논문 2009-46CI-5-3

소프트웨어 화이트박스 테스트의 교호 강도 수 기반 테스트 방법

(Computing Method for The Number of The Interaction Strength Based on Software Whitebox Testing)

최 형 섭*, 박 홍 성**

(Hyeong Seob Choi and Hong Seong Park)

요 약

소프트웨어가 복잡할수록 소프트웨어의 테스트에 들어가는 비용과 시간이 점점 증가되고 있는 문제점이 존재한다. 이러한 문제를 해결하기 위해서는 테스트에 사용되는 테스트 케이스의 수를 줄이는 것이 중요하다. 특히 단위 테스트 케이스 수를 결정하는 것에는 교호강도의 수가 중요한데 교호강도 수는 소스에서 인자간의 조합에 의해 함수의 결과에 영향을 미치는 인자들의 수를 말한다. 본 논문에서는 프로그램 코드 상에서 인자의 사용 패턴을 분석하여 특정 패턴이 되면 교호 강도 수를 늘리고 최종적으로 교호 강도수를 결정할 수 있는 방법을 제시한다. 제안하는 방법의 효과를 커버리지 및 결함 발견 수의 항목으로 실험을 통해 증명한다.

Abstract

Cost and Time for software test is gradually increasing as the software complexity increases. To cope with this problem, it is very important to reduce the number of test cases used in the software test. The interaction strength number is especially important in decision of the number of test cases for the unit test, where the interaction strength number means the number of arguments which affect the results of a function by the analysis of their combination used in source code of the function. This paper proposes the algorithm that computes the number of the interaction strength, where analyzes the patterns used in the source code of a function and increase its number when the pattern matches one of the specified patterns. The proposed algorithm is validated by some experiments finding coverage and the number of fault detection.

Keywords: Testing, Interaction strength number, Test case

I. 서 론

최근의 소프트웨어 개발에 있어 소프트웨어의 규모가 점점 늘어남에 따라 소프트웨어 개발에서 테스트에 들어가는 비용이 40-80%를 차지할 정도로 많은 비용이 들어가고 있다. 테스트 기법에는 여러 방법이 존재 하지만 소스코드에 대한 접근 방법에 의해 크게 두 가지 접근 방법이 있다. 하나는 테스트 작업이 원시 코드로부터 시작되는 화이트 박스 테스트와 코드 내부의 구조는 고려하지 않고 소프트웨어의 명세에 의해 결정되는 입력과 출력만을 고려하여 테스트를 하는 블랙박스 테스트가 있다.

소프트웨어 테스트에는 테스트케이스가 존재하게 되고 이 테스트 케이스를 이용하여 테스트를 진행한 뒤 테스트 결과를 보게 된다. 이 때 테스트 케이스를 작성

* 학생회원, ** 정회원, 강원대학교 IT 대학
(College of Information Technology, Kangwon National University)
※ 이 논문은 지식경제부의 지원에 의해 연구되어졌다.
접수일자: 2009년7월29일, 수정완료일: 2009년9월4일

하는 것과 테스트를 진행하는 것은 많은 비용과 시간이 들어간다. 이를 줄이기 위해 실제 테스트 시에 결함을 유발 할 수 있는 테스트 케이스를 작성하는 것이 중요한데 이를 위해 체계적이고 효과적인 테스트 케이스를 작성하는 방법이 필요하게 된다. 또한 테스트 케이스를 줄이면서 테스트 커버리지를 높일 수 있는 방법 역시 필요하게 된다.

많은 경우의 테스트에서 테스트 케이스를 작성하는 방법은 시스템에 입력으로 들어가는 값들을 어떻게 조합할 것인지에 대한 방법이 된다. 입력 값들의 후보들이 다른 입력 값의 후보와 동시에 입력되었을 때 결함이 발생할 수 있을 수 있다. 이러한 결함에 대처하기 위해서는 모든 경우의 입력 값들의 조합을 테스트하는 것이 가장 정확한 방법이 될 수 있다. 하지만 모든 조합의 경우를 테스트하는 것은 모든 입력 인자들의 후보들을 조합해야 하기 때문에 많은 수의 테스트 케이스가 나오게 되고 이를 테스트 하는 데는 많은 시간을 필요로 하게 된다[10]. 예를 들면 5개의 입력 인자의 후보를 가진 4개의 테스트 인자가 있을 때 모든 인자의 조합을 구하게 되면 5의 4승 즉 625개의 테스트 케이스가 나오게 되는데 이를 2 way 조합으로 하게 되면 30개 이내로 충분하게 된다.

