CREST-BV: 임베디드 소프트웨어를 위한 Bitwise 연산을 지원하는 Concolic 테스팅 기법

김윤호 O1 김문주 1 장윤규 2

1 한국과학기술원, 2 삼성전자

kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr, yoonkyu.jang@samsung.com

CREST-BV: An Improved Concolic Testing Technique with Bitwise Operations Support for Embedded Software

Yunho Kim O1 Moonzoo Kim1, Yoonkyu Jang2

1Korea Advanced Institute of Science and Technology, 2Samsung Electronics

요 약

기존 소프트웨어 테스팅 기법은 개발자가 수동으로 테스트 케이스를 작성해야 하는 비효율성으로 인해 임베디드 소프트웨어의 안정성 확보에 어려움이 있다. Concolic 테스팅 기법은 자동으로 소프트웨어 테스트 케이스를 생성함으로써 기존 소프트웨어 테스팅 기법의 문제를 해결했지만, 임베디드 소프트웨어 테스트에 필수적인 bitwise 연산을 지원하지 않는 등의 문제가 있었다. 본 논문에서는 임베디드 소프트웨어를 위해 bitwise 연산을 지원하는 Concolic 테스팅 개선 방법에 대한 연구를 수행하고 오픈 소스 임베디드 소프트웨어 Busybox에 적용하여 기존 연구와 결과를 비교하였다. Busybox의 10개 유틸리티에 적용한 결과 기존 연구 결과 대비 평균 33%의 분기 커버리지 향상이 있었다.

1. 서 론

임베디드 소프트웨어에 오류가 발생할 경우 인명 피해, 대규모 재난, 경제적 피해가 발생하는 등의 사회/경제적인 문제가 발생할 수 있다. 하지만, 기존의 소프트웨어 테스팅 기법은 테스트 케이스를 수동으로 작성하기 때문에, 비효율적이고 개발자가 생각하기 어려운 예외적인 상황에 있는 오류를 찾는데 효과적이지 못해서 임베디드 소프트웨어의 안정성 확보가 어렵다.

Concolic(CONCrete + symbOLIC) 테스팅 기법(혹은 동적 기호 실행 기법(dynamic symbolic execution))[1]은 동적 실행 기법과 기호 실행 기법을 결합하여 가능한 모든 실행 경로를 테스트하는 소프트웨어 테스트 케이스 집합을 자동으로 생성하는 기법이다. 테스트 케이스를 자동으로 생성하기 때문에 테스트에 필요한 개발자의 노력을 최소화 할 수 있고, 특히 모든 실행 가능한 경로를 테스트 할 수 있게 테스트 케이스를 생성하기 때문에 예외적인 상황에서 발생할 수 있는 숨겨진 버그를 발견하는데 효과적이다[2,3]. 하지만 사례 연구[2] 결과 현재 Concolic 테스팅 기법이 bitwise 연산(&, |, ^, << 등)을 지원하지 못하는 한계가 있어 임베디드 소프트웨어에 적용하기 위한 개선이 필요하다.

본 논문에서는 임베디드 소프트웨어 테스트에 필요한 bitwise 연산을 지원하는 Concolic 테스팅 개선 방법을 연구하고 Concolic 테스팅 도구 CREST-BV를 개발하였다. CREST-BV의 bitwise 연산 지원의 효과를 보이기 위해 오픈 소스 임베디드 소프트웨어 Busybox[4]의 10개 유틸리티에 대해 CREST-BV를 적용하였으며, 기존 Concolic 테스팅 도구 대비 평균 33%의 분기 커버리지 향상이 있었다.

2. 관련 연구

처음 Concolic 테스팅 기법이 제안된 후 다양한 프로그램 언어를 대상으로 Concolic 테스팅 관련 연구가 진행되었다. CREST[5], CUTE[6], DART[7]는 C 프로그램 언어를 대상으로 개발된 Concolic 테스팅 도구이다. 이 도구들은 경로 제약 조건식을 추출하기 위해 instrumentation 기반 기법을 사용하였다. 그 외에도 LLVM 가상 머신 언어를 사용한 KLEE[8], C# 언어를 지원하기 위해 .NET 가상 머신을 사용한 PeX[9] 등의 Concolic 테스팅 도구가 개발되었다.

