OSEK/VDX기반 전장용 운영체제의 효율적인 검증을 위한 특성기반 코드 추출 및 테스트 시나리오 생성기

박민규, 변태준, 최윤자
경북대학교 IT대학 컴퓨터학부
소프트웨어 안전공학 연구실

SSELAB KNU

배경

❖ **OSEK/VDX**

- 차량용 분산 제어장치의 공개 아키텍처에 대한 산업 표준화를 목표로 시작되어 국제적으로 통용되는 차량전장용 운영체제의 표준
- 동적 메모리 할당, 자원의 circular waiting, 멀티 쓰레딩 등의 불필요한 복잡성을 제거함

❖ **Trampoline**

- OSEK/VDX version 2.2.3을 준수하는 오픈소스 실시간 운영체제
- 하드웨어 파트로의 접근은 외부변수와 매크로로 추상화 되어있음
- ***안전 중요(Safety-Critical) 시스템***

연구 배경

—pros —cons

모델 체킹

- 완전 검증이 가능
- 기능적 안정성 분석에 적합

- 더 많은 자원과 도메인에 대한 지식을 요함
- 상대적으로 실용성이 낮음

비결정적 테스트

- 사용하기 비교적 용이함
- 적용하기 용이함
- 상대적으로 저렴한 비용

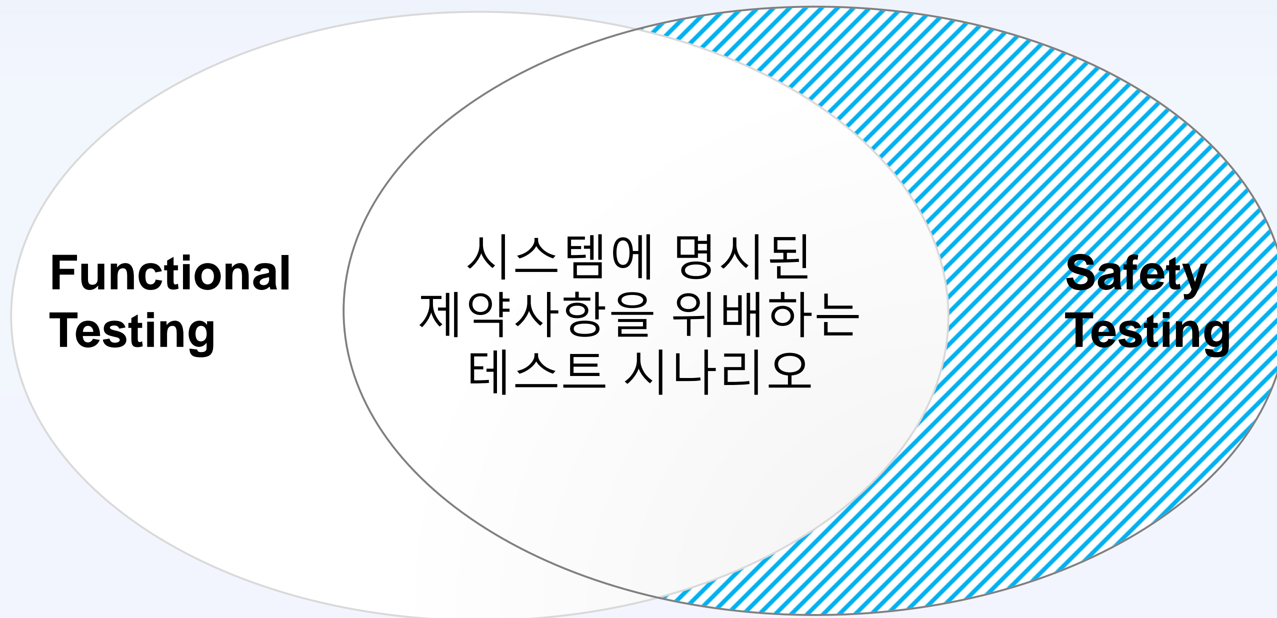
- 잘못된 작동의 부재를 검증(확신)할 수 없음

❖ 비용이 많이 소요되는 정형검증에 대한 보완적 방법으로써 비결정적 테스트를 동시에 수행

❖ 검증의 비용을 줄이기 위해 특성에 기반해 검증 대상을 추출

관련 연구 : *Property-Based Code Slicing for Efficient Verification of OSEK/VDX Operating System by Mingyu Park, Taejoon Byun and Yunja Choi, in the First International Workshop on Formal Techniques for Safety-Critical Systems, 2012*

연구 범위



❖ 본 연구에서는 제약사항을 위배하는(오류를 발생시키는) 시나리오를 최대한 배제함으로써 검증의 효율을 높이고자 하였다.

