OSEK/VDX 기반 전장용 운영체제의 안전성 검증을 위한 자동 테스트 시나리오 생성기

변태준, 최윤자
경북대학교 IT 대학 컴퓨터학부
e-mail : bntejn@gmail.com, yuchoi76@knu.ac.kr

Automatic Test Scenario Generator for OSEK/VDX-based Automotive Operating Systems

Taejoon Byun, Yunja Choi
School of Computer Science and Engineering, College of IT engineering
Kyungpook National University

요 약

차량전장용 운영체제는 안전중요(safety-critical) 소프트웨어로써 엄밀한 검증과 테스트를 필요로 한다. 엄밀한 검증은 시스템의 모든 사용 가능한 시나리오의 도출을 필요로 하며, 이것을 수작업으로 생성하는 데에는 비용과 효율성에 문제가 있다. 본 연구에서는 차량전장용 운영체제의 국제표준인 OSEK/VDX 에 명시된 제약사항을 고려한 테스트 시나리오 자동 생성기와 이를 보조하는 OSEK/VDX 시뮬레이터를 개발하여 테스트 효율의 향상과 자동화를 도모하였다. 개발된 도구는 OSEK/VDX 기반 개방형 운영체제인 Trampoline에 적용하여 효과를 입증하였다.

1. 서론

차량 전장용 운영체제는 차량의 움직임을 제어하는 전장시스템의 제어를 담당하고 있으며 차량의 안전성에 큰 영향을 미칠 수 있는 안전중요 시스템이다. 따라서, 차량 전장용 운영체제의 엄밀한 검증은 차량의 안전성을 위한 필수요소라고 할 수 있다. 그러나, 기존의 차량용 운영체제를 위한 검증은 수작업으로 생성된 테스트 시나리오에 의존한 적합성 검증(confirmation testing) 방식이 주로 사용되고 있으며, 안전성 검증을 위한 엄밀성을 충족시키지 못하고 있다.

엄밀한 검증은 정형적 모델검증, 정적 코드분석, 그리고 엄밀한 테스팅의 세가지 접근방식으로 수행될 수 있다. 본 연구에서는 개발된 운영체제가 외부환경과 상호작용을 하는 모든 가능한 경우의 시나리오를 테스트하는 엄밀한 테스팅의 관점에서 OSEK/VDX 국제표준을 기준으로 하여 기존 방식의 문제점을 파악하고, 엄밀한 테스팅을 지원하는 보조도구와 테스트 시나리오 자동생성기를 개발하였다. 도구는 OSEK/VDX 표준에서 명시한 제약사항을 테스트 시나리오에 반영하기 위한 OSEK/VDX 시뮬레이터와 이 시뮬레이터를 이용하여 각 테스트 케이스마다의 시스템 상태 변화를 예측하고 제약사항을 만족하는 다음 테스트 케이스를 자동으로 선정하는 시나리오 생성기로 구성되었다. 개발된 도구는 OSEK/VDX 기반 개방형 운영체제인 Trampoline 의 안전성 검증에 적용하여 평균 82%의 커버리지를 보였다.

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

2. 연구배경 및 관련연구

OSEK/VDX[1]는 차량용 분산 제어장치의 공개 아키텍처에 대한 산업표준화를 목표로 독일 자동차 산업계 공동 프로젝트로 시작되어 국제적으로 통용되는 차량전장용 운영체제의 표준이다. OSEK/VDX 에는 총 26 개의 API 들이 정의 되어 있으며 각 함수마다 엄격한 제약조건을 명시하고 있어 이를 고려하지 않은 테스트 시나리오의 자동생성은 오히려 테스트의 효율을 낮추는 결과를 초래하며, 이러한 이유로 기존의 적합성 테스트 케이스들은 수작업으로 생성이 되어왔다

