# 소프트웨어재난 연구센터

Software Disaster Research for Agile and Automated Response and Repair

연구3그룹: 소프트웨어안전공학 연구실
최윤자

**겨울워크샵**

**2022.2.9~2.11**

KNU 경북대학교 KYUNGPOOK NATIONAL UNIVERSITY

경북대학교 소프트웨어안전공학 연구실 *sselab.knu.ac.kr*

Safety Checking

Rigorousness

Automated

Model

Code

## 경북대학교 소프트웨어안전공학 연구실 *sselab.knu.ac.kr*

M

- Pushing acceleration pedal increases speed
- Rotating the handle x degree changes wheels y degree
- Never opens the doors while driving
- etc.

p

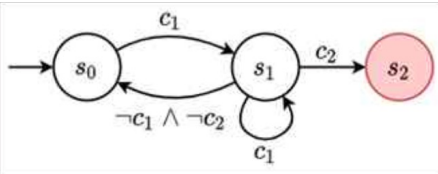
Formal checking

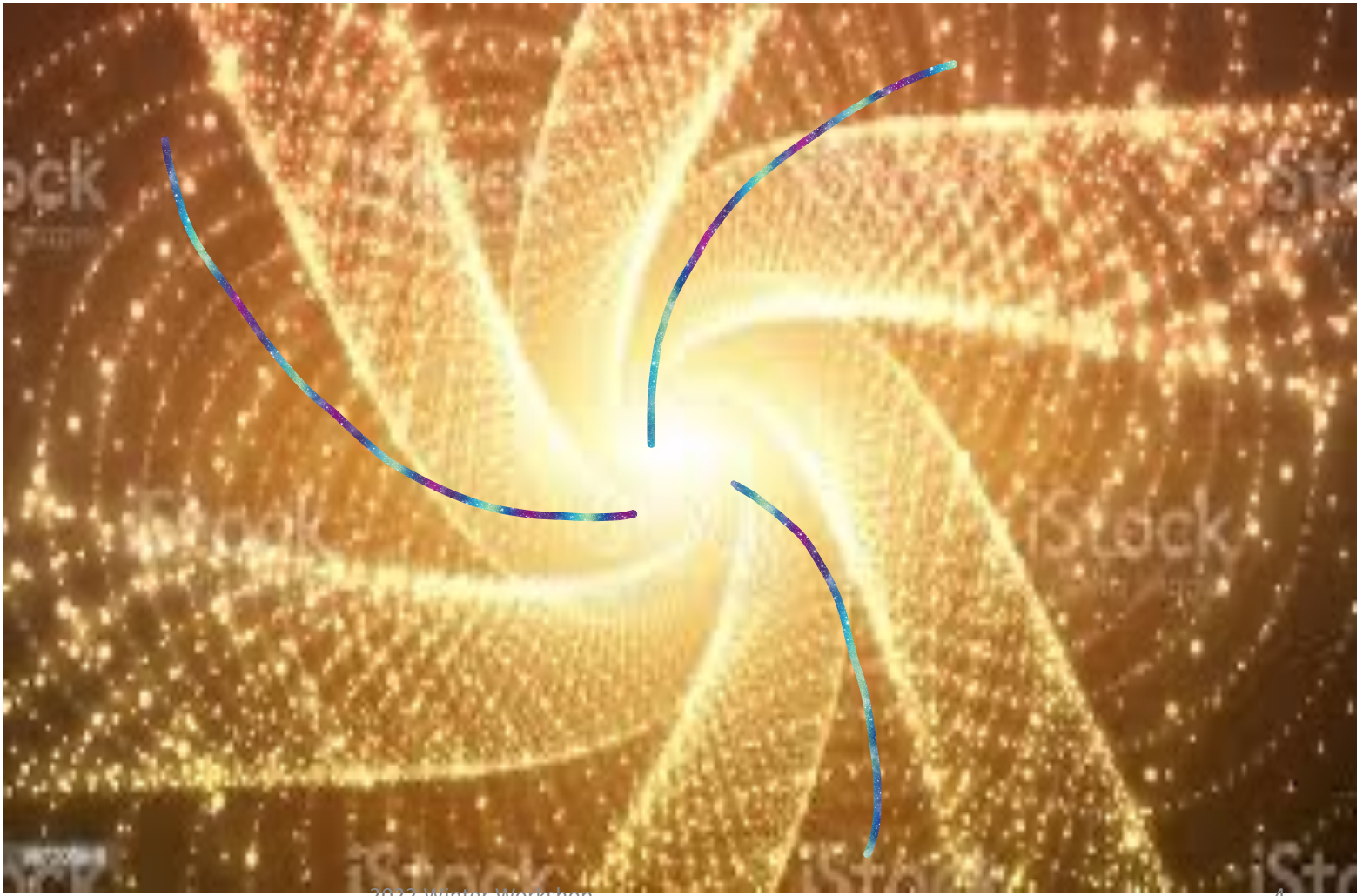
M ⊨ p

!(M ⊨ p)

Fix this and that ..

# 무한 탐색공간



Safety Checking

Rigorousness

Automated

Domain-specific

Model-based

Embedded Software
- Multi-tasking
- HW-dependent
- Shared memory
- Reactive
- Periodic

Model  Code

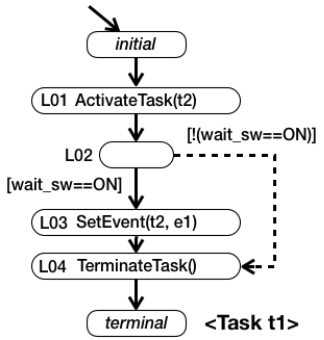
# 모델(추상화)의 필요성

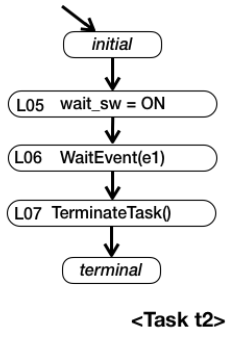
- 10만라인 이상의 프로그램 코드를 엄밀하게 검증하는 것은 현실적으로 불가능
  - 특히, 멀티태스킹 프로그램의 경우 코드 사이즈 대비 기하급수적으로 증가하는 검증 복잡도

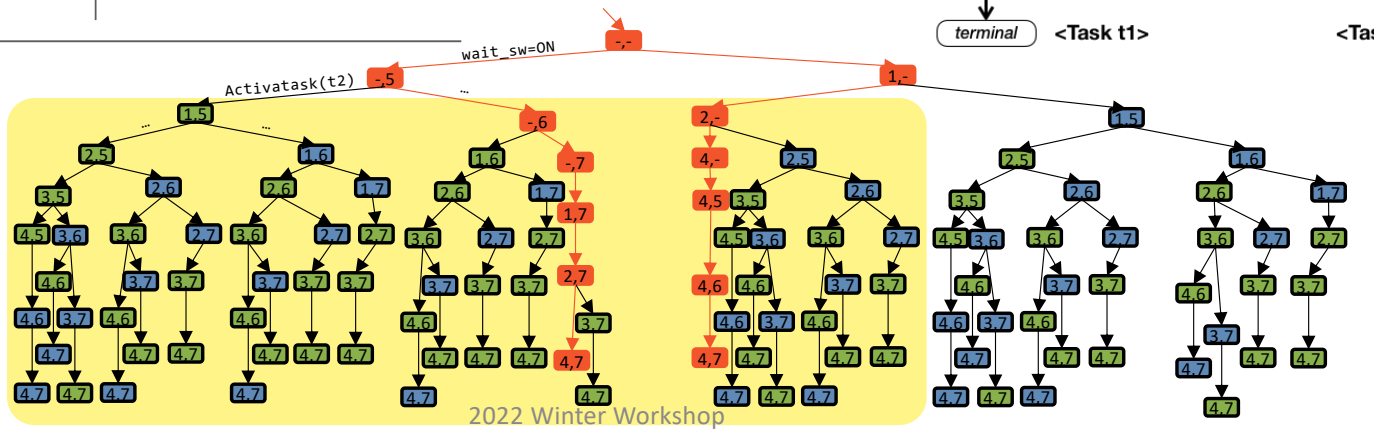
Application Program

```
   int wait_sw = OFF;
   TASK(t1) { // prio 1
01:  ActivateTask(t2);
02:  if(wait_sw==ON) {
03:    SetEvent(t2, e1);
     }
04:  TerminateTask();
   }
```

```
   TASK(t2) { // prio 2
05:  wait_sw = ON;
06:  WaitEvent(e1);
07:  TerminateTask();
   }
```