테스트 케이스를 줄이는 노력은 많은 분야에서 적용되고 연구되어 왔다. 그 중 눈에 띄는 분야는 pairwise를 시용한 테스트 방법으로 테스트 케이스의 파라미터 조합들을 가지고 테스트를 진행하는 것이다. 이는 각각의 테스트 케이스에서 모든 파라미터 쌍들이 전체 테스트 케이스에서 최소한으로만 나타나도록 하는 것이 목적이다. 이를 위해 직교 배열(Orthogonal Array)를 생성하여 테스트 케이스를 생성하는 방법들이 연구되어 왔다[2~3]. 일반적으로 모든 파라미터들의 조합을 이용하여 테스트 케이스를 생성하게 되엇을 때 보다 기하급수적으로 줄어든 테스트 케이스의 수를 얻을 수 있다. Kuhn, Wallace, Gallo[1]은 기존의 개발된 소프트웨어의 오류를 분석한 후, 많은 오류가 2개 이상의 파라미터 간 조합에 의해 발생한다는 것을 밝혀내었고 D.M Gohen 등이 AETG 시스템 테스트를 n-way 파라미터 조합을 통해 실시함으로써 테스트케이스를 줄이면서 좋은 커버리지를 가지는 실험[2]으로 교호작용을 이용한 테스트 케이스의 생성의 타당성을 증명하였다. pairwise를 이용하여 테스트를 진행할 수 있는 적합한 분야는 api테스트, GUI 테스트, Configuration test(S/W, H/W 호환성 테스트) 등이 있다.

그러나 기존의 직교배열 방법이나 n-way 파라미터 조합을 이용한 테스트 케이스의 생성 방법에 있어 기존 연구들은 테스트 케이스의 수를 줄이는 것에 초점을 맞추고 있다. 하지만 테스트의 입장에서 파라미터의 조합을 몇 가지로 할 것인지에 대한 정보는 코드 개발자만이 알 수 있거나 또는 인자들이 어떻게 사용되는지를 알 수 있는 상세한 문서가 있어야만 한다. 즉 파라미터의 어떠한 조합에 의해 버그가 발견될 수 있는지에 대한 연구가 필요하다고 할 수 있다.

본 논문에서는 이러한 테스트 인자의 조합의 수를 효과적으로 구하고 테스트 인자의 조합 수를 구할 수 있는 방법을 제안한다. 본 논문에서 사용된 방법은 화이트 박스 테스트를 기반으로 한다. 또한 테스트를 할 대상은 개발된 소스코드들의 단위테스트의 테스트 시에 사용될 수 있는 방법이다. 이를 위해 본 논문에서는 단위 테스트 시에 입력으로 들어가는 인자들의 관계를 특정 패턴으로 나누고 이를 이용해서 테스트 인자들의 조합을 몇 가지로 해야 하는지를 알려주는 알고리즘을 사용한다. 이를 이용하여 실제 모든 테스트 케이스를 적용했을 때와 축소된 테스트 케이스를 이용하여 테스트 했을 경우의 커버리지와 결함 발견 수를 비교하여 본 논문의 방법이 타당함을 보여줄 것이다.

본 논문은 Ⅱ장에서 관련 연구를 소개하고 Ⅲ장에서 테스트 입력 인자들의 패턴을 나누는 기준을 제시하고 이를 이용한 테스트 조합의 수(교호강도 수)를 결정하는 방법에 대해 기술한다. Ⅳ장에서는 본 논문의 테스트 수행을 위한 테스트 절차에 대해 기술하고 Ⅴ장에서는 본 논문의 방법을 실제 예제를 통해 보여주고 그 후 Ⅵ장에서 결론 및 향후 과제에 대해 논한다.

Ⅱ. 관련 연구

2.1 조합 테스트

조합 테스트는 다양한 입력을 요구하는 시스템이나 함수의 테스트 시에 시도할 수 있는 테스트 방법이다. 조합 테스트 방법은 주로 블랙박스 테스트 시에 시스템에 입력을 주고 그에 대한 출력을 받아서 테스트의 성공 실패 여부를 판단하는 방법이다. 그림 1의 시스템을 보게 되면 n개의 입력들이 존재하게 되고 각 입력들은 동등 분할이나 경계값 분석과 같은 방법들로 만들어진 입력 후보 값들이 존재하게 된다.

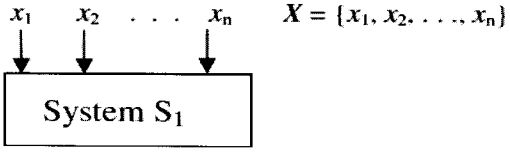


그림 1. n개의 입력을 가진 시스템 S1
Fig. 1. System S1 with n inputs.

