변형 프로그램의 동적 정보에 기반한 자동 결함 위치추정

문석현, 김문주
한국과학기술원 전산학과
대전광역시 유성구 구성동
seokhyeon.moon@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

요약: 본 논문은 프로그램 실패의 원인인 결함 구문의 위치를 정확하게 추정하는 새로운 자동 결함 위치추정 기법 MUSE 를 제안한다. MUSE 는 테스트 실패가 감지된 프로그램 P 가 주어졌을 때, P 의 결함 구문이 변형된 프로그램들과, 결함이 아닌 구문(올바른 구문)이 변형된 프로그램들을 실행한 테스트의 실행 결과가 서로 다른 특성을 이용하여 결함의 위치를 추정한다. 실제 C 프로그램에 대한 실험결과로부터 MUSE 는 최첨단 기법에 비해 매우 정확하게 결함 구문의 위치를 추정하는 것을 확인할 수 있었다.

핵심어: 디버깅(debugging), 결함 위치추정(fault localization), 변형 프로그램 (mutant).

1. 서론

일반적으로 프로그램 디버깅은 개발자에 의해 수동으로 수행되기 때문에 매우 많은 시간과 비용을 필요로 한다. 특히, 전체 디버깅 과정 중 프로그램 실패의 원인인 결함 코드의 위치를 찾는 과정인 ***결함 위치추정***은 가장 많은 시간을 소모하는 것으로 알려져 있다[1]. 이는 개발자가 직접 실패한 실행에 의해 실행된 구문을 추적하면서 결함으로 의심되는 코드를 찾아야 하기 때문이다. 따라서, 효과적인 결함 위치추정을 위한 자동 결함 위치추정 기법에 대한 연구가 활발히 진행되어 왔다.

다양한 종류의 결함 위치추정 기법 중 각광받는 연구 방향은, 테스트 실행과 프로그램 구문간 상관관계를 이용하는 Spectrum-Based Fault Localization (SBFL)이다[2-4]. SBFL 기법은 각 프로그램 구문에 결함일 가능성을 나타내는 의심도를 부여하는데, 구문이 실패 테스트에 의해 많이 실행되면서 성공 테스트에 의해 적게 실행될수록 높은 의심도를 가지게 된다. 개발자는 프로그램 구문의 의심도에 따라 매겨진 프로그램 구문의 순위를 이용하여 결함일 가능성이 높은 구문부터 프로그램을 검사함으로써 결함을 찾는 비용을 줄일 수 있다.

대표적 SBFL 기법인 Tarantula 는 다른 종류의 위치추정 기법에 비해 더 정확하게 결함의 위치를 추정하는 것으로 알려져 있다[2]. 하지만, SBFL 기법의 정확성이 다른 기법에 비해 뛰어남에도 불구하고, 개발자가 실제로 SBFL 을 결함 위치추정에 사용하기에는 그 정확성이 높지 않다는 비판을 받아 왔다[5].

본 논문에서는 새로운 결함 위치추정 기법인, 프로그램 변형 기반 결함 위치추정 기법 MUSE (**MU**tation-ba**SE**d fault localization)를 제안한다. MUSE 가 구문의 의심도를 계산하는 방법은 두 개의 관찰에 기반한다. 일반적으로, 잘못된 프로그램은 결함 구문을 수정함으로써 고쳐 진다. 따라서 결함 구문을 변형(수정)하는 것은 결함이 아닌 구문(올바른 구문)을 변형하는 것에 비해 많은 수의 실패 테스트를 성공 테스트로 만들 것이다. 반면, 올바른 구문을 변형하는 것은 결함 구문을 변형하는 것에 비해 많은 수의 성공 테스트를 실패 테스트로 만들 것이다. 왜냐하면 올바른 구문을 변형하는 것은 기존 결함 구문에 추가하여 새로운 결함 구문을 대상 프로그램에 만드는 것이기 때문이다. 이 두 가지 관찰은 MUSE 의심도 공식의 기반이 된다.