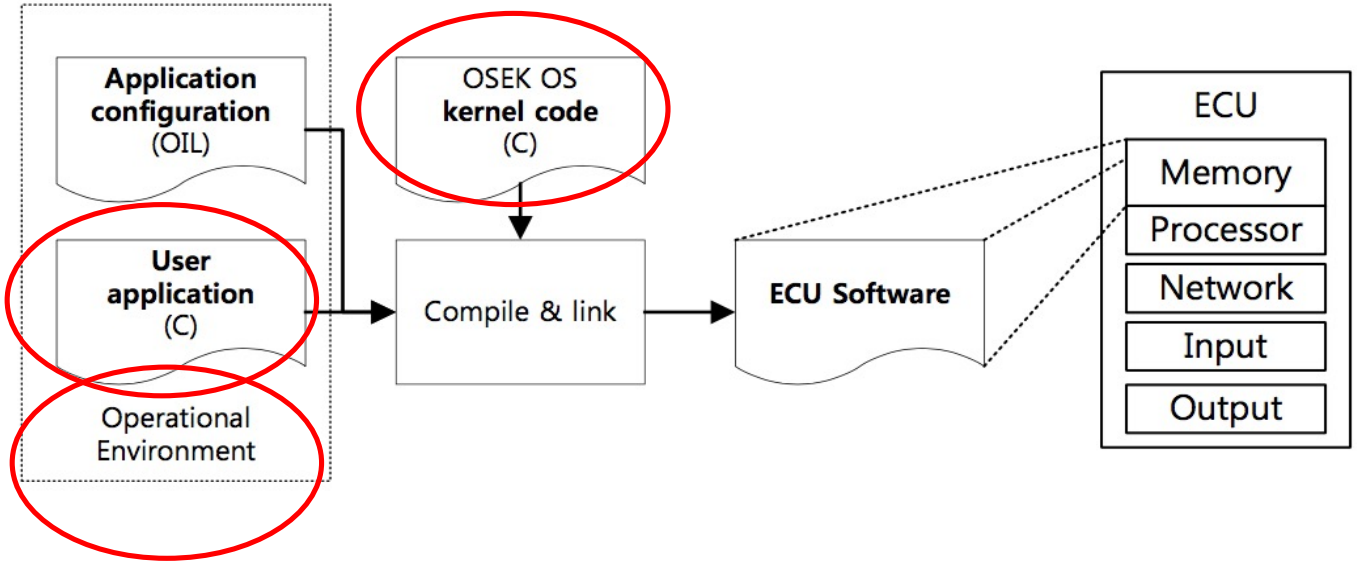


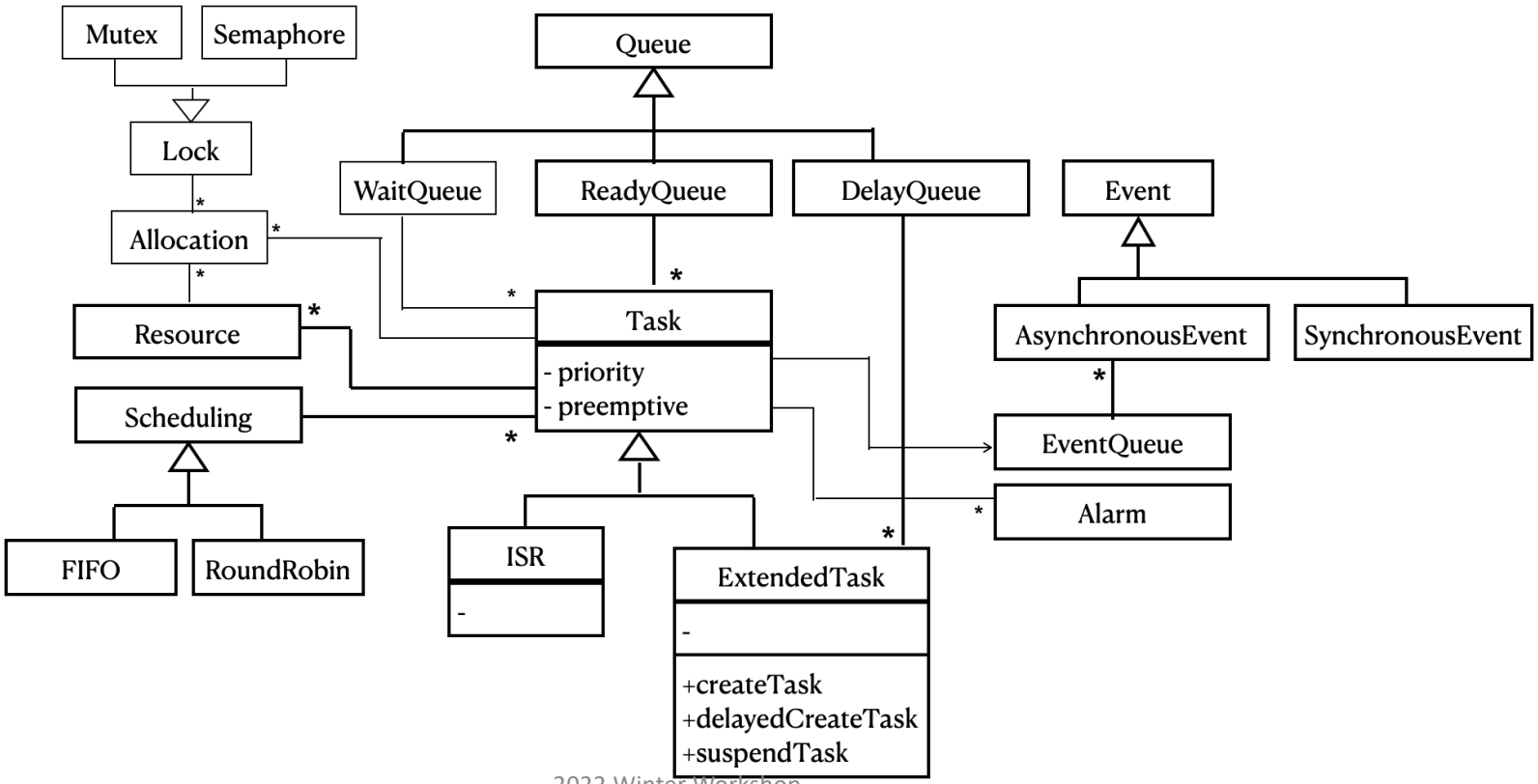
# 무엇을 추상화할 것인가?