시스템을 테스트하기 위해서 각 입력들이 가지는 후보들을 조합하여 테스트 케이스를 생성하게 된다. 각 입력 x1,x2,..xn 들의 후보들의 수를 D(x1) 이라고 하면 테스트 케이스의 총 개수는 TAllcombination과 같다.

$$TAllcombination = D(x1) \times D(x2) \times D(x3) \ldots\ldots D(xn)$$

2.2 교호 작용 테스트과 교호 강도 수

소프트웨어 테스트에서 교호 작용 테스트란 입력으로 들어가는 인자들의 모든 조합에 대해 고려하여 테스트 하는 것이 아니고 불필요한 조합을 제거 하고 실제 결함을 유발 시킬 수 있는 테스트 인자들의 조합을 포함하는 테스트 케이스를 생성하여 테스트를 진행하는 것을 말한다. 이 때 일부 조합하는 인자들의 수가 2이면 pairwise라 하고 3 이상의 인자들의 조합을 가지고 테스트 케이스를 생성하게 되면 n-way(n>3) 테스트라고 부른다. 여기서 n이라는 수를 교호강도 수 또는 조합강도 수 라고 부른다. pairwise 테스트를 처음 시도한 것은 1985년 Mandle[4]의 Ada 컴파일러 테스트 였다. 파라미터는 수학 연산자(+,-) 들과 그것의 타입(integer, long integer, float)이었는데 파라미터의 pairwise set을 만들기 위해 직교 라틴 방격이 도입되었다. 라틴 방격은 k개의 숫자를 어느 행 어느 열에서 하나씩만 있게끔 나열하여 종횡 k개씩의 숫자 또는 글자가 사 각형이 되도록 한 것이다. 라틴 방격은 파라미터의 수가 3개인 경우에만 사용할 수 있다. 예를 들어 파라미터가 A(a1,a2,a3), B(b1,b2,b3), C(c1,c2,c3)가 있을 때 테스트 케이스의 생성은 그림 2의 테이블의 행과 열의 조합을 통해 만들어낼 수 있다.

표 1을 통해 생성된 테스트 케이스를 확인할 수 있다. 테스트 케이스를 살펴보게 되면 인자들의 후보들의 조합이 2로 되어 있고(pairwise) 전체 테스트 케이스에서 각 인자 후보들의 쌍이 중복 없이 나타나는 것을 알 수 있다. 이러한 방법을 확장하여 직교 배열이라는 배열 표를 생성하여 테스트 하는 방법을 Brownlie 등[5]

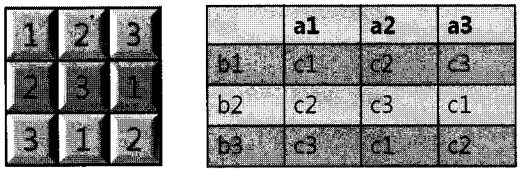


그림 2. 직교 라틴 방격과 조합 테이블의 예
Fig. 2. Examples for orthogonal Latin square and combination table.

표 1. 생성된 pair wise 테스트 케이스
Table 1. Pairwise test cases generated.

No	test case
1	a1 b1 c1
2	a1 b2 c2
3	a1 b3 c3
4	a2 b1 c2
5	a2 b2 c3
6	a2 b3 c1
7	a3 b1 c3
8	a3 b2 c1
9	a3 b3 c2

이 제안하였다. 이 논문에서 그들은 OATS(Orthogonal Array Testing System)이라는 방법을 사용하여 시스템 레벨에서 AT&T's PMX/StarMAIL라는 제품을 테스트 하였다. 그 결과 전체 조합 방법에 비해 결함발견율이 22% 높은 결과를 얻었음을 보여주고 있다. 또한 Cohen 등[2]은 1994년 AETG 시스템을 제안하였다.

AETG(Automatic Efficient Test Generator)는 상업용 테스트 도구로 n-way 테스트 케이스들을 생성할 수 있다. 이러한 n-way 테스트 케이스를 효과적으로 생성하기 위한 방법들에 대해 IPOG 방법 등 많은 연구가 있어왔다[2~3, 6]. 또한 테스트 인자들 간의 상호작용의 수를 인자별로 다르게 하여 모든 인자의 조합은 n-way로 하면서 특정 인자간의 조합은 더 높은 강도의 t-way(t>n)로 할 수 있는 방법을 시뮬레이티드 어닐링 알고리즘을 이용하여 제안한 연구도 있다[7~8].

