증가되었을 때 F(y)의 값은 2가 된다. 이 값은 y가 3일 때의 F(y)의 값 4보다는 작지만 여전히 분기 조건의 결과는 거짓이다. 따라서 y를 증가하는 방향으로 수정하여 (예를 들면, y를 7만큼 증가하여 10으로 설정) 원하는 입력 데이터를 구한다.

이와 같은 방법 외에 선형 프로그래밍을 이용하여 테스트 데이터를 생성하는 방법이 있다. 대표적인 시스템으로 TAO를 들 수 있다[14]. 이 방법은 우선 주어진 프로그램 부분에 도달 할 수 있는 프로그램 경로들을 구한 후에 심볼릭 실행 방법을 이용하여 프로그램 경로 조건들을 추출한 후 이를 선형 문제로 변환하여 테스트 데이터를 생성한다. TAO는 앞서 기술한 테스팅 시스템들과는 달리 프로그램의 실행을 요구하지 않는다.

3.3 유전자 알고리즘을 기반으로 하는 테스트 데이터 생성

최근에는 유전자 알고리즘을 이용하여 테스트 데이터를 생성하는 방법에 대한 연구가 진행 중에 있다[15,16]. 그림 3은 유전자 알고리즘 실행의 전형적인 과정을 요약해서 기술하고 있다.

```
Genetic Algorithm()
    begin
        t ← 0;
        Initialize(P(t));
        Evaluate(P(t));
        while (not Finished())
        begin
            t ←t+1;
            Select P(t) from P(t-1);
            Recombine P(t);
            Mutate P(t);
            Evaluate(P(t));
        endwhile
        solution ← BestOf(P(t))
    end
```

그림 3 유전자 알고리즘 실행을 위한 전형적인 과정

유전자 알고리즘 실행에서 첫 단계는 초기화 단계로 이 단계에서는 무작위로 초기 모집단 즉 $P(0)$을 생성하는 것이다. 이 모집단의 각 원소는 염색체라고 불리는 이진 스트링으로 표현되며 각 이진 스트링은 적합성 함수(fitness function) "F(x)"에 의해 평가된다. 테스트 데이터를 생성하는 문제에서는 각 이진 스트링을 하나의 테스트 데이터로 간주할 수 있다. 유전자 알고리즘에서는 처음 시작할 때 초기 모집단이 현재 모집단($P(t)$)이 된다. 이 모집단의 각 이진

스트링을 위에서 정의된 F(x)로 평가한 후에 선택 연산자는 현재 모집단에서 다음 교차 및 변이 연산에 참가할 두 염색체를 선택한다. 적합성 평가 과정에서 적합성이 낮은 테스트 데이터 즉 이진 스트링들은 다음 세대에서 무시될 수 있다. 이러한 과정을 거쳐 다음 모집단($P(t+1)$)을 생성한다. 이러한 반복 과정은 생성된 모집단이 문제에 대한 해에 수렴할 때 종료하게 된다.

유전자 알고리즘은 기본적으로 해를 표현하는 이진 스트링의 적합성을 평가할 수 있는 적합성 함수(fitness function)를 필요로 한다. 테스트 데이터 생성 문제에서 이러한 적합성 함수는 테스트 데이터의 품질을 평가하는 척도가 된다. 예를 들면, GADGET 시스템[15]에서는 각 분기 조건문의 형태에 따라 원하는 적합성 함수를 정의하였다. 만약 "if (c==d)..." 와 같은 문장을 만나고 이 문장이 참이 되길 원하는 입력 데이터 x를 구한다고 가정하자. 이 경우 적합성 함수 F(x)는 1/|d-c|로 표현된다. 즉, d와 c의 값을 비슷하게 하는 입력 x는 적합도가 크게 나타나지만 d와 c의 값의 차이가 큰 경우에는 적합도가 낮아 선택 과정에서 선택될 확률이 낮아진다. 근본적으로 이러한 테스트 데이터 생성 알고리즘은 3.2절에서 기술한 함수 최적화 방법에 포함 될 수 있다. 그 이유는 각 분기 조건을 적합성 함수로 나타내고 이 함수를 최대화(또는 최소화)하는 문제로 표현될 수 있기 때문이다. 또한, 테스트 데이터의 적합성 판정을 위해 프로그램을 직접 실행하는 방식을 취하기 때문에 실행 기반 테스트 데이터 생성 방법이다.