- Application program: static dependency analysis, property-based code slicing
- Operating system: standard → functional modeling
- Operational environment → log-based program synthesis



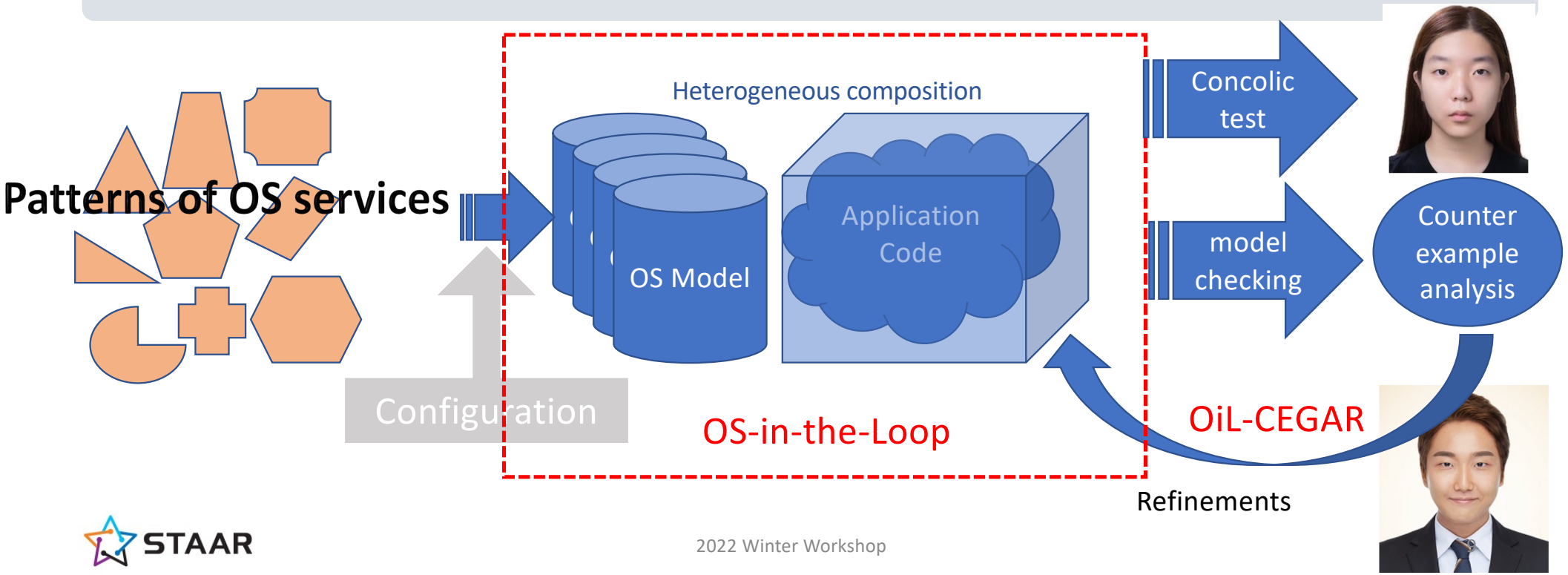
# 표준문서에 기반한 OS 패턴의 정의 및 모델 자동생성

- Common features of embedded OS



## 1. OS model + Application abstraction : 오경보를 (거의) 없애고 검증효율을 높임

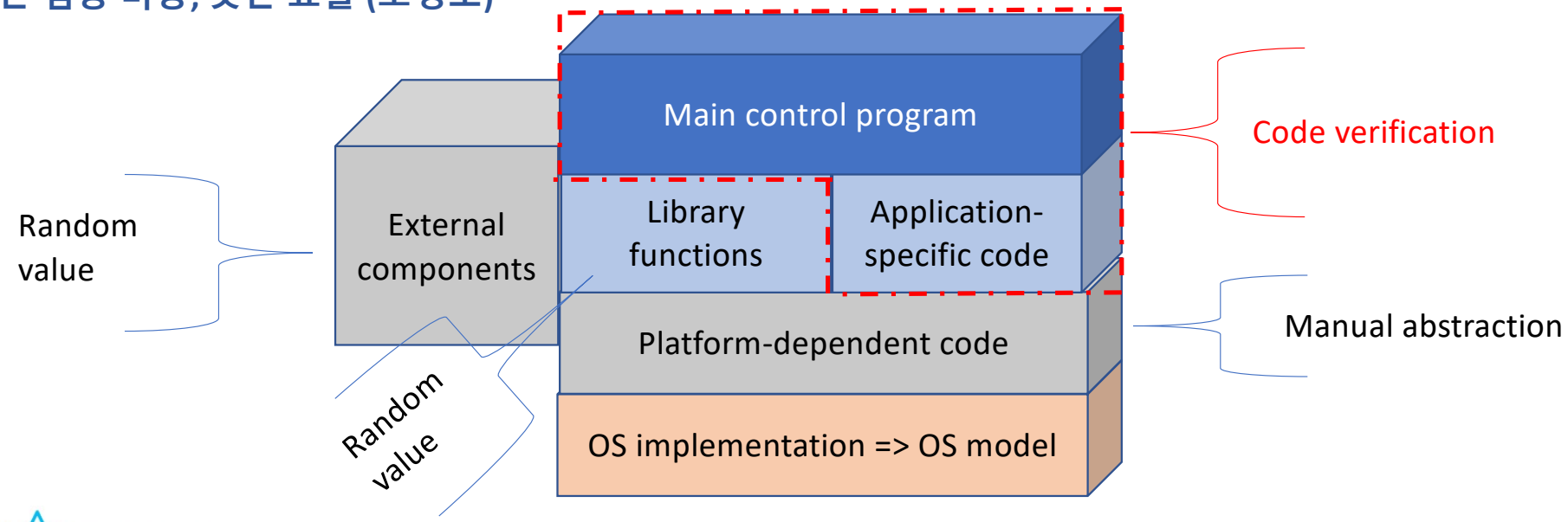
- Domain: Small-scale Embedded Software
- Language: C
- Assumption: operates on a standard embedded OS (OSEK/VDX, Zephyr, Erika, FreeRTOS ..)



## 2. Application abstraction: main control logic + operational environment

- OS-in-the-Loop verification improves verification efficiency, but it is still expensive
- We want to focus on the property of the main control logic (tasks and their interactions)
- How about library functions, user-defined functions, and platform-dependent functions?

높은 검증 비용, 낮은 효율 (오경보)