기존의 연구들은 앞 절에서 말한 바와 같이 테스트 케이스의 수를 줄이는데 초점을 맞추고 있다. 하지만 테스트 인자 간의 교호 강도 수를 어떻게 결정 할 것인지에 대한 연구는 미흡하다. 본 논문에서는 소스코드 상에서 입력 인자들 간의 관계를 정의하고 그 정의에

의해서 교호강도 수를 결정하는 방법을 제시한다.

Ⅲ. 테스트 교호 강도 수 결정 방법

소프트웨어 단위 테스트 시에 입력으로 들어가는 인자들의 모든 테스트 케이스를 적용하기에는 시간과 비용이 들기 때문에 테스트 케이스의 수를 줄이면서 커버리지를 만족시킬 수 있는 방법이 필요하게 된다. 이를 줄이기 위해 직교배열과 같이 테스트 인자의 상호 관계에 따라 테스트의 교호강도의 수를 정하여 테스트 케이스를 줄이는 것이 중요하다. 본 논문에서는 소스코드 상에 포함되어 있는 입력인자들의 관계를 살펴봄으로서 테스트 시 몇 개의 인자의 조합을 가지고 테스트 케이스를 생성할 것인지를 결정하는 방법에 대해 이야기 한다. 이를 위하여 소스코드를 분석하고 표 2에서 제시하는 방법에 의해 교호강도 수를 결정하게 된다. 테스터는 테스트 케이스를 효과적으로 줄일 수 있게 되고 이로 인해 단위 테스트 시에 들어가는 노력과 비용이 감소될 수 있고 테스트 케이스를 작성하는 과정을 자동화함으로서 빠르게 테스트를 진행할 수 있게 된다.

표 2. 교호 강도 결정 규칙
Table 2. Decision rules for interaction strength.

패턴 번호	형식	설명	예시	교호강도 수 계산법

1	A∧B	한 문장 내에서 인자가 함께 쓰이는 경우	If(A=0 && B="dd"){…
			If(A == B) {…
			If, for, while
				1 증가

2	A∨B	A또는 B가 선택적으로 사용되는 경우	If(A){} else (B){}	증가 없음

3	A←B	A의 사용 후 연속적으로 나타나는 문장에서 B가 쓰이는 경우	…
			int a=A;
			int b=B;
			…
				1 증가

4	A⊃B	A 이후의 블록문 내에서 B 가 사용되는 경우	if(A){
			int b=B;
			}
				1 증가

5	1∧2 (A∧B)∨C		If(A && C) {} else (B) {}	1 증가
6	1∧3 (A∧B)←C	관계없음		2 증가

7	1∧4
	(A∧B)∨(A⊃B)
		A나 B의 조합 다음 C가 따라오는 경우	If(A=0 && B="dd") {
			int c=C;
			}
				3 증가

8	2∧3 (A∨B)∨(A←B)	관계없음		1 증가
9	2∧4 (A∨B)⊃C	7과 동일		3 증가

10	3∧4
	(A←B)⊃C
			if(A){
			int c=C;
			int b=B;
			}
				3 증가

11	return⊃A	리턴에 영향을 주는 인자를 추적		인자의 개수 만큼 더한다.
12	출력에 관계된 인자	출력에 관계된 인자는 테스트 케이스의 조합에 포함하지 않는다.		증가 없음
13	A∧B∧C∧D 같은 경우와 같이 상호작용이 일어나는 인자의 수가 지속적으로 늘어날 경우	강도수의 증가에 영향을 주는 번호는 1,3,4로써 이 조합이 더해지게 되면 강도수를 증가시킨다.	A∧B∧C∧D 같은 경우	더해지는 인자의 수만큼 교호 강도 수를 증가시킨다.
14	중복	같은 인자가 여러곳에서 사용되었을 때는 가장 큰값만을 계산한다.		

표 2에서 테스트 인자간의 관계는 동시사용, 선택적 사용, 종속관계, 포함관계(∧,∨,←,⊃)가 있을 수 있다. 함수의 파라미터에는 입력인자와 출력인자가 있게 되는데 함수의 출력에 영향을 주게 되는 입력인자의 상호관계가 중요하게 된다.

1)동시 사용: 동시 사용이란 소스코드의 한 문장(문장이란 일반적으로 ; 로 끝나는 코드)안 에서 입력으로 들어가는 인자들이 같이 사용되는 경우를 이야기 한다. 한 문자에서 같이 인자가 사용되는 것은 두 인자 사이의 관계가 함수 출력에 영향을 줄 수 있다. 표기는 ∧으로 한다.

