특집 06

차량전장용 운영체제 검증 사례를 통한 소프트웨어 안전성 검증 기법 소개

최윤자 · 변태준 (경북대학교)

목 차 »

1. 서 론

지난 20세기는 기계와 전자, 전기의 발전을 토대로 산업사회가 급격히 성장한 하드웨어 중심의 시대였던 반면, 21세기는 이 하드웨어에 지능을 더해 자율과 창의가 상품의 핵심 요소로 작용하는 소프트웨어의 시대이다. 항공, 교통, 차량, 원자력, 통신, 국방, 의료 등 주요 사회기반시설이 모두 이러한 소프트웨어에 의해 제어되고 있을 뿐 아니라, 세금, 병역, 법률 등 관공서의 주요 업무들 역시 소프트웨어 시스템의 도움 없이는 수행될 수 없으며, 은행, 증권, 보험 등 핵심 금융 산업 또한 소프트웨어 시스템의 정확성과 신뢰성에 100% 의존할 수밖에 없다. 특히, 항공, 교통, 차량, 원자력, 국방 등 사회간접시설의 제어를 담당하는 소프트웨어 시스템은 문제가 발생할 시 막대한 인명과 재산의 피해를 초래할 수 있으므로, 시스템 개발 전과 개발 중, 그리고 개발 후의 전 과정에 걸쳐 철저한 안전성 분석과 검증이 수반되어야 한다.

소프트웨어의 안전성은 여러 면에서 하드웨어 안전성과는 다른 특성을 지니고 있다. 첫째, 소프트웨어는 눈에 보이지 않으므로 결함을 눈으로 확인하기 어렵고, 물리적인 측정방법으로는 안전성을 평가할 수 없다. 둘째, 소프트웨어가 정의하는 제어 행위들은 하드웨어에 비해 그 복잡도가 비교할 수 없이 높아 설계자나 개발자가 예측하지 못한 돌발 행위가 발생할 가능성을 내포하고 있으며, 이러한 요소들을 찾아내는 것은 매우 어렵다. 또한, 소프트웨어의 잘못된 제어논리에 의해 발생한 사고는 대다수의 경우 재연이 매우 어려우며, 이에 따라 원인 파악 및 위험요소제거가 제때에 이루어지기 힘들다. 이러한 사고의 대표적인 예로 최근의 자동차관련 사고 및 대규모 리콜사태를 들 수 있는데[1], 특히 2009년 이후 세계적인 관심을 받고 있는 토요타 렉서스의 급발진 의심 사고는 제어 소프트웨어의 결함에서 기인되었을 가능성이 높은 것으로 보고되고 있다[2].

본 논문에서는 소프트웨어 안전성의 확보가 주

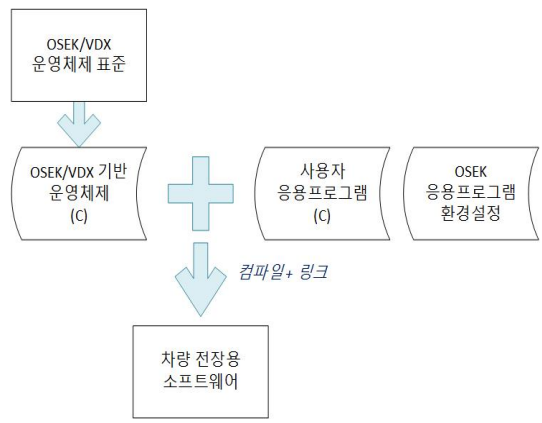
요 기간시스템의 안전을 위한 기초단계라는 인식을 확산하고자, 소프트웨어 안전성 분석 및 검증 기법을 차량전장용 운영체제를 검증한 사례 중심으로 소개하고자 한다. 업계에서는 차량 전장용 운영체제의 표준화를 위해 OSEK/VDX[3], AUTOSAR[4] 등의 국제 표준을 제정하였으며, 전 세계의 자동차 생산업체가 이를 준수하도록 협약을 채결하였다. 본문에서는 이 중 OSEK/VDX를 기반으로 한 운영체제를 대상으로 모델검증기법[5]을 적용하여 안전성 문제를 식별하는 과정을 소개한다. 소개된 검증기법들은 OSEK/VDX 기반 운영체제 뿐 아니라 AUTOSAR OS 에도 적용가능하며, 이들 국제표준을 따르는 운영체제의 안전성 검증에 공통적으로 적용될 수 있다. 소개되는 사례는 참고문헌[6]에서 발췌되었다.

