입력 커버리지를 활용한 효율적인 동적 기호 실행 탐색 기법

김윤호, 김문주

KAIST 전산학부
대전광역시 유성구 대학로 291
kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

요약: 본 논문에서는 동적 기호 실행 기법을 사용해서 효율적으로 버그를 찾기 위해 입력 커버리지를 활용한 동적 기호 실행 탐색 기법을 제안한다. 입력 커버리지란 주어진 프로그램 실행 경로를 실행할 수 있는 입력 값의 전체 가능한 입력 값 대비 비율을 나타낸다. 입력 커버리지가 낮은 프로그램 실행 경로는 개발자가 평소 생각하기 힘든 코너 케이스일 가능성이 높기 때문에 제안하는 탐색 기법은 입력 커버리지가 낮은 프로그램 실행 경로를 우선 탐색하여 코너 케이스 버그를 효과적으로 찾을 수 있다. 3 개의 SIR 벤치마크 프로그램을 대상으로 실험한 결과 제안하는 탐색 기법이 깊이 우선 탐색 기법 대비 동일한 수의 테스트 케이스를 생성했을 때 1.4~1.6 배 더 많은 소프트웨어 버그를 찾을 수 있었다.

핵심어: 소프트웨어 테스팅(software testing), 동적 기호 실행(dynamic symbolic execution), 입력 커버리지(input coverage)

1. 서론

반복문을 사용하는 복잡한 프로그램의 버그를 효율적으로 찾기 위해서 빠른 시간에 분기 커버리지 달성도를 높이기 위한 동적 기호 실행[1] 탐색 기법들이 제안되었다[2-3]. 아직 실행하지 않은 프로그램 구문을 실행할 것으로 예상되는 테스트 케이스를 생성하여 구조 커버리지를 효과적으로 높이고 아직 실행하지 못한 구문에 있는 버그를 찾을 가능성을 높이는 것이다. 하지만 단순히 실행하지 못한 프로그램 구문을 실행한다고 소프트웨어 버그가 발견되는 것이 아니고 특정한 조건을 만족하는 실행 경로에서만 소프트웨어 버그가 발견되기 때문에 구조 커버리지 달성도를 빠른 시간에 높이는 테스트 케이스가 더 효율적으로 버그를 찾을 수 있는 것은 아니다.

본 논문에서는 소프트웨어 버그를 효율적으로 찾기 위해 입력 커버리지를 활용한 동적 기호 실행 탐색 기법을 제안한다. 소프트웨어 버그 중 사용자가 평소 잘 실행하지 않는 프로그램 실행 경로에서 발생하는 코너 케이스 버그는 일반적인 소프트웨어 사용 환경에서 발생하지 않기 때문에 개발자가 테스트 과정에서 발견하기 어렵다. 특히 코너 케이스를 실행할 수 있는 프로그램 입력 값 수는 전체 가능한 입력 값 수에 비해 극히 적기 때문에 일반적인 소프트웨어 사용이나 랜덤 테스팅 등으로는 탐색하기 어렵다. 제안하는 기법은 프로그램 실행 경로를 실행할 수 있는 입력 값의 수를 측정하고 실행할 수 있는 입력 값의 수가 더 적은 프로그램 실행 경로를 우선해서 탐색한다. 따라서 일반적으로 많이 실행되는 실행 경로 대신 일반적인 사용이나 개발자 테스트에서 실행하기 어려운 코너 케이스를 먼저 탐색할 수 있다. SIR 벤치마크의 3 개 프로그램을 대상으로 적용한 결과 제안하는 탐색 기법이 깊이 우선 탐색 기법 대비 동일한 수의 테스트 케이스를 생성했을 때 1.4~1.6 배 더 많은 소프트웨어 찾았다.

2. 동적 기호 실행 기법

동적 기호 실행 기법은 동적 분석과 정적 분석 기법을 결합하여 테스트 케이스를 자동으로 생성하는 기법이다. 테스트를 생성할 대상 프로그램과 초기 테스트 케이스가 주어지면 주어진 테스트 케이스를 실행하고 실행된 프로그램 경로를 분석하여 분기문에서 어떤 조건이 수행되었는지 나타내는 경로 제약 조건식을 생성한다. 테스트 실행이 종료되면 생성된 경로 제약 조건식을 분석하여 이전에 실행되지 않은 새로운 프로그램 경로를 실행할 수 있는 테스트 케이스를 생성한다. 모든 프로그램 실행 경로를 테스트하거나 사용자가 지정한 종료 조건을 만족하면 동적 기호 실행 기법이 종료된다.

3. 입력 커버리지를 활용한 효율적인 동적 기호 실행 탐색 기법

본 장에서는 입력 커버리지와 제안하는 효율적인 동적 기호 실행 탐색 기법을 설명한다.

