제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

블록의 결합에 따른 테스트 데이터 생성 방법

곽동규, 조용윤, 유재우
숭실대학교 컴퓨터학과
{coolman, yycho}@ss.ssu.ac.kr, cwyoo@comp.ssu.ac.kr

Test Data Creator of Block Combination

Dong-Gyu Kwak*, Yong-Yoon Cho, Chea-Woo Yoo
Dept of Computing, Soongsil University

요 약

응용프로그램이 복잡해지고 사용 환경이 다양해짐에 따라 신뢰성 높은 소프트웨어 생산을 위한 테스트가 중요시 되고 있다. 소프트웨어를 테스트하기 위해서는 각 기능에 따라 동작하는 모듈이 합당한 동작에 대해서 분석해야한다. 모듈은 다수의 실행경로로 이루어져 있으며 각 실행 경로에 따라 다른 결과를 출력하도록 설계되어 있다. 본 논문은 실행 경로에 따른 모듈의 인자를 자동으로 생성할 수 있는 방법을 제안한다. 프로그램은 블록간의 제어문 결합으로 이루어져 있고 다른 실행 경로를 생성하는 요인은 프로그램내의 제어문을 통해 결정된다. 그러므로 블록간의 결합을 제어의 종류에 따라 연산으로 표현하고 연산의 특성을 분석한다. 그리고 블록의 속성은 조건식을 가지고 있어 블록이 실행되는 조건을 표현한다. 각 연산의 특성에 맞게 조건식을 연산하여 블록이 동작하는 조건을 확인하고 최종적으로 실행 경로에 따른 모듈의 인자를 구하는 방법을 제안한다. 이 방법은 조건식에 영향을 주는 명령만을 추출하는 방법으로 인자를 구하기 위한 계산의 로드를 줄이는 장점이 있다.

1. 서론

응용프로그램의 요구사항이 복잡해지고 사용 환경이 다양해짐에 따라 신뢰성이 높은 소프트웨어의 생산이 어려워지고 있다. 신뢰성이 높은 소프트웨어란 요구사항에 합당한 동작을 하는 소프트웨어를 의미한다. 소프트웨어는 모듈의 결합으로 이루어져 있고 신뢰성 있는 소프트웨어를 작성하기 위해서는 각 모듈이 개발자의 의도에 합당하게 동작해야 한다. 그러므로 신뢰성이 높은 소프트웨어를 생산하기 위해서는 그 구성 요소인 모듈의 테스트가 필요하다.

소프트웨어의 모듈은 블록의 결합으로 이루어져 있고 블록의 실행 경로에 따라 프로그램이 동작한다. 그리고 실행 경로는 제어문의 조건에 따라 정해진다. 그러므로 소프트웨어 모듈을 테스트하기 위해서는 소프트웨어의 경로를 분석하고 경로를 결정하는 조건문을 추출하고 조건문 제약식의 연산을 통해 경로에 따른 테스트 데이터를 생성하여야 한다. 이 과정에서 지정된 경로에 합당한 제약식의 연산을 통해 테스트 데이터를 생성하는 단계가 가장 노력이 많이 들고 자동화하기 어렵다[1].

본 논문은 제어문을 통해 블록의 결합을 연산으로 표현하고 그 특성을 분석한다. 일반적으로 분기를 결정하는 제어문은 두 가지로 나누어진다. 하나는 if문과 같은 분기이고 다른 하나는 while문과 같은 반복이다. 분기문은 조건에 따라 하나의 블록이 실행되면 다른 블록이 실행되지 않는 특성을 가지고 있고 반복문은 조건에 따라 블록의 실행 횟수가 결정되는 특성이 있다. 그러므로 이 두 특성을 분류하

여 다른 연산으로 표현한다.

블록은 자신이 실행되는 조건식을 속성으로 포함하여 블록의 결합을 표현하는 연산의 특성에 따라 조건식을 연산하여 프로그램 실행 경로에 합당한 인자 값을 구한다. 인자의 값을 생성하기 위해서는 조건식에 영향을 미치는 변수의 연산 정보를 확인해야 한다. 본 논문은 분기에 영향을 미치는 조건식의 연산을 추출하는 방법을 제안한다. 한 블록 조건식에 영향을 미치는 변수를 추출하고 그 연산을 통해 분기되는 조건을 찾는 식을 유도하는 방법을 사용한다. 이 방법은 모든 명령을 분석하는 방법이 아닌 분기를 위한 변수의 명령만을 확인할 수 있는 방법으로 기존의 방법보다 적은 연산을 통해 분기 조건을 확인할 수 있는 장점을 가지고 있다.

