

국방 무기 체계 SW 품질 향상을 위한 Concolic 테스팅 기술

박건우*^o 이주현+ 송형곤+ 조규태+ 김윤호* 김문주*

*한국과학기술원 전산학부 +LIG 넥스원

kunwoo1209@kaist.ac.kr, {joohyunlee, hyunggon.song, kyutae.cho}@lignex1.com, {yunho.kim03, moonzoo.kim}@gmail.com

Concolic Testing to Improve SW Quality of Defense Weapon System

Kunwoo Park*^o, Joohyun Lee+, Hyunggon Song+, Kyutae Cho+, Yunho Kim*, Moonzoo Kim*

*School of Computing, KAIST +LIG Nex1

요 약

국방 무기 체계 SW 품질 향상을 위해 노동집약적 수작업 SW 테스트 관행이 아닌, 테스트 입력을 자동으로 그리고 체계적으로 생성하는 것이 필요하다. 본 연구는 concolic 테스팅을 국방 무기 체계 SW에 적용해 높은 커버리지의 테스트 입력값을 효과적으로 생성하고, 결함을 발견하여 SW의 품질 향상에 기여했다. 또한 concolic 테스팅을 확장 적용할 수 있도록 심볼릭 모델링 방법을 예시로 제안했다.

1. 서 론

국방 무기/비무기체계 (전력지원체계)의 첨단화가 이루어지는 현 상황에서 무기체계 SW의 품질관리 능력은 필수가 되고 있다[1]. 그러나 SW 테스팅 전문 인력의 부족 및 노동집약적 수작업 SW 테스트 관행은 국방 무기체계 SW 품질 향상에 걸림돌이 되고 있다.

테스팅의 목적은 SW 내부에 존재하는 결함을 발견하는 것이고, 이를 위해서는 SW의 실행 가능한 경로를 가능한 한 많이 실행하는 TC가 필요하다. 수작업으로 만든 TC는 SW의 복잡성이 클 경우 SW의 가능한 모든 동작을 실행한다는 보장이 없기 때문에 SW에 존재하는 결함 발견이 어렵다.

Concolic 테스팅[2][3]은 상기 문제를 해결하고 테스트 커버리지를 효과적으로 증가시키는 자동화 테스팅 기법이다. Concolic 테스팅이 자동으로 TC를 생성하기 위해서 대상 프로그램의 입력값을 포함한 environment에 대한 심볼릭 모델 작성에 필요하다. 본 연구가 기여한 바는 다음과 같다.

1. Concolic 테스팅을 통해 수많은 효과적인 테스트 입력값을 생성하여, 고신뢰성이 있어야 하는 국방 무기 체계의 품질을 향상하였다 (평균 분기 커버리지 81.5% 달성 및 2개의 크래시 버그 검출).
2. 국방 무기체계 미들웨어 도메인에 concolic 테스팅을 확장 적용하는 실무자들이 참고할 수 있도록, 국방무기체계 미들웨어 도메인에 concolic 테스팅을 적용할 때 필요한 심볼릭 모델링 방법을 예시로 제안하였다.

3. 국방 무기체계 미들웨어 도메인에 concolic 테스팅의 성공적인 적용을 위해 해결해야 할 난제들을 정리하였다.

2. Concolic 테스팅 및 CROWN 소개

Concolic 테스팅은 dynamic concrete 분석과 static symbolic 분석을 결합하여 높은 커버리지를 갖는 테스트 입력값을 자동으로 생성하는 기법이다. Concolic 테스팅은 테스트 입력값을 심볼릭 변수로 설정해주면 주어진 프로그램의 실행 가능한 경로를 systematic 하게 찾고, 각 경로를 따라 실행할 수 있는 테스트 입력값을 자동으로 계산해서 알려준다[4][5]. Concolic 테스팅은 모든 실행 가능한 경로를 찾아 테스트하기 때문에 수작업으로 입력값을 만드는 것보다 효과적으로 코드 커버리지를 올릴 수 있다.

CROWN[6]은 C 프로그램을 위한 concolic 테스팅 도구이다. CROWN은 CREST[7]를 확장하여 개발되었으며, bitwise 연산, floating point 연산, 비트 필드와 같은 복잡한 형태의 C 연산을 지원한다는 강점이 있다.

3. Concolic 기술 적용 연구

3.1 대상 프로그램 코드 정보

본 연구는 LIG 넥스원에서 개발한 국방 무기 체계에 탑재될 미들웨어 프레임워크의 10개 프로그램을 테스팅하였다. 표 1은 각 타겟 프로그램에 대한 code 정보를 보인다. 마지막 열은 기존 수작업 TC로 달성한 분기 커버리지이다.

