

---

# 프로그램 합성 기법을 이용한 임베디드 소프트웨어의 모델 체킹 효율성 향상을 위한 사례연구

김요엘, 최윤자  
경북대학교 컴퓨터학부

[kimyoel2305@gmail.com](mailto:kimyoel2305@gmail.com), [yuchoi76@knu.ac.kr](mailto:yuchoi76@knu.ac.kr)

# 임베디드 소프트웨어의 검증

- 임베디드 SW: 자동차, 드론 등을 제어하는 소프트웨어.
  - 오류가 발생하면 심각한 인명/재산 피해를 초래함. (검증의 필요성)
  - 시스템이 종료될 때 까지 반복해서 수행. (메인 loop의 존재)
- 임베디드 SW 검증의 어려움:
  - 환경적인 문제: system call, external library, sensor의 처리.
  - 검증 도구의 성능적 한계: loop, recursive call이 포함된 코드의 검증 효율성 감소.

# 검증 도구의 효율성 문제

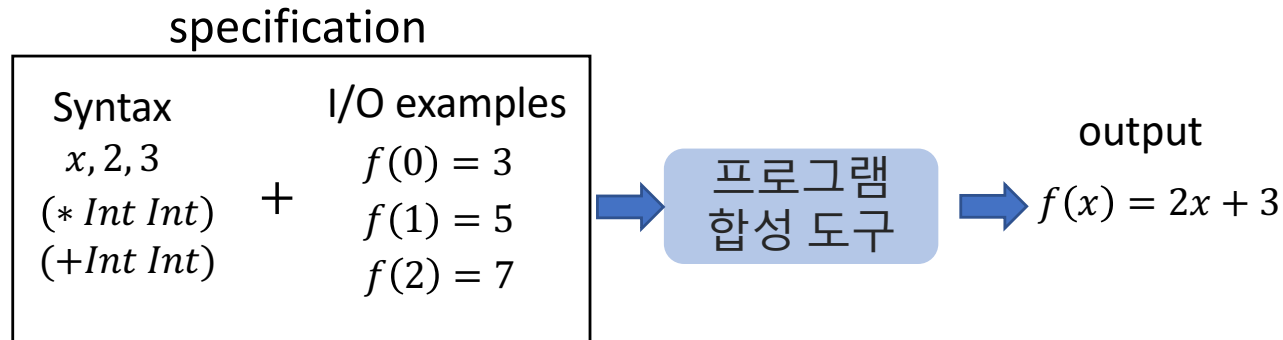
- C Bounded Model Checker (CBMC): C언어로 작성된 프로그램을 SAT formula로 변환하여 프로그램이 만족해야 하는 속성(상태)을 검증하는 도구.
- Unwinding: loop와 recursive call을 주어진 unwind (bound) 까지 수행하여 검증하는 것.
- unwind 횟수가 많아질 수록, 즉 대상 코드의 규모가 복잡할 수록 CBMC의 검증 시간은 기하급수적으로 증가함.

```
int main(int argc, char **argv) {  
    while(cond) {  
        BODY CODE  
    }  
}
```

```
int main(int argc, char **argv) {  
    if(cond) {  
        BODY CODE COPY 1  
        if(cond) {  
            BODY CODE COPY 2  
            if(cond) {  
                BODY CODE COPY 3  
                if(cond) {  
                    BODY CODE COPY 4  
                    if(cond) {  
                        BODY CODE COPY 5  
                    }  
                }  
            }  
        }  
    }  
}
```

# 프로그램 합성 기법 (Program Synthesis) 소개

- 다양한 형태의 제약조건을 가지고 프로그램 탐색 공간에서 제약조건을 만족하는 실행 가능한 프로그램을 찾아내는 기법.
- 입출력 예제 기반 프로그램 합성 기법:
  - 문법 제약: 합성된 프로그램에서 사용 가능한 인자, 상수, 연산자를 명시
  - 입/출력 예제: 합성된 프로그램들이 만족해야 하는 의미적 제약 사항