3.4 제약 논리 프로그래밍을 이용한 테스트 데이터 생성

제약 논리 프로그래밍(Constraint Logic Programming, CLP)은 선언적인 언어인 논리언어에서는 수치 제약 처리가 용이하지 않다는 단점을 해결하기 위해 논리언어에 제약 해결기(Constraint Solver)를 포함시켜 프로그램을 개발하는 개념이다[17]. 기본적으로 테스트 데이터를 생성하는 문제는 주어진 제약식을 만족하는 변수들의 값을 구하는 문제이다. 이러한 관점에서 제약 논리 프로그래밍을 이용하여 테스트 데이터를 생성하기 위한 연구가 보고되고 있다.

예를 들어 다음과 같이 Prolog 형태로 기술된 제약식을 생각해보자:

$$3*X + Y*Y = 7$$

Prolog에서는 다만 문법적으로 동일한 텀(term)들 간에만 동일하다고 판별하기 때문에 위에서 보여진 등식은 성립하지 않는다. 더욱이 X와 Y에 정수형과 같은 특정 타입이 할당되지 않는다. 그러나, Prolog언어에 제약 조건 처리 능력을 첨가한 CHIP과 같은 CLP 언어에서는 X와 Y에 타입 의미를 부여할 수 있을 뿐만 아니라 "3*X+*Y*Y=7"과 같은 형태로 제약식을 기술하는 것이 가능하다. 실제로 제약 해결기는 위에서 주어진 제약식을 입력으로 받아 비선형 제약식으로 판단하고 변수 X에 특정 값(예를 들어 1)을 대입한다. 이 과정을 거친 후의 제약식은 다음과 같다 :

$$X=1,\ Y*Y=4$$

제약식을 만족하는 Y의 값을 구하기 위해 제약 해결기는 반복적으로 Y의 값을 대입하여 결국 Y의 값이 2나 -2가 될 때 주어진 제약식이 만족됨을 보일 수 있다.

ATGen[18]은 이러한 CLP 원리를 이용하여 Ada 프로그램에 대해 테스트 데이터를 자동으로 생성하기 위해 개발된 시스템이다. 이 시스템은 우선 프로그램을 입력으로 받아 Prolog 형태로 변환한다. 변환된 프로그램에 대해 주어진 테스팅 적합성 기준에 따라 심볼릭 실행을 수행하여 프로그램의 경로 조건을 추출한 후 CLP의 제약 해결기를 이용하여 테스트 데이터를 생성한다.

CLP를 이용하는 또 다른 테스트 데이터 생성 도구로 INKA 시스템[19]이 있다. 이 시스템이 테스트 데이터를 생성하는 과정은 기본적으로 ATGen과 동일하지만 프로그램으로부터 제약식을 추출하는 방식에 차이가 있다. ATGen은 제약식을 심볼릭 실행을 통해 구하지만 INKA는 프로그램을 의미적으로 동등한 SSA(Single Static Assignment) 폼[20]으로 변환하여 제약식을 추출한다. SSA 폼에서는 모든 변수는 유일한 정의를 가지고 있으며 각 변수의 참조는 해당 변수를 정의하는 문장에 의해 도달이 가능하다는 특징을 지니고 있다. 따라서, 각 프로그램의 변수를 논리 변수(logical variable)로 처리하는 것을 가능하게 해준다. 이 두 방법의 공통점은 실제 프로그램을 실행하지 않고 테스트 데이터를 생성한다는 점이다.

3.5 수치 해석 기법을 이용한 방법

[21]에서 Gupta 등은 테스트 데이터를 생성하는 문제를 프로그램으로부터 추출한 연립방정식의 해를 구하는 수치해석 문제로 변환하여 해결하는 방법을 제안하였다. 이 방법은 주어진 프로그램 경로에 존재하는 각 분기 조건문을 입력 변수들의 선형 함수로 표현하고 초기 입력 값을 이 서술함수의 근사 해라고 가정한다. 예를 들어 (p, q)가 ax+by+c=0의 근사 해라고 가정하자. 일반적으로, 이 방정식에 p와 q를 대입하면 0이 아닌 잉여 오차 r이 결과로 나온다. 즉, ap+bq+c=r. 이 때, "a△x+b△y=-r"을 만족하도록 p와 q의 변화량(△x, △y)을 계산하면 a(p+△x)+b(q+△y)+c=0을 만족하고 (p+△x, q+△y)이 원래의 방정식 ax+by+c=0의 해가 된다. 즉 해당하는 분기 조건문이 주어진 경로에 따라 올바른 분기가 일어나도록 해주는 테스트 데이터가 된다.