2)선택적 사용: 선택적 사용이란 소스코드의 한 문장 안에서 입력인자가 둘 이상 사용 되었을 경우를 이야기 한다. 이때는 인자들 간에 서로 영향을 끼치지 않게 되기 때문에 이 인자들의 조합을 테스트할 필요가 없게 된다. 표기는 ∨로 한다.

3)종속관계: 테스트 인자들이 종속관계에 있다는 것은 소스코드 내에서 출력에 영향을 줄 수 있다는 것이다. 영향을 주기 위해서는 출력에 영향을 주는 변수와 관련된 곳에 사용되었을 경우이다. 소스 상에서 대괄호 안의 연속된 문장에서 인자들이 사용되게 될 때 종속관계에 있다고 할 수 있다. 표기는 ←으로 한다.

4) 포함관계 : 테스트 인자 하나가 다른 인자의 사용이 또다른 인자의 사용에 영향을 준다는 것이다. 이는 하나의 인자가 사용되고 그 다음 오는 문장에 앞의 인자 사용문장에 포함되는 경우를 이야기 한다. 표기는 ⊃으로 한다.

위의 4가지의 기본적인 조합의 경우를 가지고 소스를 분석하게 된다. 소스분석을 할 때 우선 4가지의 기본 조합이 있는지를 살펴보고 있다면 그 다음으로 조합수를 증가시키게 된다. 이 때 다시 기본적인 조합에 또 다른 조합이 있는지를 살펴야 한다. 이러한 방식으로 소스를 분석할 수 있는데 나올 수 있는 교호강도의 경우의 수는 매우 많아질 수 있다. 하지만 표 2에 나온 것처럼 조합 수를 증가시켜야 할 경우는 1, 3, 4번 패턴이 관련된 경우와 11번과 같이 함수의 리턴 값에 직접 영향을 주는 경우가 있다고 할 수 있다. 재귀적으로 1, 3, 4번 패턴이 관련된 경우에 테이블의 패턴대로 계산해 주면 된다. 이러한 패턴들을 계속적으로 소스를 분석하여 조합(교호강도) 수를 구할 수 있다. 그림 3은 예제 코드를 보여준다.

Test 함수에서 사용된 a,b,c,d를 표 2의 표의 패턴

```
void test(int a,int b,int c,int d)
{
int x=0,y=0;
if ( a > 0 )
{
x = 2;
}
else
{
x = 5;
}

if ( b > 0)
{
y = 1 + x;
}

if ( c > 0)
{
if ( d > 0)
{
output(x);
}
else
{
output(10);
}
}
else
{
output(1/(y-6));
}
}
```

그림 3. 예제
Fig. 3. Example.

번호에 맞추게 되면 a와 b는 연관관계가 없는 것을 알 수 있다. c와 d의 경우에 패턴 2와 매핑이 된다. 하지만 이 경우에 c와 d는 관계가 없기 때문에 초기의 교호강도 수인 2로 결정되어서 총 4개의 테스트인자들의 모든 조합을 고려하지 않고 2개의 조합만 가지고 테스트 케이스를 생성하여 실행해도 모든 커버리지를 만족할 수 있게 된다. 이 경우 모든 조합을 사용하게 되면 총 5의 5승의 개수만큼의 테스트 케이스가 만들어지게 된다. 하지만 본 논문의 방법을 이용하여 인자의 교호강도 수인 2로 하여 테스트 케이스를 만들게 되면 30가지의 테스트 케이스만 생성되게 된다.

그림 4는 테스트 교호강도 수를 결정하는 방법의 순서도이다. 각 인자가 무엇인지를 인식한 후 교호강도의 수를 2로 초기화하는데 이는 최소한의 강도수가 2가 되어야 하기 때문이다. 이후 계속적으로 소스를 분석하면서 표 2에 있는 패턴이 검출되는지를 살피게 된다. 교호 강도 수는 소스 내에서 여러 번 검출 될 수 있다. 검출 된 교호강도 수들 중에서 가장 큰 값을 가지는 교호강도수를 n-way 테스트에서 사용되는 n값으로 한다.

