테스트 케이스의 결함 실행확률을 이용한 향상된 결함 위치추정(fault localization) 기법

문석현, 김윤호, 김문주

한국과학기술원 전산학과
대전광역시 유성구 구성동 373-1
sukhyun5@kaist.ac.kr, kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

요약: 커버리지 기반 결함 위치추정 기법(CBFL)은 프로그램 디버깅에 사용되는 개발자의 노력을 줄이기 위해서, 프로그램 커버리지를 이용하여 결함(fault)으로 의심되는 코드들을 개발자에게 의심도 순으로 자동 제시한다. 하지만 CBFL 기법에서 사용되는 테스트케이스 중 결함을 실행했음에도 실패(failure)하지 않는 테스트 케이스의 수가 많을 경우, 결함 코드가 그렇지 않은 코드들보다 낮은 의심도를 가지는 문제점이 있다. 본 논문에서는 해당 문제의 해결을 위해서, CBFL 에서 사용되는 각 테스트 케이스가 프로그램 결함을 실행했을 확률에 따라 테스트 케이스에 가중치를 부여하고, 이를 통해 CBFL 의 정확성을 향상시키는 새로운 방법을 제시한다. Sir benchmark suite 의 Tcas 에 본 논문에서 제시된 방법을 이용하여 실험한 결과, 기존 방법 대비 최대 43%의 성능 향상이 있었음을 확인 할 수 있었다.

핵심어: 결함 위치추정(fault localization) 기법, 결함 가중치(fault weight)

1. 연구배경

결함 위치추정(fault localizatioin)은 프로그램을 실패(failure) 시키는 원인인 프로그램의 결함(fault)을 찾기 위한 과정이다. 하지만, 결함 위치추정은 일반적으로 개발자가 직접 실패가 발생한 프로그램 실행을 추적해서 결함을 찾아야 하기 때문에 매우 많은 노력과 시간을 필요로 한다[1]. 따라서 이러한 노력을 줄이기 위해 많은 수의 결함 위치 추정기법이 제안되었다.

많은 종류의 결함 위치 추정 기법 중 각광받는 결함 위치 추정 기법은 커버리지 기반 결함 위치추정 기법(CBFL: Coverage Based Fault Localization)[2,3,4,5,6]이다. CBFL 기법은 테스트 케이스들을 이용하여 프로그램을 실행한 후 얻은 커버리지 정보와 실행결과를 기반으로, 결함으로 의심되는 코드(e.g., 구문)들을 개발자에게 의심도 순으로 자동 제시하고, 개발자가 제시된 순으로 코드를 검사케 함으로써 결함를 찾는데 요구되는 노력을 줄이고자 한다. 각 코드의 의심도는 코드와 프로그램 실패의 상관관계로 구해지는데, 주어진 테스트 케이스 집합에서 프로그램 코드가 적은 수의 성공한 테스트 케이스에 의해 실행되는 동시에 많은 수의 실패한 테스트 케이스에 의해서 실행되면 해당 코드는 높은 의심도를 가지게 된다. 즉, 프로그램 실패가 발생했을 때 해당 코드가 실패를 발생시켰을 가능성을 의심도로 나타낸다. 하지만 사용되는 테스트 케이스 집합에서 결함 코드를 실행했음에도 프로그램 실패가 관찰되지 않은 않는 성공 테스트 케이스(CCT: Coincidentally Correct Test case)의 수가 많을 경우, 결함 코드는 그렇지 않은 많은 수의 코드들보다 낮은 의심도를 가지게 되고, 이는 CBFL 기법의 정확성을 떨어뜨린다[2].

본 논문에서는 성공 테스트 케이스가 CCT 일 확률을 추정하고, 이를 이용한 새로운 CBFL 기법을 제시한다. Sir benchmark suite[7]의 Tcas 32 개 버전들에 적용한 결과, 21 개의 버전들에서 기존 기법[3]보다 더 좋은 성능이 관찰되었다.

2. 관련 연구

Jones 와 동료들이 Tarantula[3]라고 불리는 효과적인 CBFL 기법을 제안한 이후, CBFL 기법의 정확성을 향상시키기 위한 많은 연구들이 진행되었다. [4]는 결함 코드의 의심도를 높이기 위해 의심도를 계산하는 새로운 공식을 제안하였고, [5]는 Tarantula 에서 결함 위치추정에 사용되는 구문 커버리지를 정의-사용 쌍, 분기 커버리지와 조합하여 의심도를 계산한다.