관련 연구

❖ Environment modeling for efficient model checking

- Concerning environment assumptions in verification
  - ✓ Synchronous Observers and the Verification of Reactive Systems
    by Pascal Raymond, Nicolas Halbwachs and Fabienne Lagnier, AMAST, 1993
- Specification-based environment generation
  - ✓ Partial Verification of Software Components: Heuristics for Environment Construction
    by Pavel Parizek and Frantisek Plasil,
    17th International SPIN Conference on Software Model Checking, 2007

❖ Program slicing

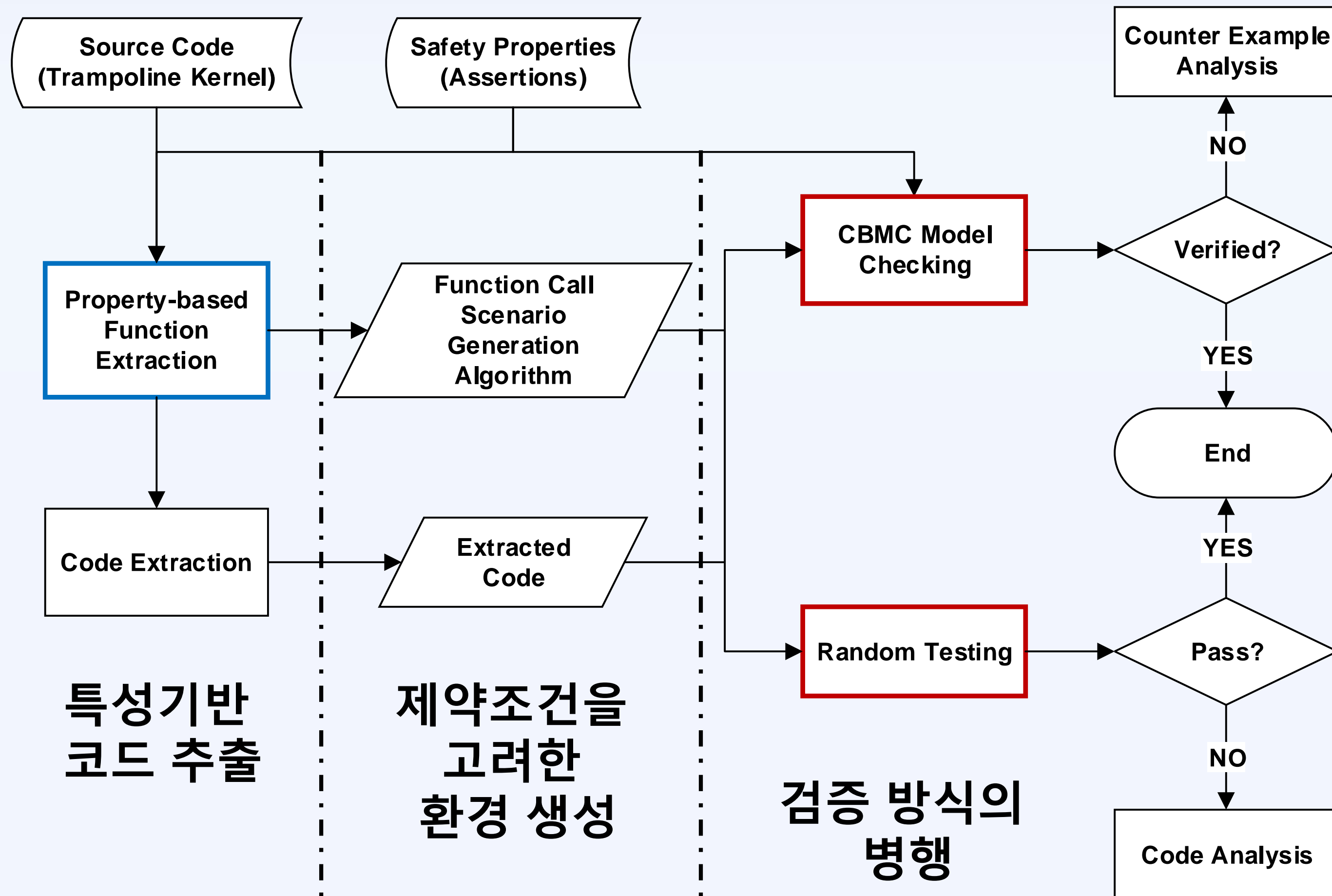
- Slicing algorithms to explicitly detect def-use associations for efficient regression testing.
  - The Application of Program Slicing to Regression Testing
    by David Binkley, Information and Software Technology, 1999
- program slicing for C programs with respect to the alarms generated from value analysis.
  - Program slicing enhances a verification technique combining static and dynamic analysis
    by Omar Chebaro, Nikolai Kosmatov, Alain Giorgetti and Jacques Julliand,
    Proceedings of the 27th Annual ACM Symposium on Applied Computing, 2012

검증 방식의 병행에 따른 중요 사안

1. **상태폭발 문제**를 피하기 위해 검증 대상의 크기를 줄여야 한다.

2. 내장형 소프트웨어의 검증을 위한 환경의 모델링
   - 시스템과의 모든 상호작용을 충분히 반영하는 환경을 어떻게 모델링 할 것인가?
   - 어떻게 하면 시스템 명세에 명시되어있는 제약조건을 준수하는 테스트 시나리오를 생성 할 것인가?

