차량 전장용 운영체제의 주기적 태스크 검증을 지원하는 테스트 케이스 생성기[1)]

최우용, 김동우, 최윤자
경북대학교 IT대학 컴퓨터학부
e-mail : whdrmt12@gmail.com

Test Case Generator Supporting The Verification of Alarm In Automotive Operating System

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

요 약

안전 필수 시스템(Safety-critical system) 중 하나인 차량 전장용 운영체제의 엄밀한 검증을 위하여 모델 기반 테스트 생성기법들이 연구되어 왔다. 그러나 기존의 연구들은 이러한 차량 전장용 운영체제에서 빈번히 사용되는 주기적인 동작을 요하는 작업들에 대한 테스트 생성 문제를 해결하지 못하였다. 본 연구에서는 주기적 태스크의 검증을 지원하지 않았던 기존의 테스트 케이스 생성기에 알람 모델을 추가하여 보완하였다. 이를 통해 차량 전장용 운영체제의 검증에 있어서 주기적 태스크를 포함한 다양한 테스트 케이스를 생성할 수 있었고 차량 전장용 운영체제의 보다 엄밀한 검증이 가능해졌다.

1. 서론

자율주행차나 커넥티드 카(Connected car) 기술의 대두로 자동차의 전장화가 가속화되면서 이를 제어하는 차량 전장용 운영체제의 중요성이 높아지고 있다. 이러한 차량 전장용 운영체제는 안전성이 사람의 생명과 직결되는 안전 필수 시스템(Safety-critical system)이다. 따라서 차량 전장용 운영체제의 엄밀한 검증은 차량의 안전성에 있어서 필수적인 요소라고 할 수 있다.

하지만 기존의 수작업으로 생성된 테스트 케이스를 통한 검증은 엄밀하지 않았다. 엄밀한 검증을 위해서는 보다 효율적인 테스트 케이스 생성이 필요하다. 이와 관련된 선행 연구[4]에 따르면 API 간에는 엄격한 제약사항이 존재하고 이러한 제약사항들을 고려하여 테스트 케이스를 생성할 때 훨씬 효율적이었다. 또한 운영체제 모델과 앞서 제시한 제약사항을 모델 기반 검증을 통해 테스트 케이스를 자동으로 생성할 수 있었다.

그러나 기존의 연구들은 차량 전장용 운영체제나 기타 임베디드 시스템의 주기적인 작업의 검증을 위한 테스트 케이스 생성을 해결하지 못하였다. 차량 전장용을 포함한 제어 소프트웨어는 주기적인 동작을 요하는 작업이 굉장히 빈번하게 사용된다. 이러한 주기적인 동작을 요하는 작업을 주기적 태스크(Periodic task)라고 한다. 따라서 차량 전장용 운영체제의 안전성을 보장하기 위해서는 주기적 태스크의 검증이 매우 중요한 부분이다.

1) “이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임” (NRF-2016R1D1A3B01011685)

본 연구는 차량 전장용 운영체제 표준인 OSEK/VDX의 알람을 모델링하고 선행 연구를 보완함으로써, 주기적 태스크를 포함하는 테스트 케이스를 생성하여 운영체제를 검증할 수 있도록 하였다. 따라서 보다 다양한 테스트 케이스가 자동 생성이 되었고 안전 필수 시스템인 차량 전장용 운영체제를 좀 더 엄밀하게 검증할 수 있게 되었다.

2. 연구배경

2.1 OSEK/VDX

OSEK/VDX는 다양한 차량 전장 제어장치(ECU)를 위한 운영체제의 공개 표준 아키텍처로써 운영체제, 통신, 네트워크에 대한 표준을 정의한다.[1] OSEK/VDX 표준에 따라 제작된 운영체제는 차량 전장 제어장치마다 탑재되어 응용프로그램과 하드웨어 간의 중계 역할을 수행한다.

그림 1 차량 전장용 소프트웨어 생성 과정

그림 1은 OSEK/VDX 기반 운영체제과 응용프로그램, 환경설정이 함께 컴파일되어 차량 전장용 소프트웨어가 생성되는 과정을 도식화하였다. 환경설정에는 응용프로그램의 태스크와 알람 등 각각의 객체의 정보를 담고 있다.