이 방법은 함수 최소화를 이용한 방법과 같이 실제 프로그램을 실행하여 테스트 데이터를 생성한다. 그러나, TESTGEN이나 ADTEST와 같은 시스템에서는 한 번에 하나의 분기 조건만을 고려하여 테스트 데이터를 생성하기 때문에 비록 분기조건이 선형일지라도 원하는 테스트 데이터를 생성하기 위해서는 많은 반복작업을 필요로 한다. 이러한 방식과는 달리 [21]에서 제안된 방법은 주어진 경로 상에 존재하는 모든 분기 조건에 대해 연립 선형 방정식을 추출하여 Gaussian 소거법을 이용하여 해를 구하는 방식을 취한다. 따라서, 주어진 프로그램 경로상의 모든 분기 조건문을 표현하는 함수들이 선형이라면 반복 작업 없이 테스트 데이터를 구할 수 있다.

4. 국내의 연구 동향

국내에서는 프로그램으로부터 테스트 데이터를 자동으로 생성하는 연구가 선진 외국에 비해 상대적으로 미흡한 실정이다. 현재 한성대학교와 한국과학기술원에서 요약 해석(abstract interpretation)에 기반한 테스트 데이터 생성에 관한 연구가 진행중이며 국내 회사에서 API 테스팅을 위한 제품이 나와 있다. 이 장에서는 이에 대해 간략하게 소개한다.

4.1 요약 해석을 이용한 테스트 데이터 자동 생성

지금까지 제안된 대부분의 테스트 데이터를 생성하는 방법은 실제 주어진 입력 데이터로 프로그램을 실행하여 원하는 테스트 데이터를 생성하는 방식이다. 그러나, 요약 해석에 기반을 둔 테스트 데이터 생성 방법은 프로그램을 실행하지 않고 정적으로 프로그램을 분석하여 테스트 데이터를 추출하는 방법이다. 요약 해석은 실제 값 대신에 요약 값을 사용하여 프로그램의 동작 정보를 계산할 수 있으며 이를 통해서 안전한(safe) 근사 정보를 제공한다[22]. 여기서 안전의 의미는 프로그램의 모든 가능한 실제 실행결과를 포함한다는 것을 의미한다. 따라서 요약 해석을 이용하여 프로그램을 분석하면 프로그램을 직접 실행하지 않고 프로그램 실행에 대한 근사 정보를 얻을 수 있다는 장점이 있다. 최근에 국내에서 요약 해석 기술을 이용하여 프로그램을 슬라이싱하는 방법이 제안되었다[23].

요약 해석을 이용하여 테스트 데이터를 생성하기 위한 과정은 프로그램으로부터 동식시스템 "X=F(X)"를 추출하여 이를 만족하는 고정점(fixed point)을 구하는 단계들로 이루어진다. 이 과정에서 구한 고정점은 실제 프로그램을 실행했을 때 초래되는 프로그램 상태의 근사 정보(요약 정보)를 나타낸다. 이렇게 추출한 근사 정보는 프로그램의 특정 블록을 수행하기 위한 입력 데이터의 필요 조건을 생성하는 기반을 제공한다[24]. 예를 들면, 다음에 주어진 프로그램에서 3번 문장을 실행하는 테스트 데이터를 생성해보자.

```
1:read(x, y);
2:if (x>10)
3:        x = x+y;
  else
4:        x = x-y;
```

이 프로그램에 대해 요약 해석을 수행하면 다음과 같은 정보를 얻을 수 있다:

x1: x=⊥, y=⊥
x2: x=[-maxint, maxint], y=[-maxint, maxint]
x3: x=[11, maxint], y=[-maxint, maxint]
x4: x=⊥, y=⊥

전방향 여기에서 xi는 i-번째 문장을 수행하기 직전의 프로그램 요약 상태를 나타낸다. x4에서 변수 x와 y가 정의되지 않은 값(⊥)을 가지는 이유는 3번 문장이 실행되면 4번 문장을 실행할 수 없다는 의미이다. 4번 문장을 실행하지 않아야 된다는 정보는 프

로그램으로부터 자동적으로 추출할 수 있다. 이렇게 추출된 정보를 제어흐름그래프 상에서 다시 역방향으로 전달하면 다음과 같은 정보가 추출된다.

x1: x=⊥, y=⊥
x2: x=[11, maxint], y=[-maxint, maxint]
x3: x=[11, maxint], y=[-maxint, maxint]
x4: x=⊥, y=⊥

x2를 보면 x가 최소한 11이상이 되어야 한다는 사실을 알 수 있다. 즉, 3번 문장을 실행하기 위해서는 x의 값이 최소한 11이상이 되어야 하며 y의 값은 어떤 정수 값을 취하더라도 3번 문장의 실행에 영향을 주지 않는다.

