

임베디드 소프트웨어 도메인에서의 프로그램 합성 기법의 성능 확인을 위한 사례연구

김요엘, 최윤자
경북대학교 컴퓨터학부

kimyoel2305@gmail.com, yuchoi76@knu.ac.kr

임베디드 소프트웨어 검증의 어려움

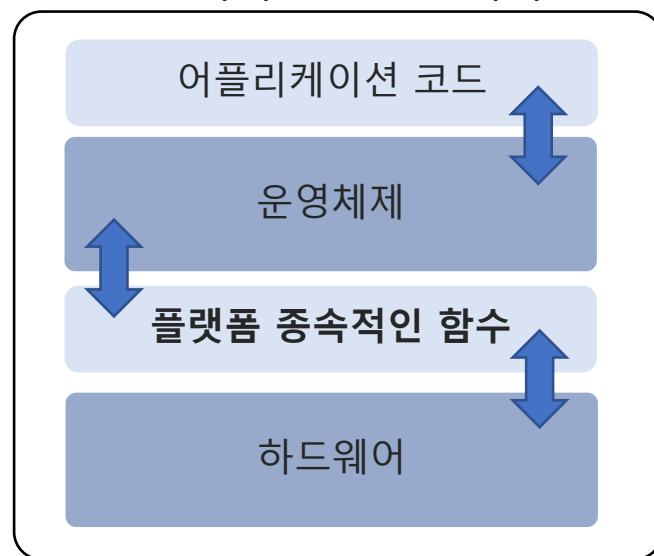
➤ 플랫폼 종속적인 함수

- 하드웨어를 직접 접근하므로 검증에 포함하기 어려움
- 수작업으로 추상화 모델을 만들어 검증에 포함

수작업을 자동화하고 검증 복잡도를 낮추고 싶다면?

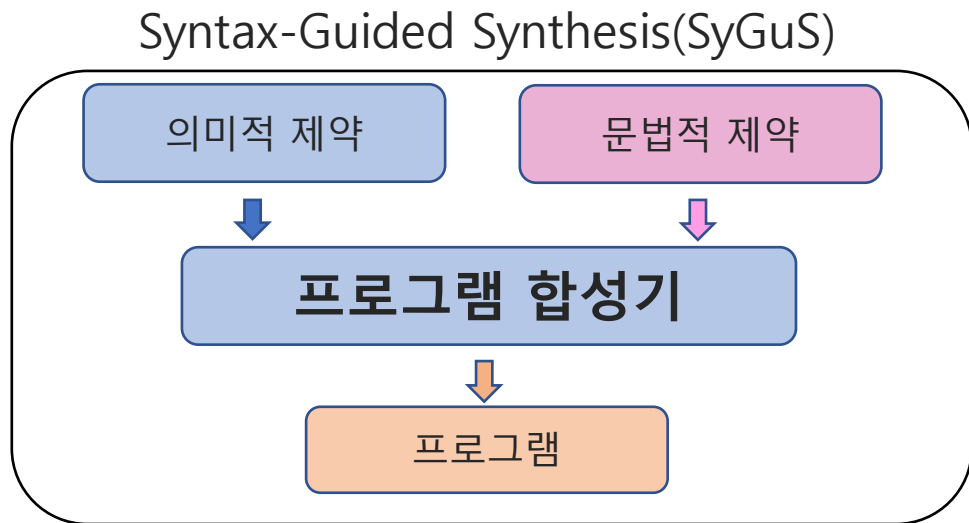
✓ 프로그램 합성 기법

임베디드 소프트웨어



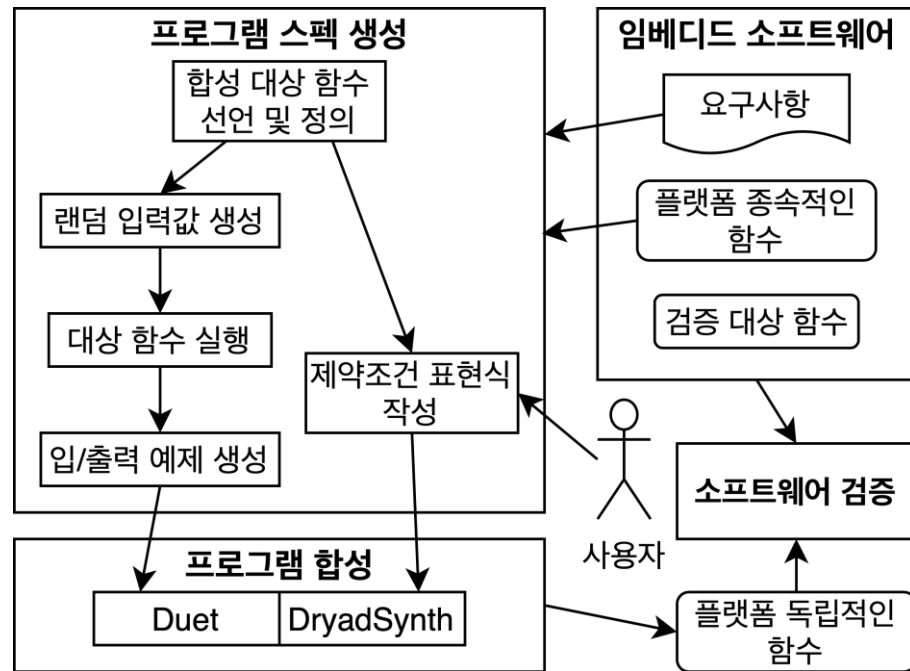
프로그램 합성 (Program Synthesis)

