프로그램 합성과 소스코드 내 결함 위치 식별을 이용한 자동 디버거 개발

팀명: WAS
팀원: 한용진
오재혁
응웬딩흐엉

2020.12.09

Contents

1 프로젝트 개요

프로젝트 추진배경

필요성 및 중요성

- 비전공자의 프로그래밍 학습에 대한 수요가 꾸준히 증가함
- COVID-19 사태로 인해 언택트 문화의 확산에 따른 **비대면 학습 증가**하고 있음
- 1:多 관계에 있는 교육 현장에서 학생들에게 피드백 제공에 어려움
- 기존 디버거는 문법 오류만 알려줄 뿐 **의미론적으로 오류인 코드를 어떻게 수정해야 하는지 알려주지 않음**
- 소프트웨어 산업에서 **디버깅에 소요되는 시간과 비용은 막대함**

독창성

- 프로그램 합성 관련 **국내 특허 1건 존재함**
- 자동으로 소스코드의 오류를 찾고 수정하는 특허는 **국내에 존재하지 않음**

프로젝트 목표 및 구현 기술

프로젝트 목표

- 본 프로젝트의 목표는 **결함 위치 식별 (Fault Localization)을 통해 소스코드의 오류 위치를 식별**하고, **프로그램 합성 (Program Synthesis)을 이용하여 식별된 소스코드의 오류를 수정**하는 자동 디버거를 개발하는 것임

