특질기반 테스트 대상 함수 추출을 위한 함수탐색기

김동우, 박민규, 최윤자
경북대학교 IT대학 컴퓨터학부
e-mail: kdw9242@gmail.com

A function finder for property-based extraction of test target functions

Dongwoo Kim, Mingyu Park, Yunja Choi
School of Computer Science and Engineering, College of IT engineering
Kyungpook National University

요 약

고안전성이 요구되는 내장형 소프트웨어의 경우 극히 낮은 확률로 발생하는 오류로 인하여 전체시스템의 안전에 치명적인 상황을 야기할 수 있으므로, 철저한 안전성 검증이 요구된다. 모든 가능한 실행경로를 고려해야 하는 안전성 검증의 고비용 문제를 해결하기 위하여, 기존연구에서는 안전성 특질 기반 테스트 대상함수를 추출하여 테스트 시나리오 생성하는 생성기를 개발하여 검증 효율을 높이는데 기여하였다. 그러나 기존의 도구는 함수포인터를 탐색 하지 못한 문제와, 변수에 대한 규칙 부족 문제 그리고 모듈화 되지 않아 유지 및 보수가 어려운 문제가 있었다. 본 논문에서는 기존도구의 문제점들을 개선하여 정확도를 높인 새로운 함수탐색기를 소개한다. 개발된 함수탐색기는 모듈화되어 차후에 수정 및 보완 문제에 대하여 유연하게 대처할 수 있게 하였다. 개선된 함수탐색기를 OSEK/VDX[1] 기반의 개방형 차량전장용 운영체제인 Trampoline을 대상으로 테스트 해 본 결과 기존 도구보다 약 68%의 높은 정확도를 보였다.

1. 서론

차량 전장용 운영체제는 시스템 오류 발생 시 치명적인 결과를 초래 할 수 있어 안전이 매우 중요한 시스템이다. 따라서 차량용 운영체제의 안전성 검증은 일반 정보시스템의 검증방식보다 철저한 기법을 요구하며, 주로 정형검증 기법과 같은 고비용을 요구하는 기법들이 사용되어 왔다. 반면, 참고문헌[2, 3] 에서는 안전성 검증을 위하여 안전성 특질과 관련된 프로그램 코드들만을 추출하고 테스트 시나리오를 자동으로 생성하는 방법을 통해 안전성 검증의 효율을 높이는 방안을 제시했다.

본 연구팀에서는 테스트시나리오 자동생성을 위한 기본 도구로써 특성기반 함수추출기[4]를 개발한 바 있다. 하지만 기존의 함수추출기는 기존 연구[4]에서 제시된 방법이 올바르게 구현되지 않아 성능의 저하가 있었으며, 기존 연구의 실험부족으로 함수포인터가 사용되었을 때의 대응규칙의 부제나, 변수간의 의존관계 분석규칙이 부족해 변수가 제대로 추출되지 않는 문제를 파악하지 못했다.

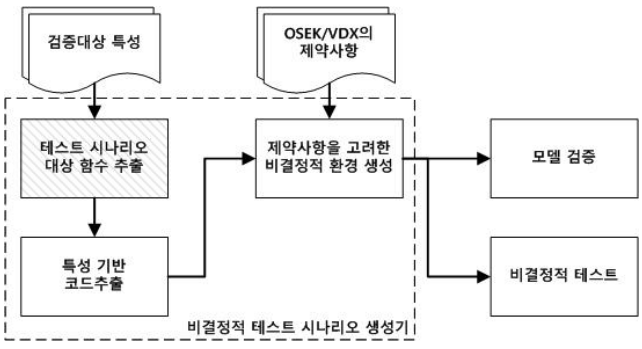
따라서 본 연구에서는 기존 도구의 문제점들을 개선하여 성능을 높이고, 기존 도구에서 분석규칙이 미흡해 추출하지 못한 부분들을 고려해 추출대상에 포함시켰다. 또한 모듈화를 통하여 차후에 추가해야할 부분에 대해서 쉽게 수정할 수 있게 개선했다. 개선된 도구는 OSEK/VDX 기반 차량전장용 운영체제인 Trampoline[5]의 안전성 검증에 테스트 대상 함수를 추출하는 도구로 활용했으며, 기존 도구보다 약 68%의 높은 정확도를 보였다.

이 논문은 2012년 정부(교육과학기술부)의 재원으로 연구재단의 지원을 받아 수행된 연구임 (2012R1A1A4A01011788).