2.2 모델 기반 테스트 케이스 생성기

선행연구에서는 OSEK/VDX 기반 운영체제의 안전성을 위한 엄밀한 검증을 위해서 테스트 케이스 자동 생성 도구를 설계하였다.[4] 이 도구는 NuSMV라는 상태 기반 모델 검증 도구를 사용하여 운영체제의 API를 무작위로 호출하는 상황을 가정하여 운영체제 모델의 모든 상태를 테스트하는 테스트 케이스를 생성한다.[3] NuSMV는 SMV 모델 언어로 작성된 모델이 어떤 속성을 만족하는지 검증하고 만족하지 않는다면 반례를 제시해주는 정형 검증(Formal verification)을 제공한다.

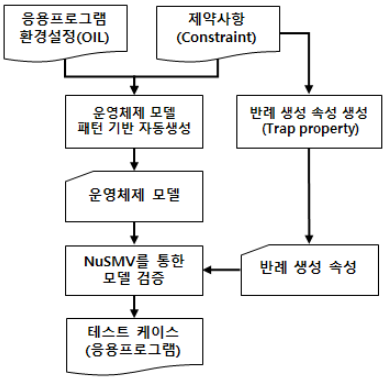

그림 2 모델 기반 테스트 케이스 생성기 흐름도

그림 2는 위에서 언급한 OSEK/VDX 기반 운영체제 검증을 위한 자동 테스트 케이스 생성기의 작동 원리를 도식화한 것이다. 환경설정인 OIL(OSEK Implementation Language)과 제약사항을 통해서 운영체제 모델을 자동 생성한다. 반례 생성 속성은 검증하고자 하는 속성이 P라면 그의 역인 not P로써, P가 참이면 NuSMV가 모델 검증을 통해 not P가 거짓인 반례를 결과로 제시하고 이를 테스트 케이스로 변환하게 된다.

선행 연구를 통해 효율적인 테스트 케이스 생성이 가능해졌지만, 기존 운영체제 모델은 OSEK/VDX의 태스크, 리소스와 이벤트 객체를 다뤘다. 일반적인 태스크의 검증에 초점을 두었고 알람이 모델에 포함되지 않았으므로 기존 도구로는 주기적 태스크에 대한 검증이 어려웠다.

2.3 OSEK/VDX의 알람

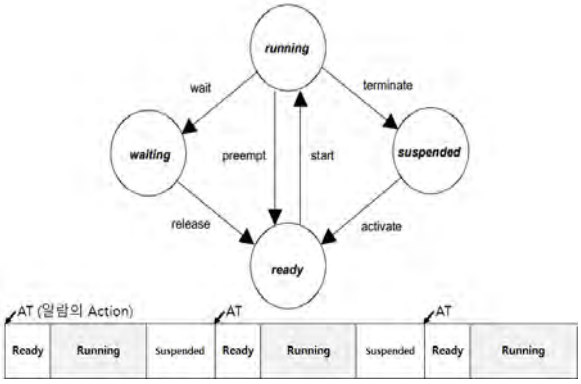

그림 3 태스크의 상태 다이어그램 및 주기적 태스크의 시뮬레이션

그림 3은 태스크의 상태 다이어그램과 알람을 통한 주기적 태스크 시뮬레이션[5] 결과를 가시화 한 것이다. 알람은 주기마다 환경설정에 따른 행위를 하고 이를 액션이라 한다. 알람의 액션은 OSEK/VDX 기반 API[1] 중 이벤트를 설정하는 SetEvent와 태스크를 활성화하는 ActivateTask가 있다. 그림 3의 경우 알람의 액션이 ActivateTask이다.

OSEK/VDX의 알람에 관련된 API는 SetRelAlarm, SetAbsAlarm, CancelAlarm이 있고, 각각 알람의 시작이나 응용프로그램의 시작을 기준으로 알람을 시작하거나 알람을 정지시킬 때 사용된다. 이 밖에도 알람은 주기 알람(Cyclic alarm) 혹은 일회성 알람(Single alarm)이 되는데 주기가 0ms이면 일회성 알람이다.

3. 연구내용

3.1 알람의 모델링

본 연구에서는 OSEK/VDX 기반 알람을 모델링함으로써 주기적 태스크를 포함하는 테스트 케이스를 생성하고자 한다. 우선 OSEK/VDX에서 응용프로그램의 환경설정에 명시된 알람의 정보를 상태 기계로 모델링하게 된다.

```
ALARM alarm0 {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = task1; };
    AUTOSTART = TRUE {
        ALARMTIME = 500; CYCLETIME = 1000; APPMODE = std;
    };
}
```

그림 4 알람 명세

그림 4은 환경설정의 알람에 대한 정보이다. 알람에 대한 카운터와 액션, 그리고 자동 시작 여부가 명시된다. 만약 자동 시작 여부가 참이라면 위 그림과 같이 시작 시간(Start time)과 주기(Cycle time)를 명시해야 한다. 알람은 시작 시간 후 울리고 다음부터 주기마다 울리게 된다.