4.2 CodeScroll

CodeScroll[25]은 국내 MacroImpact라는 회사에서 개발된 테스트 데이터 자동 생성 도구이다. 현재 이 도구는 현재 C와 Java로 작성된 API들을 테스팅할 수 있는 테스트 데이터를 자동으로 생성할 수 있다. CodeScroll에서는 프로그램에서 사용하는 각 입력 변수의 도메인을 동적으로 분할하여 테스트 데이터의 경로 적합성(path coverage)을 높이는 방식을 취하고 있다. 즉, 초기 입력 도메인의 각 분할 영역으로부터 테스트 데이터들을 선정하여 프로그램을 실행한다. 선택된 테스트 데이터들이 다른 프로그램 경로를 실행한다면 이를 바탕으로 테스트 데이터 집합을 선정했던 분할 영역을 더 세분화하는 과정을 반복하여 최종적으로는 각기 다른 경로를 실행할 수 있도록 입력 도메인이 분할되도록 한다.

많은 테스트 데이터 생성 도구들이 정수나 실수와 같은 기본 자료형만 다루거나 배열, 포인터가 있는 경우에는 적용되지 못하거나 효율적이지 못하다는 단점이 있는 반면 CodeScroll은 이러한 제약점 없이 사용할 수 있다는 장점이 있다. 또한 Java API 테스팅의 경우 원시코드가 아닌 바이트 코드를 대상으로 테스트 데이터 생성 작업을 수행할 수 있다.

5. 결 론

최근에 정형적 방법을 이용하여 프로그램을 검증하는 많은 연구가 이루어지고 있다. 그러나 현실적으로 정형적 명세가 갖는 여러 제한점(사용하기 어려움, 프로그램을 직접 대상으로 하기 보다 프로그램의 모델에 적용 등) 때문에 프로그램 테스팅 방법을 이용하여 검증을 수행하는 것이 일반적이다. 그러나, 프로그램 테스팅에 있어서 가장 어려운 점은 테스트 데이터를 자동으로 생성하는 문제이다. 대부분의 상업용 테스팅 도구들에서 테스트 데이터 생성 과정은 자동화가 이루어지지 않고 있는 실정이다. 또한, 대부분의 테스팅 자동화에 관한 연구는 명세로부터 테스트 데이터를 생성하는 방식이며 이를 위해서는 정형적인 명세가 필수적이다.

본 고에서는 명세가 없이 프로그램으로부터 자동으로 테스트 데이터를 생성하는 방법들을 실제 사용하고 있는 기술을 중심으로 분류하고 논의하였다. 또한, 국내에서 진행되고 있는 테스팅 자동화 연구에 대해서도 간략하게 소개하였다. 프로그램으로부터 자동으로 테스트 데이터를 생성하는 방법은 프로그램 개발 비용을 현저하게 줄일 수 있으리라 판단된다. 현재 국내외에서 소프트웨어 품질에 대한 관심이 증폭되고 있는 현실에 비해 아직까지 이렇다할 만한 자동화 도구가 없는 현실을 감안한다면 테스팅 자동화 도구의 개발은 매우 의미 있는 일로 여겨진다. CodeScroll과 같이 국내 기업에서도 테스팅 도구에 대한 연구가 어느 정도 성숙한 만큼 세계적으로 경쟁력 있는 테스팅 도구를 만들기 위한 많은 작업과 연구가 필요한 시점이다.

참고문헌

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

[2] McMillan, K. L., Symbolic *Model Checking: An Approach to the State Explosion Problem*, Kluwer Academy, 1993.

[3] Holzmann, G. J. *Design and Validation of Computer Protocols*, Prentice Hall, 1991.

[4] Larsen, K., Pettersson, P., and Yi. W., "UPPAAL in a Nutshell". *Springer International Journal of Software Tools for Technology Transfer*, 1(1+2), 1997.

[5] Griffioen, D. and Huisman, M., "A Comparison of PVS and Isabelle/HOL", *Theorem Proving in Higher Order Logics: 11th International Conference*, 1998.

[6] Ince, D., "The Automatic Generation of Test

Data", *The Computer Journal*, vol. 30, no. 1, pp. 63-69, 1987.

[7] Offutt, J. and Pan, J., "The Dynamic Domain Reduction Approach to Test Data Generation", *Software-Practice and Experience*, vol 29, no 2, pp. 167-193, 1999.

[8] Clarke, L. A., "A System to Generate Test Data and Symbolically Execute Program", *IEEE Trans. on Software Eng.*, vol. 2, no. 3, pp. 215-222, 1976.