2. 연구 배경

2-1. 테스트 시나리오 생성기

기존 연구[2, 3]에서 제안한 안전성 검증 테스트 시나리오는 정형검증기법을 적용할 때 발생하는 기하급수적인 비용증가를 완화하기 위해 제안된 방법이다. 본 방법은 검증하고자 하는 특성과 밀접한 연관이 있는 테스트 시나리오 대상 함수를 추출하고, 대상 함수 및 검증할 특성과 연관된 코드를 추출해 검증대상의 크기를 줄인다. 추출된 코드는 OSEK/VDX의 제약사항을 고려한 비결정적 환경과 함께 테스트 시나리오 및 모델검증 등에 활용된다. 본 연구의 함수추출기는 특성과 밀접한 연관이 있는 테스트 대상 함수를 추출하는데 활용되었다.



(그림 1) 테스트 시나리오 생성기 구성

2-2 기존 도구의 문제점

기존의 함수탐색기는 세 가지 정도 문제가 있었다.

첫째로, 기존의 함수탐색기의 경우 호출 관계만 가지고 함수를 추출하였기 때문에, 다음과 같이 funcA에서 funcB를 함수포인터로 사용하는 경우 추출할 수 없었다. 하지만 funcA 함수가 funcB 함수 호출에 영향을 미치므로 funcA도 추출해야 한다.

```
void funcA() {
    signal(1, funcB);
}
```

둘째로, 변수를 뽑아내는 규칙이 부족하여 기존에 만들어진 함수 탐색기는 필요한 변수를 뽑아 낼 수 없었다. 예를 들어 y = a.b->c처럼 구조체의 연속이 이어진 상황이 있을 경우, 변수로 a와 b와 c를 추출하는 것이 아니다. 필요한 변수인 c를 추출해야 한다. 하지만 기존의 함수 탐색기는 셋 중 아무것도 추출 해 내지 못 하였다.

셋째로, 기존의 함수 탐색기의 경우 모듈화가 되어 있지 않아, 유지 및 보수의 경우 새로운 수정을 할 수가 없었다.

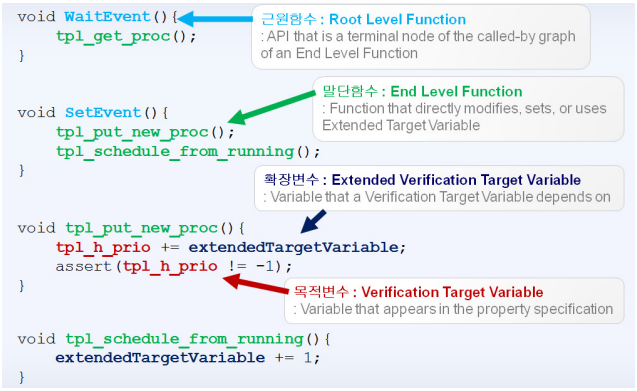
이러한 세 가지 문제점들로 인하여 기존의 함수탐색기는 정확도가 떨어지고 수작업이 많이 요구 되었다.

3. 특성 기반 테스트 대상 함수 추출 기법

3-1 추출 대상

시나리오를 만들기 위한 함수 및 변수를 추출하기 위해서, 한 개 혹은 여러 개의 안전성 특질과 관련된 목적변수를 입력 받아야 한다. 입력 받은 목적변수의 값을 변경 시키는 변수를 확장변수라고 한다. 이러한 목적변수와 확장변수의 값을 변경하는 문장이 있는 함수를 말단함수라고 한다.

근원함수는 이러한 말단함수를 호출하는 함수를 재귀적으로 찾아 나온 함수이다. 근원함수를 호출하면 말단함수를 모두 호출하여 확장변수의 값을 변경한다. 그래서 시나리오 생성 시 근원함수만 임의로 호출하면 해당 목적변수 값의 변화를 알 수 있으므로 근원함수의 목록을 추출해야 한다. (그림 2) 는 분석 하고자 하는 대상의 예제이다.