3.2 연구 문제 (Research Questions)

Concolic 테스팅이 random 테스팅보다 얼마나 효과적으로 타겟 프로그램의 분기 커버리지를 높일 수 있는지 확인한다.

이 논문은 과학기술정보통신부의 재원으로 한국연구재단의 지원(NRF-2016R1A2B4008113), 과학기술정보통신부의 재원으로 한국연구재단-차세대 정보 컴퓨팅기술개발사업의 지원(NRF-2017M3C4A7068177), 교육부의 재원으로 한국연구재단의 지원(NRF-2017R1D1A1B03035851), LIG 넥스원의 지원을 받아 수행한 연구임

표 1. LIG Nex1 타겟 프로그램 코드 정보

타겟 프로그램	함수 수	LOC	분기 수	기존 TC 달성분기 커버리지
bit_bse_task	15	271	110	40.4%
task_tse	4	77	80	25.0%
xml	20	294	203	4.9%
frw_node_cfg	21	347	260	68.7%
frw_node_setAttribute	32	588	390	54.9%
ipc_toString	2	29	40	35.0%
RestCmd	28	149	46	32.6%
cli_CmdRoutineChange	7	123	64	64.1%
cli_Monitor_cb	10	136	50	36.0%
data_showVars	15	184	133	10.5%
평균	15.4	219.8	137.6	37.2%

RQ1. CROWN이 생성한 TC가 같은 수의 random 테스트로 생성한 TC보다 분기 커버리지를 얼마나 더 높이 달성했는가?

RQ2. CROWN이 생성한 TC가 같은 시간 동안 random 테스트로 생성한 TC보다 분기 커버리지를 얼마나 더 높이 달성했는가?

3.3 실험 설계

각 타겟 프로그램에 대해 concolic 테스트를 적용해 생성된 TC 수와 걸린 시간을 측정한 후, 같은 수의 TC를 생성하도록 (RQ1) 혹은 같은 시간만큼 (RQ2) random 테스트를 적용해 커버리지를 비교하였다.

CROWN이 생성할 최대 TC 수를 10만 개로 설정하였고, concolic 테스트에 대해서 DFS 탐색 전략을, random 테스트에 대해서 random_input 탐색 전략을 사용하였다.

실험은 Ubuntu Linux 16.04.5 LTS가 설치된 i5-4670K @ 3.40GHz의 CPU와 8GB의 RAM을 가진 서버에서 진행하였다.

3.4 심볼릭 변수 설정 예시

3.4.1. primitive 변수 타입

Primitive 타입 변수는 CROWN이 제공하는 API 함수를 사용해서 심볼릭 설정한다. 예를 들어 `SYM_int(x)`; 구문은 integer 타입 x를 심볼릭 변수로 설정하겠다는 의미이다.

3.4.2. 구조체 타입

구조체 타입 안의 멤버 변수는 멤버 변수가 primitive인 경우 각각에 대해 심볼릭 설정한다. 멤버 변수가 구조체인 경우 그 구조체의 멤버 변수를 다시 한번 재귀적으로 심볼릭 선언한다.

3.4.3. enum 타입

enum 타입 변수는 정의한 값 이외의 값을 가질 경우 false alarm을 일으킬 수 있기 때문에 되도록 정의한 값을 갖도록 심볼릭 설정을 하는 것이 중요하다. 그림 1은 enum 타입 변수인 level을 enum에서 정의한 값과 예외 처리를 위해 정의가 안 된 값 중 하나를 가지도록 심볼릭 설정한 모습이다.

enum 타입 정의

```
typedef enum
{
    NS_LOG_LEVEL_FATAL = 10,
    NS_LOG_LEVEL_SEVERE = 20,
    NS_LOG_LEVEL_ERROR = 30,
    NS_LOG_LEVEL_WARN = 40,
    NS_LOG_LEVEL_CAUTION = 50,
    NS_LOG_LEVEL_NOTICE = 60,
    NS_LOG_LEVEL_INFO = 70,
    NS_LOG_LEVEL_DEBUG = 80,
    NS_LOG_LEVEL_TRACE = 90,
    NS_LOG_LEVEL_VERBOSE = 100,
} NS_log_level_t;
```