Concolic 테스팅 기법은 임베디드 소프트웨어를 테스트 하기 위한 방법으로도 사용되고 있다. [10]에서는 Concolic 테스팅 기법을 사용해서 Flash 메모리 드라이버를 테스트 하였으며 모델 체킹 기법과의 장, 단점 비교 분석을 하였다. [2]에서는 모바일 폰 플랫폼 소프트웨어에 Concolic 테스팅 기법을 적용하여 Concolic 테스팅 기법을 실제 소프트웨어 프로젝트에 적용할 때 발생하는 어려움과 현재 개발된 테스팅 도구의 한계에 대해 분석하였다. [3]에서는 임베디드 플랫폼에서 사용되는 소프트웨어 라이브러리 libexif를 테스트하고 임베디드 소프트웨어에 적합한 Concolic 테스팅 기법을 분석하였다.

```
1 /* x, y, z: 심볼릭 변수 */
2 int max(int x, int y, int z){
3   if (x>=y){
4     if (x>=z)
5       return x;
6     else return z;
7   }else if (y>=z)
8     return y;
9   else return z;}
```

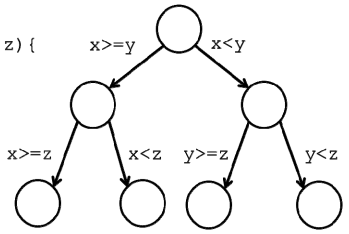

그림 1. 3개의 분기를 갖는 예제 프로그램 및 실행 경로

3. CREST-BV: Bitwise 연산을 지원하는 Concolic 테스팅 도구 확장

3.1 Concolic 테스팅 기법

Concolic 테스팅 기법은 주어진 테스트 케이스를 사용해서 테스트 대상 프로그램을 수행하고, 실행된 프로그램 경로를 따라 경로 제약 조건을 생성한다. 테스트 대상 프로그램이 종료되면 경로를 따라 생성한 경로 제약 조건식을 활용해서, 기존에 실행되지 않은 경로를 실행하기 위한 테스트 케이스를 생성한다. 이 과정은 모든 실행 가능한 경로가 실행되거나, 사용자가 지정한 종료 조건을 만족할 때까지 반복된다.

그림 1의 경우를 예로 살펴보자. 그림 1의 왼쪽 예제 프로그램은 세 정수 값을 입력으로 받아 가장 큰 값을 돌려주는 함수이고, 오른쪽 그래프는 해당 예제 프로그램의 전체 실행 경로를 나타내는 그래프이다. 초기 입력값이 x=1, y=1, z=1로 주어졌을 때 3번째 줄의 if 조건문과 4번째 줄의 if 조건문을 만족하게 되고 테스트 대상 프로그램이 종료될 때, 경로 조건 $(x>=y)\wedge(x\geq z)$가 생성된다. 기존에 실행되지 않은 경로를 실행하기 위해 마지막 분기 조건 x>=z을 부정해서 새로운 경로 조건 $(x>=y)\wedge(x<z)$을 생성하고 constraint solver를 사용해서 이 새로운 경로 조건을 풀어내서 새로운 경로 조건을 만족하는 테스트 케이스 x=1, y=1, z=2를 생성한다. 이렇게 얻은 새로운 테스트 케이스를 사용해서 테스트 대상 프로그램을 다시 실행하고 이 과정을 모든 실행 경로가 실행되거나 사용자 지정 종료 조건을 달성할 때까지 반복하게 된다.

3.2 CREST-BV 확장

CREST-BV는 C 프로그램을 대상으로 하며, 기존 오픈 소스 instrumentation 기반 Concolic 테스팅 도구 CREST를 확장해서 구현되었다. 기존의 CREST는 bitwise 연산을 지원하지 않아 bitwise 연산을 많이 사용하는 임베디드 소프트웨어에 적용했을 때 분기 커버리지 달성도 및 버그 발견 능력이 떨어졌다. 이와 같은 문제를 해결하기 위해 CREST-BV는 bitwise 연산을 지원하도록 CREST를 확장하였다.