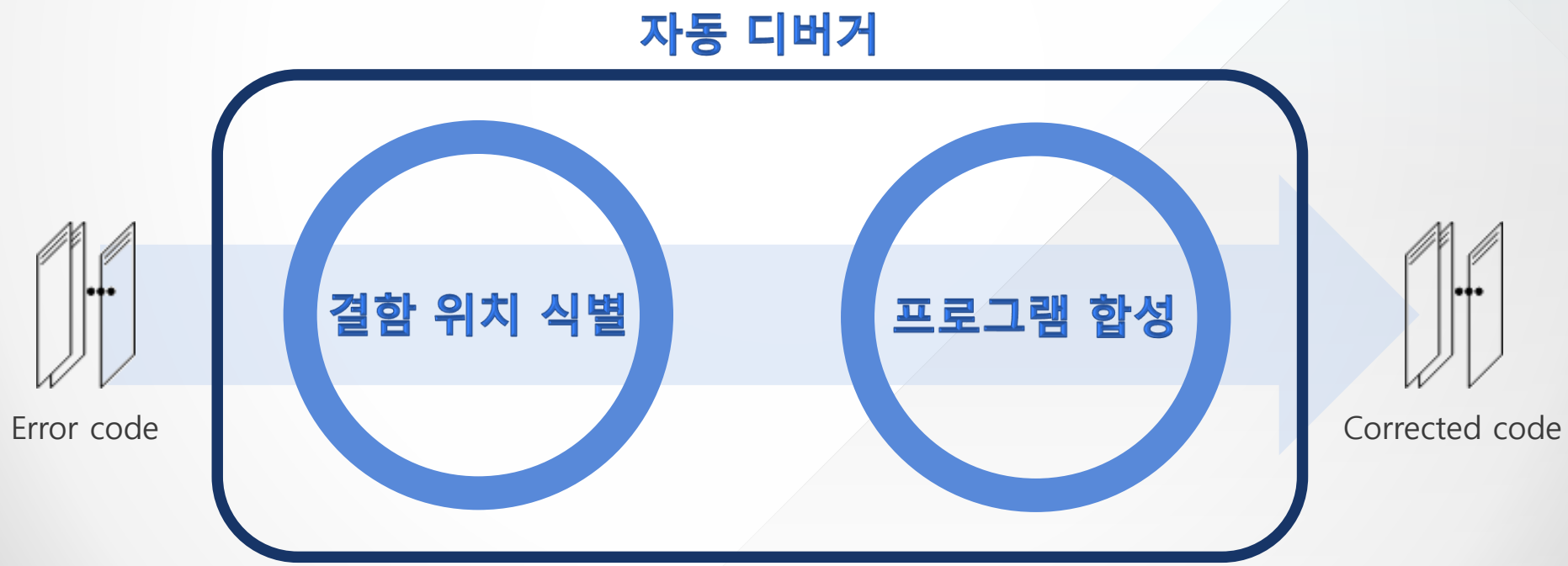


그림 1. 자동 디버거 구조

2 자동 디버거 설계

언어 설계

Turing Complete

- *튜링 기계와 동일한 계산능력을 갖음으로써 계산적 문제를 풀 수 있음을 의미함
- 반복문
- 조건문

```
OP := “+” | “-” | “*” | “/” | “%”
INEQ := “==“ | “!=” | “<“ | “>”
NUM := \d
VAR := [a-z]+(\[[a-z]+\])?
OPERAND := NUM | VAR
EXP := OPERAND | OPERAND OP OPERAND | “??”
TRUTH := “true” | “false” | OPERAND INEQ OPERAND | “!!”
CLAUSE := VAR “=” EXP | CLAUSE CLAUSE |
          “if” “(” TRUTH “)” “:” CLAUSE (“else” “:” CLAUSE)? |
          “while” “(” TRUTH “)” “:” CLAUSE
```

그림 2. 튜링 완전한 언어 설계

*튜링 기계: 긴 테이프에 쓰여 있는 여러 가지 기호들을 일정한 규칙에 따라 바꾸는 기계로서 적당한 규칙과 기호를 입력하면 일반적인 컴퓨터의 알고리즘을 수행할 수 있음.

자동 디버거 모델 세부 설계 – 결함 위치 식별

향상된 정확도를 갖는 결함 위치 식별 모듈 구현

테스트케이스 구성

- 테스트케이스 구성에 따라 **결함 위치에 대한 의심도 값이 달라질 수 있음**
- **여러 개의 의심도 계산식을 사용**

동일한 커버리지

- 동일한 커버리지를 갖는 테스트케이스가 존재하면 **해당 코드에 가중치가 증가됨**
- 동일한 커버리지를 갖는 테스트케이스를 **하나의 테스트케이스로 간주함**

Tarantula: $Sus(s) = \frac{failed(s)/totalfailed}{\frac{failed(s)}{totalfailed}+\frac{passed(s)}{totalpassed}}$

Ochiai: $Sus(s) = \frac{failed(s)}{\sqrt{totalfailed*(failed(s)+passed(s))}}$

Op2: $Sus(s) = failed(s) - \frac{passed(s)}{totalpassed+1}$

그림 3. 의심도 계산식 예시

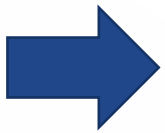
자동 디버거 모델 세부 설계 - 프로그램 합성

계산 가능한 시간 내에 오류를 수정하는 프로그램 합성 모듈 구현

프로그램 공간

- **열거형 탐색 방식은 프로그램 후보를 나열**하여 명세를 만족하는 프로그램을 찾기 때문에 **시간 복잡도가 매우 큼**
- **주어진 프로그램과 동등한 의미를 갖는 논리식을 세우고 이를 검증**

```
def max(x, y):
    if (x > y):
        z = x
    else:
        z = y
    return z
```