# 모델 자동생성의 필요성

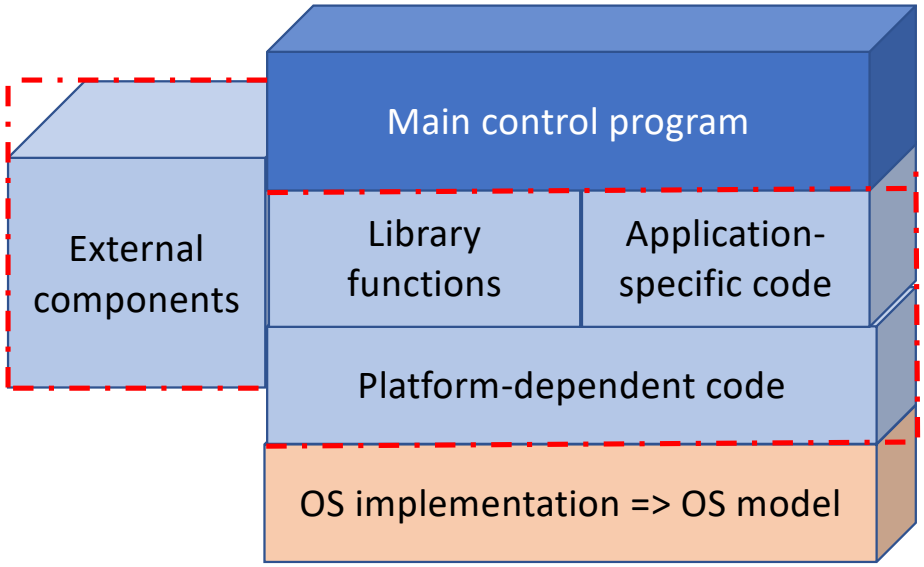
- (흔하지 않은) Open source 인 경우에도 코드를 그대로 검증하는 것은 불가능하거나 검증비용이 비현실적으로 높음
- 대부분의 현실적인 시스템에서 모델이 존재하지 않음
  - 개발과정에서 모델을 만들지 않는 경우
  - 모델을 만들어도 공개하지 않는 경우
  - 컴포넌트 별로 (각각 다른 기업에서) 개발하여 실행파일을 통합하는 방식이 통상적
    - 이경우, (불완전한) 컴포넌트 스펙이 존재할 가능성
  - 외부 라이브러리를 사용하는 경우
    - 함수원형과 메뉴얼 문서 존재 가능
- 검증의 효과적 적용을 위해서는 코드 추상화와 모델이 필수이나 수작업으로 모델링을 할 수 있는 전문가가 거의 없음

# 프로그램 합성기를 이용한 로그 기반 모델 자동생성

- External components : random stubs → log-based, spec-based synthesis
- Library functions: random stubs → log-based, spec-based synthesis
- Platform-dependent code : manual abstraction → log-based synthesis
- Application-specific code: code verification → log-based synthesis