CREST-BV는 크게 instrumentation을 수행하는 front-end, 기호 실행을 수행하는 middle-end, 경로 제약 조건식을 풀고 테스트 케이스를 생성하여 테스트 대상 프로그램을 실행하는 back-end로 구성되어 있다(그림 2). 먼저 front-end를 확장하여 bitwise 연산자에 대한 probe 함수를 개발하고 CIL(C Intermediate Language)의 instrumentation 엔진을 확장해서 bitwise 연산자에 대한 probe를 삽입할 수 있도록 확장하였다. 특히, 향후 middle-end 및 back-end 에서 활용하기 위해 프로그램 변수 타입에 따른 크기 정보를 probe 함수를 통해 추출할 수 있도록 확장하였다. 그 다음, middle-end를 확장하여 기호 실행 라이브러리 함수들이 bit 수준 연산을 지원할 수 있도록 확장하였다. 기존 선형 정수 표현식의 경로 제약 조건식은 각 변수의 계수들과 비교 연산자만 사용해서 표현할 수 있었으나, bitwise 연산자 및 비선형 정수 표현식의 경로 제약 조건식은 기존 방법으로 표현할 수 없었다. 따라서 일반적인 수식 나무 형태로 경로 제약 조건식을 확장하였다. 마지막으로 back-end를 확장하여 bitwise 연산을 지원하는 경로 제약 조건식을 풀 수 있도록 개선하였다. Bitwise 연산을 지원하는 경로 제약 조건식을 정확하게 풀기 위해서는 bitwise 연산자의 피연산자가 서로 같은 크기를 가져야 하기 때문에 C 프로그램 언어의 연산자 승급 규칙을 따라 피연산자의 크기를 동일하게 만들어서 경로 제약 조건을 풀었다. CREST는 선형 정수 표현식의 경로 제약 조건식을 풀기 위해서 Yices[11] SMT solver를 사용하였다. 하지만, Yices의 경우 라이브러리로 사용할 때 bit 수준 정확도를 갖는 /, % 연산을 지원하지 않는 한계가 있어, CREST-BV는 Z3[12] SMT solver를 사용하였다.

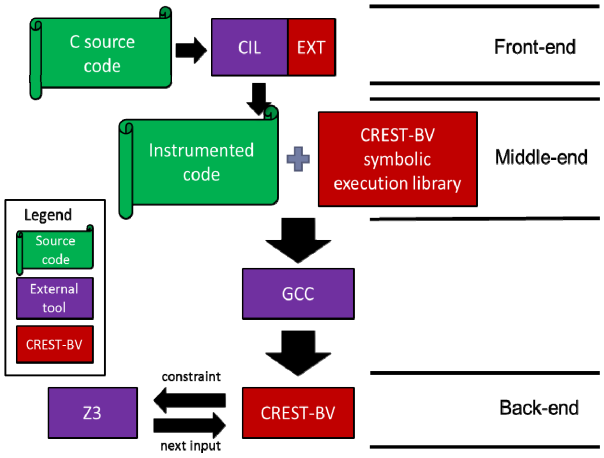

그림 2. CREST-BV 구조

4. CREST-BV 적용 사례 연구

4.1 실험 환경

Busybox는 임베디드 환경에서 리눅스/유닉스의

표 1. busybox 10개 유틸리티에
CREST-BV 와 CREST를 적용한 결과

Target	# of Brs.	Cov.(%)		Time(s)	
		BV	LIA	BV	LIA
cp	22	**50.0**	36.4	1.4	0.1
cut	102	**45.1**	2.9	30.0	0.2
expr	152	67.1	**67.8**	252.7	34.8
grep	168	**76.8**	41.7	1245.3	0.1
ls	240	**69.2**	35.8	232.3	0.1
mv	46	17.4	17.4	0.2	0.2
od	72	**94.4**	79.2	178.6	21.9
printf	148	**72.3**	67.6	302.9	44.3
tr	130	**46.2**	35.4	495.6	27.7
vi	1512	46.4	**55.4**	8961.2	2464.4
Avg.	259	**58.5**	44.0	1170.0	259.4

커맨드 라인 유틸리티들을 제공하는 소프트웨어다. 본 논문에서는 Busybox 1.17.0 버전의 유틸리티 가운데 10개 유틸리티 cp, cut, expr, grep, ls, mv, od, printf, tr, vi 를 대상으로 CREST-BV와 기존 CREST를 적용하여 분기 커버리지와 실행 시간을 비교하였다. 모든 실험은 Intel Core2Duo E8600@3.3GHz CPU, 8GB 메모리를 장착한 하드웨어와 Debian 6.0.1 32bit OS 환경에서 수행되었다. 분기 커버리지 효과 및 실행 성능 비교를 위해서 CREST-BV(Z3 2.19 사용)와 선형 정수 표현식만 지원하는 CREST(Yices 1.0.29 사용)를 사용해서 결과를 비교하였다. 경로 탐색 방법으로는 깊이 우선 탐색방법을 사용하였으며, 최고 10,000개의 테스트 케이스를 생성할 수 있도록 제한하였다.