검증 방식 병행의 개요



특성과 연관된 변수 및 함수

```
void WaitEvent(){
     tpl_get_proc();
}
```

Root Level Function
: API that is a terminal node of the called-by graph of an End Level Function

End Level Function
: Function that directly modifies, sets, or uses Extended Target Variable

```
void SetEvent(){
     tpl_put_new_proc();
     tpl_schedule_from_running();
}
```

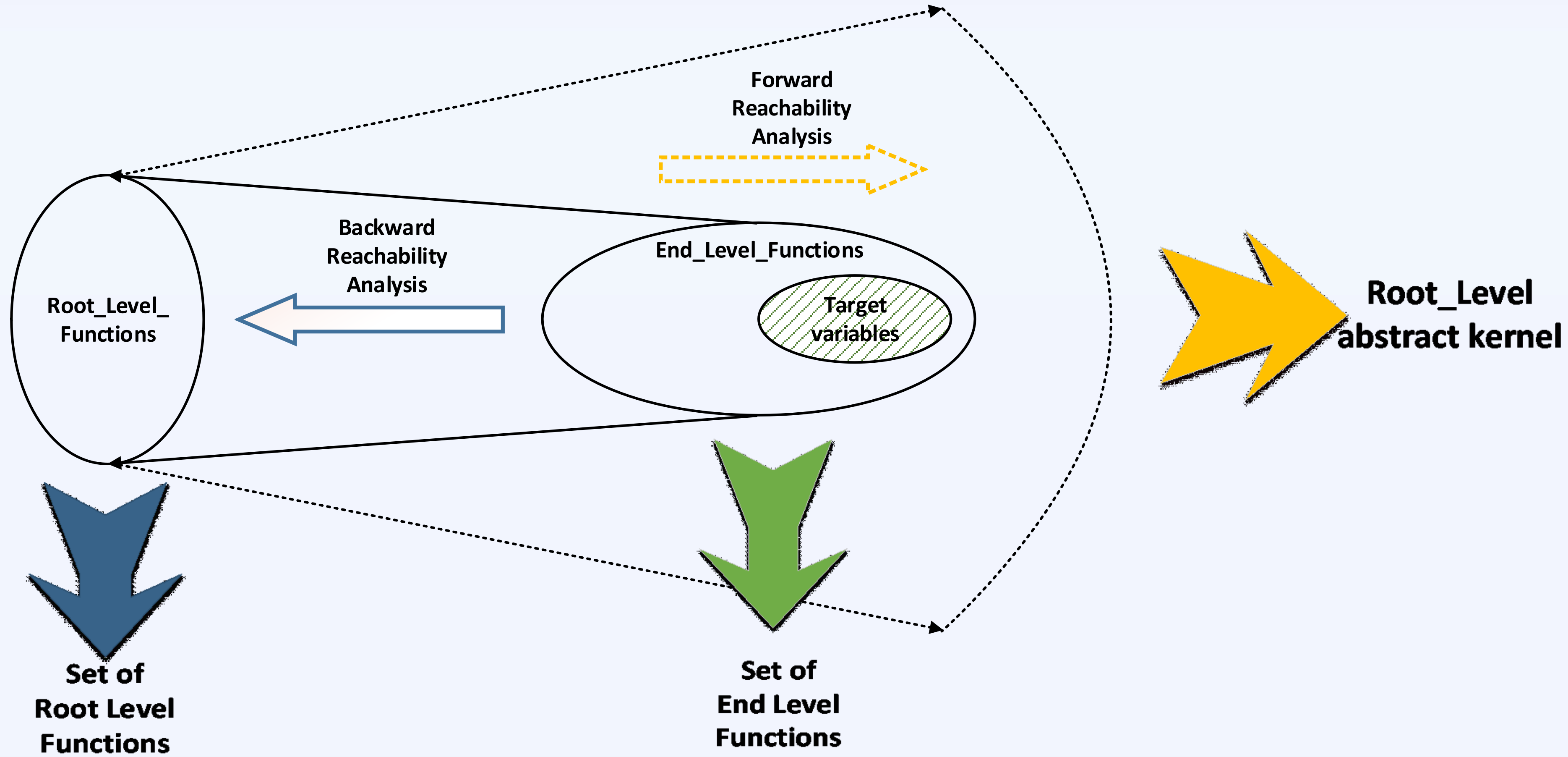
Extended Verification Target Variable
: Variable that a Verification Target Variable depends on

```
void tpl_put_new_proc(){
     tpl_h_prio += extendedTargetVariable;
     assert(tpl_h_prio != -1);
}
```