높아질 수 밖에 없는 정확도

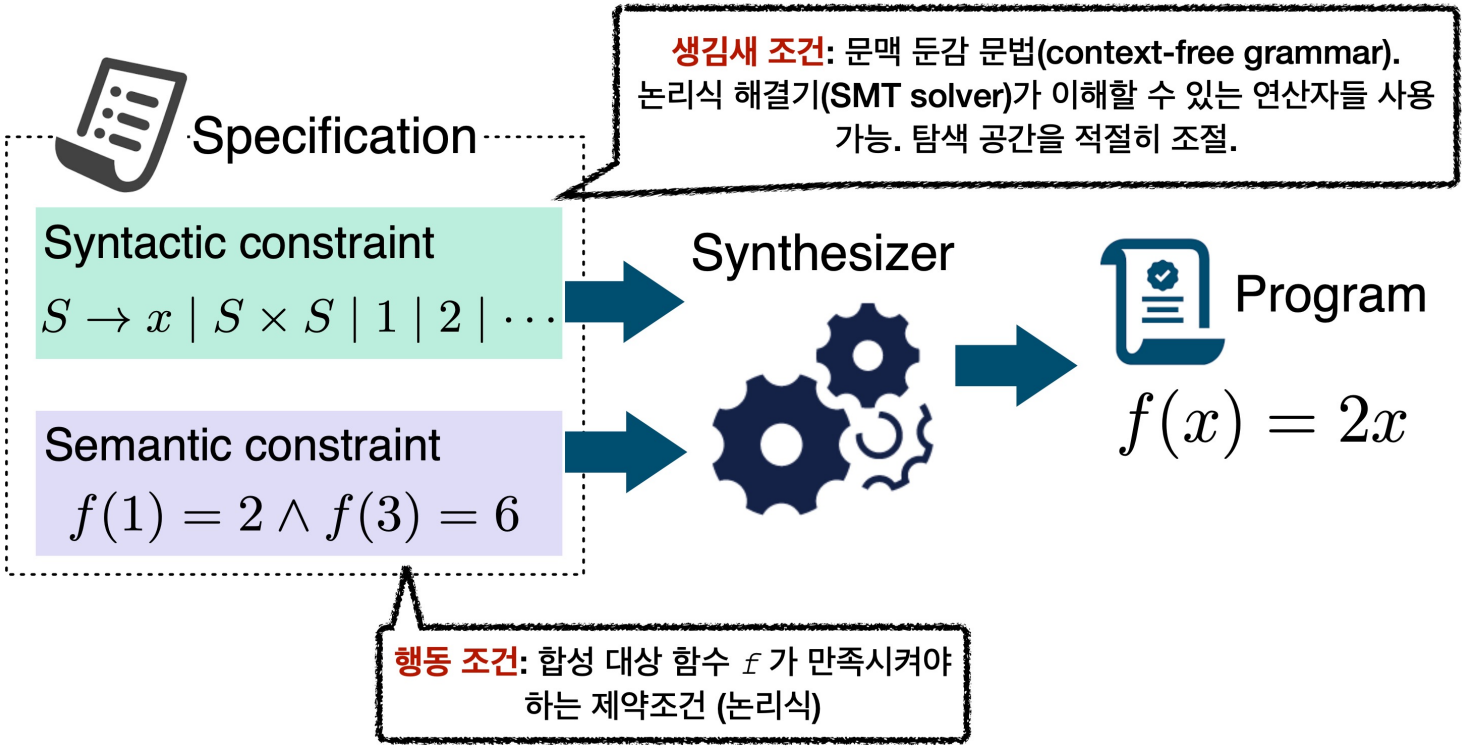
낮아지는 정확도




목표 모델:
높은(?) 정확도와
작은(?) 사이즈를 가진 모델

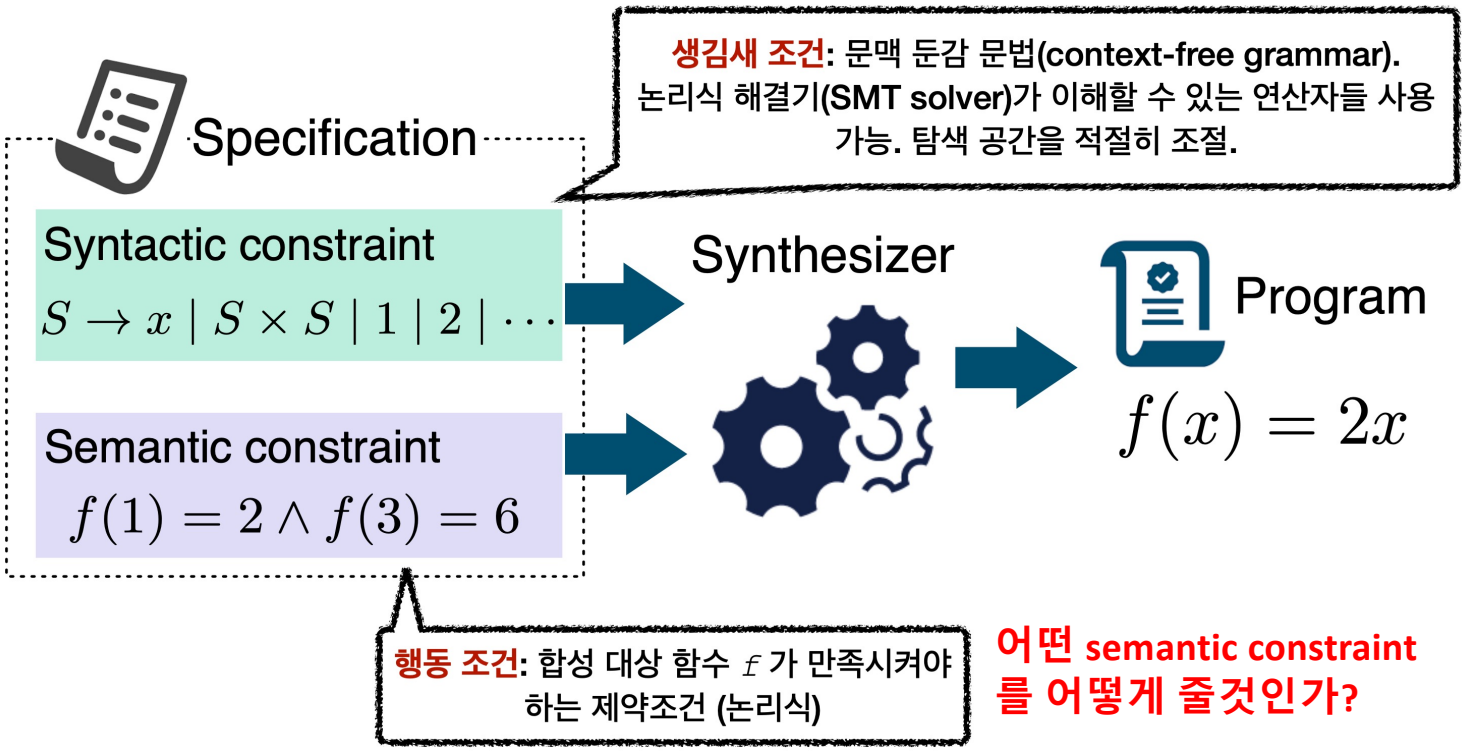
낮은 정확도 → 많은 오경보
큰 사이즈 → 높은 검증 비용

# 로그(입출력값) 기반 프로그램 합성




* 한양대학교 이우석 교수님 강의자료에서 발췌

# 로그(입출력값) 기반 프로그램 합성 : 현실적인 문제점




어떤 syntactic constraint 를 줄것인가?

어떤 semantic constraint 를 어떻게 줄것인가?

* 한양대학교 이우석 교수님 강의자료에서 발췌

# 로그(입출력값) 기반 프로그램 합성 : 실험 사례 1

```
// array size: 6, each value : 0~9
int containsDuplicate(int* array){
    int values[10] = {0};
    for (int i = 0; i < SIZE; i++) {
        values[array[i]]++;
        if (values[array[i]] == 2) return 1;
    }
    return 0;
}
```

원본 소스 코드

```
(set-logic LIA)
;;
(synth-fun f ((x0 Int) (x1 Int) (x2 Int) (x3 Int) (x4 Int)
              (x5 Int)) Int
    ((Start Int ( x0 x1 x2 x3 x4 x5 1 0
;;              (+ Start Start)
;;               (- Start Start)
                (ite StartBool Start Start)
                ))
       (StartBool Bool (
                true false
                (= Start Start)
                (< Start Start)
                (>= Start Start)
                (and StartBool StartBool)
                (or StartBool StartBool)
                (not StartBool)     ;; added 2022.1.25
))))
```

Syntactic constraints

```
;; contraint list - 10 randomly generat
(constraint (= (f 9 8 7 0 1 3 ) 0 ))
(constraint (= (f 6 8 4 0 8 5 ) 1 ))
(constraint (= (f 5 8 5 5 1 8 ) 1 ))
(constraint (= (f 0 9 9 1 4 2 ) 1 ))
(constraint (= (f 1 3 0 3 2 3 ) 1 ))
(constraint (= (f 0 3 3 9 3 4 ) 1 ))
(constraint (= (f 2 1 2 6 1 3 ) 1 ))
(constraint (= (f 1 6 1 6 4 4 ) 1 ))
(constraint (= (f 6 4 6 6 7 0 ) 1 ))
(constraint (= (f 0 1 5 0 6 7 ) 1 ))

;;adding one constraint
(constraint (= (f 0 3 1 8 9 9 ) 1 ))

;;adding one constraint
(constraint (= (f 9 1 1 3 6 5 ) 1 ))

;;adding one constraint
(constraint (= (f 7 7 6 9 1 7 ) 1 ))
```

Semantic constraints

```
(define-fun f ((x0 Int) (x1 Int) (x2 Int) (x3 Int) (x4 Int) (x5 Int)) Int (ite (= x1
x0) 1 (ite (= x2 x0) 1 (ite (= x3 x0) 1 (ite (= x2 x1) 1 (ite (= x5 x1) 1 (ite (= x5
x2) 1 (ite (= x4 x0) 1 (ite (= x4 x2) 1 (ite (= x3 x2) 1 (ite (= x5 x0) 1 (ite (= x5
x4) 1 (ite (= x4 x1) 1 (ite (= x3 x1) 1 (ite (= x4 x3) 1 (ite (= x5 x3) 1 0))))))))))
))))))
size : 76
time : 3.59 sec
```

```
int DuplicateSynthesized(int* array){
if(array[1]==array[0]) return 1; else if(array[2]==array[0]) return 1; else if(array[
3]==array[0]) return 1; else if(array[2]==array[1]) return 1; else if(array[5]==array
[1]) return 1; else if(array[5]==array[2]) return 1; else if(array[4]==array[0]) retu
rn 1; else if(array[4]==array[2]) return 1; else if(array[3]==array[2]) return 1; els
e if(array[5]==array[0]) return 1; else if(array[5]==array[4]) return 1; else if(arra
y[4]==array[1]) return 1; else if(array[3]==array[1]) return 1; else if(array[4]==arr
ay[3]) return 1; else if(array[5]==array[3]) return 1; else return 0;
}
```

합성결과
- 100% 정확도
- 72 iterations
- 총 82개 입출력값 소요

# 로그(입출력값) 기반 프로그램 합성 : 실험 사례 2

```c
// array size: 6, each value: 0~9
int containsNearbyDuplicate(int* array){
    int k = array[SIZE-1];
    if (k == 0) return 0;
    for (int i = 0; i < SIZE-1; i++) {
        int temp = array[i];
        int start = i - k;
        int end = i + k;
        if (start < 0) start = 0;
        if (end > SIZE-2) end = SIZE-2;
        for (int j = start; j <= end; j++) {
            if (i == j) continue;
            if (temp == array[j]) return 1;
        }
    }
    return 0;
}
```

원본 소스 코드

Synthesis process 1

1. Start with 10 randomly generated input/output constraints
2. Synthesize using Eusolver and measure accuracy
3. Go to 6 if desired accuracy is achieved, continue, otherwise
4. Use CBMC to generate a counterexample for f(x)== f_synth(x)
5. Add the counterexample as a new constraint and go to step 2
6. Stop

```
 93 average success rate: 77.12
 94 average success rate: 82.17
 95 average success rate: 90.57
 96 average success rate: 75.58
 97 average success rate: 91.83
 98 average success rate: 93.07
 99 average success rate: 92.98
100 average success rate: 92.89
101 average success rate: 86.31
102 average success rate: 82.24
103 average success rate: 76.66
104 average success rate: 77.33
105 average success rate: 93.26
106 average success rate: 66.25
107 average success rate: 73.60
108 average success rate: 71.27
109 average success rate: 79.83
110 average success rate: 79.23
111 average success rate: 80.66
112 average success rate: 76.15
113 average success rate: 71.71
114 average success rate: 72.74
115 average success rate: 94.40
116 average success rate: 94.61
117 average success rate: 90.04
118 average success rate: 95.78
```

size : 121
time : 4.67 sec

```
;;adding one constraint
;; adding this one made it take too much time -- not finished after 290 minutes
;;(constraint (= (f 8 3 8 8 9 0 ) 0 ))
```