CBFL 기법에 사용되는 테스트 케이스 집합에서 CCT 의 수가 많을 경우 정확도가 떨어지는 문제점을 해결하기 위해 많은 방법들이 제안되었다. [2]에서는 프로그램 결함의 유형에 따라, 프로그램 실행 후 생성된 커버리지를 변형하여 정확성을 향상 시킨다. 하지만, 커버리지를 변형시키기 위해 결함의 유형을 알아야 한다는 문제점이 있다. [6]은 CBFL 에서 사용되는 테스트 케이스 집합에서 CCT 집합을 예측하고, 예측된 CCT 집합을 제거한다. 하지만, CCT 를 잘못 예측(거짓 참)할 경우 CBFL 의 정확성이 더 떨어지는

	TC1	TC2	TC3	TC4	TC5	TC6		
void Example (int x, int y){	5.1	9.1	9.0	10.1	1.2	4.7	Tarantula	WTFL
1: if(x>y){	●	●	●	●	●	●	0.50	1.72
2: x = x-2; // x = x+2;	●	●	●	●			0.62	1.8
3: printf("x>y");	●	●	●	●			0.62	1.8
4: }else{					●	●	0.00	0.6
5: y = y+2;}					●	●	0.00	0.6
6: if(x<y+6){	●	●	●	●	●	●	0.50	1.72
7: printf("x<y+6");}	●				●	●	0.71	1.6
} Pass/Fail	F	P	P	P	P	P		

[그림 1] 틀린 프로그램과 테스트 케이스 및 각 테스트 케이스의 커버리지와 각 구문의 의심도

문제점이 있다. 본 논문에서는 [2]의 제약과 [6]의 거짓 참 없이, CCT 를 고려하여 CBFL 의 정확성을 높이는 방법을 제시하고, CBFL 기법의 대표적 연구인 [3]과 비교한다.

3. 가중치가 부여된 테스트 케이스 기반의 프로그램 결함 위치추정(WTFL: Weighted Test cases based Fault Localization)

3.1 커버리지 기반 결함 위치추정(CBFL) 기법

CBFL 기법은 테스트 케이스를 이용하여 프로그램을 실행한 후 얻은 커버리지와 실행결과를 이용해, 각 코드가 결함 코드일 가능성(의심도)을 계산한다. Tarantula[3]에서는 구문 커버리지를 기반으로 공식 (1)을 이용해 구문 s와 프로그램 실패의 상관관계를 이용해 의심도를 계산한다.

$$Susp(s)_{Tarantula} = \frac{\frac{|failed(s)|}{|totfailed|}}{\frac{|failed(s)|}{|totfailed|} + \frac{|passed(s)|}{|totpassed|}} \quad (1)$$

공식(1)에서 failed(s)는 s를 실행한 실패 테스트 케이스의 집합, passed(s)는 s를 실행한 성공 테스트 케이스의 집합, totfailed는 대상 프로그램 테스팅에 사용되는 모든 실패 테스트 케이스의 집합, totpassed는 모든 성공 테스트 케이스의 집합을 나타낸다.

그림 1 의 경우를 예로 살펴보자. 예제는 결함(라인 2)을 가지고 있는 프로그램으로, x 와 y 간의 크기 비교 후 x 또는 y 에 대한 연산을 하고 조건에 따라 printf 구문을 실행한다. 결함은 2 번 라인에 있으며, 정상 프로그램(결함이 없는 프로그램)은 라인 2 를 제외한 모든 코드가 틀린 프로그램과 같다(정상 프로그램에서 라인 2 는 'x = x+2'). 그림의 오른쪽 열에는 6 개의 테스트 케이스들(TC1,2,…,6)과 각 테스트 케이스를 실행하고 얻은 구문 커버리지 정보가 주어져 있다. TC1(x=5,y=1)의 실행결과는 옳은 프로그램에서의 실행 결과(라인 6 의 'x<y+6'이 출력되지 않음)와 다르므로 프로그램 실패를 발생시킨 것으로 간주되고, 그림 1 의 가장 아랫행에 실패를 의미하는 F 로 표시된다(같으면 P(성공)로 표시). CBFL 기법에서 개발자는 정상 프로그램의 실행 결과를 알고 있다고 가정한다. 공식 (1)을 이용한 각 구문의 의심도는 그림의 가장 오른쪽에서 두 번째 행(Tarantula)과 같이 계산된다. 예를 들면, 라인 2 는 1 개의 실패 테스트 케이스(TC1)와 3 개의 성공 테스트 케이스(TC2,3,4)에 의해 실행되므로 약 0.62 의 의심도를 가지게 된다(1 / (1 + 3/5) ≈ 0.62). 개발자는 의심도 순으로 최대 3 개의 구문(라인 7,3,2)을 검사함으로써 결함 구문(라인 2)을 찾을 수 있다.