심볼릭 변수 세팅

```
unsigned char coin;
SYM_unsigned_char(coin);
switch (coin) {
    case 0: level = NS_LOG_LEVEL_FATAL; break;
    case 1: level = NS_LOG_LEVEL_SEVERE; break;
    case 2: level = NS_LOG_LEVEL_ERROR; break;
    case 3: level = NS_LOG_LEVEL_WARN; break;
    case 4: level = NS_LOG_LEVEL_CAUTION; break;
    case 5: level = NS_LOG_LEVEL_NOTICE; break;
    case 6: level = NS_LOG_LEVEL_INFO; break;
    case 7: level = NS_LOG_LEVEL_DEBUG; break;
    case 8: level = NS_LOG_LEVEL_TRACE; break;
    case 9: level = NS_LOG_LEVEL_VERBOSE; break;
    default: level = 110; break;
}
```

그림 1. enum 타입 심볼릭 변수 설정

3.4.4. 스텝의 리턴 값

대상 프로그램 코드뿐 아니라 스텝의 리턴값도 심볼릭 선언해야 한다. 스텝의 고정된 리턴값이 분기 조건에 사용되는 경우 분기 조건의 값이 바뀌지 않기 때문에 부득이하게 탐색하지 못한 경로가 생긴다. 스텝의 리턴 값을 개발자가 설정한 가능한 값의 범위 내에서 심볼릭 설정을 해서 이 문제를 해결할 수 있다. 그림 2는 가능한 값이 2개인 스텝의 리턴 값을 심볼릭 설정한 모습이다.

```
NS_ret_t NS_task_isApplicationTasksStopped() {
    unsigned char coin;
    SYM_unsigned_char(coin);
    if (coin) return NS_SUCCESS;
    else return NS_FAIL;
} // RestCmd 프로그램 test_main.c 파일 line 37
```

그림 2. 스텝의 리턴 값 심볼릭 설정

5. 실험 결과

표 2는 각 타겟 프로그램에 대해 concolic 테스트, 같은 TC 수의 random 테스트, 그리고 같은 시간 동안의 random 테스트 분기 커버리지 결과이다. 10만 개 미만 TC로도 모든 경로가 커버된 경우, TC 생성을 중단하였다.

5.1 RQ1에 대한 결론

같은 TC 수로 concolic 테스트가 random 테스트보다 효과적으로 분기 커버리지를 높였다. Concolic 테스트가 생성한 평균 33445.5개 TC는 평균 81.5%의 분기를 달성하여, random 테스트가 달성한 분기 커버리지보다 34.2%p 더 높이 달성했다. 또한, 10개 프로그램 모두 concolic 테스트가 최소 11%p 더 높은 분기 커버리지를 달성했다.

5.2 RQ2에 대한 결론

같은 시간 동안 concolic 테스트가 random 테스트보다 효과적으로 분기 커버리지를 높였다. Concolic 테스트가 평균 1174.3초 동안 생성한 TC는 평균 81.5%의 분기를 달성하여, random 테스트가 달성한 분기 커버리지보다 30.6%p 더 높이 달성했다. 또한, 10개 프로그램 모두 concolic 테스트가 최소 9%p 더 높은 분기 커버리지를 달성했다.

표 2. Concolic 테스트 및 random 테스트 달성 커버리지

타겟 프로그램	생성 수	TC 시간 (s)	Concolic TC 달성 커버리지 (%)	같은 숫자의 Random TC 달성 커버리지 (%)	같은 시간의 Random TC 달성 커버리지 (%)
bit_bse_task	35	16	78.2	50.0	50.0
task_tse	20	1	72.5	32.5	45.0
xml	100,000	4900	73.1	4.9	4.9
frw_node_cfg	7	1	78.5	64.2	66.5
frw_node_setAttribute	100,000	3912	80.0	55.4	55.4
ipc_toString	6,144	111	100	77.5	77.5
RestCmd	27,841	366	87.0	2.2	2.2
cli_CmdRoutineChange	96	3	82.8	71.8	71.8
cli_Monitor_cb	100,000	2427	72.0	54.0	54.0
data_showVars	312	6	91.0	60.2	82.0
평균	33445.5	1174.3	81.5	47.3	50.9

5.3 발견한 결함

Concolic 테스트로 xml와 frw_node_cfg에서 크래시 버그를 발견하였다. Random 테스트로 발견 못 했고 LIG 넥스원에서 그동안 발견 못 했던 새로운 결함이다.

5.3.1 xml 결함

xml은 입력값으로 받은 string 형태의 XML 파일을 파싱해서 트리 자료 구조에 저장하는 프로그램이다. Concolic 테스트가 생성한 10만 개의 TC 중 9164개가 모두 같은 line에서 크래시 버그를 일으켰다. 결함 분석 결과, XML 파일에 여는 태그 없이 닫는 태그만 있는 경우(예.) NULL 포인터 역참조가 발생하여 크래시가 발생했다.