2. OSEK/VDX 기반 전장용 운영체제

2.1 차량 전장용 운영체제

한 대의 차량에는 많게는 100여개의 작은 계산기기가 탑재되며, 차량 시스템은 이들 간의 상호작용을 통해 제어된다. 이러한 계산기기를 ECU(Electrical Control Unit)라 하며, 각각의 ECU에는 응용프로그램과 하드웨어기기 간의 중계역할을 하는 운영체제가 탑재되어 있다. 전 세계의 자동차 업계와 부품업체들은 이러한 운영체제를 표준화하고 규격화하였으며, 같은 표준을 준수하여 생산된 부품 간에는 호환이 가능하다. OSEK/VDX에서 정의한 운영체제 표준은 AUTOSAR 컨소시엄에 의해서도 채택되었으며, 이는 현재 가장 널리 사용되고 있는 국제표준이다.

(그림 1)은 OSEK/VDX 기반 운영체제의 구성요소와 생성과정에 대한 도식이다. 먼저 OSEK/VDX 기반 운영체제는 OSEK/VDX의 운영체제



(그림 1) 차량전장용 소프트웨어의 생성

표준에 따라 C언어로 작성되며, 전장용 소프트웨어의 코드는 OSEK/VDX에서 정의하는 문법구조와 시스템함수를 이용하여 C코드로 작성된다. 차량 전장용 소프트웨어는 메모리의 동적 할당이나 자원(Resource), 작업(Task) 등 시스템 구성요소의 동적 생성을 허용하지 않으며, 이러한 특질들은 “OSEK 응용프로그램 환경설정”에 정적으로 기술된다. 운영체제의 소스코드와 응용프로그램의 소스코드, 그리고 환경설정은 시스템 생성 단계에서 함께 컴파일과 링크되어 최종적으로 차량 전장용 소프트웨어가 생성된다.

2.2 ISO 26262

차량 전장용 시스템은 자동차 기능 안전성 국제표준인 ISO 26262[7]를 준수하여 개발되어야 하며, 이는 일반적인 전장용 장비 개발 전 과정에 적용되어야 할 기능적 안전성 표준을 제시하고 있다. 이 표준은 특히 차량 전장용 시스템을 개발할 시 요구사항 정의 단계에서부터 코드 개발에 이르기까지, 소프트웨어 생명주기 전반에 걸쳐 안전성 보장기법을 적용할 것을 권장하고 있으며, 차량 안전생명주기와 차량 위험수준 분석기법 및 수용 가능한 위험수준의 측정방법 등을 구체적으로 제시하고 있다[8].

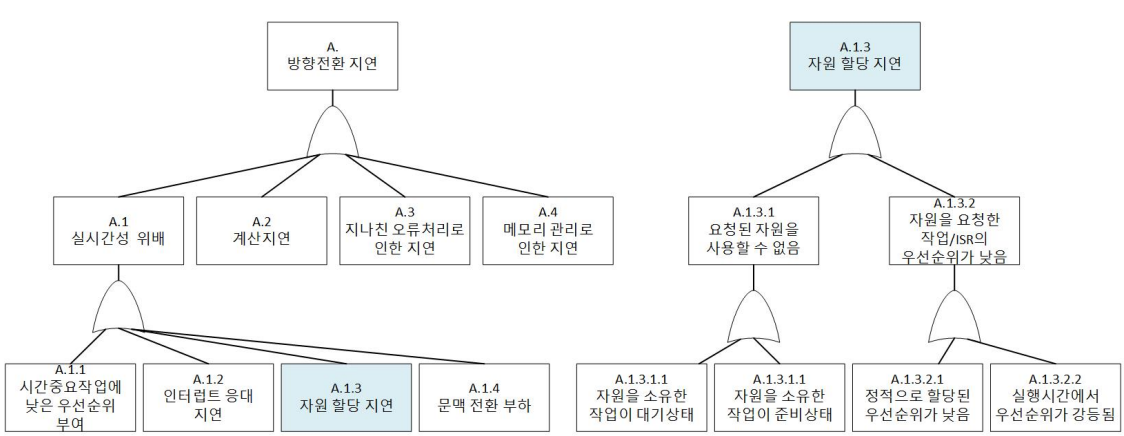
특히, 고위험군 소프트웨어의 단계별 산출물에 대한 핵심평가기법들을 권고하고 있는데, 대표적 기법들로는 정형검증, 제어흐름분석, 요구명세기반 테스트, 안전요건 테스트 등을 들 수 있다. 본 논문에서는 ISO 26262에서 권장하고 있는 안전성 검증 수행 프로세스와 기법들 중, 정형검증기법에 해당하는 모델검증기법을 차량전장용 운영체제에 적용한 사례를 중심으로 소개하고자 한다.