본 논문은 2장 관련 연구에서 기존의 테스트 데이터 생성 방법을 소개하고 3장에서 제어문의 결합을 연산으로 정의한 후 4장에서 테스트 데이터 생성기의 구조를 소개한다. 그리고 5장에서 결론과 향후 연구 방향에 대해 기술한다.

2. 관련 연구

2.1 임의의 값을 이용한 테스트 데이터 생성 방법[2]

Bird는 "Automatic generation of random self-checking test cases"[]에서 임의의 값을 생성하여 테스트 케이스를 생성하고 테스트 대상에 대해 실험하였다. 이 방법은 테스트 데이터를 쉽게 생성할 수 있는 장점을 가지고 있지만 개발자가 선택한 경로에 따른 테스트 데이터를 생성하기 위해서는 반복적인 많은 작업을 요구하는 단점을 가지고 있다.

2.2 함수 최소화 방법을 이용한 테스트 데이터 생성 방법[3]

Korel은 임의의 값을 사용하는 방법이 경로에 따른 테스트가 어려운 점을 해결하기 위해 임의의 값을 생성하여 실행하는 방법을 변형하여 개발자의 경로에 합당하게 실행될 경우에 테스트를 진행하고 경로에 합당하지 않은 경우 테스트를 중지하고 다른 테스트 데이터를 생성하는 방법을 제안하였다. 이 방법은 기존의 임의의 값을 이용한 방법이 가지고 있는 경로를 결정할 수 없는 약점을 극복하지만 임의의 값이 선택한 경로를 실행하지 않을 경우 비효율적으로 테스트 데이터를 생성한다는 약점을 가지고 있다.

2.3 일차연립방정식을 이용한 테스트 데이터 생성 방법[4]

Gupta는 추출된 조건문의 제약식의 계산을 일차연립방정식으로 적용하여 연립방정식의 해를 계산함으로써 지정된 경로에 따른 테스트 데이터를 생성하는 방법을 제안하였다. 이 방법은 추출된 조건문의 분기 범위를 계산하여 주어진 경로에 따라 올바른 분기가 일어나는 테스트 데이터를 생성하게 한다. 이 방법은 기존의 한 번에 하나의 분기 조건만을 고려한 방법에 비해 적은 연산을 통해 테스트 데이터를 생성할 수 있는 장점이 있다. 하지만 Gupta가 제안한 방법은 분기에만 적용 가능하고 반복과 같은 제어문에 대한 분석을 고려하고 있지 않고 제어문에 영향을 주는 명령문의 추출방법에 대해서는 소개하고 있지 않는다.

3. 제어문 연산

모듈에서 실행의 경로를 결정하는 블록의 제어문은 두 가지로 종류를 가진다. 한 종류는 if문과 같은 분기이고 다른 하나는 while문과 같은 반복이다. 분기가 가지는 특성은 두 블록이 연속적으로 결합하여 조건에 따라 하나의 블록이 실행되면 다른 하나의 블록이 실행되지 않는 특성을 가지고 있다. 그리고 반복은 하나의 블록이 조건에 따라 조건에 참인 경우 블록이 반복되는 특성을 가지고 있다. 또한 모든 분기문은 "if(condition) <block> else <block>"의 형태로 변환할 수 있고 모든 반복문은 "while(condition) <block>"으로 변환할 수 있다[5]. 그러므로 if_else문과 while문의 분석만으로 모든 제어문을 같은 방법으로 적용할 수 있다.

본 논문은 앞에서 소개한 두 가지 종류의 제어를 각각 연산으로 표현한다. 한 종류인 분기는 "+"로 표현한다. 예를 들어 "if(<condition>) <$block_1$> else <$block_2$>" 같은 프로그램 코드는 "$block_1$ <condition> + $block_2$ <~condition>"로 표현한다. 예제에서의 분기는 <condition>이 참일 경우 $block_1$가 실행되고 거짓일 경우 $block_2$가 실행되는 $block_1$과 $block_2$의 결합이다. 또한 "$block^{<condition>}$" 표현에서 "<>"은 block이 실행될 조건식을 표현한다. 또 다른 한 종류인 반복은 "()"로 표현한다. 즉, "while(<condition>) <block>"과 같은 반복문이 있는 프로그램 코드는 "($block^{<condition>}$)"으로 표현법을 사용한다. 또한, 연속적인 블록의 결합은 "·"으로 표현한다. 표 1은 간단한 C 프로그램 모듈이다.