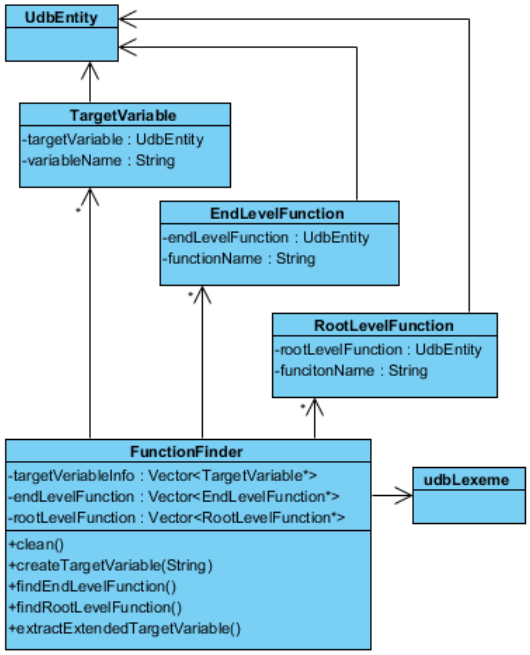


(그림 2) 목적변수, 확장변수, 말단함수, 근원함수

안전성 특질을 가진 assert(tpl_h_prio != -1)에서 변수 tpl_h_prio를 목적변수라고 하였을 경우, tpl_h_prio를 변경하는 extendedTargetVariable은 확장변수이다. 그리고, 목적변수인 tpl_h_prio 값의 변경이 일어나는 tpl_new_proc은 말단함수이며, 확장변수인 extendedTargetVariable 의 값을 수정하는 tpl_schedule_from_running 도 말단함수이며, 말단함수를 호출하는 SetEvent는 말단함수의 상위함수이고, SetEvent를 호출하는 WaitEvent는 SetEvent의 상위함수이다. WaitEvent의 상위함수는 없으므로, WaitEvent는 근원함수가 된다.

3-2. 함수 탐색기

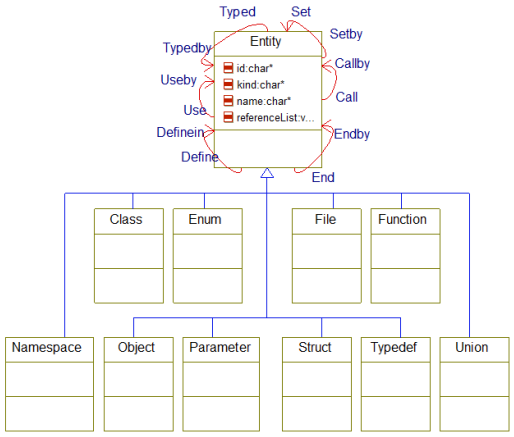
(그림 3)은 새로이 개발된 함수탐색기의 구성도이다. FunctionFinder는 추출 할 목적변수를 입력 받아, 확장변수, 말단함수, 근원함수를 추출하여 vector에 저장한다. 저장된 함수 및 변수는 시나리오 생성을 위해 사용 된다. TargetVariable, EndLevelFunction, RootLevelFunction 클래스들은 각각 목적변수, 말단 함수, 근원함수, 들을 저장하는 클래스 이다.



(그림 3) 함수 탐색기 구성

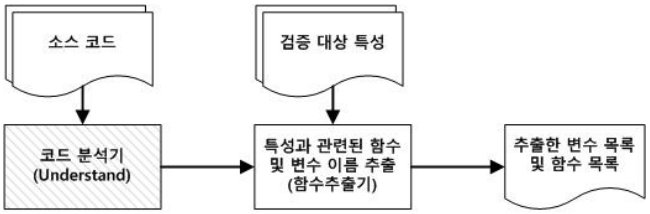
테스트 대상 함수 및 변수를 찾기 위해 정적 분석 도구인 Understand를 사용한다. 분석을 위해 사용된 Understand에는 세 가지 개념이 있다. 첫째로 형, 이름, 식별자를 가지고 함수 및 변수를 나타내는 entity(UdbEntity)가 있다. 둘째로 entity 사이의 연관관계와 타입, 소스코드에서의 위치를 나타내는 reference가 있다. 셋째로, reference가 어떤 관계를 가지고 있는지를 나타내는 kind가

존재한다. (그림 4)는 Entity 의 계층 구조이다.



(그림 4) Entity 계층 구조[6]

reference의 경우 변수와 함수사이 혹은 함수와 함수사이의 참조만 알 수 있으므로, 변수와 변수사이의 참조를 알 수 없어, 확장변수를 찾을 때 사용할 수 없었다. 그리하여, 변수 간의 참조를 프로그램으로 찾기 위해 파서인 lexeme을 사용하여, 목적변수 뒤에서 값을 설정하는 확장변수를 찾는다. 다음은 함수 탐색기의 동작 과정이다.