3.2 CCT: Coincidentally Correct Test case

CCT 는 프로그램 결함을 실행했지만 프로그램 실패가 발생하지 않는 테스트 케이스로, CBFL 기법의 정확성을 떨어뜨리는 가장 큰 요소 중 하나다[2]. CCT 의 수가 많을수록 공식 (1)에서 결함 코드의 passed(s) / totpassed 값이 커지므로, 결함 코드의 의심도는 떨어지고, 이는 결함 코드가 정상 코드들보다 낮은 의심도를 가지게 한다. 예를 들면, 그림 1 에서 TC2,3,4 가 결함 구문(라인 2)의 passed(s) / totpassed 의 값을 증가시켜서, 결함 구문은 정상 구문(라인 7)에 비해 낮은 의심도를 가지게 된다.

Tarantula 기법에서는 많은 수의 성공 테스트 케이스들이 결함 코드를 실행하면 할수록 해당 테스트 케이스들이 실행한 구문들의 의심도는 낮아진다. 하지만 성공 테스트 케이스와 실패 테스트 케이스가 비슷한 구문을 많이 실행할수록, 성공 테스트 케이스가 결함 코드를 실행했을 가능성은 점점 높아지므로, 해당 성공 테스트 케이스가 실행한 구문들의 의심도는 높아져야 한다. 예를 들면, 성공 테스트 케이스와 실패 테스트 케이스가 동일한 구문을 실행하면 성공 테스트 케이스는 결함 코드를 반드시 실행하기 때문에 해당 성공 테스트 케이스가 실행한 구문들의 의심도는 높아져야 한다. 3.3 장에서는 성공 테스트 케이스가 결함 코드를 실행했을 확률을 추정하는 방법과 그 확률에 따라 해당 테스트 케이스가 실행한 구문들의 의심도를 높일 수 있는 공식을 제시한다.

3.3 테스트 케이스에 대한 가중치 부여와 의심도 공식

3.3.1 테스트 케이스에 대한 가중치 부여

각 성공 테스트 케이스의 가중치는 각 성공 테스트 케이스가 결함을 실행했을 확률, 즉 해당 성공 테

스트 케이스가 각 실패 테스트 케이스가 실행한 구문들 중 몇 개를 실행했는지에 따라 결정된다.

$$Weight(p) = \frac{\sum_{f \in totfailed} \frac{|S_f \cap S_p|}{|S_f|}}{|totfailed|} \qquad (2)$$

S_f는 실패 테스트 케이스 f가 실행한 구문들의 집합, S_p는 성공 테스트 케이스 p가 실행한 구문들의 집합을 나타내고, totfailed는 모든 실패 테스트 케이스의 집합이다. 예를 들면, [그림 1]에서 TC2 는 TC1(실패 테스트 케이스)이 실행하는 5 개 구문(라인 1,2,3,6,7) 중 4 개(라인 1,2,3,6)를 실행하므로 0.8 의 가중치를 가진다(4/5 = 0.8).

3.3.2 가중치가 부여된 테스트 케이스 기반 의심도 공식

공식(1)은 가중치가 부여된 테스트 케이스를 각 구문의 의심도에 반영할 수 없으므로, 본 논문에서 제안하는 Weighted Test cases based Fault Localization(WTFL) 기법에서는 이를 반영할 수 있는 새로운 의심도 공식(3)이 사용된다.

$$\text{Susp(s)}_{\text{WTFL}} = \frac{\text{failed(s)}}{\text{totfailed}} + \frac{\sum_{\text{p} \in \text{passed(s)}} \text{weight(p)}}{|\text{passed(s)}|} \qquad (3)$$

공식(3)의 왼쪽 항은 구문 s와 실패 테스트 케이스의 상관관계를 나타내는 것으로, s가 많은 실패 테스트 케이스에 의해 실행될수록 의심도는 증가한다. 오른쪽 항은 구문 s를 실행한 각각의 성공 테스트 케이스들이 CCT 일 가능성을 평균한 것으로, 그 값이 높을수록 의심도는 증가한다. 예를 들면, 라인 2 는 1 개의 실패 테스트 케이스와 4/5 의 가중치를 가진 3 개의 성공 테스트 케이스에 의해서 실행되므로 1.8 의 의심도를 가진다(1/1 + (4/5+4/5+4/5)/3 = 1.8)

그림 1 의 예제에 WTFL 기법을 적용하면, 기존 CBFL 기법에 비해 정확성이 향상됨을 확인할 수 있다. Tarantula 기법에서는 결함을 찾기 위해 최대 3 개의 구문을 검사하지만, WTFL 기법에서는 최대 2 개 구문(라인 3,2)의 검사를 통해 결함을 찾을 수 있다.