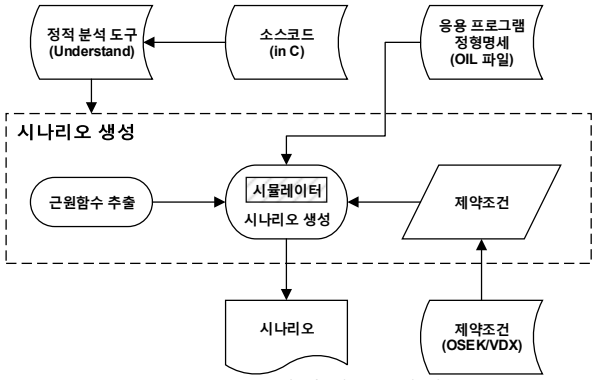
이러한 문제의 해결과 엄밀한 안전성 검증을 위하여 최근 몇 년간 OSEK/VDX 기반 운영체제를 위한 정형적, 비정형적 검증기법이 연구되고 있으나[5,6], 소스코드 분석을 기반으로 한 테스트 시나리오 자동생성 방식은 본 연구에서 처음으로 시도되었다.

3. 제약조건을 고려한 시나리오 자동 생성 개요

본 연구에서는 OSEK/VDX 에 명시되어 있는 제약조건들을 충족시키는 테스트 시나리오를 자동으로 생성하는 시나리오 자동 생성기를 구현하여 Trampoline

을 테스트하였으며, 이를 통해 유효하지 않은 시나리오를 배제하여 테스트의 효율을 높일 수 있었다.

그림 1 은 테스트 시나리오의 자동 생성을 위한 전체적인 절차를 도식화한 것이다. 우선 정적 분석도구인 Understand 를 이용해 Trampoline 의 소스코드를 분석하고, 테스트의 목적 변수에 접근하는 모든 근원함수를 추출한다. 그리고 OSEK/VDX 에 명시되어있는 모든 제약조건을 추출한다. 끝으로, 이렇게 추출된 제약조건을 모두 고려하여 임의의 시나리오를 생성한다. 본 연구에서는 시나리오 생성 과정에서 OSEK/VDX 응용프로그램의 정형명세(OIL 파일)를 입력 받아 생성될 테스트 환경의 작업, 자원, 이벤트와 같은 가변적인 요소들을 고정해두고 이 환경에서 작동하는 테스트 시나리오를 무작위로 생성하도록 하였다.



(그림 1) 시나리오 생성

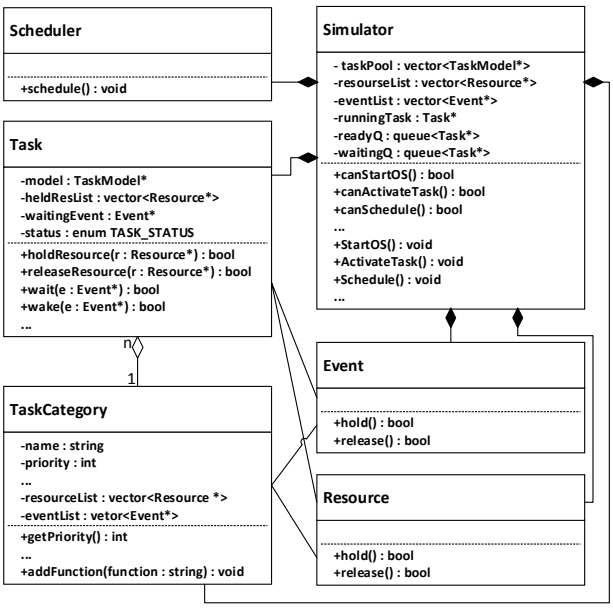
제약조건들을 충분히 반영 하기 위해서는 이전에 생성된 근원함수들이 호출됨으로써 발생한 모든 변화들을 기록하고 있어야 한다. 예를 들어 *ActivateTask(t1)* 이 시나리오에 더해졌다면, 다음 시나리오 생성 단계에서는 제약 조건들을 충분히 고려하기 위하여 작업 t1 을 *READY* 상태로 기록해야 하는 것이다. 일련의 과정은 모두 자동화 되었으며, 이러한 OSEK/VDX 의 명세에 따라 작성된 모의 운영체제를 OSEK/VDX 시뮬레이터라 명하였다. 그림 1 의 시나리오 자동 생성기의 주요 구성들의 기능들은 다음절에서 설명하였다.