3. 안전성 검증 기법 및 프로세스

소프트웨어의 안전성을 향상시키기 위한 기술요소들에는 소프트웨어 산출물(명세서, 디자인, 프로그램 코드 등) 검토에서부터, FTA, FMEA, Safety Case와 같은 시스템 요구분석 단계에서 이루어지는 안전요건 도출 기법, 코딩 단계에서 적용되는 방어적 프로그래밍 기법, 도출된 안전요건의 충족여부를 정형적으로 검증하는 모델검증기법 등이 존재한다. 본 연구에서는 이러한 기술요소들 중, 오류나무분석 기법을 통하여 안전성 요건을 도출하고, 도출된 안전성 요건들을 모델검증 기법으로 검증하는 과정을 소개한다.

3.1 오류나무분석(Fault Tree Analysis)을 이용한 안전성 저해요소 파악

오류나무분석(Fault Tree Analysis)[9] 기법은 시스템의 오작동 원인을 하위수준의 오작동 원인들의 부울식으로 분석해 나가는 하향식 분석기법이다. (그림 2)는 차량의 운행 중 발생 가능한 시스템 안전성 저해요소인 “방향전환 지연”을 발생시킬 수 있는 소프트웨어적 위험요소들을 분석한 오류나무의 일부이다. “방향전환 지연”은 “실시간성 위배”, “계산지연”, “지나친 오류처리로 인한 지연”, 또는 “메모리 관리로 인한 지연” 등으로 나뉠 수 있으며, 그 중 “실시간성 위배” 는 또다시 “시간중요작업에 낮은 우선순위부여”, “인터럽트 응대지연”, “자원할당 지연” 또는 “문맥전환 부하” 등으로 그 원인들을 세분화 될 수 있다. (그림 2)의 오른쪽 그림은 왼쪽에서 하향식으로 분석해낸 저해요소들 중 “A.1.3” 에 해당하는 저해요소를 세분화한 결과이다. 이와 같이, 주어진 시스템 수준의 안전성 저해요소들을 하향식으로 분석하여 하위수준의 상세한 안전성 저해요소들을 도출하는 것이 소프트웨어 안전성 분석의 첫 번째 단계이다.