본 논문에서는 MUSE 의 결함 위치추정 정확성을 측정하기 위해 SIR benchmark[6]이 제공하는 2 개 실제 C 프로그램의 2 개 결함 버전에 대해 MUSE 를 적용했다. 실험 결과로부터 MUSE 는 결함 구문의 순위를 2 개 결함 버전에서 모두 1 등으로 매긴 반면, 최첨단 Spetrum-Based Fault Localization 기법인 Op2[3]는 평균 459 등으로 매겼다. 이로부터 MUSE 가 결함 위치추정을 위해 소모하는 비용을 혁신적으로 줄일 수 있다는 것을 확인 할 수 있었다.

2. 변형 프로그램 기반 결함 위치추정 기법

2.1 핵심 발상

테스트 집합(test suite) T 와, T 의 어떤 테스트에 대해 실패하는 결함이 있는 프로그램 P 가 주어졌을 때, P 의 결함 구문을 변형한 프로그램을 m_f, 올바른 구문을 변형한 프로그램을 m_c라고 하자. MUSE 는 아래의 두 개 핵심 가정에 기반하여 결함의 위치를 추정한다.

가정 1 : P 에서 실패한 테스트들은, m_f에서 성공할 가능성이 m_c에서 성공할 가능성에 비해 더 크다.
가정 2 : P 에서 성공한 테스트들은, m_c에서 실패할 가능성이 m_f에서 실패할 가능성에 비해 더 크다.

2.2 의심도 공식

MUSE 의 의심도 공식 mu는 위의 두 개 가정에 기반하여 정의된다. 먼저 공식의 정확한 정의를

`

표 1 실험 대상 프로그램, 프로그램 크기 (Lines Of Code), 실패 및 성공 테스트 수

Target	*Ver.*	*LOC*	$\lvert f_P \rvert$	$\lvert p_P \rvert$	**Description**
flex	v1	12,423	2	40	Lexical Analyzer Generator
grep	v3	12,653	5	175	Pattern Matcher

위하여 몇 가지 기호를 아래와 같이 정의하자.

P 의 어떤 구문 s 에 대하여, $f_P(s)$를 P 에서 s 를 실행한 실패 테스트 집합, $p_P(s)$ 를 P 에서 s 를 실행한 성공 테스트 집합이라 하자. 또한, 고정된 변형 연산자(mutation operator)가 주어졌을 때, mut(s) = $\{m_1, \ldots, m_k\}$를 s 를 변형한 P 의 변형 프로그램들 중 적어도 한 개의 테스트 실행 결과가 P 에서의 결과와 다른 변형 프로그램 집합이라고 하자. mut(s)의 각 변형 m_i에 대해서, f_{m_i} 와 p_{m_i}를 각각 m_i에서의 실패 및 성공 테스트 집합이라고 한다. 그리고, $f2p$ 를 P 의 모든 변형 프로그램들(mut(P))에 대해서 P 에서의 실패 테스트가 성공으로 바뀐 총 개수($f2p = \sum_{m \in mut(P)} |f_P \cap p_m|$) , $p2f$ 를 mut(P)에 대해서 P 에서의 성공 테스트가 실패로 바뀐 총 개수($p2f = \sum_{m \in mut(P)} |p_P \cap f_m|$)라고 하자. MUSE 의심도 공식 mu는 다음과 같이 정의된다.

$$\mu(s) = \frac{1}{|mut(s)| + 1} \sum_{m \in mut(s)} \left(\frac{|f_P(s) \cap p_m|}{f2p + 1} - \frac{|p_P(s) \cap f_m|}{p2f + 1} \right)$$

3. 실험 환경

본 논문에서는 위치추정 기술의 정확성을 측정하기 위해서, 개발자가 위치추정 기법에 의해 각 구문에 부여된 의심도에 따라 순위화된 구문을 높은 순위부터 차례대로 검사할 때, 몇 개의 구문을 검사해야 결함 구문을 찾을 수 있는지 측정한다. 본 논문에서는 2 개의 최첨단 SBFL 기법(Ochiai[4], Op2[3])과 MUSE 의 성능을 비교했다.