3.1.1 상태 전이

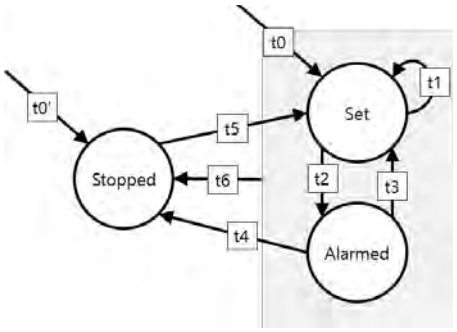

그림 5 알람의 상태 다이어그램

그림 5는 알람의 행위를 상태 다이어그램으로 표현한 것이다. 3.1에서 언급한 알람 명세를 모델에서 표현하기 위해 Set, Alarmed, 그리고 Stopped의 3가지 상태를 제시한다. Set은 알람이 울리기를 기다리는 상태, Alarmed의

표 1 알람의 상태 전이

전이	시작상태	이벤트와 조건	행위	도착상태
t0	-	autostart = TRUE	start = 초기 값, cycle = 초기 값, counter = start	Set
t0'	-	autostart = FALSE	start = 0, cycle = 0	Stopped
t1	Set	counter > 0	counter--	Set
t2	Set	counter = 0	bip = TRUE	Alarmed
t3	Alarmed	cycle > 0	counter = cycle, bip = FALSE	Set
t4	Alarmed	cycle = 0	start = 0, cycle = 0, counter = 0, bip = FALSE	Stopped
t5	Stopped	API = SetAlarm	start = 무작위 값, cycle = 무작위 값	Set
t6	Set 또는 Alarmed	API = CancelAlarm	start = 0, cycle = 0, counter = 0, bip = FALSE	Stopped

경우 알람이 울려서 액션을 취할 상태, 마지막으로 Stopped는 알람이 정지해서 작동하지 않는 상태를 의미한다. 또한 t0부터 t6까지 이들 간의 상태 전이가 존재한다.

표 1은 t0부터 t6까지의 전이를 자세하게 설명했다. t0 전이의 경우, start와 cycle이 환경설정에 명시된 값을 토대로 초기 값이 설정되게 된다. t2 전이에서 bip의 경우 알람이 울렸음을 표시하는 것이고 t5에서는 start와 cycle에 무작위 값을 전달하였다.

3.1.2 시작 시간과 주기의 크기 조정

환경설정의 시작 시간과 주기는 실제 시간인데 반해, 운영체제 모델에서 시작 시간과 주기는 상대적 시간 기반이다. 따라서 환경설정의 알람 정보를 모델에 적용하기 위해 실제 시간에서 상대적 시간으로 크기 조정(Scaling)이 필요하다. 본 연구에서는 밀리세컨드 단위의 환경설정 속 시작 시간과 주기를 100분의 1로 환산해 상태의 스텝 개수로 표현했다. 예를 들어 시작 시간이 1000ms이고 주기가 500ms라면, NuSMV에서 검증 시에는 10개 스텝 후 처음 알람이 울리고 그 후 5개 스텝마다 알람이 울린다.

3.1.3 알람 모델 생성

```
MODULE main
    alarm : Alarm(env, 0, TRUE, 10, 5, SE, 1, 0);

MODULE Alarm(env, aid, autostrt, st, cy, api, tid, eid)
    VAR
    state : {STOPPED, SET, ALARMED};
    bip : boolean;
    counter : 0..10;
    start : 0..10;
    cycle : 0..10;

    DEFINE
    aapi := api;
    atid := tid;
    aeid := eid;
                ⋮
```

그림 6 알람 모델

그림 6는 3.1에서 제시한 그림 4의 알람 명세를 토대로 제작한 알람 모델의 일부이다. 메인 모듈에서 환경설정을 토대로 한 알람에 대한 정보들을 알람 모듈로 전달하는데, 시작 시간과 주기가 크기 조정되어 전달된다. 예를 들어서 OIL에서의 ALARMTIME와 CYCLETIME은 크기 조정을 통해 모델의 start와 cycle로 바뀐다. bip의 경우는 알람이 울렸을 때 확인하기 위한 변수이다. DEFINE에 있는 aapi는 알람의 액션을, atid와 aeid는 액션의 태스크와 이벤트의 대상을 나타낸다. 이 밖에도 알람 모델에는 t0부터 t6까지의 전이와 그에 따른 변수의 변화가 있다. 예를 들어 t3 전이의 경우는 다음과 같다.

next(state) := state = ALARMED & cycle > 0 : SET;

개선된 모델에서는 알람의 bip에 따라 알람의 액션을 수행할지 원래대로 무작위 API를 생성할지 판단하고, 마찬가지로 알람의 액션의 대상을 택할지 원래대로 대상 태스크나 이벤트를 무작위로 선택할지 판단한다.