(그림 5) 함수 탐색기의 프로세스

(그림 5) 의 프로세스와 같이 소스코드를 코드 분석기(Understand)에 넣어서 얻은 데이터와 추출하고자 하는 검증 대상의 특성을 함수 탐색기에 넣어 관련 있는 함수 및 변수들을 소스코드에서 찾은 후 추출하여 출력한다.

3-3 기존 도구의 문제 해결

3-3-1 함수포인터

새로운 함수탐색기에서는 호출 외 함수포인터의 사용을 고려하기 위해서 filter에 useby ptr을 추가 하였다. 추가한 usebyptr 에 의해 함수포인터 외 포인터 변수나 파일포인터가 추출되어 함수 포인터인지 확인하는 작업이 필요하다. <표 2> 는 함수포인터를 고려하여 근원함수를 추출하는 코드이다.

<표 2> 함수포인터를 고려하여 추출하는 코드

```
findRootLevelFunction(말단함수 Entity) {
  상위함수 리스트 : vector<entity>;
  endLevelFunctionEntity를 상위함수 리스트에 추가
  while(모든 상위함수 리스트에 대하여 ) {
    functionReference
      = filter(endLevelEntity, "callby, useby ptr");
    funcionReferece.함수포인터가 아닌 포인터 제거
    if(functionReference.size != 0)
      상위함수들을 상위함수 리스트에 추가
    else
      근원함수 리스트에 추가
  }
}
```

말단함수 entity 하나를 입력 받아 자신을 호출하는 상위함수를 찾는다. 상위함수가 없다면 근원함수 리스트에 추가한다. 하지만 상위함수가 있다면 근원함수가 아니므로 상위함수 리스트에 추가하여 근원함수를 찾는다.

3-3-2 확장변수 추출

확장변수를 찾기 위해 기존도구에서는 Understand DB Entity 의 reference만을 이용하여 하였지만, 변수와 변수사이의 의존관계는 파악할 수 없었다. 새로 개발된 도구에서는 Understand에서 지원하는 파서인 lexeme을 이용하여 변수간의 의존관계까지도 분석할 수 있었다.

lexeme은 Understand의 구문 분석기로 소스코드에 있는 모든 단어 및 공백을 하나 하나씩 구분하여 분석한다. entity의 reference로 set이나 modify를 이용하여 위치를 찾은 후, parser인 lexeme을 사용하여 목적변수 다음에 오는 확장변수를 추출을 하였다. Understand를 이용한 확장변수 추출에 대한 규칙은 다음과 같다.

1. set 이나 modify 되는 위치에서 문장이 종료 될 때까지 존재하는 모든 변수의 entity를 추가한다.
2. index와 casting 하는 entity를 확장변수 리스트에 추가하지 않는다.
3. 구조체의 연속이 이어질 경우 연속된 구조체의 마지막 변수만 확장변수 entity에 추가한다.
4. 함수 entity는 추가하지 않고, 매개변수 entity는 추가한다.

목적 변수가 set이나 modify되는 위치에서 reference로 소스코드에서의 위치를 확인하고, lexeme으로 entityID와 위치가 같은 lexeme을 찾아 문장의 끝인 ';'이 나올 때까지 존재하는 확장변수를 저장한다. 확장변수만을 저장하여야 하므로, castingEntity나 index 를 저장하지 않도록 하여야 한다.

예를 들어, y = x + (a)b + c[d]와 같은 상황이 있다면, y값을 수정하는 값은 x와 b와 c이므로 확장변수는 b와 c가 된다. a는 casting entity 이므로 확장변수가 아니고 d

는 index 이므로 확장변수가 아니다.

구조체의 연속이 이어질 경우, 마지막 구조체 변수만 저장한다. 예를 들어 y = a->b.c처럼 구조체의 연속이 이어질 경우 앞 부분인 구조체의 머리는 확장변수 리스트에 추가하지 않고, 구조체의 꼬리 변수인 c만 확장변수 리스트에 추가한다.

a = func(b)와 같은 상황에서는 func도 한 entity 이므로, 추가된다. 하지만 함수는 확장변수가 아니므로 확장변수 리스트에 추가하지 않고, 매개변수 b는 확장변수 리스트에 추가한다.