# 프로그램 합성 기법의 기대 역할과 한계

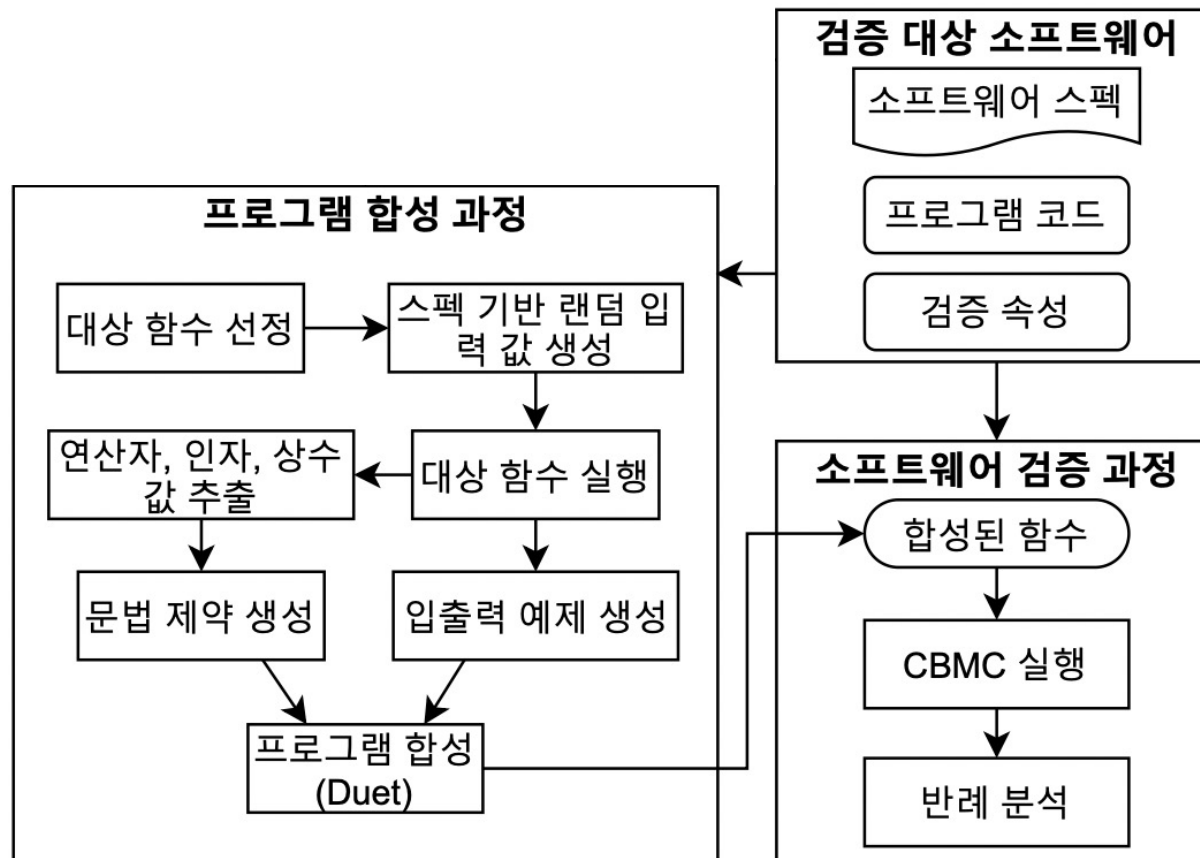
기대 역할:

1. 기존 함수와 비슷하게 동작하지만 더 단순한 형태의 함수를 합성.
2. 기존 함수를 대체하여 검증 시간을 단축 시킬 수 있을 것으로 기대.

한계:

1. 그러나 일종의 under-approximation 기법이기 때문에, 검증 결과의 신뢰도를 포기하는 대신 검증 시간을 단축함.
2. 검증 결과의 신뢰도는 합성된 함수의 정확도와 직결되어 있음.

# 프로그램 합성 기법의 구체적인 적용 방법



# 실험 1: 기존 합성 기법의 정확도 측정

## ➤ 사례: Object Follower

- 근처의 표적 물체를 카메라 센서로 포착하여 따라 이동하는 로봇 자동차.

## ➤ 실험: Duet\*을 사용

- 각 함수 별로 인자, 상수, 연산자를 수집하여 문법 제약 작성.
- 각 함수 별로 랜덤 입출력 예제 10개를 Duet에게 주고 합성.
- 별도로 준비한 테스트 입/출력 예제 1,000개로 정확도를 측정하고 코드 사이즈도 측정.

## ➤ 결과:

- 대부분의 합성된 함수가 검증에 사용할 수 없을 정도로 정확도가 매우 낮았음.
- 이는 큰 사이즈의 배열이 입력으로 사용되어 입력 범위가 매우 커지면서 합성 난이도가 증가했기 때문.

| 대상 함수          | 정확도   | 사이즈 |
|----------------|-------|-----|
| getX           | 1.5%  | 50  |
| getArea        | 1.1%  | 94  |
| getbiggestrect | 64.7% | 16  |
| median_filter  | 21.8% | 26  |
| fisqrt         | 9.7%  | 82  |

\*W. Lee, "Combining the Top-down Propagation and Bottom-up Enumeration for Inductive Program Synthesis," Principles of Programming Languages, 2021