$$(x > y \cap z' = x) \cap (\neg(x > y) \cap z' = y)$$

그림 4. 프로그램에 대응되는 논리식 생성 예시

자동 디버거 모델

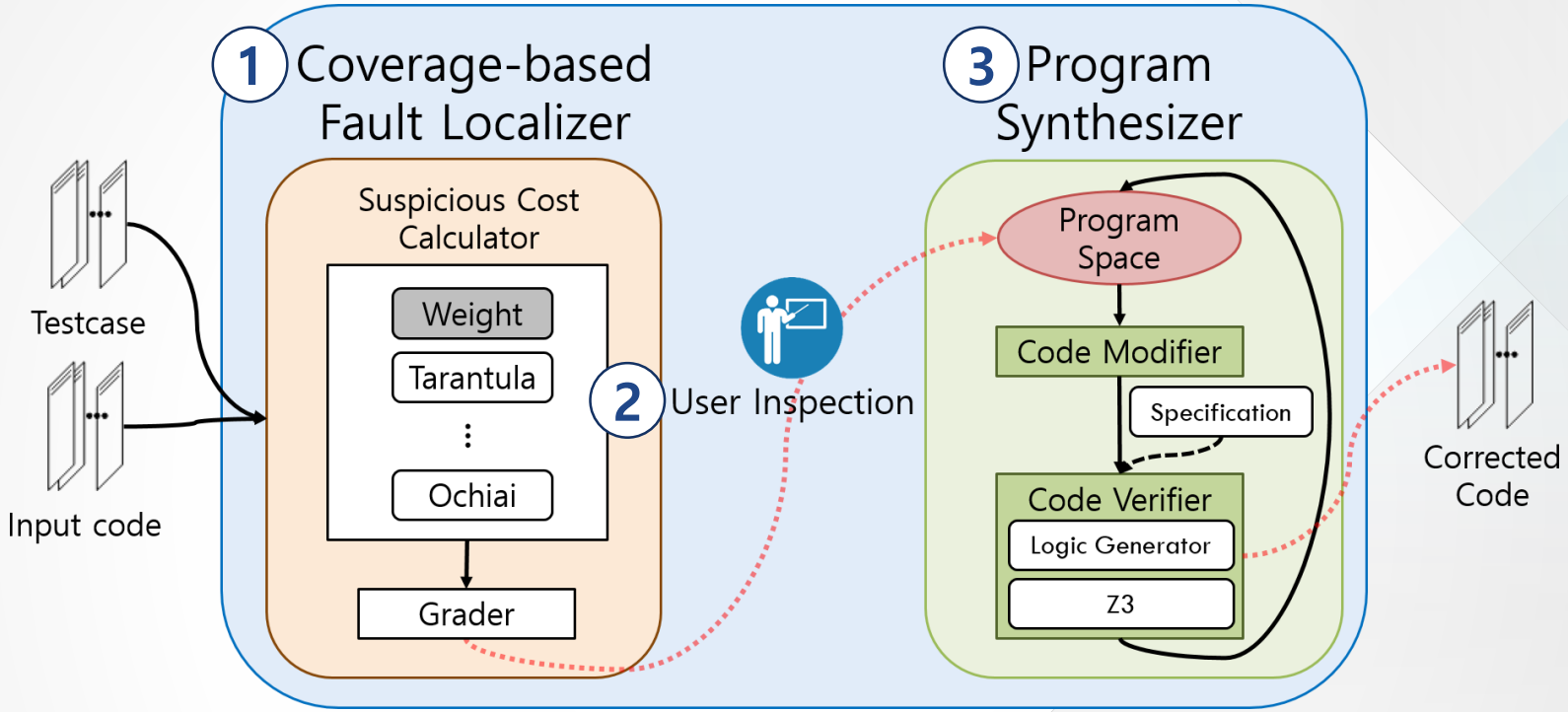


그림 5. 자동 디버거 모델

1) 여러 개의 의심도 계산식을 사용하여 **코드 의심도 계산**

2) 사용자가 계산된 의심도 값을 참고하여 **오류 코드를 결정**

3) 프로그램 합성기를 통해 주어진 명세를 만족하는 **수정된 소스코드를 생성**

3 자동 디버거 구현

프로젝트 데모

프로젝트 데모

자동 디버깅 예시 – 소스코드 및 테스트케이스

- 오류가 존재하는 소스코드 및 입출력 예제로 구성된 테스트케이스 명제 생성

```
def max(x, y):
    if (x > y):
        z = y
    else:
        z = y
    return z
```

```
(1,2), 2
(4,3), 4
(6,9), 9
(5,7), 7
(-10,15), 15
(5,-10), 5
(5,0), 5
(0,5), 5
(0,10), 10
```