표 1 C 프로그램 모듈

```
1  int input[10];
2
3  int binarySearch(int low, int high, int target){
4    int mid = (low + high) / 2;/*block0*/
5    while(low <= high){/*block1*/
6      mid = (low + high) / 2;
7      if(input[mid] == target){/*block2*/
8        return mid;
9      }else{/*block3*/
10       if(target > input[mid]){/*block4*/
11         low = mid + 1;
12       }else{/*block5*/}
13       if(target < input[mid]){/*block6*/
14         high = mid - 1;
15       }else{/*block7*/}
16     }
17   }
18   return -1;
19 }
```

표 1은 input 배열을 정렬하는 C 프로그램 소스이다. 표 1의 제어문 구조를 분석하여 앞에서 제안한 표현으로 바꾸면 표 2와 같이 나타난다.

표 2 코드를 분석한 표현식

$$block0 \cdot (\{block1 \cdot \{block2^{<input[mid]-target>} + \{\{block4^{<target>input[mid]>} + block5^{<target<-input[mid]>}\} \cdot \{block6^{<target<input[mid]>} + block7^{<target>-input[mid]>}\}\}\}^{<input[mid]!-target>}\}^{<low<-high>}$$

표 2의 표현식에서 block4가 실행되는 조건식을 계산하기 위해서는 선행되는 명령 중에서 block0 · block1의 연산을 계산하여야한다. 실제 추출되는 계산식은 표 3과 같다.

표 3 추출되는 조건식과 계산식

조건식	계산식
target > input[low+high] low <= high	mid = (low+high)/2

표 3의 조건식에서 target과 input, low, high의 변수가 조건식을 분기시키는 변수이지만 mid는 low와 hight를 통해 생성되는 변수이다. 조건식의 연산 방법은 수치해석적인 방법[4]를 사용한다.

4. 시스템 구조

표 2와 같은 표현식은 구조를 분석하기 어렵다. 그러므로 그림 1과 같이 연산으로 블록이 결합되는 추상 구문 트리를 제안한다. 생성된 추상 구문 트리를 사용자에게 테스트 대상이 되는 실행 경로를 입력 받을 때와 조건식을 추출할 때 사용된다.

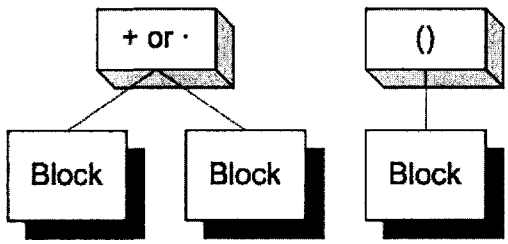

그림 1 연산으로 표현된 추상 구문 트리

본 시스템은 테스트 코드를 분석하여 경로에 합당한 테스트 데이터를 생성하는 프로그램이다. 테스트 데이터 생성기는 소스를 분석하여 블록의 연산 구조에 합당한 추상 구문 트리를 생성하고 생성된 추상 구문 트리를 확인하여 사용자가 테스트할 실행 경로를 입력한다. 실행 경로에 따라 실행될 블록의 조건식과 조건식에 영향을 주는 명령을 추출하여 연산한다. 연산에 따라 조건식이 참이 되거나 거짓이 되는 식을 추출하여 모듈의 입력을 찾는다. 그림 2는 제안한 추상 구문 트리를 생성하고 생성된 추상 구문 트리를 이용하여 테스트 데이터를 생성하는 시스템의 구조도이다.

테스트 데이터 생성기는 소스 분석기와 실행 경로 입력기, 조건식 추출/분석기로 이루어져 있다. 소스 분석기는 테스트 소스를 입력으로 받아 제안한 추상 구문 트리를 생성하는 모듈이고 실행 경로 입력기는 추상 구문 트리를 사용자에게 제공하여 사용자가 테스트할 실행 경로를 입력할 수 있도록 하는 모듈이다. 또한, 사용자가 입력한 실행 경로와 추상 구문 트리를 이용하여 조건식과 조건식에 영향을 주는 명령을 추출하고 명령과 조건식을 연산, 분석하여 테스트 데이터를 생성하는 조건식 추출/분석기를 가지고 있다.