# 해결 방법: 배열 인덱스 계산식 분석

오직 배열 인덱스 계산식에서만 사용되는 입력 변수: **rectindex**을 찾음.

```
static int nextcamda
```

0 ~ 7

(1)에서 접근 되는  
원소들의 입력 범위: 0~144

```
int getX(int rectindex) {  
(1)  int xul = nextcamdata[5 * rectindex + 2];  
(2)  int xlr = nextcamdata[5 * rectindex + 4];  
(3)  return (xlr + xul) / 2;  
}
```

(2)에서 접근 되는  
원소들의 입력 범위: 0~88

따라서, 임의의 **rectindex** 값 (예: 0)을 사용해도 (3)에서 반환되는 출력 범위가 동일함.

# 해결 방법: 배열 인덱스 계산식 분석

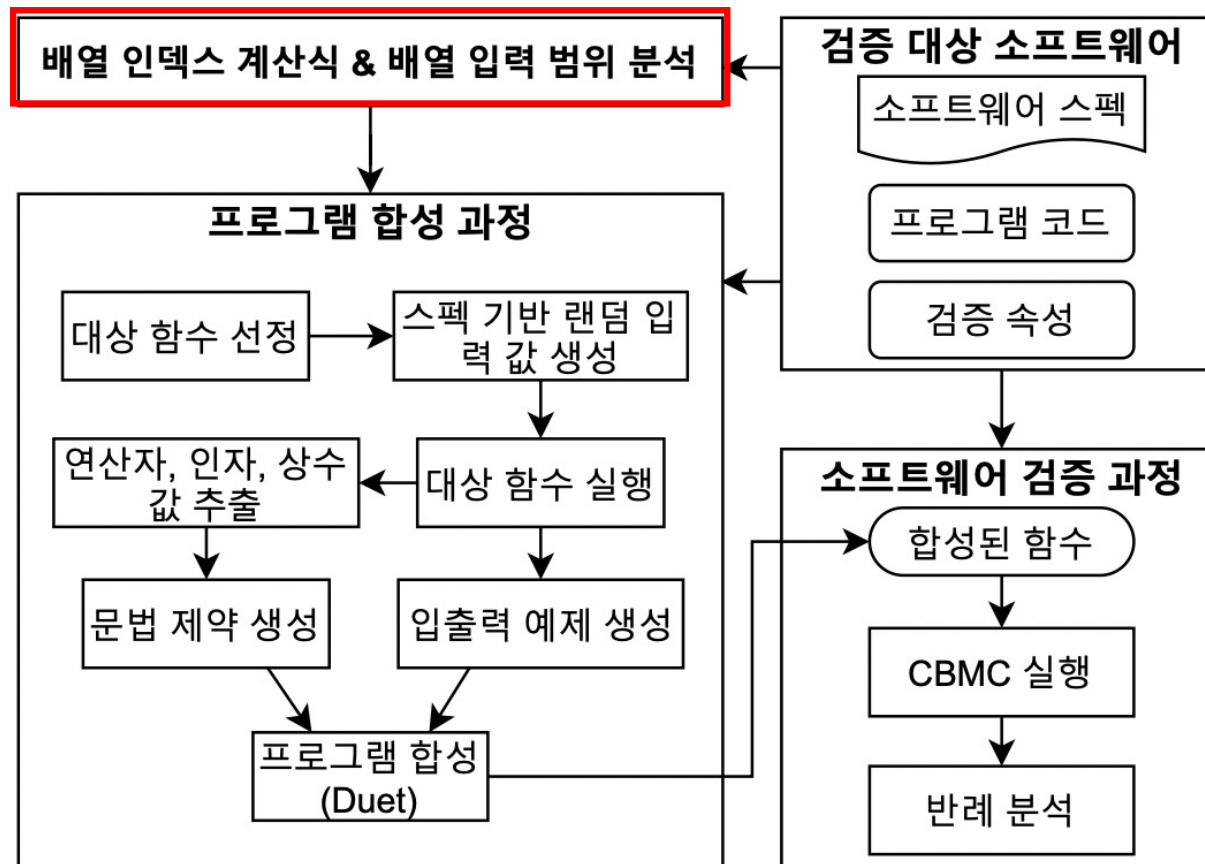
- rectindex가 0으로 고정되면서 getX가 호출된 라인 (9)을 기준으로 가장 최근에 rectindex를 정의한 라인 (4)이 rectindex = 0;로 대체됨.
- 위와 같은 방법으로 getX와 getArea 함수의 입력 범위를 대폭 줄일 수 있음.
- 사용되지 않는 배열 원소를 합성 스펙에서 제외함으로써 합성 난이도가 대폭 감소함.
- 그 결과 합성 정확도가 100%로 향상됨. (줄어든 입력 범위 기준)