그림 6. 입력 코드 및 입출력 테스트케이스

자동 디버깅 예시 – 결함 위치 식별

- 결함 위치 식별을 통해 결함 위치에 대한 의심도 계산

```
def max(x, y):
    if (x > y):
        z = y
    else:
        z = y
    return z
```

line	suspiciousness	code
1	0.349	def max(x,y):
2	0.349	if (x > y):
3	1.0	z = y
4	0.0	else:
5	0.2	z = y
6	0.349	return z

그림 7. 식별된 오류가 존재하는 코드 및 의심도 계산 결과

자동 디버깅 예시 - 프로그램 합성

- 프로그램 합성을 이용해 식별된 오류를 수정
- 입력 프로그램에 대응하는 논리식 생성
- Z3, SMT solver를 이용한 논리식 검증

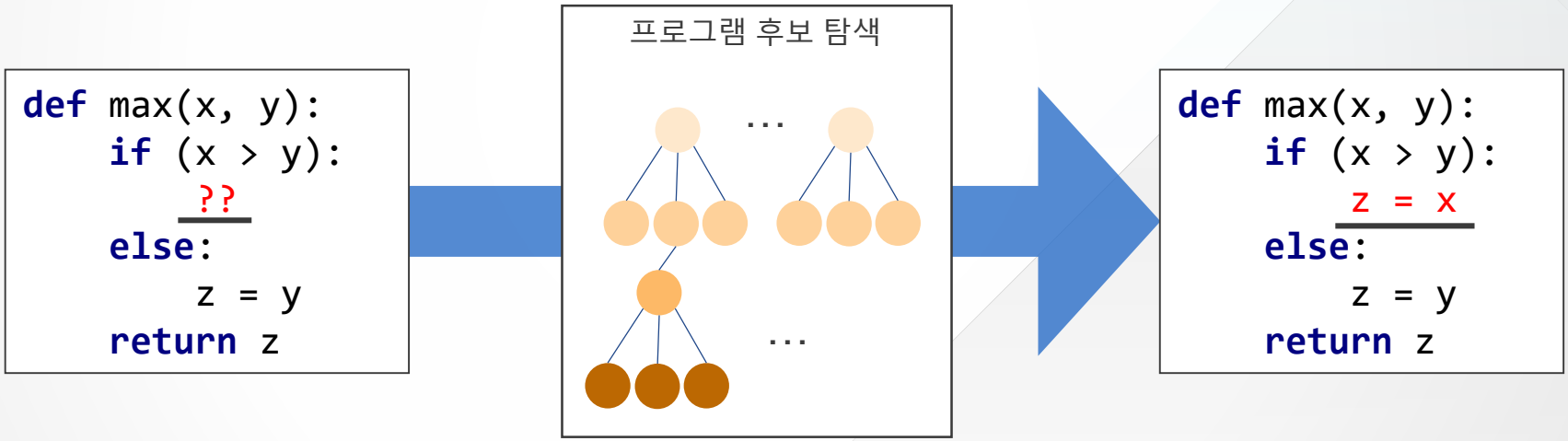
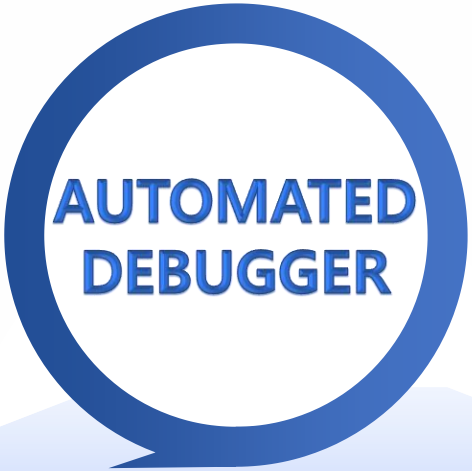


