Bitfield 심볼릭 지원을 통한 Concolic 테스팅 효과 향상

이아청 김현우 김윤호 김문주
한국과학기술원
dkcjd2000@gmail.com hyunoo@kaist.ac.kr yunho.kim03@gmail.com moonzoo@cs.kaist.ac.kr

Improvement of Concolic Testing Effectiveness by Supporting Bitfield Symbolic Variable

Ahcheong Lee Hyunwoo Kim Yunho Kim Moonzoo Kim
Korea Advanced Institute of Science and Technology

요 약

Concolic testing은 효과적인 테스트 입력을 자동으로 생성하기 때문에, 신뢰성이 중요한 임베디드 SW 분야에서 중요한 기술이다. 하지만, 기존 concolic testing 도구는 임베디드 시스템에 자주 사용되는 bitfield를 심볼릭하게 지원하지 않는 제약점이 있었다. 본 논문에서는 concolic testing 도구인 CREST에 bitfield 변수를 심볼릭하게 지원하는 기능을 추가하고, bitfield 기능을 지원하지 않는 CREST와 지원하는 CROWN을 자동차 SW에 적용하여 분기 커버리지를 비교하였다. 그 결과 CROWN이 CREST 대비 75% 정도의 분기 커버리지 향상을 달성했다.

1. 서 론

Concolic 테스팅 기법은 심볼릭 수행(Symbolic execution)을 이용한 테스트 케이스 생성 기법으로 모든 가능한 동작을 검사하는 테스트 케이스 생성을 목표로 한다. 소프트웨어의 신뢰성이 중요한 임베디드 시스템 분야에 특히 유용한 기술이며, C언어를 대상으로 하는 Concolic 테스팅 도구로는 CREST[1]와 KLEE[2]가 있다.

하지만 CREST와 KLEE 모두 임베디드 SW 등에서 많이 사용하는 bitfield 변수를 심볼릭하게 선언하는 기능을 지원하지 않는 문제가 있다. Bitfield는 프로그램이 사용하는 메모리의 크기를 줄이기 위해, 구조체(struct)와 공용체(union)의 field들을 Byte 단위가 아닌 Bit 단위의 메모리만을 사용하도록 정의하는 기능이다. 그림 1에서 `int x1:3` 은 `x1`이 3 bit 만 메모리 사용하도록 선언한다. CREST는 프로그램의 변수를 Byte Address를 이용하여 처리하기 때문에 Byte 크기 단위의 변수만을 다룰 수가 있다. 이러한 제약점을 극복하기 위한 naïve한 해법으로, 심볼릭하게 선언할 field들의 bitfield 정보를 지우고, 각 field에 상응하는 크기 제한을 두는 방법으로 bitfield와 비슷한 효과를 만들 수 있으나, 이경우 공용체를 사용하는 경우 전체 struct 구조의 메모리 할당이 변하여 프로그램의 semantic이 변하는 단점이 있다.

본 논문에서는 CREST를 개량한 CROWN을 이용한 concolic 테스팅에서, 구조체와 공용체 안의 bitfield 변수들을 심볼릭하게 선언하여 사용하는 기술에 대해 논하고, case study를 통해 bitfield를 심볼릭하게 다루기 전과 후의 테스트 분기 커버리지 변화를 다룬다.

2. 배경

이 장에서는 concolic 테스팅 기법과 bitfield가 포함된 프로그램 예시에 대해 설명한다.

```
1  #include<stdio.h>
2  struct {
3    int x1 : 3;
4  } a1;
5  int main(){
6    //CREST_int(a1.x1)
7    if (a1.x1 == 2)
8      printf("line 8 reached\n");
9    return 0;
10 }
```

그림 1. bitfield가 포함된 프로그램 예시

Concolic 테스팅은 대상 프로그램의 테스트 케이스 수행과 동시에 심볼릭 정보를 얻어낸뒤 이 심볼릭 정보를 분석하여 도달하지 못한 분기를 실행하는 테스트 케이스를 생성한다.

그림 1의 프로그램에 대해 CREST를 이용하여 concolic 테스팅을 실행하면, line 6에서 변수 `a1.x1`의 심볼릭 정보를 기록하도록 지정한다. 초기 실행에서 `a1.x1`에 초기값 0을 넣고 프로그램을 실행하며, line 7에서 else 분기를 실행하고 프로그램이 종료된다. 이때 생긴 심볼릭 path formula인 `(a1.x1 != 2)`를 negate한 `!(a1.x1 != 2)`에서 `a1.x1`을 구하여 새로운 분기에 도달하는 테스트 케이스를 만든다.