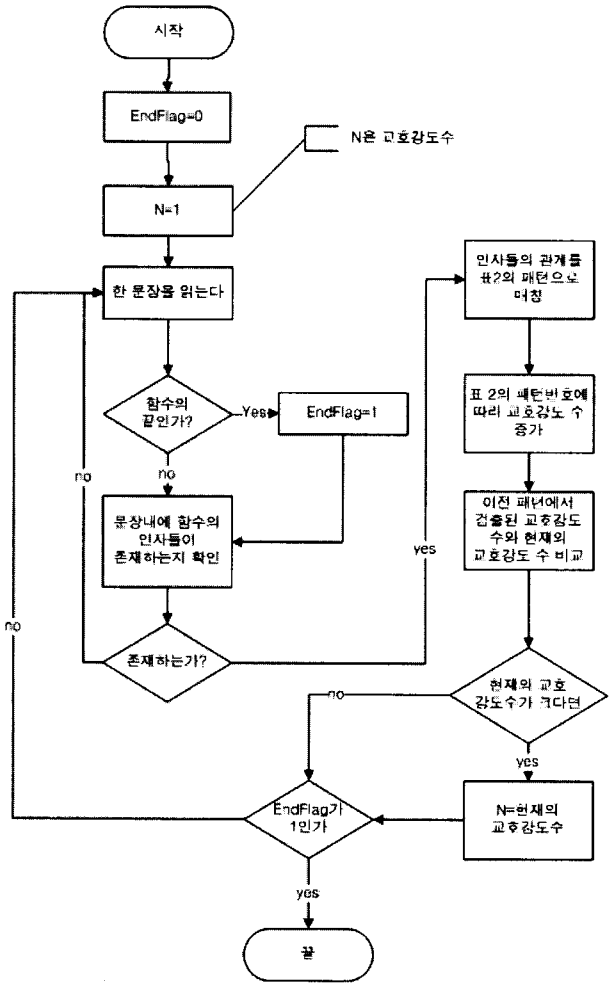


그림 4. 교호강도 수 결정 알고리즘
Fig. 4. Algorithm computing the number of interaction strength.

IV. 교호 강도 수 기반 테스트 방법

3절에서 제안한 방법을 이용해서 소프트웨어의 단위 테스트를 수행할 수 있게 된다. 테스트 시에 가장 중요한 단계는 테스트 케이스를 잘 작성하는 것이다. 테스트 케이스에는 일반적인 경우의 테스트 케이스들이 들어가지만 시스템이나 함수의 결함을 유발 시킬 수 있는 의도적인 잘못된 인자 값들도 들어갈 수 있어야 한다. 테스트 케이스가 생성되고 나면 실제 테스트를 수행할 수 있는 테스트 드라이버가 준비되어야 한다. 본 논문에서 제안한 교호 강도 수의 파악을 위해서 테스트 케이스 생성 전 단계에서 소스 코드를 분석하여 3절에서 보인 교호강도 수를 결정하는 알고리즘이 수행되어야 한다. 테스트 드라이버는 테스트 케이스를 입력으로 받아 테스트를 수행하는 실행파일이 될 수 도 있고 컴파일 가능한 코드 또는 스크립트가 될 수 도 있다. 테스트 수행 시 테스트 결과를 리포트 해주는 역할도 테스트 드라이버가 하게 된다. 조합 테스트 방법에서 테스트 드라이버는 테스트 레벨에 따라 자동으로 생성될 수 도 있고(하위레벨, 함수 테스트) 테스터에 의해 수동으로 수행될 수 도 있다. 테스트 드라이버에 의해 나오는 결과 파일에는 테스트 케이스 별로 입력에 대한 결과들을 가지고 있어야 하고 수행된 커버리지 계산결과등도 가지고 있을 수 있다. 커버리지 테스트란 테스트 케이스에 의해 테스트 대상 프로그램의 어떤 부분들이 수행이 되었는지를 확인하는 것이다. 테스트 결과 분석 단계에서는 테스트 오라클과 실제 수행 결과를 비교하여 테스트의 성공 실패 여부를 판단하게 된다. 테스트 오라클이란 테스트 수행 시 성공과 실패를 판가름 할 수 있는 기준을 말한다. 단위 테스트 시에는 주로 예상결과 값이 있어야 하는데 이 예상 결과 값을 테스트 오라클이라고 할 수 있다. 또한 커버리지 결과를 분석하여 100%가 아닐 경우에는 어떤 부분의 코드가 수행되지 않았는지 또는 어떤 수행 경로가 수행되지 않았는지를 살피게 된다. 이를 이용해서 결함을 찾을 수 있게 된다.

V. 실 험

교호강도 수를 측정하기 위해 총 3가지의 함수를 대상으로 실험을 하였다.