4. 가중치가 부여된 테스트 케이스 기반의 결함위치추정 기법의 성능평가: Tcas 에 대한 기법 적용

4.1 실험환경

Tcas 는 비행기의 충돌 회피를 위한 소프트웨어의 컴포넌트이며 총 173 개의 라인을 가지고 있다. Sir benchmark suite[7]는 fault 의 위치/유형에 따른 41 개 버전의 Tcas 와 1608 개의 테스트 케이스들을 제공한다. 본 실험에서는 41 개 버전들 중 주어진 테스트 케이스가 결함 구문을 실행 할 수 없는 버전들과 결함 구문을 실행하는 실패 테스트 케이스가 없는 버전들을 제외한 총 32 개의 버전들에 대해서 Tarantula 와 WTFL 기법을 비교한다. 이를 위해, GCOV[8]를 사용하여 각 테스트케이스의 커버리지를 기록하고, 이를 기반으로 각 구문의 의심도를 계산한다. 실험은 Intel Core2Duo@3.00GHZ CPU, 3GB 메모리를 장착한 하드웨어와 Ubuntu 10.04.4 32bit OS 환경에서 수행되었다.

[표 1] Tcas 32 개 버전들에 대한 Tarantula 와 WTFL 기법 적용결과

Ver.	Taran. (%)	WTFL (%)	Ver.	Taran. (%)	WTFL (%)
1	**4.69**	10.94	21	35.94	**34.38**
2	29.69	**28.13**	22	37.50	37.50
3	78.13	**73.44**	23	37.50	37.50
4	4.69	**3.13**	24	37.50	**35.94**
5	84.38	**68.75**	25	**3.13**	6.25
6	10.94	**7.81**	26	81.25	**71.88**
7	76.56	76.56	27	84.38	**68.75**
8	81.25	81.25	28	29.69	**21.88**
9	37.50	37.50	29	31.25	**29.69**
10	10.94	**6.25**	30	29.69	**28.13**
12	78.13	**71.88**	33	85.94	**76.56**
16	76.56	76.56	34	78.13	**71.88**
17	76.56	76.56	35	29.69	**21.88**
18	85.94	**68.75**	37	4.69	**3.13**
19	76.56	76.56	39	**3.13**	6.25
20	35.94	**34.38**	41	4.69	**3.13**

4.2 실험결과

[표 1]은 Tarantula 와 WTFL 기법을 Tcas 에 적용한 실험결과를 나타낸다. 'Ver.' 행은 Tcas 의 버전번호, 'Taran.(%)'와 'WTFL(%)'행은 Tarantula 와 WTFL 기법을 적용한 후 결함 구문를 찾기 위해 최대로 검사해야 하는 구문의 비율을 각각 나타낸다. 예를 들면, 그림 1 에서 WTFL 기법을 사용하면, 결함 구문(라인 2)을 찾기 위해서 최대 28%(2/7 = 0.28)의 구문을 검사한다.

[표 1]의 실험 결과에서 볼 수 있듯이, WTFL 기법은 대부분의 tcas 버전에서 Tarantula 보다 더 정확하다. 총 32 개의 버전 중 21 개의 버전에서 검사해야 하는 구문의 비율이 Tarantula 보다 작고, 8 개의 버전에서는 같으며, 단 3 개(버전 1,25,39)의 버전에서만 크다. 특히, 버전 10 에서는 WTFL 기법의 정확성이 Tarantula 보다 약 43% 증가했음을 알 수 있다((10.94 – 6.25)/10.94 ≈ 0.43).

5. 결론

본 논문에서는 CBFL 기법의 정확성을 향상시키기 위해서 Coincidentally Correct Test case 를 고려하여 각 테스트 케이스에 가중치를 부여하였으며, 이를 기반으로 한 새로운 CBFL 기법을 제시하였다. 기존 연구와의 비교를 위해 Sir benchmark suite 의 Tcas 32 개 버전에 새로운 기법을 적용한 결과 21 개 버전들에서 성능 향상이 있었음을 확인할 수 있었다.

참고 문헌

[1] I. Vessey, Expertise in Debugging Compute Programs,
Inter. J. of Man-Machine Studies, 23(5):459-494,1985.

[2]Wang X, Cheung S.C, Chan W.K, and Zhang Z, Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization, ICSE'09, Pages 45-55, 2009

[3]James A. Jones, Mary Jean Harrold, and John Stasko, Visualization of Test Information to Assist Fault Localization, ICSE'02, Pages 466-467, 2002.

[4]R. Abreu, P.Zoeteweij, and A. J. C. van Gemund, On the accuracy of spectrum-based fault localization, TAIC'07, Pages 89-98, 2007.

[5]R. Santelices, J.A.Jones, Y.Yu, and M.J.Harrold, Lightweight fault-localization using multiple coverage types, ICSE'09, Pages 56-66, 2009

[6]Wes Marsi, and Rawad Abou Assi, Cleansing Test Suites from Coincidental Correctness to Enhance Fault-Localization, ICST'10, Pages 165-174, 2010

[7] Sir benchmark suite. http://sir.unl.edu

[8] GCOV.
http://gcc.gnu.org/onlinedocs/gcc/Gcov.html