3.2 테스트 케이스 생성

앞서 환경설정과 제약사항을 통해 만든 운영체제 모델과 반례 생성 속성을 NuSMV로 검증을 하게 되면, 해당 속성의 반례를 결과로 얻을 수 있다.

$G\ (S_0 \rightarrow !F\ S_i)$

위는 반례 생성 속성의 일반적인 형태로 LTL(Linear temporal logic) 식으로 적게 된다. G는 조건이 항상 만족해야 한다는 의미이고 F는 조건이 언젠가 만족해야 한다는 의미이다. 위의 식은 S_0 상태에서 S_i 상태까지 도달할 수 없음을 의미한다. 따라서 검증을 통해 얻을 수 있는 반례는 S_0 상태에서 S_i 상태까지 도달한 경우를 의미한다.

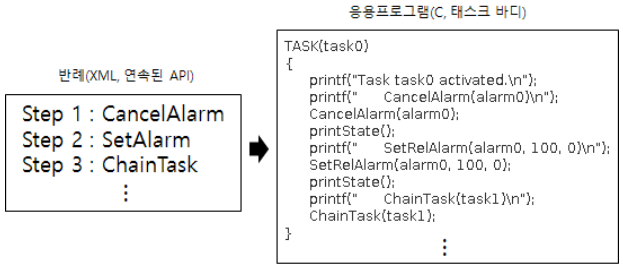

그림 7 반례를 이용한 테스트 케이스 생성

그림 7은 반례를 통해 연속된 API를 추출한 후[4] 생성된 테스트 케이스이다. 테스트 케이스는 OSEK/VDX 기반 운영체제 중 하나인 Trampoline용으로 생성했다.[2]

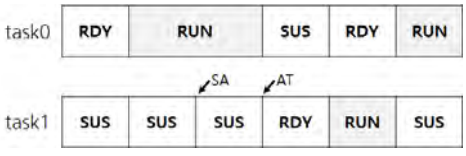

그림 8 테스트 케이스의 태스크 시뮬레이션

그림 8은 그림 7의 태스크 바디를 포함한 테스트 케이스의 태스크 시뮬레이션 결과이다.[5] 자동 시작하지 않게 설정된 task1이 일회성 알람으로 인해 주기적 태스크가 되어 Running 상태로 전이된 것을 확인할 수 있다.

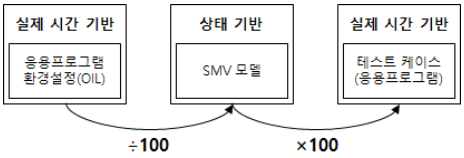

그림 9 시작 시간과 주기의 크기 조정

그림 9는 본 연구에서 알람의 시작 시간과 주기를 어떻게 크기 조정했는지에 대한 전반적인 흐름에 대한 그림이다. 3.1.2에서는 실제 시간 기반 값을 상태 기반 값으로 바꿀 때의 괴리로 시작 시간과 주기를 100분의 1로 줄였다. 반대로 테스트 케이스를 생성할 때는 반대로 상태 기반 값을 실제 시간 기반 값으로 바꿔야하므로 다시 복구시키기 위해서 100배 늘려준다.

4. 실험 및 평가

본 연구에서는 두 가지 실험을 통해 알람이 추가된 SMV 모델의 유효성 검사와 성능 측정을 했다. 실험은 본 연구에서 개선한 내용을 제외하고는 최대한 이전 도구와 현재 도구가 동일한 환경에서 진행되도록 하였다. 태스크 3개, 리소스 1개, 이벤트 1개, 그리고 개선된 도구의 경우 알람 1개를 추가한 환경에서 진행되었다. 총 5가지의 제약사항을 사용해서 이전도구와 현재도구 모두 실험을 진행했다. 모델 검증 기법으로는 LTL 방식으로 모델 검증하였다. 실험에는 Linux Fedora 24에서 Intel Xeon E5-1680 v4 3.40GHz와 64GB의 메모리로 진행하였다.