4. OSEK/VDX 시뮬레이터

4-1. 모델

OSEK/VDX 운영체제의 실행시간 정보들을 예측하고 변화를 추적하기 위해 운영체제의 필수요소들을 모델링 하였다. 그림 2 는 이벤트(event), 자원(resource), 작업, 그리고 작업 스케줄러(task scheduler) 등을 포함한 시뮬레이터 모델을 도식화한 것이다.

TaskCategory 클래스는 OSEK/VDX 에 정의되어있는 작업(task)의 종류를 분류하기 위해 정의되었다. OSEK/VDX 응용프로그램(application)의 작업명세에는 각 작업의 이름, 우선순위, 이벤트의 목록, 자원의 목록, 동시에 생성될 수 있는 동일한 작업의 최대 개수가 있다. 작업에 대한 정적인 명세는 TaskCategory 에 표현할 수 있지만, 작업의 동적인 정보를 표현하고 관리하기 위해서는 다른 클래스가 필요하다. 더욱이 OSEK/VDX 에서는 하나의 작업 정적 명세에 따라 생성 가능한 작업의 숫자가 여러 개일 수 있기 때문에 동적인 정보는 별도로 Task 클래스에서 정의 하였다. 이 클래스에는 할당된 자원의 목록과 대기중인 이벤트, 작업의 상태 등이 정의되었다. 따라서 Task 클래스의 각 사례(instance)가 작업의 동적인 상태를 표현하며 이는 하나의 프로세스 제어 블록(PCB)에 해당한다 할 수 있다.



(그림 2) OSEK/VDX 모델

스케줄러는 작업의 우선순위와 선점가능(pre-emptive) 여부에 따라 작업의 실행 순서를 정해준다. 스케줄링이 발생하는 시점은 현재 수행중인 작업이 선점 가능하며, ReleaseResource, Schedule, ActivateTask, ChainTask 등 스케줄링을 발생시키는 함수가 호출될 때이다. 스케줄링이 발생하면 스케줄러는 준비상태 큐(ready queue)에서 가장 우선순위 값이 높은 작업을 추출해 RUNNING 상태로 만든 다음 해당 작업을 수행하며, 작업의 몸체(task body)에서 아직 수행되지 않았던 함수부터 수행하게 된다.

끝으로 Simulator 클래스가 OSEK/VDX 운영체제 모델의 모든 자원과 기능들을 관리하게 된다. 작업의 생성과 종료, 스케줄링 등 모든 기능들이 서비스로 제공되며, 외부에서 추출한 임의의 함수가 현재 상황에서 수행 가능한지 여부도 Simulator 를 통해 확인할 수 있다. 시뮬레이터에 관해서는 뒷장에서 자세히 설명한다.