# 로그(입출력값) 기반 프로그램 합성 : 실험 사례 2

```
// array size: 6, each value: 0~9
int containsNearbyDuplicate(int* array){
    int k = array[SIZE-1];
    if (k == 0) return 0;
    for (int i = 0; i < SIZE-1; i++) {
        int temp = array[i];
        int start = i - k;
        int end = i + k;
        if (start < 0) start = 0;
        if (end > SIZE-2) end = SIZE-2;
        for (int j = start; j <= end; j++) {
            if (i == j) continue;
            if (temp == array[j]) return 1;
        }
    }
    return 0;
}
```

원본 소스 코드

Synthesis process 2

1. Start with 10 randomly generated input/output constraints
2. Synthesize using Eusolver and measure accuracy
3. Go to 6 if desired accuracy is achieved, continue, otherwise
4. Randomly generate 10 more input/output constraints
5. Add the constraints and go to step 2
6. Stop

```
747 average success rate: 72.27
748 average success rate: 72.55
749 average success rate: 72.29
750 average success rate: 72.20
751 average success rate: 72.22
752 average success rate: 72.13
753 average success rate: 72.06
754 average success rate: 72.28
755 average success rate: 72.29
756 average success rate: 72.30
757 average success rate: 72.25
758 average success rate: 72.05
759 average success rate: 72.26
760 average success rate: 72.32
761 average success rate: 72.30
762 average success rate: 72.28
763 average success rate: 72.32
764 average success rate: 72.05
765 average success rate: 72.39
766 average success rate: 72.35
```

size : 76
time : 163.14 sec

**Using iterative refinement is more efficient???**

# 로그(입출력값) 기반 프로그램 합성 : 실험 사례 3 - 1

```
// array size: 10, each distinct value: 0~10
int missingNumber(){
    int nums[11] = {0};
    for (int i = 0; i < SIZE; i++) {
        nums[array[i]] = 1;
    }
    for (int i = 0; i <= SIZE; i++) {
        if (nums[i] == 0) return i;
    }
    return -1;
}
```

```
422 average success rate: 55.89
423 average success rate: 54.83
424 average success rate: 54.46
425 average success rate: 53.66
426 average success rate: 53.23
427 average success rate: 54.51
428 average success rate: 58.65
429 average success rate: 58.56
430 average success rate: 57.95
431 average success rate: 58.46
432 average success rate: 57.96
433 average success rate: 57.97
434 average success rate: 58.80
435 average success rate: 55.45
436 average success rate: 54.81
437 average success rate: 54.86
438 average success rate: 54.89
439 average success rate: 55.22
440 average success rate: 51.21
441 average success rate: 51.90
442 average success rate: 51.85
443 average success rate: 54.96
444 average success rate: 53.90
445 average success rate: 53.18
446 average success rate: 53.12
447 average success rate: 57.78
```

Iterative refinements
size : 654
time : 378.96 sec

```
(set-logic LIA)
;;
(synth-fun f ((x0 Int) (x1 Int) (x2 Int) (x3 Int) (x4 Int)
              (x5 Int)) Int
   ((Start Int ( x0 x1 x2 x3 x4 x5 1 0
;;              (+ Start Start)
;;               (- Start Start)
                (ite StartBool Start Start)
                ))
       (StartBool Bool (
                true false
                (= Start Start)
                (< Start Start)
                (>= Start Start)
                (and StartBool StartBool)
                (or StartBool StartBool)
                (not StartBool)    ;; added 2022.1.25
))))
```

```
344 average suc
345 average suc
346 average suc
347 average suc
348 average suc
349 average suc
350 average suc
351 average suc
352 average suc
353 average suc
354 average suc
355 average success rate: 89.55
356 average success rate: 89.56
357 average success rate: 89.64
358 average success rate: 89.01
359 average success rate: 88.67
360 average success rate: 88.73
361 average success rate: 88.97
362 average success rate: 88.88
363 average success rate: 88.67
364 average success rate: 90.45
365 average success rate: 89.50
366 average success rate: 89.67
```

Random
size : 2475
time : 1905.73 sec

# 로그(입출력값) 기반 프로그램 합성 : 실험 사례 3 - 2

```
// array size: 10, each distinct value: 0~10
int missingNumber(){
    int nums[11] = {0};
    for (int i = 0; i < SIZE; i++) {
        nums[array[i]] = 1;
    }
    for (int i = 0; i <= SIZE; i++) {
        if (nums[i] == 0) return i;
    }
    return -1;
}
```

```
457 average success rate: 60.01
458 average success rate: 59.48
459 average success rate: 56.66
460 average success rate: 57.77
461 average success rate: 58.88
462 average success rate: 58.93
463 average success rate: 59.10
464 average success rate: 58.70
465 average success rate: 58.82
466 average success rate: 58.60
467 average success rate: 58.61
468 average success rate: 58.13
469 average success rate: 57.52
470 average success rate: 57.66
471 average success rate: 59.56
644 average success rate: 53.51 : 59.18
645 average success rate: 53.22 : 59.73
646 average success rate: 48.23 : 71.06
647 average success rate: 49.66 : 71.90
648 average success rate: 55.08 : 71.70
649 average success rate: 55.28 : 71.59
650 average success rate: 72.75 : 70.65
651 average success rate: 72.77 : 68.89
652 average success rate: 72.50
653 average success rate: 68.45
654 average success rate: 68.40
655 average success rate: 71.25
656 average success rate: 71.09
657 average success rate: 68.32
658 average success rate: 67.86
659 average success rate: 49.42
660 average success rate: 45.93
661 average success rate: 49.06
```

```
(set-logic LIA)
;;
(synth-fun f ((x0 Int) (x1 Int) (x2 Int) (x3 Int) (x4 Int) (x5 Int)) Int
    ((Start Int ( x0 x1 x2 x3 x4 x5
                -1 1 0 2 3 4 5 6 7 8 9
;;                (+ Start Start)
;;                 (- Start Start)
                  (ite StartBool Start Start)
                  ))
        (StartBool Bool (
                  true false
                  (= Start Start)
                  (< Start Start)
                  (>= Start Start)
99 average success rate: 91.08 ool StartBool)
100 average success rate: 91.14 ol StartBool)
101 average success rate: 91.40 ool)    ;; added 2022.1.25
102 average success rate: 91.68
103 average success rate: 91.60
104 average success rate: 91.07
105 average success rate: 91.06
106 average success rate: 91.43
107 average success rate: 91.25
108 average success rate: 91.23
109 average success rate: 91.29
110 average success rate: 91.56
111 average success rate: 91.51
112 average success rate: 91.48
113 average success rate: 91.49
114 average success rate: 91.67
115 average success rate: 91.84
116 average success rate: 91.93
117 average success rate: 91.83
```

Iterative refinements
size : 1111
time : 197.5 sec

Random
size : 536
time : 322.75 sec

# 로그(입출력값) 기반 프로그램 합성 : 실험 사례 4

```
// int array[6] = {0};, value: 0~9
int majorityElement(int* array){
    int max = -1, maxIndex = -1;
    int values[10] = {0};
    for (int i = 0; i < SIZE; i++) {
        values[array[i]]++;
    }
    for (int i = 0; i < 10; i++) {
        if (max <= values[i]) {
            max = values[i];
            maxIndex = i;
        }
    }
    return maxIndex;
}
```

```
1019 average success rate: 96.17
1020 average success rate: 95.89
1021 average success rate: 97.71
1022 average success rate: 96.84
1023 average success rate: 97.46
1024 average success rate: 95.15
1025 average success rate: 94.93
1026 average success rate: 95.16
1027 average success rate: 97.45
1028 average success rate: 96.55
1029 average success rate: 94.85
1030 average success rate: 94.57
1031 average success rate: 96.66
1032 average success rate: 95.78
1033 average success rate: 94.38
1034 average success rate: 94.67
1035 average success rate: 95.91
1036 average success rate: 96.24
1037 average success rate: 96.36
1038 average success rate: 95.08
1039 average success rate: 97.80
1040 average success rate: 97.08
1041 average success rate: 96.44
1042 average success rate: 97.52
1043 average success rate: 98.44
1044 average success rate: 98.00
1045 average success rate: 98.22
1046 average success rate: 98.31
1047 average success rate: 96.04
1048 average success rate: 97.78
1049 average success rate: 99.21
```