4. 실험

차량 전장용 OS인 Trampoline 소스코드 중 assert 함수 안의 여러 변수들을 목적변수로 함수탐색기에 입력하여, 기존의 함수탐색기와 비교해 보았다. 아래는 차량 전장용 OS인 Trampoline 소스코드 중 assert함수를 사용한 문장이다.

```
1. assert((tpl_h_prio >= 0) && (tpl_h_prio < 3));
2. assert(tpl_fifo_rw[tpl_h_prio].size > 0);
3. assert((prio >= 0) && (prio < 3));
4. assert(tpl_fifo_rw[prio].size < tpl_ready_list[prio].size);
5. assert(tpl_kern.running != NULL);
6. assert((tpl_kern.running->state) == RUNNING);
7. assert(tpl_h_prio != -1);
8. assert(tpl_locking_depth>0);
```

<표 3>은 assert 함수 안의 변수를 목적변수로 지정하여 기존의 함수탐색기와, 새로 만들어 낸 함수탐색기로 추출한 결과를 Trampoline 소스코드와 비교한 것이다.

<표 3> 각각의 목적 변수에 대한 말단함수와 근원함수와 확장변수의 개수 비교 (기존의 프로그램 추출에서 수작업 제외)

목적변수	실제	기존 함수탐색기 (정확도)	새 함수탐색기 (정확도)
tpl_h_prio	27	8(29.6%)	27(100%)
tpl_fifo_rw.size	13	0(0%)	13(100%)
tpl_kern.running->state	16	0(0%)	16(100%)
prio(tpl_put_preempted_proc 지역변수)	21	8(38%)	21(100%)
prio(tpl_put_new_proc 지역변수)	8	8(100%)	8(100%)
tpl_kern.running	34	0(0%)	34(100%)
tpl_ready_list.size	1	0(0%)	1(100%)
tpl_locking_depth	33	25(75%)	33(100%)

<표 3>의 결과를 비교 해 보았을 때, 새로 만든 함수탐색기는 기존의 함수 탐색기와 비교 했을 경우, 완벽한 정확도를 가지고 있어, 시나리오 생성 시 함수나 변수 추출로 인해 생기는 오차율을 줄이고, 수작업으로 인해서 생기는 시간문제도 해결 하였다.

6. 결론

함수와 변수를 추출하기 위하여 만들어진 기존의 함수탐색기의 함수포인터 문제를 수정하고, 확장변수에 대한 규칙을 추가하여 새로운 함수 탐색기를 만들었다. 새로운 함수 탐색기는 보다 정확한 함수와 변수의 추출을 가능하게 하여, 차량 전장용 OS 인 Trampoline의 시나리오 생성과 검증을 보다 정확하게 하도록 기여를 하였다. 함수 탐색기의 개선된 점은 다음과 같다.

- 첫째로, 함수 포인터의 사용 문제가 있어, 필터링하는 부분을 추가하여, 함수포인터를 추출하여 사용할 수 있게 하였다.
- 둘째로, 확장 변수를 추출하는 방법에 있어서 기존에 존재하지 않던, 규칙들을 추가하여, 확장변수를 추출 가능하게 하였다.

하지만 아직 문제가 있는 부분도 있다. 예를 들어 구조체 안의 변수로 포인터가 많이 걸려 있는 경우, 이 구조체를 확장변수로 추출할 시에 많은 변수가 추출된다. 이렇게 추출하기 힘든 경우, 많은 변수들을 그대로 추출해야 하는지, 아니면 추출하지 않아야 되는지에 대한 연구가 아직 필요하다.

참고문헌

[1] OSEK/VDX. http://portal.osek-vdx.org.

[2] Mingyu Park, Taejoon Byun, Yunja Choi, “Property-based CodeSlicing for Efficient Verification of OSEK/VDX Operating Systems”, Proceedings First International Workshop on Formal Techniques for Safety-Critical Systems, 2012.

[3] 변태준, 최윤자, “OSEK/VDX 기반 전장용 운영체제의 안전성 검증을 위한 자동 테스트 시나리오 생성기”, 한국정보처리학회 추계학술발표대회 논문집 제 18 권 제 2 호, 2012.

[4] Nahida Sultana Chowdhury, Yunja Choi, “Automated Scenario Generation for Model Checking Trampoline Operation System”, 한국정보처리학회 추계학술발표대회 논문집 제 19 권 제 2 호, 2011.

[5] Trampoline. http://trampoline.rts-software.org/

[6] 박민규, 최윤자, 김진삼, “코드 정형검증을 위한 특성기반 코드 추출기”, 한국정보처리학회 추계학술발표대회 논문집 제 17 권 제 2 호, 2010