3.1 실험 대상 및 구현

MUSE, Ochiai, Op2 를 SIR benchmark 가 제공하는 2 개 실제 C 프로그램에 대해 적용했다. 표 1 은 실험에 사용된 프로그램과 benchmark 가 제공하는 각 프로그램의 테스트 집합 정보를 나타낸다. 본 실험에서는 위의 각 C 프로그램에 대해 benchmark 가 제공하는 결함 버전들 중, 무작위로 1 개 결함 버전을 선택하여 총 2 개의 결함 버전을 선택했다. 본 논문에서는 gcov 를 이용하여 각 테스트의 커버리지를 측정했다. 변형 프로그램 생성을 위해서 C 언어를 대상으로 하는 Proteum 변형 도구[7]를 이용하여, 변형 연산자 별로 구문당 1 개의 변형 프로그램을 무작위로 생성했다. MUSE, Op2, Ochiai 는 6400 라인의 C++코드로 구현했고,

표 2 Ochiai, Op2, MUSE 의 정확도

Target Program	*Rank of a faulty stmt*		
	Ochiai	Op2	MUSE
flex v1	1,248	887	**1**
gzip v2	31	31	**1**
Average	639.5	459.0	**1**

모든 실험은 Intel i5 3.6Ghz CPU 와 8GB 메모리를 장착한 Debian Linux 6.05 머신 10 대에서 수행했다.

4. 실험 결과

표 2 는 결함 구문을 찾기 위해 검사해야 하는 실제 구문 수를 나타낸다. MUSE 는 2 개의 결함 버전 모두에서 결함 구문의 순위를 1 등으로 메긴다 (완벽한 결함 위치추정 결과). 반면에 Op2 는 결함 구문의 순위를 평균적으로 459 등으로 매기는데, 이를 통해 MUSE 가 SBFL 기법에 비해 매우 정확하게 결함의 위치를 추정한다는 것을 확인할 수 있다.

5. 결론

본 논문에서 제시한 두 개의 가정에 기반하는 MUSE 는 결함일 가능성이 높은 구문의 의심도를 높일 뿐 아니라(가정 1), 올바른 구문의 의심도를 낮춤으로써(가정 2), 결함의 위치를 매우 정확하게 추정한다. 실험 결과로부터, MUSE 가 최첨단 SBFL 기법에 비해 매우 정확하게 결함의 위치를 추정하는 것을 확인 할 수 있었다. 향후 연구 방향으로, 서로 다른 변형 연산자가 MUSE 의 결함 위치추정 정확성에 어떤 효과를 미치는지 연구하는 것을 목표로 하고 있다.

감사의 글

본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학 IT 연구센터 지원사업 (NIPA-2013-H0301-13-5004), 한국연구재단의 중견연구자지원사업 핵심연구 (NRF-2012R1A2A2A01046172), 미래창조과학부가 지원한 2013 년 정보통신·방송(ICT) 연구개발사업의 연구결과로 수행되었음.

참고문헌

[1]I. Vessey, "Expertise in debugging computer programs: A process analysis," *IJMMS*, vol. 23, pp. 459-494, 1985.
[2]J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *ASE*, 2005, pp. 273-282.
[3]L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *TOSEM*, vol. 20, p. 11, 2011.
[4]R. Abreu, P. Zoeteweij, and A. J. C. Van Gemund, "On the accuracy of spectrum-based fault localization," in *MUTATION*, 2007, pp. 89-98.
[5]C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?," in *ISSTA*, 2011, pp. 199-209.
[6]H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *ESE*, vol. 10, pp. 405-435, 2005.
[7]M. E. Delamaro, J. C. Maldonado, and A. M. R. Vincenzi, "Proteum/IM 2.0: An integrated mutation testing environment," in *Mutation testing for the new century*, ed: Springer, 2001, pp. 91-101.

`