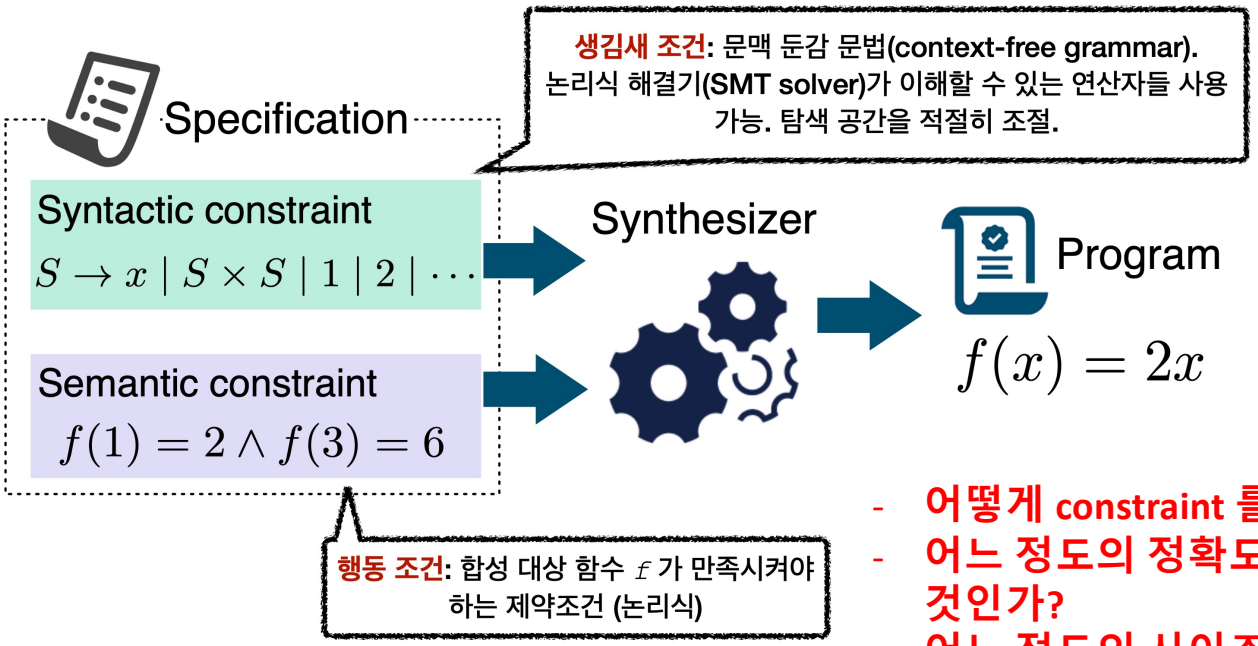
```
184 average success rate: 94.47
185 average success rate: 94.46
186 average success rate: 92.83
187 average success rate: 94.66
188 average success rate: 94.25
189 average success rate: 95.12
190 average success rate: 94.92
191 average success rate: 94.30
192 average success rate: 95.54
193 average success rate: 93.64
194 average success rate: 93.75
195 average success rate: 93.85
196 average success rate: 94.30
197 average success rate: 94.26
198 average success rate: 94.28
199 average success rate: 95.21
200 average success rate: 94.69
201 average success rate: 95.13
202 average success rate: 94.80
203 average success rate: 94.98
204 average success rate: 95.82
205 average success rate: 95.83
206 average success rate: 94.58
207 average success rate: 95.00
208 average success rate: 93.96
209 average success rate: 96.29
210 average success rate: 93.66
211 average success rate: 93.97
212 average success rate: 93.95
213 average success rate: 95.24
```

**High accuracy**
**Bigger size**

Iterative refinements
size : 1156
time : 386.7 sec

random
size : 1076
time : 1034.21 sec

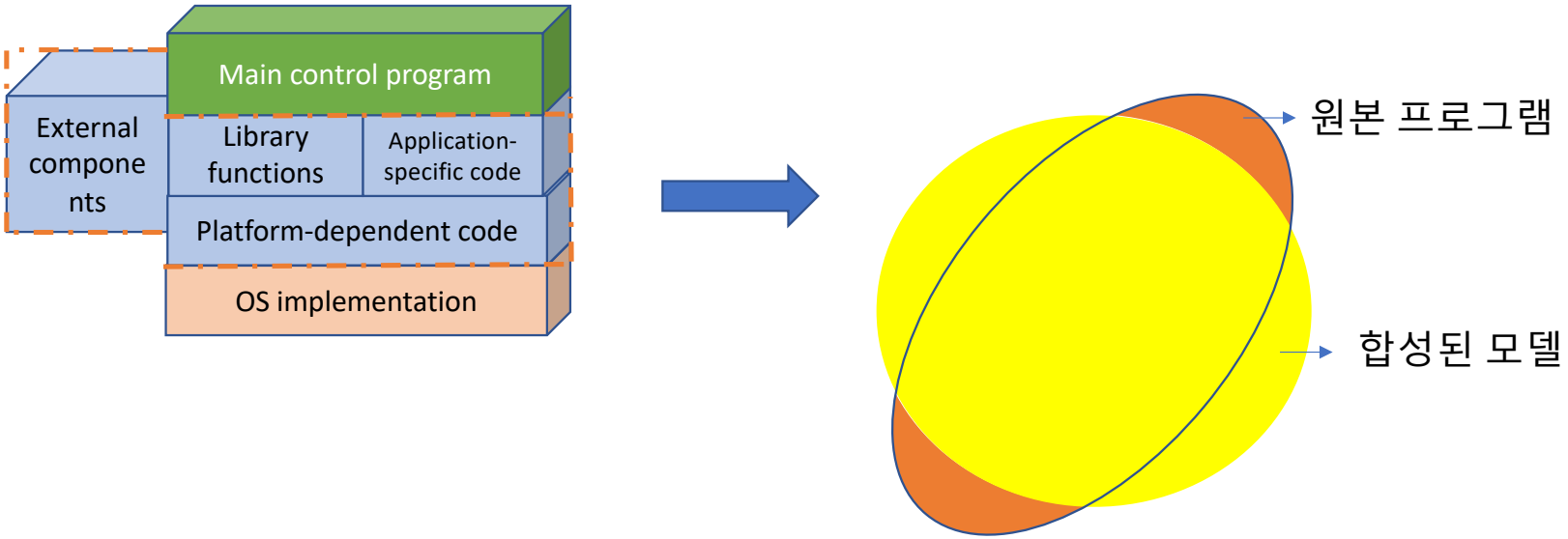
# 로그(입출력값) 기반 프로그램 합성 : 앞으로 해결해야 할 일들




코드의 스펙이 없고 소스코드도 없다고 했을때, 어떤 상수와 연산자들을 활용할 것인가?

- 어떻게 constraint 를 생성할 것인가?
- 어느 정도의 정확도를 목표로 할 것인가?
- 어느 정도의 사이즈를 목표로 할 것인가?
- 정확도가 100% 미만일때 오경보를 어떻게 식별하고 처리할 것인가?