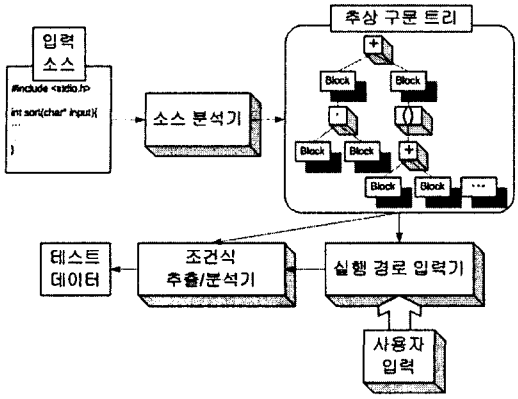

그림 2 테스트 데이터 생성기의 시스템 구조도

5. 결론 및 향후 연구 과제

현재 소프트웨어의 요구사항은 다양하게 증가하고 있다. 그에 따라 소프트웨어의 복잡도가 증가하고 복잡도 증가에 따라 개발자에 의도에 합당한 신뢰성이 높은 소프트웨어 생산이 어려워지고 있다. 그에 따라 기존의 소프트웨어를 테스트하기 위한 연구[6][7][8]가 활발히 이루어졌지만 테스트를 진행하기 위한 테스트 데이터는 개발자가 직접 작성하는 경우가 많았다. 소프트웨어를 테스트하기 위한 테스트 데이터는 프로그램의 실행 경로에 따라 분석되어야 한다. 왜냐하면 소프트웨어의 동작은 실행 경로에 따라 결정되므로 실행 경로에 따른 분석은 소프트웨어 동작 경우에 따른 분석 방법이 된다.

소프트웨어는 모듈로 구성되어 있고 모듈은 다수의 블록의 결합으로 이루어져 있다. 블록은 제어를 통해 결합되어 있는데 제어의 동작에 따라 실행 경로가 결정된다. 본 논문은 모듈의 실행 경로를 분석하기 위해서 모듈을 구성하고 있는 블록간의 결합을 연산으로 표현하는 방법을 제안한다. 또한, 테스트 데이터를 생성할 수 있는 테스트 데이터 생성기의 구조를 소개한다. 테스트 데이터 생성기는 제안한 추상 구문 트리를 생성하고 생성된 추상 구문 트리를 이용하여 사용자로부터 테스트 대상이 되는 실행 경로를 입력 받는다. 해당 실행 경로에 대한 조건식과 명령을 추출, 분석하여 테스트 데이터를 생성한다.

본 논문에서 제안하는 블록 결합의 표현과 테스트 데이터 생성은 모듈의 모든 명령을 분석하지 않고 조건에 영향을 주는 명령만을 분석하여 테스트 데이터를 생성하는 방법으로 경로 분석에 영향을 주는 명령만을 분석하여 테스트 데이터에 적용한다. 그러므로 상대적으로 적은 연산에 의해 테스트 데이터를 생성할 수 있다. 또한, 연산으로의 표현은 수학적인 표현으로 표현이 단순하고 효과적인 문제해결 방법을 제공한다.

본 시스템에서는 사용자에 잘못된 실행 경로 입력에 대해 많은 실행을 거쳐 사용자의 오류를 찾을 수 있다. 예를 들어 표 1에서 block4가 실행되면 block6이 실행되지 않는다. 즉, 해당 인수를 찾을 수 없다. 이 경우 인수의 해가 불능이 된다. 그러므로 소프트웨어의 구조적 분석을 통한 실행 경로 분석은 더욱 효과적인 테스트 데이터를 생성하게 한다.

참고문헌

[1] 정인상, "소프트웨어 테스팅을 위한 테스트 데이터의 자동 생성", *한국정보과학회 논문지*, 제9권, 11호, pp10-18, 2001.11.

[2] Bird, D.L, Munoz, C.U "Automatic generation of random self-checking test cases", *IBM Systems Journal*, Vol 22, pp229-245, 1983.

[3] Korel, B, "Automated Software Test Data Generation", *IEEE Trans. on Software Engineering Methodology*, Vol 5, no.1 pp63-86, 1996.

[4] Gupta, N, Mathur, A, Soffa, M, "Automated Test Data Generation Using An Iterative Relaxation Method" *In Proceedings of Foundations of Software Engineering*, ACM Press, Nov, 1998

[5] Steve McConnell, *CODE COMPLETE*, 1993.

[6] 공기석, 손승우, 임채덕, 김홍남, "내장형 실시간 소프트웨어의 원격디버깅을 위한 디버그에이전트의 설계 및 구현", *한국정보과학회 가을 학술발표논문집 Vol. 26. No 2*, pp.125~127, 1999.

[7] Dr. Neal Stollon, Rick Leatherman, Bruce Ableidinger, "Multi-Core Embedded Debug for Structured ASIC Systems", *proceedings of DesignCon 2004*, Feb, 2004.

[8] L.Hatton, Embedded software testing, Software Testing Congress, 2000