- 다양한 형태의 제약조건을 가지고 프로그램 탐색 공간에서 제약조건을 만족하는 실행 가능한 프로그램을 찾아내는 기법



프로그램 합성 기법의 기대 역할

- 소프트웨어 테스트, 검증에 사용
 - 테스트 스텝, 모의 객체(test mock)*
 - 복잡한 함수를 합성하여 검증에 대신 사용
- 임베디드 소프트웨어 도메인:
 - ✓ 플랫폼 종속적인 함수를 플랫폼 독립적인 함수로 변환



*M. Fazzini, A. Gorla, A. Orso, "A Framework for Automated Test Mocking of Mobile Apps," Automated Software Engineering, 2020

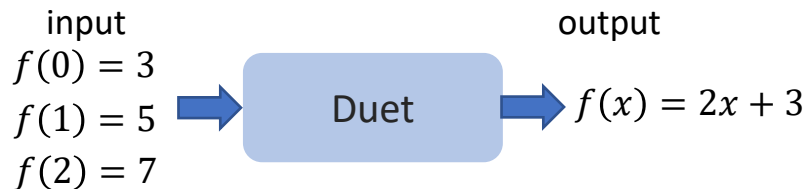
연구의 목적

- ✓ 합성된 프로그램의 정확도와 복잡도 측정을 통하여 임베디드 소프트웨어 검증에서의 **활용 가능성 판단**
- 프로그램 합성 기법의 **실제 적용 사례** 연구
- 다양한 합성 기법 중 특성이 서로 다른 합성 기법을 선택하여 **장단점** 정리
- 어떤 합성 기법을 선택해야 하는지에 대한 **가이드라인** 제시

프로그램 합성 기법 비교

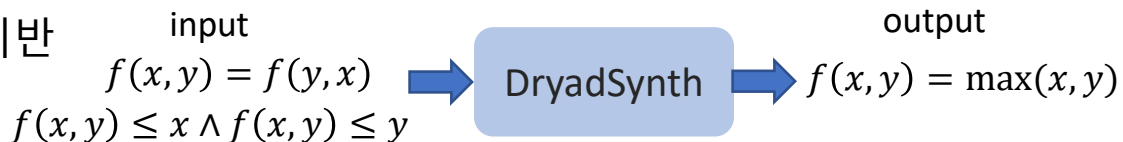
➤ Duet*

- 입/출력 예제 기반
- 스펙(제약조건) 생성 **자동화** 가능



➤ DryadSynth**

- 입/출력 사이의 논리적 관계 기반
- 사용자가 직접 스펙 생성



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

**K. Huang, X. Qiu, P. Shen, et al., "Reconciling Enumerative and Deductive Synthesis," Programming Language Design and Implementation, 2020

프로그램 합성 기법 비교

➤ Duet

- 입/출력 예제 기반
- 스펙 생성 **자동화** 가능
- **과적합(overfitting)** 발생 가능성

표 1 Duet의 합성 결과

함수 명	입/출력 예제 수	합성 시간	합성 사이즈	정확도	연산/상수 합계	출력 범위
stock	100개	0.3초	24	100%	10개	0~4
area	100개	0.1초	40	100%	5개	0~
median	600개	4.5초	461	100%	4개	int
bool	500개	9.1초	282	99.5%	7개	0~1
max	500개	6.1초	1821	52.2%	10개	0~7
taxi	200개	284.9초	3262	6.5%	15개	3300~

➤ DryadSynth

- 입/출력 사이의 논리적 관계 기반
- **사용자가 직접** 스펙 생성
- Duet의 성능이 좋지 않았던 함수도 **합성 가능**

```
(define-fun macro((meter Int) (second Int)) Int (  
  + 3300 (+ (* 100 (/ meter 131)) (* 100 (/ second 31)))  
)  
(constraint (= (taxi meter second time)  
  (ite (or (and (<= 0 time) (<= time 4)) (= time 24))  
    (+ (macro meter second) (* 2 (/ (macro meter second) 10)))  
    (macro meter second`)  
  )  
)  
)
```