하지만 이 코드는 bitfield 변수인 `a1.x1`가 포함되어 있다. CREST에서는 bitfield 변수가 포함된 path formula를 다룰 수가 없어 새로운 테스트 케이스

생성을 하지 못하고, 초기 입력값을 이용한 실행 후 종료되어 버린다. CROWN에는 그림 1과 같이 bitfield가 포함된 프로그램을 변환하여 concolic 테스팅이 가능하도록 기능을 추가하였다.

3. CROWN에서 심볼릭 bitfield 변수 사용 기술

(1) 구조체와 공용체의 정의 변환

각 구조체와 공용체의 field들의 메모리 할당을 파악하여, bitfield 변수들이 할당되는 메모리 영역을 Byte 단위의 변수들로 치환한다.

(2) 코드 내의 bitfield 변수 치환

코드 내에서 사용된 bitfield 변수를 (1)에서 정의한 Byte 단위 변수에 대한 expression으로 치환한다. 치환 시 bitwise operator를 이용하여 bitfield 변수에 할당된 메모리 영역만을 읽고 쓰도록 한다.

3.1. 구조체와 공용체의 정의 변환

각 구조체와 공용체에서 bitfield 변수를 Byte 단위의 변수로 치환한다. bitfield 변수를 안전하게 치환하기 위해서는 메모리 할당을 먼저 파악하여야 한다. 각 field들은 textual order에 따라 메모리 할당을 하며, 각 field들의 메모리 할당 위치와 크기는 각 field의 타입과 크기에 따라 정해진다. 이전의 field들의 메모리 할당과 현재 field의 타입과 크기를 고려하여 다른 field들 사이에 적절한 padding을 넣거나 넣지 않게 할당을 결정한다.

그림 2와 그림 3에서 구조체의 정의를 변환하는 예시를 나타내었다. 그림 3에서 굵은 테두리는 1Byte, 얇은 테두리는 1 bit를 의미한다.

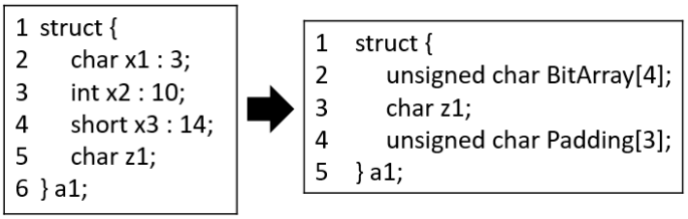

그림 2. bitfield를 포함하는 구조체를 변환하는 예시

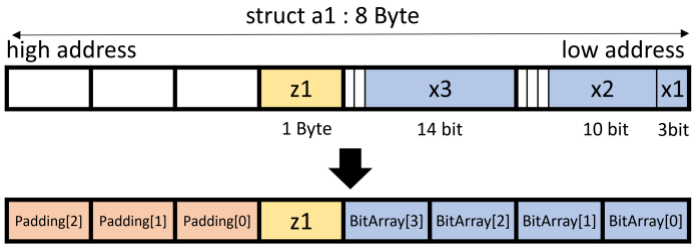

그림 3. 그림 2의 메모리 레이어

크기가 8Byte인 구조체 a1을 대상으로 변환을 한 예시이다. bitfield가 아닌 변수 z1은 메모리 상의 위치가 변하지 않도록 그대로 정의되고, 나머지 bitfield 변수 x1, x2, x3가 unsigned char array로 치환된 것을 볼 수 있다. 이와 같이 bitfield 변수들이 연속되어 메모리에 할당되어 있다면 이들을 함께 하나의 array로 치환한다.