| 대상 함수          | 전체 입력 범위 |     | 인덱스 계산식 분석 후  |     |
|----------------|----------|-----|---------------|-----|
|                | 정확도      | 사이즈 | 정확도           | 사이즈 |
| getX           | 1.5%     | 50  | 100%          | 5   |
| getArea        | 1.1%     | 94  | 100%          | 28  |
| getbiggestrect | 64.7%    | 16  | return 0으로 대체 |     |
| median_filter  | 21.8%    | 26  | N/A           |     |
| fisqrt         | 9.7%     | 82  | N/A           |     |

```

(4) rectindex = getbiggestrect(0, -1);
(5) if (rectindex >= 0) {
(6)     area = getArea(rectindex);
(7)     area = median_filter(area);
(8)     size = fisqrt(area);
(9)     x = getX(rectindex);
    }
  
```

# 배열 인덱스 분석이 추가된 프로세스



## 실험 2: 검증 시간 및 신뢰도 비교

검증 효율 비교를 위해서 다음 3개의 코드의 검증 시간과 신뢰도를 비교

- (1) 기존 코드, (2) 정확도 64.7%인 getbiggestrect 함수를 대체했을 때의 코드, (3) 배열 인덱스 계산식 분석 결과를 적용했을 때의 코드를 CBMC에 적용.

- 검증 속성:

- ‘일정 이상의 속력일 때 급회전을 하면 안 된다.’ (p1, p2),
- ‘급정거 시 특정 힘 이상으로 감속하면 안 된다.’ (p3) 등

- unwind: 120 (모든 거짓 속성이 반례를 생성할 수 있는 횟수)

실험 환경: Ubuntu 18.04, 3.3-GHz Intel Xeon W-2155 CPU, 256GB 메모리, CBMC 버전: 5.11

## 실험 2: 검증 시간 및 신뢰도 비교

| 속성 | (1) 기존 함수 |       | (2) getbiggestrect 함수만 대체 |       |          | (3) 배열 인덱스 계산식 분석 후 |       |           |
|----|-----------|-------|---------------------------|-------|----------|---------------------|-------|-----------|
|    | 시간 (초)    | 검증 결과 | 시간(초)                     | 검증 결과 | 신뢰 여부    | 시간 (초)              | 검증 결과 | 신뢰 여부     |
| p1 | 10120     | TRUE  | 5621                      | TRUE  | NO (0/0) | 5325                | TRUE  | NO (0/0)  |
| p2 | 57536     | FALSE | 6246                      | FALSE | NO (0/1) | 5848                | FALSE | YES (2/2) |
| p3 | 9578      | TRUE  | 5632                      | TRUE  | NO (0/0) | 5331                | TRUE  | NO (0/0)  |
| p4 | 68143     | FALSE | 6135                      | FALSE | NO (0/1) | 5651                | FALSE | YES (2/2) |
| p5 | 66315     | FALSE | 6171                      | FALSE | NO (0/1) | 6106                | FALSE | YES (2/2) |
| p6 | 76757     | FALSE | 6633                      | FALSE | NO (0/1) | 5759                | FALSE | YES (2/2) |

- (2)와 (3) 모두 복잡한 함수(getbiggestrect)를 단순하게 합성된 함수로 대체한 결과 (1)에 비해 검증 시간이 최대 10배 이상 감소하였음.
- (2)의 경우 모든 거짓 속성의 반례에서 부정확한 출력 값을 반환하여 신뢰할 수 없었음.
- 반면 (3)의 경우 입력 범위가 감소되어 합성된 함수의 정확도가 충분히 높아 모든 거짓 속성의 검증 결과를 신뢰할 수 있었음.

\*신뢰 여부: (합성된 함수가 정확한 출력 값을 반환한 횟수)/(반례에서 합성된 함수의 호출 횟수)

# 결론 및 향후 연구

## 결론:

- 프로그램 합성 기법을 이용하여 CBMC 검증 시간을 최대 10배 이상 단축하였음.
- 배열의 인덱스 계산식을 분석했을 때 합성 결과의 정확도 향상이 이루어져 검증 결과의 신뢰도가 증가하였음.

## 향후 연구:

- 하지만 소개한 분석 방법은 접근된 배열 원소들이 서로 같은 입력 범위를 가지고 있어야 하기 때문에 다른 사례에 적용하기엔 제한적임.
- 따라서 일반적으로 적용될 수 있는 분석 기법을 만드는 것이 향후 연구임.