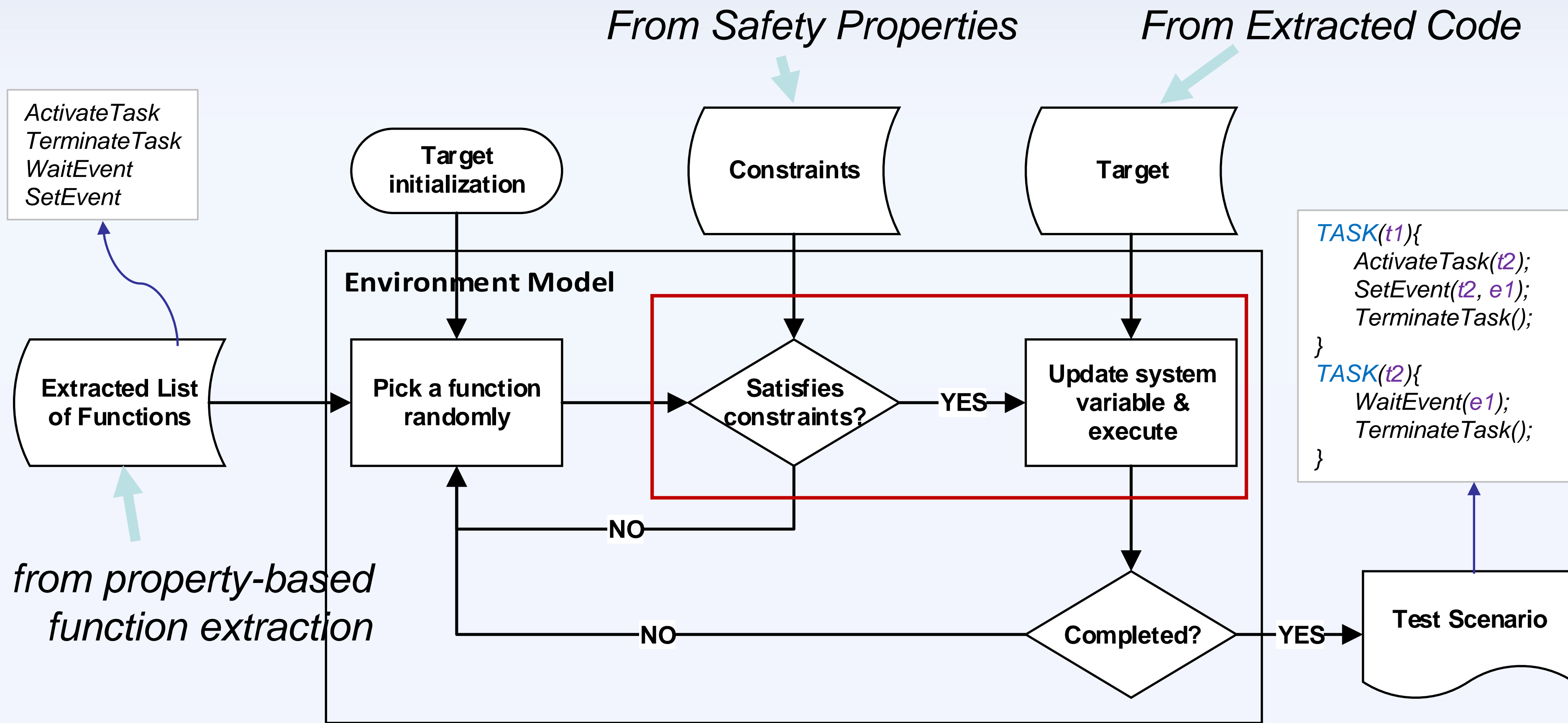
Verification Target Variable
: Variable that appears in the property specification

```
void tpl_schedule_from_running(){
     extendedTargetVariable += 1;
}
```

특성기반 코드 추출



제약조건을 고려한 환경 생성 – root-level function

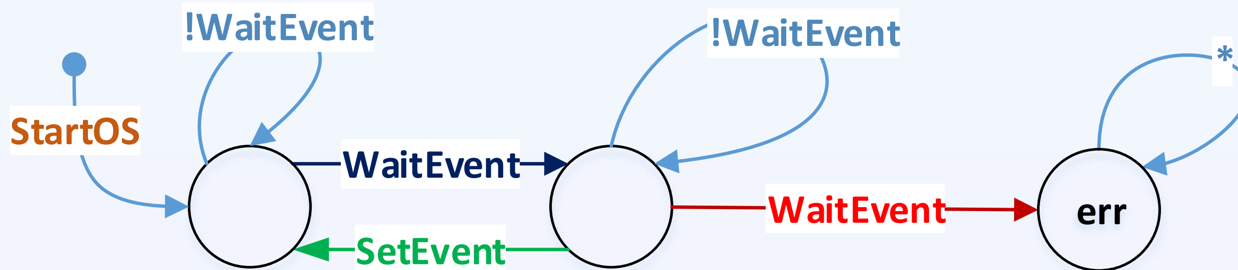


- 환경은 Root-Level function의 목록에서 가능한 모든 호출을 수행
- 제약조건에 기반해 false alarm이 발생하는 호출을 배제

제약조건 도출

❖ **예시**

WaitEvent가 호출되면 호출한 작업은 대기상태로 변경되어, 다른 작업에서 SetEvent가 호출되기 전까진 아무런 일도 할 수 없다. 따라서 WaitEvent는 중첩되어 호출될 수 없으며 다른 작업에서 SetEvent가 호출된 후에야 다시 호출될 수 있다.