구조체를 변환할 때에는 구조체 자체의 크기도 보존되어야한다. 구조체와 공용체의 크기는 가장 큰 field의 타입에 따라 정해진다. 그림 2의 구조체의 field들 중 가장 큰 타입이 int형이므로 4Byte의 배수로 구조체의 크기가 정해진다. 이 구조체는 실제로 5Byte만 사용하지만 8Byte가 할당된다. 변환에서 구조체의 크기를 보존하지 않으면 semantic이 변할 가능성이 있다. 따라서 크기를 보존하기위해 추가로 하나의 array를 더 선언한다. 그림 2의 예시에서는 마지막에 Padding이라는 이름의 array를 추가로 넣은 것을 볼 수 있다. 추가로 변수를 선언하지 않으면 구조체 a1의 크기가 5Byte로 줄어들게 된다.

gcc extension으로 packed, aligned와 같은 attribute를 구조체에 부여할 수 있다. 두 attribute 모두 field의 메모리 할당을 변화시키는 역할을 한다. 이때도 attribute가 주어지지 않았을 때와 마찬가지로 메모리 할당을 추적하여 처리할 수 있다.

3.2. 코드 내의 bitfield 변수 치환

구조체와 공용체의 정의를 바꾸었기 때문에, 코드 내에서도 bitfield변수를 치환해야한다. 코드 내의 모든 bitfield 변수를 그에 상응하는 array 변수의 expression으로 치환하면 된다. 그림 4와 그림 5에서 bitfield 변수들이 치환되는 예시를 볼 수 있다. 크게 bitfield 변수의 값을 읽어 사용할 때(그림 4), 그리고 bitfield 변수에 값을 assign 할 때(그림 5) 두가지 경우로 나눌 수 있다. 구조체 a1의 정의는 그림 2와 같다.

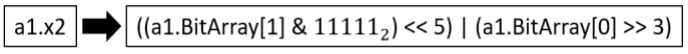

그림 4. bitfield 변수의 값을 읽을 때의 치환 예시

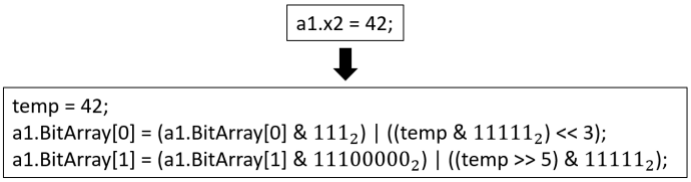

그림 5. bitfield변수가 포함된 assignment의 치환 예시

x2 변수의 값이 새로 정의된 BitArray[0]과 BitArray[1]에 나뉘어 저장되기 때문에 Bitwise operator를 이용하여 expression을 만들어 치환한다. x2 변수가 할당되는 메모리 영역만을 읽고 쓰는 것을 볼 수 있다.

CREST에서는 매 execution에서 심볼릭 변수들을 포함하는 path formula를 구성하고, constraint solver인 Z3[4]를 이용하여 다음 execution을 위한 program input을 생성한다. 이 input은 다음 execution 에서 심볼릭 변수의 값으로 들어가게 된다. CROWN에서는 bitfield 변수 역시 심볼릭 변수로 지정 될 수 있도록 기능을 추가하였다. bitfield 변수를 심볼릭 변수로 다룰 때, 실제 symbolic execution을 수행할 때에는 bitfield 변수를 모두 제거한 상태이기 때문에 실제 심볼릭

변수로 지정되는 것은 치환된 Byte단위의 변수가 되도록 하여야 한다. 그러나 단순히 치환된 변수를 심볼릭하게 지정할 경우, 원래 원하던 bitfield 변수 외에 다른 변수의 값도 바꾸는 input을 만들 가능성이 있기 때문에, 그에 관련된 formula도 추가하였다.

예를 들어, 그림 2의 예시에서 x2가 심볼릭 변수로 지정될 경우, 실제 execution에서는 x2가 BitArray[0]와 BitArray[1]으로 대체되었기 때문에 이 두 변수를 심볼릭 변수로 다루게 된다. 하지만 변수 x1은 심볼릭 변수로 선언되지 않았으므로 BitArray[0]의 하위 3bit는 변하지 않는 테스트 케이스를 만들도록 constraint를 추가해주었다.

4. Case Study 설계

실제 임베디드 SW에서의 분기 커버리지를 비교하기 위해 A모사에서 사용되는 한 모듈에 대해 case study를 진행하였다. CREST와 bitfield를 지원하는 CROWN를 적용한 후 분기 커버리지를 비교하였다. CREST는 bitfield 변수가 포함된 path formula는 다룰 수 없기 때문에, bitfield를 지원하게 되면 search space가 넓어지는 효과를 만든다.