그림 2 taxi 함수의 제약조건이 담긴 DryadSynth의 스펙

사례: Object Follower

- 근처의 표적 물체를 따라 이동하면서도 항상 표적으로부터 일정 거리를 유지하는 로봇
- 실험 대상 함수는 모두 카메라에서 측정된 센서 값(nxtcamdata)을 해석하는 함수들임
- 배열과 전역 변수를 사용하는 등 전형적인 임베디드 소프트웨어의 형태를 지님
- 각 합성 기법이 Object follower의 함수들을 얼마나 잘 합성하는지 확인하기 위한 실험

```
typedef unsigned char U8;
static U8 nxtcamdata[41];
int getWidth(U8 rectindex);
int getHeight(U8 rectindex);
int getArea(U8 rectindex);
int getbiggestrect(U8 pcolorid, int minarea);

#define MYABS(x) (((x) >= 0)? (x):- (x));
int getWidth(U8 rectindex) {
    int xul = (int) nxtcamdata[5 * rectindex + 1 + 1];
    int xlr = (int) nxtcamdata[5 * rectindex + 1 + 3];
    return MYABS(xlr - xul);
}

int getHeight(U8 rectindex) {
    int yul = (int) nxtcamdata[5 * rectindex + 1 + 2];
    int ylr = (int) nxtcamdata[5 * rectindex + 1 + 4];
    return MYABS(ylr - yul);
}

int getArea(U8 rectindex) {
    return getWidth(rectindex) * getHeight(rectindex);
}

int getbiggestrect(U8 pcolorid, int pminarea) {
    int rectindex = -1;
    int maxarea = pminarea;
    for (int i = 0; i < nxtcamdata[0]; i++) {
        int colorid = (int) nxtcamdata[1 + 5 * i + 0];
        if (colorid == pcolorid) {
            int area = getArea(i);
            if (area >= maxarea) {
                maxarea = area;
                rectindex = i;
            }
        }
    }
    return rectindex;
}
```

실험 방법

- 실험 대상 함수는 모두 nextcamdata라는 전역변수를 사용
 - 프로그램 스펙에서 해당 전역변수를 매개변수로 추가
- Duet과 DryadSynth 모두 배열을 지원하지 않음
 - 배열의 각 요소(element)들을 정수형 일반 변수로 변환하여 스펙에 추가 (41개)
- Duet
 - 랜덤 입/출력 예제를 100개씩 생성해 스펙에 추가하고 합성했을 때 가장 정확도가 높은 합성 결과를 사용
- DryadSynth
 - 대상 함수의 코드를 보면서 스펙을 직접 작성

실험 결과

함수 명	Duet						DryadSynth		
	예제 수	합성 시간	합성 사이즈	정확도	컴포 넌트	연산/상수 합계	합성 시간	합성 사이즈	정확도
getWidth	200개	177.7초	1341	2.7%	3	6개	0.3초	687	100%
getHeight	300개	458.1초	2006	2.2%	3	6개	0.3초	687	100%
getArea	300개	325.9초	3129	4.8%	3	6개	0.4초	2292	100%
getbiggestrect	100개	5.2초	416	37.9%	1	15개	3.0초	56411	100%

➤ Duet

- getWidth/Height/Area:
 - 출력값의 경우의 수가 각각 256, 256, 65026개
 - 매개변수 42개 중 사용되지 않는 37개 변수가 합성을 방해
- getbiggestrect:
 - 출력값의 경우의 수가 -1~7로 9가지
 - 사용 가능한 연산자와 상수 값이 많음

➤ DryadSynth

- 정확도는 100%
- 매우 큰 사이즈(복잡도)
- 매우 높은 스펙 작성 난이도 (getbiggestrect)

실험 결과

합성 기법	장단점
Duet	‘간단한’ 함수의 경우 성능이 뛰어남
	예제로 스펙을 생성하기 때문에 자동화가 쉬움
	‘복잡한’ 함수의 경우 성능이 낮음
DryadSynth	‘복잡한’ 함수를 합성할 수 있고, 성능이 뛰어남
	스펙 작성을 직접 해야 하기 때문에 자동화가 힘들
	스펙 작성이 어렵고 시간이 오래 걸림

➤ ‘간단한’ 함수

- 프로그램 탐색 공간이 **좁은** 경우
- 출력값의 범위가 **몇 가지로 고정**되는 경우

➤ ‘복잡한’ 함수

- 출력값의 범위가 **넓은** 경우
- 매개변수, 사용 가능한 연산자 및 상수 값이 **많은** 경우
- 사용되지 않는 매개변수가 합성에 **방해**를 하는 경우

결론 및 향후 연구

- ✓ 실험 결과 '복잡한' 함수를 합성해야 할 경우 DryadSynth가 좋으나 **매우 긴 스펙 작성 시간**이라는 단점이 존재
- ✓ 그에 비해 **스펙 생성 자동화**가 가능하다는 점에서 Duet의 합성 가능성과 정확도를 높이는 방법을 조금 더 연구해볼 필요 있음
 - Duet에서 실험 시 사용한 입력값 생성 방식이 랜덤하기 때문에 결과가 좋지 않음
- **향후 연구**
 - 파악된 단점을 극복하고, 프로그램 합성 기법이 검증에 사용될 수 있는 수준까지 정확도를 끌어 올릴 수 있는 방법 연구