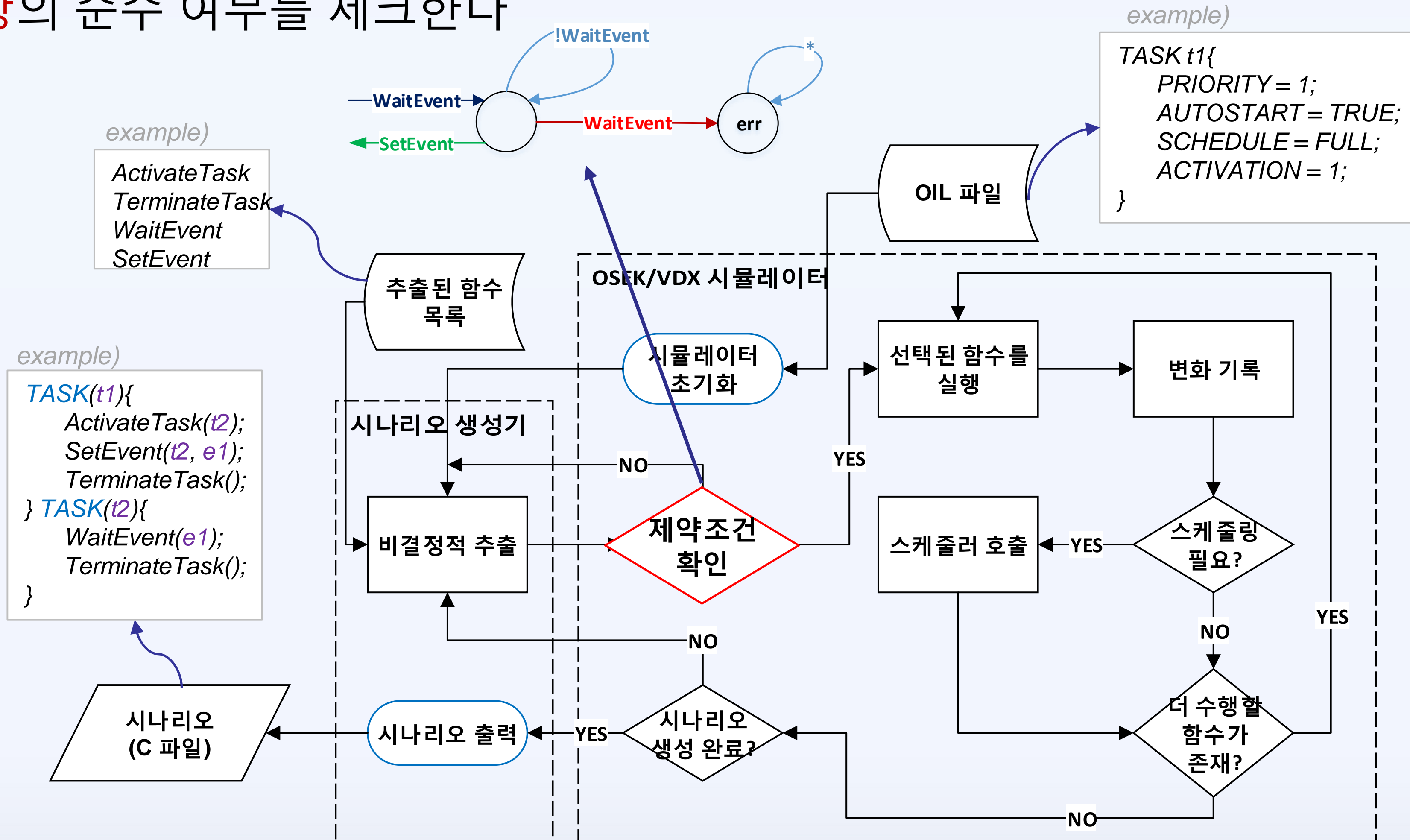


❖ **제약조건을 고려하지 않은 비결정적 시나리오 생성의 문제점들**

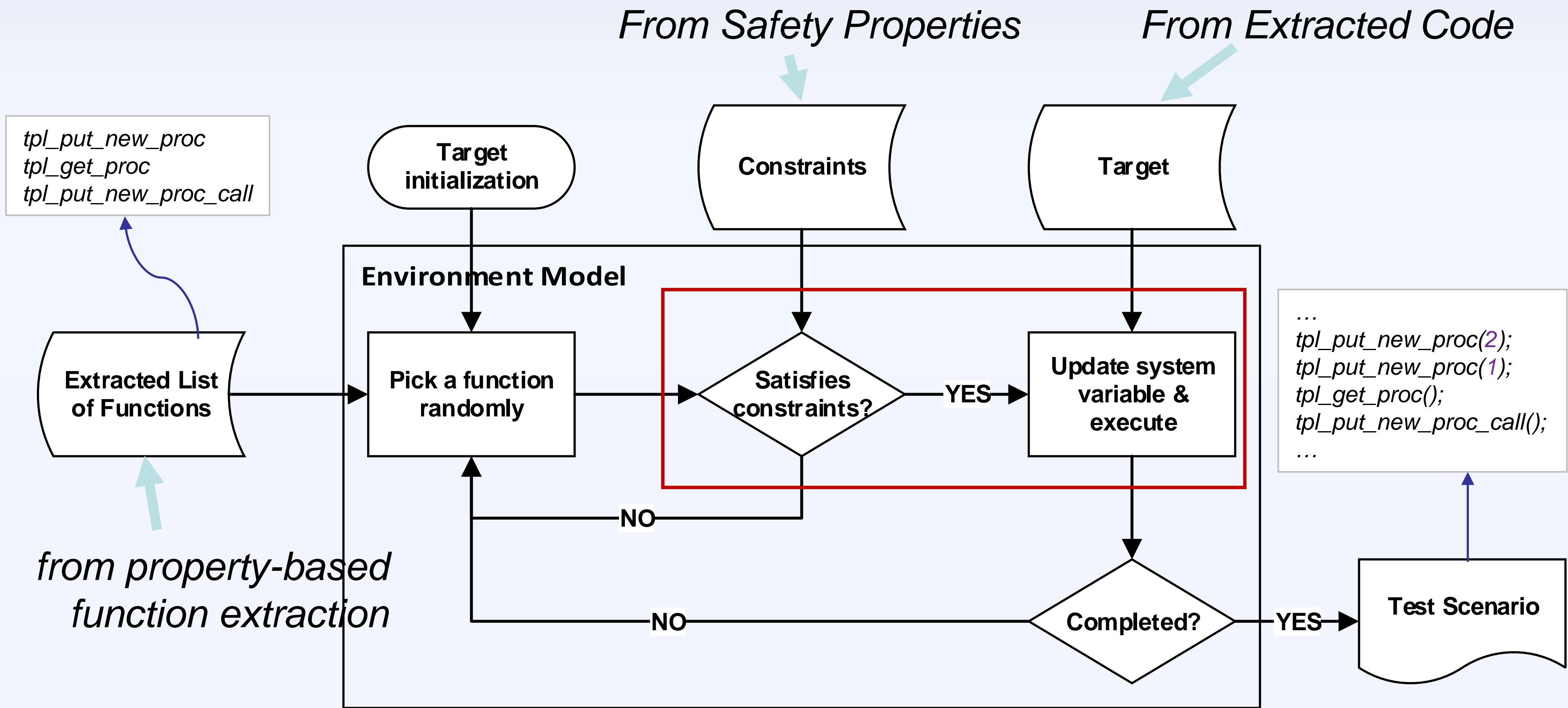
- *SetEvent*가 이미 호출되었는지 어떻게 판단 할 것인가?
  - ➢ Task와 Resource, Event의 변화를 모두 기록할 필요가 있음
- *WaitEvent*가 호출된 다음에는 어떤 Task를 활성화 시킬 것인가?
  - ➢ 우선순위 큐로 준비상태 Task들을 관리한다

OSEK/VDX 제약사항 확인기

OSEK/VDX 제약사항 확인기는 시스템 상태의 변화를 추적함으로써 제약사항의 준수 여부를 체크한다



제약조건을 고려한 환경 생성 – end-level function

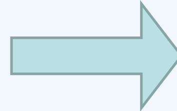


- 환경은 End-Level function의 목록에서 가능한 모든 호출을 수행
- 제약조건에 기반해 false alarm이 발생하는 호출을 배제

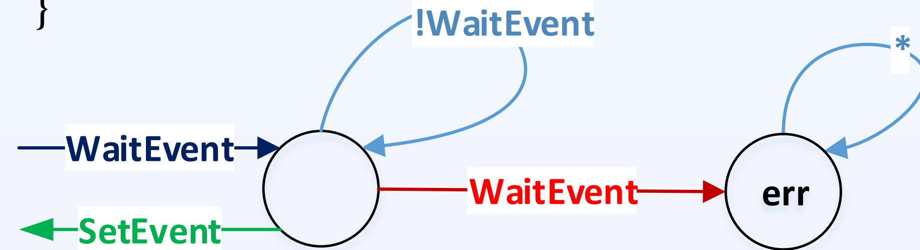
제약조건을 고려한 환경 생성 – False Alarm의 제거