4.2 실험 결과

표 1은 CREST-BV와 CREST를 적용한 실험 결과를 나타낸다. 첫 열과 두 번째 열은 각각 테스트 대상 프로그램의 이름과 총 분기문의 수를 나타내며 세 번째, 네 번째 열은 각각 CREST-BV(열 이름 BV)와 CREST(열 이름 LIA)를 사용했을 때의 분기 커버리지를 나타낸다. 다섯 번째, 여섯 번째 열은 총 수행에 걸린 시간을 초 단위로 나타낸 것이다.

CREST의 평균 분기 커버리지가 44.0%인데 반해 CREST-BV는 분기 커버리지 58.5%를 달성하였다. 이는 기존 결과 대비 약 33%((58.5-44.0)/44.0*100) 더 향상된 결과이다. 총 10개 유틸리티 가운데 7개 유틸리티에서(expr, mv, vi 제외) CREST-BV가 더 높은 분기 커버리지를 달성할 수 있었다. 특히 cut의 경우 기존 CREST로는 2.9%의 분기문 밖에 커버할 수 없었으나, CREST-BV를 사용해서 42.2%의 분기문을 더 커버할 수 있었다. vi, expr의 경우 기존 bitwise 연산 지원으로 인해 Concolic 테스팅으로 실행 가능한 경로가 더 늘어나 10,000개의 테스트 케이스로 충분히 가능한 실행 경로를 테스트 할 수 없었기 때문에 분기 커버리지가 더 떨어지는 결과가 발생하였다.

실행 시간의 경우 9개 유틸리티(mv만 제외)에서 CREST-BV가 더 오래 걸렸다. 이는 선형 정수 표현식으로 표현된 경로 조건이 더 간단하기 때문에 효과적인 풀이 알고리즘이 있는 반면, bit 수준 경로 조건식은 그 복잡도로 인해 효율적인 알고리즘이 없기 때문이다.

5. 결 론

본 논문에서는 임베디드 소프트웨어 테스트를 위한 bitwise 연산을 지원하는 Concolic 테스팅 기법을 연구하고 이를 구현한 CREST-BV를 개발하였다. CREST-BV를 오픈 소스 임베디드 소프트웨어 Busybox에 적용한 결과 기존의 CREST보다 약 33.1% 더 향상된 분기 커버리지를 달성할 수 있었다.

참고 문헌

[1] C. Pasareanu and W. Visser, A Survey of New Trends in Symbolic Execution for Software Testing and Analysis, STTT 11(4), 2009.

[2] M. Kim, Y. Kim and Y. Jang, Industrial Application of Concolic Testing on Embedded Software: Case Studies, ICST, 2012

[3] Y.Kim, M.Kim, Y.Kim, and Y.Jang, Industrial Application of Concolic Testing Approach: A Case Study on libexif by Using CREST-BV and KLEE, ICSE, 2012

[4] Busybox. http://www.busybox.net

[5] CREST. http://code.google.com/p/crest

[6] K. Sen, D. Marinov, G. Agha, CUTE: A Concolic Unit Testing Engine for C, ESEC/FSE, 2005

[7] P. Godefroid, N. Klarlund, K. Sen, DART: Directed Automated Random Testing, PLDI, 2005

[8] C. Cadar, D. Dunbar, and D. Engler, KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs, OSDI, 2008

[9] N. Tillmann and W. Schulte, Parameterized Unit Tests, ESEC/FSE, 2005

[10] M. Kim, Y. Kim and Y. Choi, Concolic Testing of the Multi-sector Read Operation for Flash Storage Platform Software, FACJ 24(2), 2012

[11] B. Dutertre and L. Moura, A Fast Linear-arithmetic Solver for DPLL(T), CAV, 2006

[12] L. Moura and N. Bjorner, Z3: An Efficient SMT Solver, TACAS, 2008.