4-2. 제약조건 도출

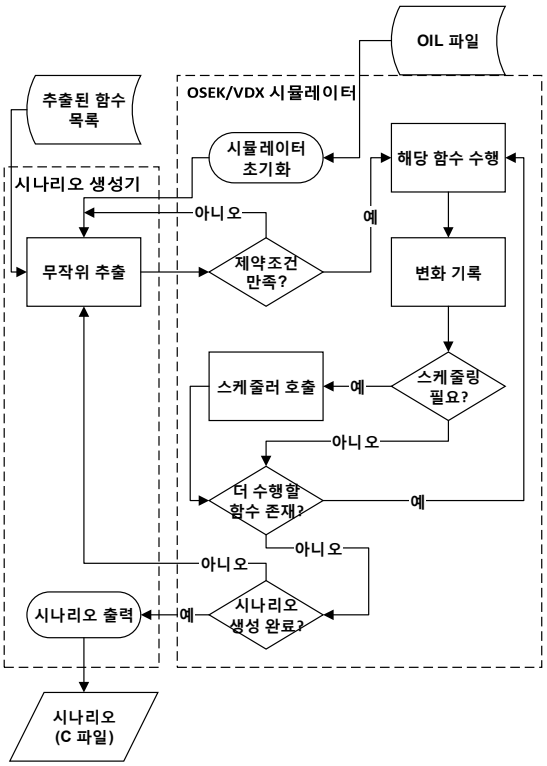
OSEK/VDX 에는 API 함수들간의 제약조건들이 명시되어있는데, 이들은 다음 API 함수가 호출될 수 있는 전제조건이 된다. 예를 들어, 작업을 종료하는 API 함수인 TerminateTask 가 호출될 수 있으려면 앞서

ChainTask 나 ActivateTask 가 호출되어 작업이 활성화 되어 있어야 한다. 이 경우 TerminateTask 의 선행조건은 {ActivateTask, ChainTask}라 할 수 있다. 표 1 에는 OSEK/VDX 표준에서 파악된 제약조건들이 일부 정리되어있다.

<표 1> 제약조건

근원함수	제약조건
StartOS	StartOS 는 최초에 한 번만 호출될 수 있다.
WaitEvent	WaitEvent 가 호출되면 호출한 작업은 대기상태로 변경되어, 다른 작업에서 SetEvent 가 호출되기 전까진 아무런 일도 할 수 없다. 따라서 WaitEvent 는 중첩되어 호출될 수 없으며 다른 작업에서 SetEvent 가 호출된 후에야 다시 호출될 수 있다.
ReleaseResouce	ReleaseResource 는 해당 자원이 이미 할당된 경우에만 호출될 수 있다.
TerminateTask	TerminateTask 는 ActivateTask 혹은 ChainTask 로 인해 활성화된 작업이 있을 때에만 수행이 가능하다.
Schedule ChainTask	Schedule 과 ChainTask 는 GetResource 에 의해 점유되어있는 자원이 없을 때에만 호출될 수 있다.

4-3. 시뮬레이터



(그림 3) OSEK/VDX 시뮬레이터

OSEK/VDX 시뮬레이터는 자원, 이벤트, 현재 실행상태인 작업의 참조 값, 준비상태 큐, 작업들의 상태와 같은 모든 실행 시간 정보들의 변화를 기록한다. 이 함수들 중 하나가 호출되면 시뮬레이터는 작업을 활성화하거나 스케줄러를 호출하는 등 OSEK/VDX 명세에 따라 동작을 수행하게 된다. 그림 3 은 시뮬레이터의 동작을 포함한 시나리오 생성의 전반적인 과정을 도식화하였으며, 그 중 핵심 과정들은 다음과 같다.

4-3-1. 시뮬레이터 초기화

첫 번째 단계에서는 OIL 파일로부터 OSEK/VDX 응용프로그램의 정형명세를 읽어와 시뮬레이터를 초기화한다. 정형명세에 정의되어있는 각각의 작업과 자원, 그리고 이벤트에 대한 정보는 위 그림 0 에 있는 Task, Resource, 그리고 Event 클래스의 오브젝트로 각각 사례화된다. 이로써 각 클래스의 사례들은 최초에 입력 받은 정형명세의 상세 정보들을 지니게 되는 것이다.

4-3-2. 무작위 추출과 제약조건 확인

시나리오 생성기에서 무작위로 추출된 근원함수는 제약조건을 충족여부의 확인 후 테스트케이스로 확정된다.