## 로그(입출력값) 기반 프로그램 합성 : 부정확성에 따른 오경보(false negative) 및 오양성(false positive)




동적테스트 vs. (프로그램 합성을 이용한) 자동 추상화와 정형검증 vs. 수작업 추상화와 정형검증

# 연구3그룹: SW유사재난 재발방지

FORMALWORKS SW Development & Verification

**포멀웍스**
실시간 내장형 시스템 분석/검증의 난제 해결책 연구

+

rgblab

**무지개연구소**
드론 제어시스템 제작 및 시험 검증

SW 재난 오류 데이터베이스


SW재난 패턴 데이터베이스화

SW재난 패턴학습

SW재난 모델 자동생성

프로그램합성 기법 기반 모델 자동생성

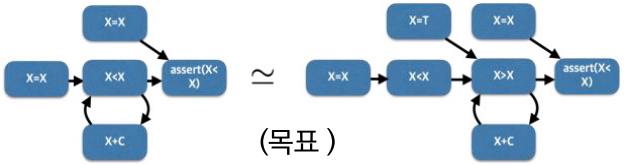
유사 SW 오류 식별을 위한 엄밀검증

**SW 재난 DB를 구축하여 재난 패턴을 학습하는 최초의 연구**

정적 분석을 위한 프로그램 패턴 자동 생성 (허기홍, OOPSLA 2017)

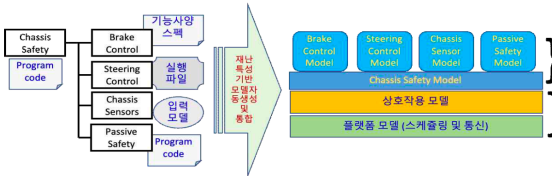
grep diff git

(현재)



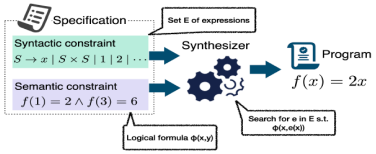

**프로그램 자동합성 기법을 정형모델 자동생성에 도입하는 최초의 연구**

패턴기반 모델자동생성(최윤자, JSS 2018, IEEE Access 2020)



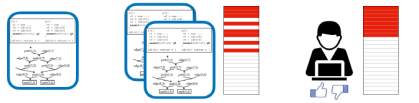

+

프로그램 자동생성 (이우석, POPL 2021)



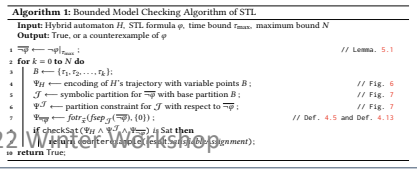

**오류 패턴 학습과 모델검증을 통합하여 전이검증을 구현하는 최초의 연구**

프로그램 버전에 따른 전이 분석 (허기홍, PLDI 2019)

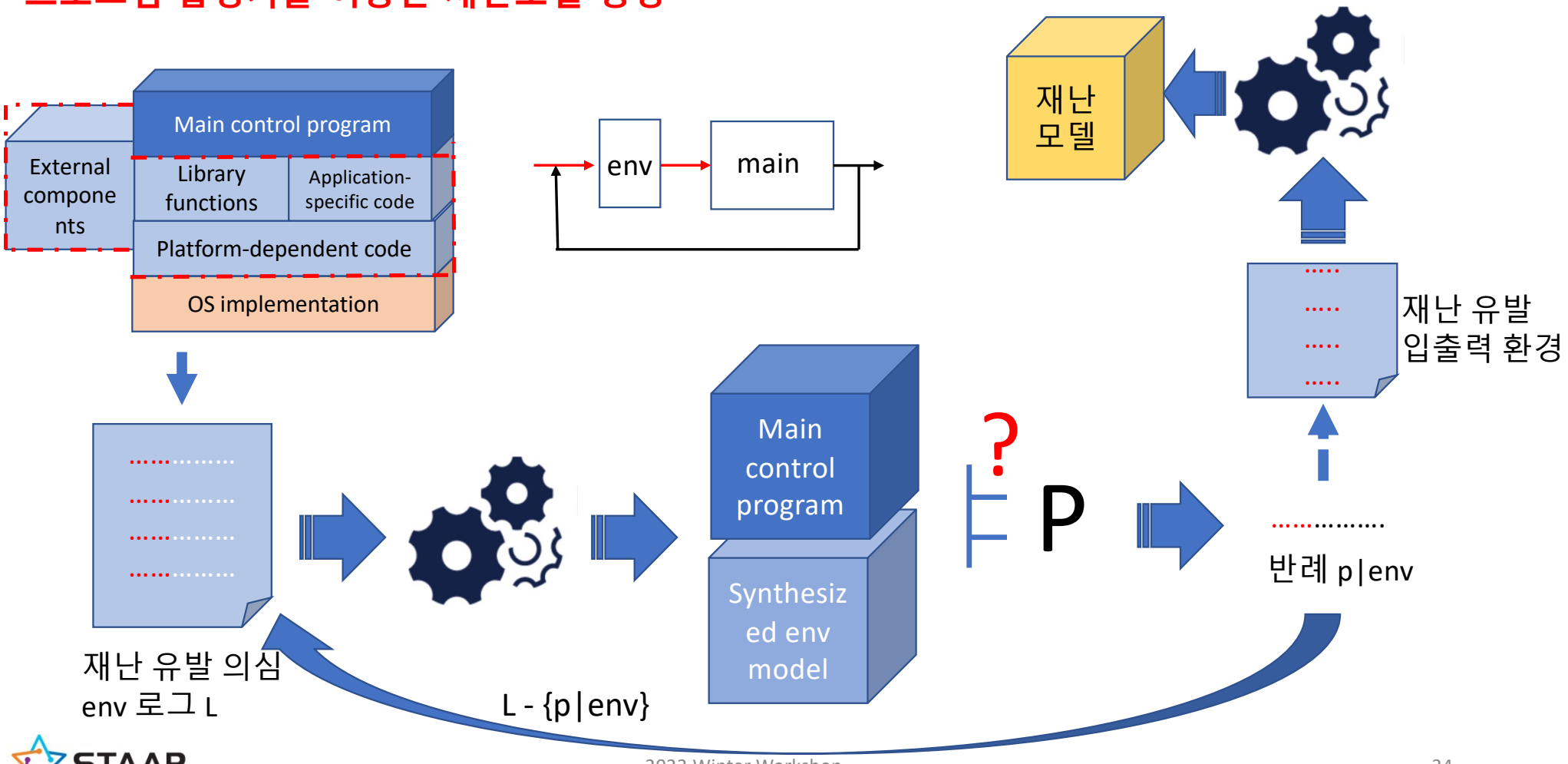



+

모델검증 알고리즘 (배경민, POPL 2019)



# 프로그램 합성기를 이용한 재난모델 생성



# 향후 계획

- 합성의 정확도를 높일 수 있는 로그 선택 기법
  - 점증적 합성: 모델검증, 심볼릭 테스트 입력 생성 등 기존 도구의 활용
  - 로그 분류기법: 시스템 로그로 부터 점증적 합성의 효과를 높이기 위한 로그 분류기법
- 낮은 정확도의 합성결과로 부터 얻은 검증결과의 해석
  - 오경보 식별기법 및 합성모델의 정제
  - 검증된 속성에 대한 해석
- 재난모델 생성 기법
  - 재난 모델의 정의
  - 합성 → 속성검증 → 모델 정제 → 합성의 사이클을 통해 재난모델과 정상모델 분리 실험