5.3.2 frw_node_cfg 결함

frw_node_cfg은 트리 자료 구조에 저장된 XML 파일의 파싱 데이터를 다시 읽는 프로그램이다. Concolic 테스트가 달성하지 못한 분기를 분석하면서, XML 파일에 닫는 태그 없이 70개 이상의 여는 태그(예. <a>)만 있는 경우 크래시가 발생하는 버그를 검출했다.

5.4 해결해야 할 기술적 난제

국방 무기체계 미들웨어 도메인에 concolic 테스트의 성공적인 적용을 위해 해결해야 할 난제들은 다음과 같다.

5.4.1 수작업의 심볼릭 변수 설정

테스터가 소스 코드상에서 구조체 멤버 변수들의 타입을 찾아야 하는 번거로움이 있다. 특히, bit_bse_task의 경우, 구조체가 4중 구조체로 정의되어 있어 (구조체 A의 멤버 b가 구조체 B 타입이고, B가 구조체 멤버 c를 갖고 등등) 14,416개 변수에 대해 수작업으로 심볼릭 설정을 진행했다. 따라서 수작업을 줄이기 위해 심볼릭 변수 설정을 자동화하는 연구가 진행된다면 이러한 수고를 줄일 수 있을 것이다.

5.4.2 경로의 폭발적 증가

경로의 폭발적 증가(path explosion)로 인한 concolic 테스트의 확장성(scalability) 문제가 발생한다. 그림 3은 xml의 프로그램 구조 중 일부분이다.

```

1 for (... ; ... ; p++) { // (loop iterating 22 times)
2 ...
3 if(string_begins_no_case("![CDATA[ ", p)) { ... } ...}
    
```

그림 3. 경로의 폭발적 증가 코드 예시

3번째 줄의 분기는 string p가 “![CDATA[”로 시작하는지 확인한다. Concolic 테스트가 해당 분기 조건을 참으로 만드는 string p를 생성하기 위해서는 적어도 10번의 iteration이 필요하다 (즉, “![CDATA[” 길이가 9이므로). 그러한 분기가 22번 반복하는 loop 안에 존재하기 때문에 10의 22제곱 수 만큼의 iteration을 통해 해당 코드의 모든 경로를 탐색할 수 있다. 이러한 경로의 폭발적 증가를 해결하기 위해서 path pruning[8]이나 효율적인 경로 탐색 알고리즘을 개발[9] 등 많은 연구가 진행되고 있다.

5.4.3. 국방 미들웨어 동적 할당 디버깅의 어려움

타겟 국방 무기 체계 SW에서는 평균 7.3번의 메모리 동적 할당이 사용되고 있어 메모리 누수나 의도치 않은 메모리 할당 실패로 인한 내부적인 결함이 발생할 수 있다. 그러나 concolic 테스트는 구체적인 성능 관련 assert가 있지 않으면, 이러한 동적 성능 결함을 자동으로 발견하지는 못한다.

6. 결론

Concolic 테스트를 LIG 넥스원에서 개발한 국방 무기 체계 미들웨어 도메인에 적용해 높은 커버리지를 달성하고 결함을 발견하여 SW의 품질을 향상했다. 또한, 실무자들이 확장 적용할 수 있게 심볼릭 모델링 방법을 예시로 제안하였다. 그러나 concolic 테스트 적용 시 발생하는 기술적인 이슈가 존재하기 때문에 이를 해결하려는 노력이 필요하다.

7. 참고 문헌

[1] 박철현 외 3명, “무기체계 임베디드 소프트웨어의 유지보수 체계 개선 및 정보보호체계 구축 방안.” 보안공학연구논문지 Journal of Security Engineering 12.4, 2015
 [2] Kannavara et al., Challenges and Opportunities with Concolic Testing. In NAECON, 2015
 [3] Cadar, et al., KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs In OSDI, 2008
 [4] Kim et al., Industrial application of concolic testing on embedded software: Case studies. In ICST, 2012
 [5] Kim et al., A scalable distributed concolic testing approach: An empirical evaluation. In ICST, 2012
 [6] <https://github.com/swtv-kaist/CROWN>
 [7] CREST, <https://github.com/jburnim/crest>
 [8] Jaffar et al., Boosting concolic testing via interpolation. In FSE, 2013
 [9] Ma et al., "Directed Symbolic Execution". In SAS, 2011