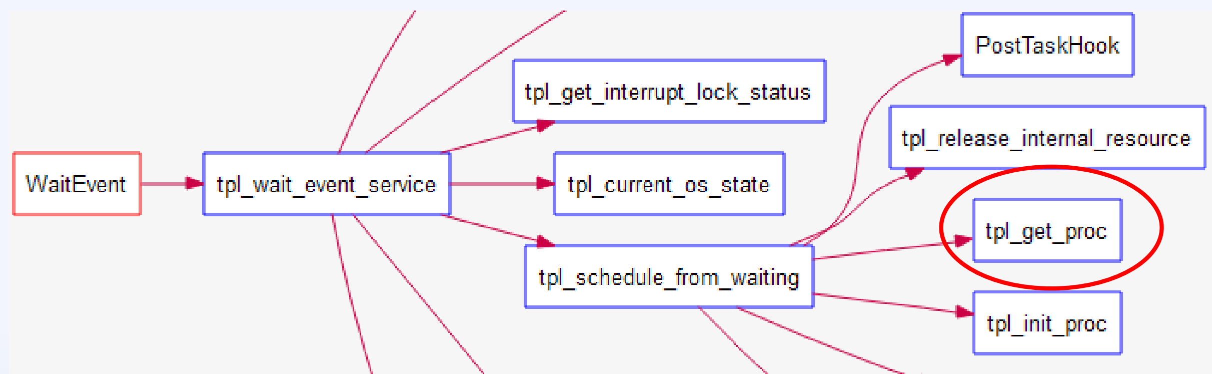
API function	End-Level Function
StartOS	tpl_put_new_proc*2, tpl_get_proc
WaitEvent	tpl_get_proc

```
function end_level_scenario(){
        tpl_put_new_proc();
        tpl_put_new_proc();
        tpl_get_proc();
        tpl_get_proc();
        tpl_get_proc();
}
function tpl_put_new_proc(){
        tpl_h_prio++;
        assert(tpl_h_prio != -1);
}
function tpl_get_proc(){
        tpl_h_prio--;
        assert(tpl_h_prio != -1);
}
```



```
function root_level_scenario(){
        StartOS();
        WaitEvent();
        WaitEvent(); //Constraint Violation
}
```