[9] King, J. C., "Symbolic Execution and Program Testing,", Comm. ACM, vol. 19, no. 7, pp. 385-394, 1976.

[10] Howden., "Symbolic Testing and the DISSECT Symbolic Evaluation System", *IEEETrans on Software Eng.,* vol. 4, no. 4, pp. 266-2781, 1977.

[11] Korel, B., "Automated Software Test Data Generation", *IEEE Trans. on Software Eng.*, vol. 16, no. 8, pp. 870-879, 1990.

[12] Roger, F. and Korel, B., "The Chaining Approach for Software Test Data Generation", *ACM Trans. on Software Engineering Methodology.*, vol. 5, no.1, pp. 63-86, 1996.

[13] Gallagher, M. J. and Narasimhan, V. L., "ADTEST: A Test Data Generation Suite for Ada Software Systems", *IEEE Trans. on Software Eng.*, vol. 23, no. 8, pp. 473-484, 1997.

[14] Lapierre, S., Merlo, E., Antoniol, G., Fiutem, R., and Tonella, P., "Automatic Unit Test Data Generation Using Mixed-Integer Linear Programming and Execution Trees". *In Proceedings of International Conf. on Software Maintenance.* 1999.

[15] McGrew, G., Michael, C., and Michael, S., "Generating Test Data by Evolution", RST Corporation, RSTR-018-97-01, 1997.

[16] Chung, I. S., "Automatic Test Generation for Mutation Testing Using Genetic Algorithms", *In Proceedings of International Conf. on Software Eng., and Knowledge Eng.,* 1998.

[17] Jaffar, J. C. and Mather, M J., "Constraint Logic Programming: A Survey", *Journal of Logic Programming*, vol. 20, no. 19, pp. 503-581, 1994.

[18] Meudec, C., "ATGen: Automatic Test Data Generation using Constraint Logic Programming and Symbolic Execution", *IEEE/ACM First International Workshop on Automated Program Analysis, Testing and Verification, 22nd International Conference on Software Engineering (ICSE 2000)*, pp. 22-31, 2000.

[19] Gotlieb, A., Botella, B., and Rueher, M., "Automatic Test Data Generation using Constraint Solving Techniques", *In Proceedings of ISSTA*, 1998.

[20] Cytron, R. Ferrante, J., Rosen, B. K., Wegman, M. N., and Zadeck, F. K., "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph", *ACM Trans on Programming Languages and Systems*, vol. 13, no. 4, pp. 451-490, 1991.

[21] Gupta, N., Mathur, A., and Soffa, M. L., "Automated test data generation using an iterative relaxation method". *In Proceedings of Foundations of Software Engineering.* ACM Press, Nov. 1998.

[22] Cousot, P. and Cousot,R., "Systematic design of program analysis frameworks," in *Proc. of 4th ACM Symp. on Principle of Programming Languages*, pp. 269-282, ACM Press, 1979.

[23] 정인상, 창병모, "요약 해석을 이용한 프로그램 슬라이싱", *정보과학회 논문지: 소프트웨어 및 응용*, 제28권, 8호, pp551-559, 2001.

[24] 한승희, 강제성, 정인상, 권용래, "요약 해석을 이용한 테스트 데이터 자동 생성 기법", 한국정보과학회 추계학술발표회, 서울여자대학교, 2001.

[25] http://www.macroimpact.com

정 인 상

1983~1987 서울대학교 컴퓨터공학과(학사)
1987~1989 한국과학기술원 전산학과(석사)
1989~1993 한국과학기술원 전산학과(박사)
1994~1998 한림대학교 부교수
1997~1998 미국 purdue 대학 방문교수
1999~현재 한성대학교 컴퓨터 공학부 부교수

관심분야:병렬 및 분산 프로그램 테스팅, 테스트 데이터 자동생성, 모형 검사(model checking)
E-mail:insang@hansung.ac.kr

• 2001 프로그래밍언어연구회 추계 학술대회 •

- 일 자 : 2001년 11월 17일
- 장 소 : 숭명여자대학교
- 내 용 : 논문발표
- 주 최 : 프로그래밍언어연구회
- 문 의 처 : 숭실대학교 컴퓨터학부 유재우 교수
  Tel. 02-820-0675

• 데이타베이스연구회 2001년 추계 튜토리얼 •

- 일 자 : 2001년 11월 23일
- 장 소 : 대한상공회의소
- 주 제 : "전자 카달로그 구축 기술"
- 주 최 : 데이타베이스연구회
- 문 의 처 : 서울대학교 컴퓨터공학부 이상구 교수
  Tel. 02-883-9235
  E-mail : tutorial@europa.snu.ac.kr