<표 2> 함수 추출 및 확인절차

```
While(!simulator.scenarioGenCompleted()){
    API extracted = getRandomFunction();
    switch(extracted){
    case START_OS:
        if(simulator.canStartOS())
            simulator.StartOS();
        break;
    case WAIT_EVENT:
        if(simulator.canWaitEvent())
            simulator.WatiEvent();
        break;
    …
}
```

시뮬레이터에는 각 근원함수의 제약조건이 정의되었다. 예를 들어 WaitEvent(e1)라는 함수가 무작위로 추출되었다면 다음과 같은 제약조건들을 충족해야 한다.

1) 실행중인 작업의 상태가 RUNNING 이어야 한다.
2) 실행중인 작업이 점유중인 자원이 없어야 한다.
3) 실행중인 작업의 정형명세에 이벤트 e1 이 선언되어 있어야 한다.

이러한 제약조건들은 각 API 함수들마다 별도로 작성되어 있으며 다른 API 함수의 경우도 각각 별도의 확인 과정을 거친다.

4-3-3. 시나리오 생성완료 여부 판단

시나리오 자동생성의 완료시점은 어느 한 작업의 시나리오가 TerminateTask 나 ChainTask 로 끝맺은 경우와 시나리오 자동생성 과정에서 특정한 작업에 시나리오가 전혀 추가되지 않는 경우이다. 수행 가능한 모든 작업의 명세(TaskCategory)에 대해 시나리오 생성이 완료되었거나 생성이 전혀 되지 않았다면 전체적인 시나리오 생성이 완료되었다고 판단하고 시나리오 생성을 종료한다. 그리고 마지막으로 실행 가능한 시나리오를 C 파일로 출력한다.

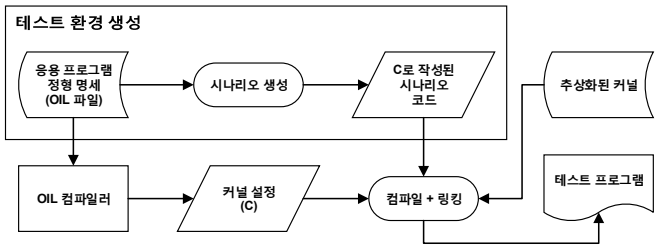
5. 실험

개발된 시나리오 생성기를 이용하여 Trampoline 운영체제의 assertion 들을 자동 검증하였으며, 이 assertion 들은 다음과 같다.

assert(tpl_h_prio != -1) (1)
assert(tpl_kern != NULL) (2)
assert(tpl_kern->state == RUNNING) (3)

테스트를 수행하기 위해서 그림 1 과 같이 테스트 환경을 구성하였다. 먼저 Trampoline 내부에서 하나의 목적 변수와 관련된 assert 조건을 이용하여 이 변수에 직접적으로 접근하는 함수들인 말단함수들을 추출하였다. 그리고 앞서 추출된 말단함수를 호출하는 근원함수(API)들을 함수 호출관계를 파악하여 추출한 뒤 시나리오 생성에 활용하였다.

이렇게 생성된 시나리오는 테스트를 위해 아래 그림 4 와 같은 절차를 거친다. 생성된 시나리오와 OIL 파일로부터 생성된 커널 설정, 그리고 Trampoline 의 추상화된 커널 코드가 함께 최종 테스트 프로그램으로 컴파일되어 최종 테스트 환경이 완성된다.



(그림 4) 테스트 환경

실험에서는 정적분석 도구 SquishCoCo[2]를 이용해 추출된 말단함수의 코드에 대한 측정범위(coverage)를 측정하였다. 아래의 표는 각 말단함수 별 코드에 대한 측정범위와 측정에 사용된 메모리 공간을 보여준다.