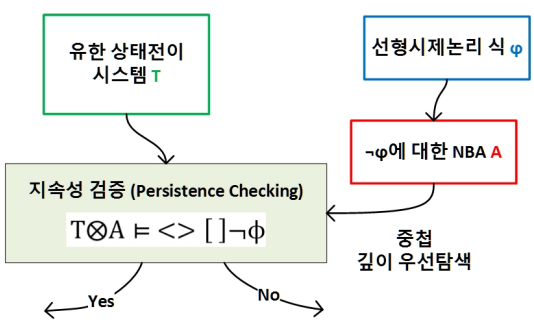


(그림 2) 차량용 시스템을 위한 오류나무분석 사례

3.2 모델검증

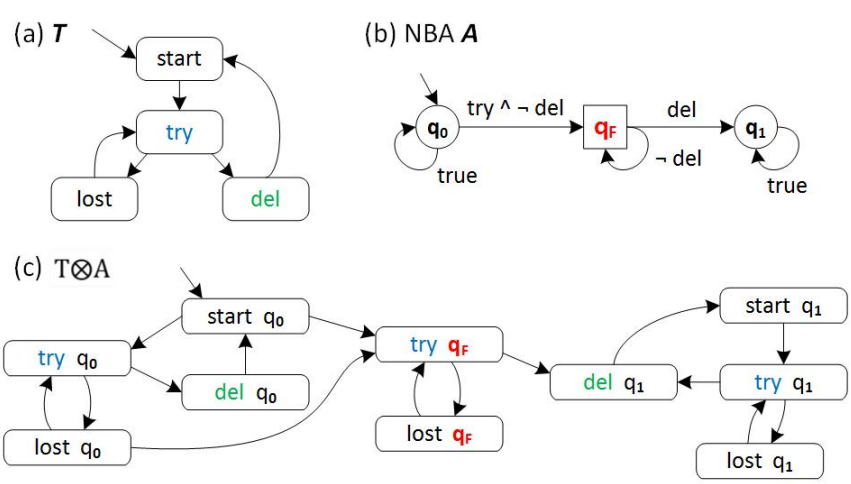
모델검증(Model Checking)[10] 기법은 검증대상 시스템을 상태전이 시스템의 형식으로 정형명세한 후, 오류나무분석 등과 같은 분석기법을 통해 도출된 안전성 저해요소들이 검증대상시스템에 발현되지 않음을 상태 도달가능성 분석(Reachability Analysis)을 통해 증명하는 기법이다.

(그림 3)은 모델검증 기법 중 선형시제논리(Linear-time Temporal Logic, LTL)를 사용한 모델검증의 적용방식에 대한 도식이다. 먼저, 검증대상 소프트웨어는 유한 상태전이 시스템 T로 모델링되고, 안전성 특질은 선형시제논리 ϕ로 정의된다. ϕ의 논리역인 "*not* ϕ"는 시스템에서 발생해서는 안 되는 "불량행위"를 나타내게 되며, 이는 비결정적 부키 오토마타(Nondeterministic Büchi Automata) A의 형태로 표현된다. 모델검증기는 유한 상태전이 시스템 T와 오토마타 A의 곱이 최종상태에 도달하지 않음을 중첩깊이우선 탐색(nested DFS) 기법을 사용하여 보임으로써, 시스템 T가 안전성 특질 ϕ를 만족함을 증명한다.



(그림 3) 모델검증기법

(그림 4)는 간단한 메일전송시스템 모델에서 만족해야할 시스템 특성을 모델검증기법으로 검증하는 방법을 도식화하고 있다. 먼저, 메일전송시스템의 간략한 모델이 상태전이 시스템의 형식으로 정의되었다 (그림 4 (a)). 이 시스템은 메일이 전달될 때까지(*del*) 전송을 계속하는(*try*) 방식으로 모델링되어있다. 이 시스템을 대상으로 검증하고자하는 성질 ϕ가 "메시지가 반복적으로 전송되면 언젠가는 전달되어야한다" 라고 하면, 이 성질의 역을 표현하는 Büchi Automata A 를 (그림 4) (b)와 같이 표현할 수 있다. (그림 4) (c)는 T와 A의 곱의 결과이며, 불안전상태인 q_F 에 무수히 도달 가능하므로, 이 시스템 T 는 성질 ϕ를 만족하지 않는다. 이때 모델검증기는 성질 ϕ를 만족하지 않는 반례를 도출하는데,



(그림 4) 모델검증의 예

이는 초기상태에서 출발하여 불안전 상태에 무수히 도달하는 경로 중 하나이며, (그림 4) (c)의 예에서는 $(start, q_0) \rightarrow (try, q_0) \rightarrow (lost, q_0) \rightarrow ((try, q_F) \rightarrow (lost, q_F))^*$ 이 하나의 반례라 할 수 있다.

모델검증기법은 이론적 완성도와 실질적 파급효과를 인정받아, 그 창시자들이 2008년 튜링상을 수상하였으며, 앞서 소개된 검증과정이 전 자동화되어 다양한 도구들이 개발되어있다[11,12].

4. 검증사례: Trampoline 운영체제

본 장에서는 3장에서 소개한 안전성 검증 기법을 차량전장용 운영체제 Trampoline[13]에 적용한 과정과 결과를 소개한다. Trampoline 은 C 언어로 작성된 OSEK/VDX 기반 운영체제이며, 초기버전은 국제인증기관에서 인증을 받았던 만큼, 어느 정도 품질을 갖춘 공개소프트웨어이다. Trampoline의 커널은 174개의 함수와 4530 라인의 코드로 구성되어 있으며, 이는 동작 환경을 모방(emulate)하는 코드와 하드웨어 종속적인 코드를 포함하고 있다. Trampoline은 이 하드웨어 모방 코드를 통해 POSIX 환경을 지원함으로써, 복잡한 하드웨어 이식 과정을 거치지 않고도 간편하게 실행시간 테스트를 수행할 수 있다.