3가지의 함수는 수치해석에서 방정식의 근사 해를 찾기 위한 Regular-falsi 함수와 픽셀 간 거리를 측정하기 위한 get_distance함수 그리고 행렬계산을 위한 matrix_operation 함수이다. 모든 함수들은 C코드로 작성되어 있으며 각 함수들의 인자들의 정보는 표 3과 같다.

표 3. 함수와 관련 인자
Table 3. Function and related arguments.

함수 이름	인자 이름	인자 타입	범위(min,max)
Regular-falsi	a	double	-100, 100
	b	double	-100, 100
	error	double	0.00001, 1
	Max	int	1, 200
get_distance	x1	int	1, 800
	y1	int	1, 600
	x2	int	1, 800
	y2	int	1, 600
matrix_operation	a1	int	1, 5
	a2	int	1, 5
	b1	int	1, 5
	b2	int	1, 5
	operator	char	+,-,*

테스트 진행 방법은 Ⅳ장에서 이야기한 테스트 수행 절차에 따라 진행하였다. 테스트 케이스를 생성하기 전 논문에서 제시한 방법에 따라 각 함수들의 교호 강도 수를 측정하였다. 함수 인자들의 후보들은 동등분할 방법을 이용하여 작성하였다. 그리고 각 함수 인자들의 범위를 설정하고 각 인자별로 범위 바깥의 값들과 경계값 그리고 범위 내의 값들을 선택하여 각 인자들의 후보 값들을 결정하였고 이를 이용하여 테스트 케이스들을 생성하였다. 테스트 케이스를 생성하는데 사용된 도구는 jenny[9]를 사용하였다. jenny는 사용자의 입력에 따라 원하는 n-way의 테스트 케이스들을 생성해주는 도구이다. 테스트 케이스는 각 인자들의 후보들을 조합하여 함수로 들어가는 인자들의 조합을 가지는 리스트이다. 각 함수의 교호강도 수는 Regular-falsi 의 경우 3이 나왔고 get_distance 의 경우는 2가 나왔고 matrix_operation의 경우 3의 강도수가 나왔다. 실제 테스트 케이스를 생성했을 때의 각각의 함수의 테스트 케이스의 수들은 표 4에 나타나 있다. 테스트 생성은 각 함수마다 2way 에서부터 all combination 까지의 테스트 스윗들을 생성하여 실행을 하였다. matrix_operation 의 경우에는 인자의 수가 5개 이므로 5way 까지 테스트 케이스를 생성할 수 있다. 표 4에서 음영이 표시된 칸은 각 함수들에서 계산된 최적의 교호강도 수를 나타낸다.

테스트 대상 함수들은 실제로 모든 테스트 케이스에서 결함을 유발시키지 않는 다는 것을 확인 후 결함을 검출하기 위해서 각 함수 소스코드에 결함을 발생시킬 수 있는 코드를 집어넣었다. 결함을 유발시키는 방법은 코드상의 인자들이 조합되는 경우에 함수의 기대 값과 다른 값을 출력시킬 수 있는 결함을 삽입하였다. 결함 삽입의 종류는 if문 같은 제어문의 판단 조건을 바꾸는 결함이 있고, 리턴 되는 결과 값 자체를 바꾸도록 하는 경우가 있었다. Regular-falsi 함수의 경우에는 2가지의 결함을 넣었고, get_distance 함수의 경우에는 4가지, 그리고 matrix_operation 함수의 경우에는 5가지의 결함이 삽입되었다.

표 4. 교호강도 결정 알고리즘에 의해 결정된 교호강도 수

Table 4. The number of interaction strength computed by the suggested algorithm.

함수	2way	3way	4way	5way
Regular -falsi	30	153	625	N/A
get_dist ance	30	153	625	N/A
matrix_o peration	38	207	971	3750

표 5. 커버리지 및 결함 발견 수

Table 5. Coverage results and the number of fault detection

함수이름	테스트 결과			
	교호강도 수	테스트 케이스 수	커버리지(%)	결함 발견
Regular-falsi	2	30	89	1
	3	153	100	2
	4	625	100	2
get_distance	2	30	100	4
	3	153	100	4
	4	625	100	4
matrix_operation	2	38	92	3
	3	207	100	5
	4	971	100	5
	5	3750	100	5

표 5는 각 테스트 함수들을 각각의 n-way 테스트로 실행하였을 때의 결과를 나타낸다. 모든 함수에서 커버리지는 논문에서 제시한 방법에 의해 선택된 교호강도 수의 커버리지보다 낮은 수의 n-way 테스트의 경우가 적게 나온 것을 볼 수 있다. 선택된 교호강도 수 보다 낮은 수로 테스트 케이스를 생성하게 되면 조합되지 않는 경우의 테스트 케이스가 생길 수 있게 되기 때문이다. 또한 결함의 발견 빈도도 커버리지 경우와 비슷한 결과를 얻을 수 있었다. 커버리지 경우와 마찬가지로 결함이 있는 라인을 테스트 하지 못하는 경우가 생겼기 때문이다. 즉 본 논문에서 제시한 방법을 이용한 것이 더 높은 커버리지 결과와 결함 발견 결과를 얻을 수 있는 것으로 볼 수 있다.