첫 번째 실험에서는 선행 연구에서 개발한 테스트 케이스 생성기에 알람관련 환경을 추가하여 추가하기 전과 비교해서 같은 결과가 도출되어서 알람의 추가가 다른 부분에 영향을 주지 않는지 검사해보았다.

두 번째 실험에서는 선행 연구의 기존 도구와 알람을 추가한 개선된 도구의 테스트 케이스를 생성하는데 소요되는 시간과 메모리 사용량을 비교해보았다.

표 2 실험 결과

	결과 개수		소요 시간		메모리 사용량	
	이전 도구	현재 도구	이전 도구	현재 도구	이전 도구	현재 도구
제약사항1	30개	30개	12초	5분18초	61.8MB	89.5MB
제약사항2	30개	30개	10초	14분4초	61.5MB	84.4MB
제약사항3	19개	19개	4초	1분16초	44.3MB	70.7MB
제약사항4	20개	20개	3초	32초	28.8MB	63.8MB
제약사항5	20개	20개	3초	40초	29.6MB	66.4MB

표 2는 실험의 결과이다. 동일한 제약사항으로 이전 도구와 현재 도구를 비교했다. 이전의 도구와 알람이 추가된 현재 도구의 테스트 케이스 결과 개수가 동일하게 도출됐다. 이는 검증 시에 이전 도구에서 도달 가능한 상태와 도달 불가능한 상태가 현재 도구에서도 도달 가능성이 유지됨을 의미한다. 따라서 본 연구의 운영체제 모델에서의 알람이 정상적으로 작동하여 다른 객체에 영향을 끼치지 않고 주기적 태스크를 검증할 수 있었음을 알 수 있었다.

소요 시간에 있어서는 이전 도구와 현재 도구가 비교적 큰 차이를 보였다. 본 연구에서는 상태 기계를 기반으로 한 NuSMV의 검증을 위한 운영체제 모델에 시간 속성을 추가함으로써 많은 상태가 상태 기계에 추가된다. 따라서 이러한 차이는 실제 시간을 기반으로 한 알람을 상태 기반으로 옮기면서 초래된 것으로 보인다.

메모리 사용량의 경우 시간 속성을 추가하였음에도 상태 기반 검증의 특성상 소요 시간에 비하여 소폭 증가하였음을 확인할 수 있다.

5. 관련 연구 및 결론

선행 연구[4]는 태스크를 중점적으로 효율적인 테스트 케이스를 생성하고 시뮬레이션 하는 것이 주요 관심사였다. 또한 참고문헌[6]은 본 연구와 같이 주기적 태스크를 정형화했고 이를 이용해서 실시간 시스템의 태스크가 주기적으로 활성화되는지 확인하였다. 하지만 본 연구와 달리 주기적 태스크의 시작시간을 정형화하지 않았고 주기적 태스크가 설정되고 해지되는 경우를 고려하지 않았다.

본 연구에서는 OSEK/VDX 운영체제 기반 차량 전장용 운영체제의 주기적 태스크 검증을 위해서 알람 모델을 추가하여 주기적 태스크를 다룰 수 있는 테스트 케이스를 생성하는 도구를 개발했다. 따라서 전보다 다양한 테스트 케이스를 생성할 수 있게 되었고 차량 전장용 운영체제의 안전성을 보다 엄밀하게 검증할 수 있게 되었다.

선행 연구에 따르면 이전 도구로 Trampoline[2]의 안전성 검증에 적용하여 평균 82%의 커버리지를 보였는데 개선된 도구로 Trampoline에 적용하여 커버리지를 측정해 볼 계획이다. 또한 시작 시간과 주기의 크기 조정에 있어서 좀 더 개선된 방법을 적용하고 이를 증명할 예정이다.

참고문헌

[1] OSEK Group. “OSEK/VDX operating system specification 2.2.3”, 2005

[2] J.-L. Bechennec, M. Briday, S. Faucou, and Y. Trinquet. “Trampoline : an opensource implementation of the OSEK/VDX RTOS specification”, 2006

[3] NuSMV http://nusmv.fbk.eu/

[4] Taejoon Byun and Yunja Choi: “Automated System-level Safety Testing Using Constraint Patterns for Automotive Operating Systems”, ACM Symposium on Applied Computing, 2015.

[5] 이수경, 김동우, 최윤자 “차량 전장용 제어 소프트웨어 응용프로그램의 검증을 위한 모델 기반 Task Simulation 도구”, 추계학술발표대회 논문집 제22권 제2호, 2015.

[6] Vahid Garousim, A Formalism for arrival time analysis of real-time tasks based on UML models, 2008.