4.1 오류나무 분석을 통한 안전 요소 도출

먼저, 검증하고자하는 안전성 특질들을 식별해내기 위하여 (그림 1)의 오류나무분석을 통해 56개의 운영체제 관련 안전 저해요소들을 도출하였다[14]. 다음은 식별된 안전 저해요소들을 기반으로 정의한 안전성 특질들의 몇 가지 사례들이다.

SR1. 작업들과 인터럽트 서비스 루틴들은 자원을 점유한 상태로 종료되어서는 안 된다.
SR2. 작업은 자원을 점유한 채로 이벤트를 대기해서는 안 된다.
SR3. 작업은 이벤트를 무한정 기다려서는 안 된다.
SR4. 활성화된 작업은 반드시 수행되어야 한다.
SR5. 정적 우선순위가 높은 작업은 정적 우선순위가 낮은 작업보다 항상 먼저 수행되어야 한다.

<표 1> 은 이러한 안전성 특질들은 Trampoline 코드의 자료구조를 참조하여 선형시제논리로 변환한 결과들이다. 예를 들어, SR1 은 "*TerminateTask* 또는 *ChainTask* 가 호출되는 시점에서 해당 작업이 리소스를 점유하고 있으면 오류상태에 다다른다." 는 안전성 요건을 시제논리로 표현한 것이다. 여기

〈표 1〉 안전성 특질의 시제논리표현

SR	정형명세	검증결과
SR1	[] ((TerminateTask || ChainTask) && taskhasResource → 〈〉 (error))	성공
SR2	[] (WaitEvent && taskhasResource → 〈〉 (error))	성공
SR3	[] ((wait_id == i) → 〈〉 (tpl_kern.running_id == i))	실패
SR4	[] (dyn_proc_table[i].state == ready → 〈〉 (tpl_kern.running_id == i))	실패
SR5	[] ((stat_proc_table[i].priority 〈 stat_proc_table[j].priority && stat_proc_table[j].priority 〉 max_ceiling_priority && dyn_proc_table[i].state == ready && dyn_proc_table[j].state == ready) → dyn_proc_table[j].state != running U (dyn_proc_table[j].state == (running || error)))	성공

서 [ ] q 는 모든 실행 경로에 대하여 논리식 q가 참이라는 의미이고, <> q는 언젠가는 q가 참이 된다는 의미이며, p U q는 q가 참인 상태 이전의 모든 상태에서 p는 참이라는 의미이다.

4.2 모델검증

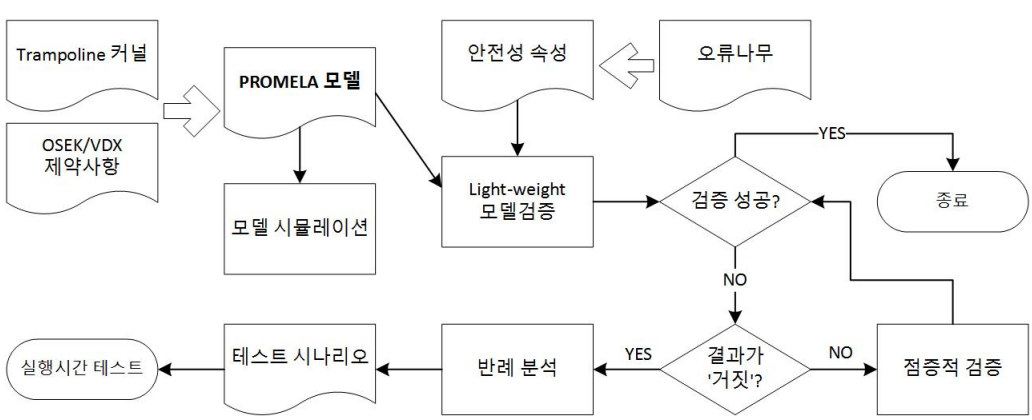
모델검증기법을 통해 Trampoline을 검증하기 위해서는 검증대상인 Trampoline 커널과 이에 대한 외부적 환경에 해당하는 작업 모형을 정형적으로 명세해야한다. 작업 모형이란 Trampoline 커널과 상호작용하는 작업(Task)에 대한 모형으로써, OSEK/VDX에 명시된 제약조건하에서 커널의 시스템함수를 무작위로 호출한다. 본 연구에서는 모델검증기 SPIN[11]을 사용하였으며, 검증 대상인 Trampoline커널과 작업 모형은 SPIN에서 요구하는 정형명세언어인 PROMELA로 명세하였다.