표 1. 테스트 대상 모듈의 정보

# of c File	# of functions	# of branches	total LOC
129	1124	24723	85125

테스팅 대상 모듈의 정보는 표 1과 같다. 모듈에 포함된 각 함수 대해 driver를 만들어 유닛 테스트[5]를 시행하였다. 각 함수의 모든 input 변수들을 심볼릭 변수로 지정하였으며, CREST의 경우 bitfield 변수들은 심볼릭 변수로 지정할 수 없어 하지 않았다. 파일 외부의 함수들은 모두 stub 함수로 만들어 새로운 심볼릭 변수를 return하도록 하였다. 각 함수 당 최대 테스트 케이스 개수는 1000개, 각 execution의 timeout은 0.5초로 설정하였다. 경로 탐색 방법으로는 DFS(depth-first search) 보다 분기 커버리지를 빨리 달성할 수 있는 reverse DFS(depth-first search)를 사용하였다. CREST와 CROWN를 이용해 만든 테스트 케이스를 다시 입력으로 사용하여 실행한 후 gcov를 이용하여 분기 커버리지를 측정하였다.

추가로, 1장에서 설명한 naïve한 approach에 대해서도 분기 커버리지를 측정하였다. 단순히 구조체 definition에서 bitfield attribute를 지워주는 방식으로, 프로그램의 semantic이 변할 가능성이 있지만 그 방법이 단순해 사용했던 방식이다. 마찬가지로 bitfield였던 변수를 포함하여 모든 input 변수를 심볼릭 변수로 지정하였다.

Intel Xeon X5650 2.67Hz 프로세서와 4GB RAM이 장착된 Ubuntu 서버에서 진행하였으며, 추가로 테스트 케이스 생성에 걸린 시간도 측정하였다.

5. 결과

표 2. 테스트 결과

	CREST	CREST +bitfield attribute 제거	CROWN
Branch cov.	25.67%	31.00%	**44.91%**
Time (s)	1693	2013	3542
# of Test cases	13108	39853	50840
Z3 time(s)	136	435	1432

표 2의 첫번째 열은 CREST 제약사항때문에 bitfield 변수를 심볼릭하게 사용하지 못한 결과, 두번째 열은 CREST로 bitfield attribute를 지우고 심볼릭 설정한 결과, 마지막 열은 CROWN으로 bitfield 변수를 정확하게 심볼릭 선언하고 테스트한 결과이다. 두번째 행은 각 기법이 테스트 케이스 생성에 총 걸린 시간, 세번째 행은 생성한 테스트 케이스 개수, 네번째 행은 Z3가 path formula를 푸는데 걸린 시간이다.

CREST가 bitfield 변수가 포함된 path formula를 다루지 못하기 때문에 상대적으로 적은 branch만을 타겟할 수 밖에 없어 분기 커버리지가 25.67%로 CROWN 보다 낮게 나왔다. CROWN을 CREST와 비교했을 때 분기 커버리지가 상대적으로 75% 증가하였으므로 분기 커버리지 상승 효과가 나타났다고 볼 수 있다.

CROWN은 bitfield 변수를 다루기 위해 긴 expression들을 타겟 프로그램에 넣으므로 Z3가 풀어야 할 path formula가 길어지고 푸는데 걸리는 시간도 길어진다고 예상할 수 있고, 결과도 이를 보여주었다.

6. 결론

타겟 프로그램의 semantic을 바꾸지 않으면서 bitfield 변수를 치환하는 기능을 CROWN에 추가하였다. 그에 따라 bitfield 변수를 포함하는 프로그램도 concolic testing이 가능하게 되었다. bitfield 변수가 사용된 실제 프로그램에서 도입한 결과 분기 커버리지가 상대적으로 75%까지 증가하는 것을 볼 수 있었다.

참고문헌

[1] J. Burnim and K. Sen. Heuristics for scalable dynamic test generation. ASE. 2008.

[2] C. Cadar, D. Dumbar, and D. Engler. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008). 2008.

[3] Y. Kim, M. Kim, and Y. Jang. CREST-BV: An Improved Concolic Testing Technique Supporting Bitwise Operations for Embedded Software. Journal of KIISE: Software and Applications. 40(2):90-98. 2013.

[4] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. TACAS. 2008.

[5] Y. Kim, Y. Kim, T. Kim, G. Lee, Y. Jang and M. Kim. Automated Unit Testing of Large Industrial Embedded Software using Concolic Testing. ASE. 2013