Ⅵ. 결 론

본 논문에서 최적의 교호강도 수를 결정하는 방법을 제안하였고, 이를 실험을 통해 검증하였다. 테스트 케이스는 테스트 작업의 입력으로 들어가는 파라미터들을 말한다는 가정 하에 이 입력들의 관계를 살핌으로써 교호작용의 강도를 최적의 값으로 선택하였다는 것을 보여주었다. 실험에서 보였듯이 제안하는 방법에 의해 결정된 교호강도 수가 더 효과적으로 테스트 커버리지를 높이면서 결함을 찾을 가능성도 더 높게 나오는 것을

알 수 있었다.

본 논문은 테스트 교호강도 수를 임의로 설정하는 것이 아닌 소스를 사용하여 최적의 테스트 케이스의 수를 결정하는 방법을 제안하였다. 앞으로는 본 논문에서 제시한 방법을 블랙박스 테스트나 테스트 케이스를 조합하는 테스트 종류에 적용할 수 있는 방법에 대해 연구할 예정이다.

참 고 문 헌

[1] D. Richard Kuhn, Dolores R. Wallace, Albert M. Gallo Jr., "Software Fault Interactions and Implications for Software Testing," IEEE transactions on Software Engineering, vol. 30, no. 6, pp. 418-421, June 2004.

[2] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, Gardner C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE transactions on Software Engineering, vol. 23, no. 7, July 1997.

[3] Lei, Y and Tai, K. C., In-Parameter-Order: A Test Generating Strategy for Pairwise Testing, High-AssuranceSystems Engineering Symposium, 1998. Proceedings. Third IEEE International 13-14 pp. 254-261, November 1998.

[4] R. Mandle, "Orthogonal Latin squares: An application of experimental design to compiler testing," Communication of the ACM, vol. 28 no. 10 pp. 1054-1058, 1985.

[5] R. Brownlie, J Prowse, and M.S. Phadke, "Robust Testing of AT&T PMX/StarMAIL Using OATS," AT&T Technical Journal, Vol. 71 No. 3, pp. 41-47, May/June 1992.

[6] Soumen Maity, Amiya Nayak, "Improved Test Generation Algorithms for Pair-Wise Testing," Proc. 16th IEEE International Symposium on Software Reliability Engineering, pp. 244-253, November 2005.

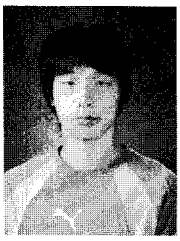
[7] M. B. Cohen, C. J. Colbourn, J.S. Collofello, P. B. Gibbons and W. B. Mugridge, "Variable Strength Interaction Testing of Components," In Proc. of the Intl. Computer Software and Applications Conference, (COMPSAC 2003), Dallas TX, pp. 413-418, 2003.

[8] M. B. Cohen, C. J. Colbourn and A.C.H. Ling, "Augmenting simulated annealing to build interaction test suites," 14th IEEE Intl. Symp. on Software Reliability Engineering(ISSRE 2003), Denver CO, pp. 394-405, November 2003.

[9] http://burtleburtle.net/bob/math/jenny.html

[10] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. IPOG: A general strategy for t-way software testing. In Fourteenth Int. Conf. Engineering Computer-Based Systems. pp.549 - 556, March 2007.

저 자 소 개



최 형 섭(학생회원)
2008년 강원대학교 IT학부 전자통신공학과 학사 졸업.
2009년 강원대학교 IT학부 전자통신공학과 석사 과정.
<주관심분야 : 컴퓨터 언어, 소프트웨어 테스팅, 컴퓨터 네트워크>

박 홍 성(정회원)-책임저자
1983년 서울대학교 제어계측공학과 학사 졸업.
1986년 서울대학교 제어계측공학과 석사 졸업.
1992년 서울대학교 제어계측공학과 박사 졸업.
<주관심분야 : 로봇 S/W, 무선데이터통신, 실시간 통신, 매체 제어 분석>