(그림 5)는 모델검증기법으로 Trampoline을 검증하는 절차에 대한 도식이다. 첫 번째 단계는 Trampoline 커널과 작업 모형을 PROMELA로 명세하는 과정이며, 이는 정해진 규칙[6]에 따라 수동으로 이루어진다. 생성된 모델은 SPIN의 시뮬레이션 기능을 통해 일차적 검증(validation) 과정을 거친 후, 오류나무 분석을 통해 도출된 안전성 속성을 모델검증하는데 쓰인다. 검증 과정은 시간과 자원의 제약을 점증적으로 완화해가며 반복 진행되며, 이는 검증이 성공하거나 반례가 도출될 때 까지 반복된다. 검증 결과가 거짓으로 판명되었을 경우 모델검증기는 그 반례를 하나 도출하며, 이 반례를 분석해 오류를 유발하는 시스템 함수 호출 순서를 유추할 수 있다. 끝으로, 반례로부터 유추된 시스템 함수 호출 순서는 Trampoline 테스트 프로그램으로 작성되어 실제 Trampoline상에서 오류를 일으키는지 테스트한다.

4.3 식별된 안전 저해 요소

<표 1>의 안전성 속성에 대한 검증을 실시한 결과 SR3에 대한 검증 결과가 거짓으로 판명되었으며, 다음과 같은 시나리오가 도출되었다.

1) 자동 실행 작업이면서 우선순위가 1인 작업 t_1이 시스템 시작과 함께 시작되어 자원 1을 점유하고 t_2를 실행시킨 후 자원 1을 해제하고 종료한다.
2) 우선순위가 5인 작업 t_2는 t_1을 선점하며 시작되고, 작업 t_3를 실행시킨 다음 이벤트 2를 기다리다 종료한다.

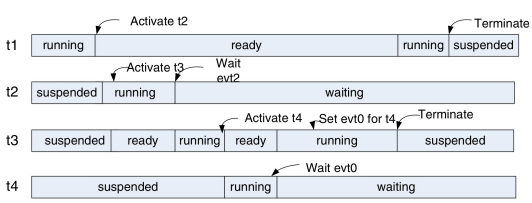


(그림 5) Trampoline 커널코드의 안전성 분석을 위한 모델검증 적용 프로세스

3) 우선순위가 2인 작업 t_3는 t_4를 실행시키고 t_4에 이벤트 0을 보낸 다음 종료한다.
4) 우선순위가 4인 작업 t_4는 이벤트 0을 기다리고 t_2에 이벤트 2를 보낸 다음 종료한다.

위 시나리오가 정상적으로 수행될 경우 모든 작업은 정상적으로 종료되어야하지만 모델검증 결과(그림 6)과 같이 두 작업 t_2와 t_4가 이벤트를 무한정 기다리며 작업을 종료하지 않음으로써 "작업은 이벤트를 무한정 기대려서는 안 된다"는 안전성 속성 SR3을 위배하였다. 이는 t_3이 이벤트 0를 t_4에 보냈지만 이벤트가 정상적으로 전달되지 않아 t_4가 무한대기상태에서 벗어나지 못하며, 이에 따라 t_4 역시 t_2에 이벤트를 보낼 수 없게 되어 발생한 반례이다.

<표 2>의 코드는 위 반례를 분석하여 오류가 발생하는 원인인 코드결함을 식별한 것이다. 위 코드는 작업 t_3에서 t_4에 이벤트 0을 보낼 경우 호출되는 코드의 일부이며, 이벤트를 받는 작업 t_4의 식별자와 보내고자 하는 이벤트 0이 위 함수의 매개변수로 전달된다. 이 때 3번 줄에서는 이벤트 0이 t_4가 기다리고 있던 이벤트가 맞는지를 확인하고자 비트단위 논리곱 연산을 수행하는데, 0과의 논리연산 결과는 언제나 0이 되어 해당 조건문을 통과하지 못한다. 그 결과 해당 이벤트를 대기 중이던 작업을 깨우는 루틴이 수행되지 못하게 되고 따라서 작업이 무한대기상태에 빠지는 오류가 발생하게 된다.