본 연구는 한국연구재단 한-아프리카 협력기반조성사업(NRF-2014K1A3A1A09063167), 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업(IITP-2015-H8501-15-1012), 초소형, 고신뢰(99.999%) OS 와 고성능 멀티코어 OS 를 동시 실행하는 듀얼 운영체제 원천 기술 개발(R0101-15-0081)의 지원으로 수행되었음

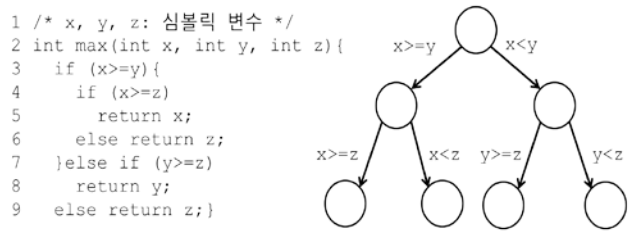

그림 1 분기문 3 개를 갖는 예제 프로그램 및 실행 경로 ([4]에서 발췌)

3.1 입력 커버리지

입력 커버리지는 프로그램의 실행 경로가 얼마나 많은 서로 다른 입력 값을 테스트하는지를 나타내는 커버리지 지표이다. 예를 들어 x, y, z 의 입력 값이 모두 1~10 사이인 그림 1 의 프로그램에서 x=y=z=1 인 테스트 케이스를 실행할 때 생성되는 경로 제약 조건식 (x>=y)∧(x>=z) 만족하는 입력 값은 모두 385 개로 전체 입력 값 1,000 개 중 38.5% 의 입력 커버리지를 갖는다고 볼 수 있다.

3.2 입력 커버리지를 활용한 동적 기호 실행 탐색 기법

소프트웨어 버그를 찾을 수 있는 테스트 케이스를 효율적으로 생성할 수 있도록 동적 기호 실행의 탐색 휴리스틱 알고리즘을 개발하였다. 동적 기호 실행에서 새로 생성할 테스트 케이스가 실행할 실행 경로의 예상 동적 커버리지를 계산하고 예상 동적 커버리지가 낮은 실행 경로를 먼저 실행하도록 테스트 케이스를 생성하였다. 따라서 제안하는 기법을 사용하면 일반적인 입력 값으로 실행하기 힘든 코너 케이스 실행 경로를 먼저 테스트할 수 있다.

4. 실험

제안하는 기법이 실제 버그를 효율적으로 탐색하는지 확인하기 위해 SIR 벤치마크[5]의 3 개 프로그램을 대상으로 실험을 수행하였다. 소프트웨어 버그 탐지 효율을 비교하기 위해 프로그램 변이 분석 (mutation analysis) 기법을 적용하여 소프트웨어 버그를 삽입하고 SIR 벤치마크의 기존 테스트 케이스의 90% 이상이 탐지 가능한 변형 프로그램을 제거하여 코너 케이스 버그를 삽입하였다. 표 1 은 실험에 사용한 프로그램과 프로그램의 크기, 테스트 수 및 변형 프로그램 수를 나타낸다.

제안하는 탐색 기법은 CREST-BV[4] 동적 기호 실행 도구를 사용하여 구현하였으며 CRES-BV 에서 제공하는 탐색 기법인 깊이 우선 탐색 기법과 제안하는 탐색 기법을 비교하였다.

표 2 는 테스트 케이스 생성 횟수를 최대 1,000, 2,000, 3,000 개로 제한했을 때 탐지한 어려운 변형 프로그램의 수를 나타낸다. 테스트 케이스 생성 횟수를 1,000 개로 제한했을 때의 replace 실험 결과를 제외하면 제안하는 탐색 기법이 깊이 우선 탐색 기법보다 더 많은 변형 프로그램을 탐지하였다. 테스트 케이스 생성 횟수를 1,000, 2,000, 3,000 으로 제한했을 때 평균적으로 제안하는 탐색 기법이 깊이 우선 탐색 기법보다 1.4, 1.6, 1.5 배 더 많은 변형 프로그램을 탐지할 수 있었다.

표 1. 실험 대상 프로그램의 LOC, 입력값 제한 및 테스트 케이스 수

이름	LOC	테스트 수	변형 프로그램 수
printtoken1	725	4130	5,624
printtoken2	569	4115	2,391
replace	563	5542	2,116

표 2. 제한된 테스트 케이스 생성 횟수 내에서 탐지된 어려운 변형 프로그램 수

이름	탐색기법	1000	2000	3000
printtoken1	제안기법	1,083	1,809	2,741
	깊이우선	608	1,107	1,605
printtoken2	제안기법	873	1,293	1,518
	깊이우선	640	933	1,117
replace	제안기법	386	1,106	1,607
	깊이우선	486	671	1,098
평균	제안기법	780.7	1,402.7	1,955.3
	깊이우선	578.0	903.7	1,273.3

5. 결 론

본 논문에서는 입력 커버리지를 활용한 소프트웨어 버그를 효율적으로 탐지하는 동적 기호 실행 탐색 기법을 제안하였다. 3 개의 SIR 벤치마크 프로그램을 대상으로 실험한 결과 제안하는 탐색 기법이 깊이 우선 탐색 기법 대비 동일한 수의 테스트 케이스를 생성했을 때 1.4~1.6 배 더 많은 소프트웨어 버그를 찾을 수 있었다.

참고문헌

[1] Cadar et al., “Symbolic Execution for Software Testing in Practice-Preliminary Assessment”, ICSE, 2011

[2] H. Seo and S. Kim, “How We Get There: A Context-Guided Search Strategy in Concolic Testing”, FSE, 2014

[3] J. Burnim and K. Sen, “Heuristics for Scalable Dynamic Test Generation”, ASE, 2008

[4] 김윤호 외 2 인, “CREST-BV: 임베디드 소프트웨어를 위한 Bitwise 연산을 지원하는 향상된 Concolic 테스팅 기법”, 정보과학회논문지: 소프트웨어 및 응용, Vol.40, No.2, 2013

[5] Do et al., “Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and Its Potential Impact”, ESE, Vol.10, No.4, 2005