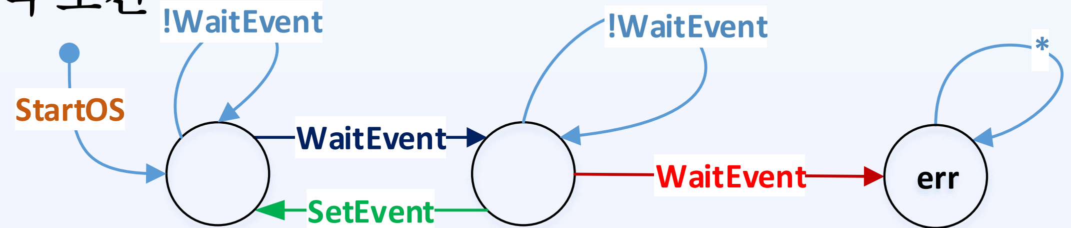




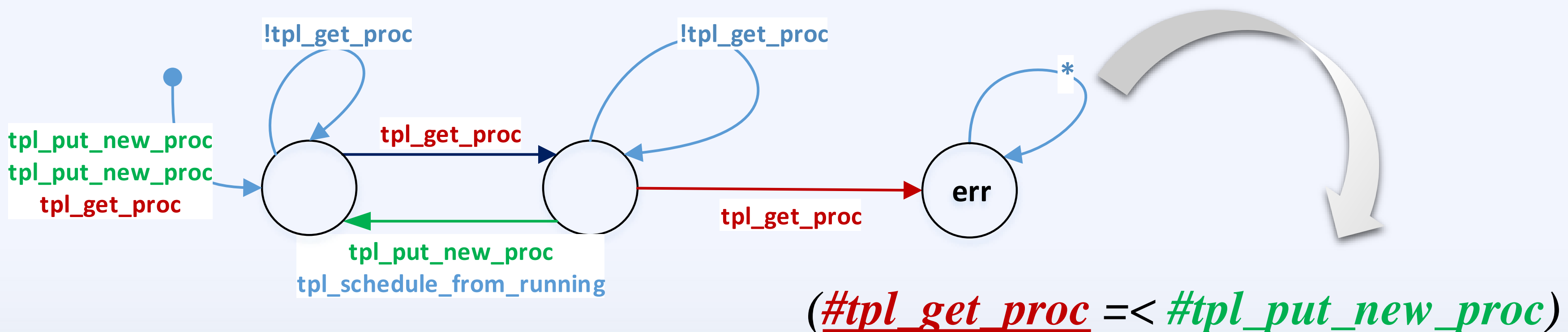
제약조건을 고려한 환경 생성 – 내부 제약조건

API function	End-Level Function
WaitEvent	tpl_get_proc
SetEvent	tpl_put_new_proc, tpl_schedule_from_running
StartOS	tpl_put_new_proc, tpl_put_new_proc, tpl_get_proc

OSEK/VDX 명세에서 도출한 제약 조건



호출관계를 고려해 도출한 내부 제약조건



(#tpl_get_proc =< #tpl_put_new_proc)

실험

- ❖ **대상 검증 속성 (Target Verification Property)**
  - Trampoline 커널에 있는 세 가지 기능적 안전 속성(functional safety property)에 대하여 환경 생성 및 검증을 수행하였다.

assert(tpl_h_prio != -1)

assert(tpl_kern != NULL)

assert(tpl_kern -> state == RUNNING)

- ❖ **위 세 개의 assert 문에 대하여** 모델 검증을 수행
- ❖ **위 세 개의 assert 문에 대하여** 비결정적 테스트를 실시한 뒤 코드 분기 커버리지(Code Branch Coverage)를 측정함

**실험 환경 : Linux Fedora 16 OS, with Intel Xeon 3.4GHz e3-1270 processor and 32GB of 1333MHz DDR3 RAM.*

*** 모델검증은 모델 검증기 CBMC 를 사용함.*

**** 코드 분기 커버리지는 SquishCoCo 툴로 측정되었음.*

실험 결과 – Model Checking

❖ **CBMC Model Checking**

- Unwind 값이 10 이하에서는 counter-example을 생성하지 않고 수행
- Unwind 값이 15 이상인 경우 검증의 수행이 불가능함