(그림 6) 안전요건 SR3를 위배하는 반례

〈표 2〉 반례의 원인이 되는 코드결함

```
1: tpl_status tpl_set_event(tpl_task_id task_id, tpl_event_ma
   sk in_event){
2:     ....
3:     if((events->evt_wait & in_event) != 0){
4:       .... // 대기(waiting)상태인 작업을 깨워 준비상태 큐에 등록
5:         ....
6:     }
7:     ....
9: }
```

5. 결 론

이상 자동화된 정형검증기법의 하나인 모델검증기법을 차량전장용 운영체제의 안전성 검증에 적용한 사례를 소개하였다. 모델검증기법은 기존에 주로 사용되어 온 동적 테스트 기법에 비해 매우 엄밀한 검증기법이며, 다양한 안전요건을 시제논리로 표현함으로써 자동화된 검증을 수행할 수 있다. 소개된 기법과 적용방식은 전장용 운영체제의 검증뿐 아니라 차량용 응용시스템의 검증 및 여타 내장형 소프트웨어의 검증에도 유사한 방식으로 적용될 수 있다.

참 고 문 헌

[1] "Toyota Announces Voluntary Recall of Certain Toyota Prius, RAV4, Tacoma and Lexus RX 350 Vehicles" : http://toyotanews.pressroom.toyota.com/releases/toyota+voluntary+recall+021214.htm
[2] Michael Barr, Bookout V. Toyota, http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf
[3] "OSEK/VDX operating system specification 2.2.3," http://portal.osekvdx.org/files/pdf/specs/os223.pdf.
[4] "AUTomotive Open Source ARchitecture," http://www.autosar.org/, http://www.autosar.org/.
[5] Ed Clarke, Orna Grumberg, and Doron Peled,

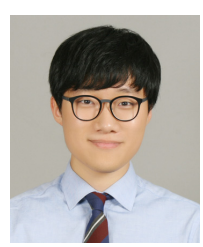
Model Checking, MIT Press 1999.
[6] Yunja Choi. "Model checking Trampoline OS: a case study on safety analysis for automotive software." Software Testing, Verification and Reliability 24.1, 38-60, 2014.
[7] ISO 26262-1:2011(en) Road vehicles — Functional safety — Part 10: Guideline on ISO 26262
[8] 최윤자, 이우진, 자동차 전장용 소프트웨어 검증 동향, 정보과학회지 제31권 제5호, 2013.5, 18-24
[9] Oh Y, Yoo J, Cha S, Son HS. "Software safety analysis of function block diagrams using fault trees" Reliability Engineering and System Safety, 88(3):215–228, 2005.
[10] Clarke, Edmund M., Orna Grumberg, and Doron Peled. Model checking. MIT press, 1999.
[11] Gerard Holzmann, The SPIN Model Checker:Primer and Reference Manual, Addison-Wesley Publishing Company, 2003
[12] NuSMV: A New Symbolic Model Checking, http://nusmv.irst.itc.it
[13] Bechennec, Jean-Luc, et al. "Trampoline an OpenSource Implementation of the OSEK/VDX RTOS Specification", 2006.
[14] 최윤자, 윤성현, 김연준, Lee S. 차량용 실시간 운영체제를 위한 안전성 인증연구, 기술보고서, 한국전자통신연구소, 2009

저 자 약 력

최 윤 자

이메일 : yuchoi76@knu.ac.kr

- 1991년 연세대학교 수학과(학사)
- 1993년 연세대학교 수학과(석사)
- 1993년~1996년 삼성데이타시스템 소프트웨어엔지니어
- 1999년 (미)미네소타대학교 컴퓨터과학과(석사)
- 2003년 (미)미네소타대학교 컴퓨터과학과(박사)
- 2003년~2006년 (독)프라운호퍼 소프트웨어공학연구소 연구원
- 2006년~현재 경북대학교 컴퓨터학부 부교수
- 관심분야 : 소프트웨어공학, 소프트웨어 정형분석, 컴포넌트 소프트웨어, 모델검증



변 태 준

이메일 : bntejn@gmail.com

- 2013년 경북대학교 컴퓨터학부(학사)
- 2013년~현재 경북대학교 컴퓨터학부 석사과정
- 관심분야 : 소프트웨어공학, 모델검증, 명세기반 테스트