<표 3> 말단함수 코드 측정범위

속성	*assert(tpl_h_prio != -1)* 코드 길이 : 1337 lines 8 개의 근원함수 / 50 개의 함수				
시나리오의 길이	14	20	22	32	34
tpl_schedule_from_running	100%	100%	100%	100%	100%
tpl_get_proc	100%	100%	100%	100%	100%
tpl_put_new_proc	66.7%	66.7%	66.7%	66.7%	66.7%
메모리(MB)	2.64	2.64	2.64	2.64	2.64
속성	*assert(tpl_kern != NULL)* *assert(tpl_kern->state == RUNNING)* 코드 길이 : 1378 lines 9 개의 근원함수 / 52 개의 함수				
시나리오의 길이	14	20	22	32	34
tpl_schedule_from_running	100%	100%	100%	100%	100%
tpl_schedule_from_dying	80%	80%	60%	80%	100%
tpl_schedule_from_waiting	0%	100%	66.7%	66.7%	66.7%
tpl_start_schedule	100%	100%	100%	100%	100%
tpl_wait_event_service	0%	60%	60%	60%	60%
tpl_activate_task	100%	100%	100%	100%	100%
tpl_set_event	0%	80%	80%	80%	80%
메모리(MB)	2.64	2.64	2.64	2.64	2.64

시나리오 자동 생성을 통해 각기 다른 두 개의 목적 변수에 관련해 추출된 근원함수들에 따라 각기 다른 시나리오를 여러 개 만들 수 있었다. 그리고 위 표 3 과 같은 실험결과를 얻을 수 있었다.

표 3 의 실험결과에서 시나리오의 길이가 증가함에도 불구하고 측정범위 값이 더 이상 오르지 않는 경우를 볼 수 있다. 이런 현상이 발생하는 이유로는 크게 두 가지가 있을 수 있는데, 첫 번째는 예외처리 코드의 경우이다. 예외처리 코드의 경우 예외가 발생하지 않는 한 통과할 수 없으므로 측정범위가 더 이상 올라가지 않은 것이다. 두 번째 경우는 추출되지 않은 말단함수와 연관된 조건문이 있는 경우이다. 이 경우 역시 조건문이 항상 거짓이 되어 일부 코드는 어떤 경우에도 수행되지 않는다. 따라서 코드의 측정범위는 더 이상 증가하지 않고 테스트가 종료된 것이다.

6. 결론

엄격한 제약사항을 갖는 시스템의 엄밀한 검증을 위해서는 제약사항을 위배하는 테스트 시나리오와 제약사항을 만족하는 테스트 시나리오가 모두 필요하다. 제약사항을 만족하는 테스트 시나리오의 가능한 경우의 수는 제약사항을 위배하는 테스트 시나리오의 가능한 경우에 수에 비해 월등히 많은데, 무작위 시나리오 생성 방식으로는 테스트 환경에서 정상적으로 수행되는 결과를 거의 얻을 수 없었고 수작업으로 예측 가능한 시나리오를 작성하는 것은 부정확할뿐더러 많은 시간이 소요되었다. 본 연구에서 개발한 시나리오 자동 생성기는 이러한 문제를 해결하여 제약사항을 만족하는 테스트 시나리오를 자동 생성하였고, 그 적용결과 테스트의 효율을 높이고 테스트 시나리오 작성에 소요되는 시간을 획기적으로 단축할 수 있었다.

참고문헌

[1] OSEK/VDX Portal. http://portal.osek-vdx.org.
[2] Squish Coco Code Coverage. http://www.froglogic.com/squish/coco/
[3] Trampoline - OpenSource RTOS project. http://trampoline.rts-software.org.
[4] Understand: Source Code Analysis and Metrics. http://www.scitools.com/.
[5] Yang et al., Model-based Design and Verification of Automotive Electronics Compliant with OSEK/VDX, In Second International Conference on Embedded Software and Systems, 2005.
[6] Jiang Chen and Toshiaki Aoki, Conformance Testing for OSEK/VDX Operating System Using Model Checking, In 18th Asia-Pacific Software Engineering Conference, 2011.