Property / **Unwind**	**tpl_h_prio != -1** code size : 437 lines 3 End-Level Functions / 19 Functions in total			**tpl_kern != NULL** **tpl_kern->state == RUNNING** code size : 787 lines 7 End-Level Functions / 32 Functions in total		
Unwind	VCC	Time(s)	Memory(MB)	VCC	Time(s)	Memory(MB)
Unwind 3	15	2	55.72	53	15	200.94
Unwind 7	43	91	297.14	157	4,100	1288.9
Unwind 10	64	2,379	923.67	235	42,241	1942.54
Unwind 15	99	30,748	2693.75	365	> 6 days	> 7366.89

실험 결과 – 비결정적 테스트

❖ **생성된 시나리오로 비결정적 테스트를 수행한 결과 – 분기 커버리지 측정**

- 테스트 시나리오의 길이가 34 이상으로 늘어나도 커버리지가 증가하지 않는 것을 확인할 수 있음

Property	tpl_h_prio != -1 code size : 1337 lines (8 Root-Level, 3 End-Level Functions) / 50 Functions					tpl_kern != NULL tpl_kern->state == RUNNING code size : 1378 lines (9 Root-Level, 7 End-Level Functions) / 52 Functions				
Length of the Scenario	14	20	22							
tpl_schedule_from_running	100%(3/3)	100%(3/3)	100%(3/							
tpl_schedule_from_dying	-	-	-							
tpl_schedule_from_waiting	-	-	-							
tpl_start_scheduling	-	-	-							
tpl_wait_event_service	-	-	-							
tpl_activate_task	-	-	-							
tpl_set_event	-	-	-	-	-	0%(0/5)	80%(4/5)	80%(4/5)	80%(4/5)	80%(4/5)
tpl_get_proc	100%(3/3)	100%(3/3)	100%(3/3)	100%(3/3)	100%(3/3)	-	-	-	-	-
tpl_put_new_proc	66.67%(2/3)	66.67%(2/3)	66.67%(2/3)	66.67%(3/3)	66.67%(2/3)	-	-	-	-	-
Time(s)	-	-	-	-	-	-	-	-	-	-
Memory(MB)	2.64	2.64	2.64	2.64	2.64	2.64	2.64	2.64	2.64	2.64

```
void tpl_schedule_from_running(void){
      ......
      // READY_AND_NEW
      if (tpl_kern.running->state == READY_AND_NEW)
      {
            tpl_init_proc(tpl_kern.running_id);
      }
      ......
}
```

실험 결과 – Overflow오류를 발견

```
tpl_get_proc             -> tpl_h_prio :  2, taskNum : 4, activationCount(T1, T2, T3) : 1, 255, 2
tpl_put_new_proc         -> tpl_h_prio :  2, taskNum : 5, activationCount(T1, T2, T3) : 1, 255, 1
---put_new_proc : 1
tpl_get_proc             -> tpl_h_prio :  2, taskNum : 4, activationCount(T1, T2, T3) : 1, 0, 1
tpl_get_proc             -> tpl_h_prio :  0, taskNum : 3, activationCount(T1, T2, T3) : 1, 0, 0
tpl_schedule_from_running -> tpl_h_prio : -1, taskNum : 3, activationCount(T1, T2, T3) : 0, 0, 0
RandomTest: RandomTest.c:505: tpl_schedule_from_running: Assertion `tpl_h_prio != -1' failed.
```

- End-level function를 대상으로 비결정적 테스트를 수행한 결과 overflow문제를 발견
  - Task2의 activation count가 255인 상태에서 activation이 하나 더 증가하면 0으로 변화
    → *tpl_h_prio 의 값이 -1로 변화*
  - 모델 검증기에서도 역시 unwind 값이 255이상이라면 동일한 오류를 검출할 수 있을 것이라 예상.
    → *하지만 한정적인 비용에서 해당 검증은 불가능에 가까움*

결론

❖ 본 방법의 이점

1) 제약사항을 만족하는 환경을 자동으로 생성함으로써 안정성 분석에서 관심 있는 부분만을 집중적으로 검증
2) 특성과 연관된 코드만을 추출하여 검증대상의 크기를 효과적으로 줄임

❖ 모델 검증과 비결정적 테스트의 사용방법 제안

1) End-Level Random Testing 을 stress testing 에 활용
2) Root-Level Random Testing 을 End-Level Testing에서 발견한 오류의 확인에 활용
3) 모델 검증을 마지막으로 제한적인 범위에서 포괄적인 검증에 활용

한계 및 향후 연구 방향

❖ 커버리지

- 일부 조건문에 나타나는 변수들이 완전히 추출되지 않음.
- 목적변수 관련된 조건의존성(conditional dependency)를 고려하여 변수를 추출함으로써 branch 커버리지를 향상시킬 수 있다.

❖ 가변적 OIL 설정 파일

- 본 연구에서는 커널 설정 파일인 OIL을 고정한 채로 시나리오를 생성함
- OIL 파서를 제작하여 시스템 설정의 변화를 손쉽게 반영할 수 있도록 할 예정