그림 8. 프로그램 합성을 통한 식별된 오류 수정 과정 예시

자동 디버거 부가기능



자동 디버깅

- 오류 코드 위치 탐지
- 오류 코드 수정

프로젝트 관리

- 개별 프로젝트 생성
- 프로젝트 다운로드
- 디버깅 기록 조회

회원 가입

- 회원가입
- 로그인

결함 위치 식별기 성능 향상 결과

- 오류 코드가 아닌 코드들에 대한 의심도 값이 감소

```
1 def max(x, y):
2     if (x > y):
3         z = y
4     else:
5         z = y
6     return z
```

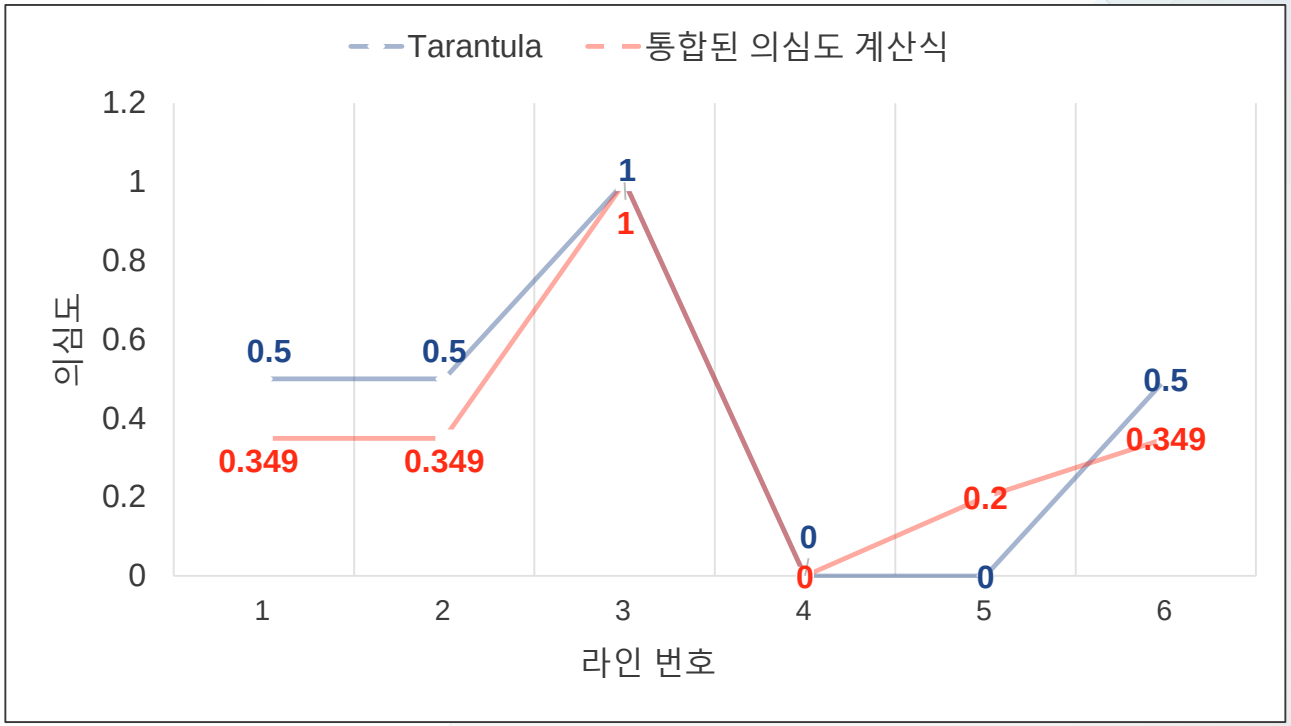


그림 9. max 알고리즘에 대한 의심도 계산식 성능 비교

결함 위치 식별기 성능 향상 결과

- 오류 코드가 아닌 코드들에 대한 의심도 값이 감소

```
1 def gcd(x, y):
2     while(y != 0):
3         if (x > y):
4             x = x - y
5         else:
6             y = y - x
7     return x
```

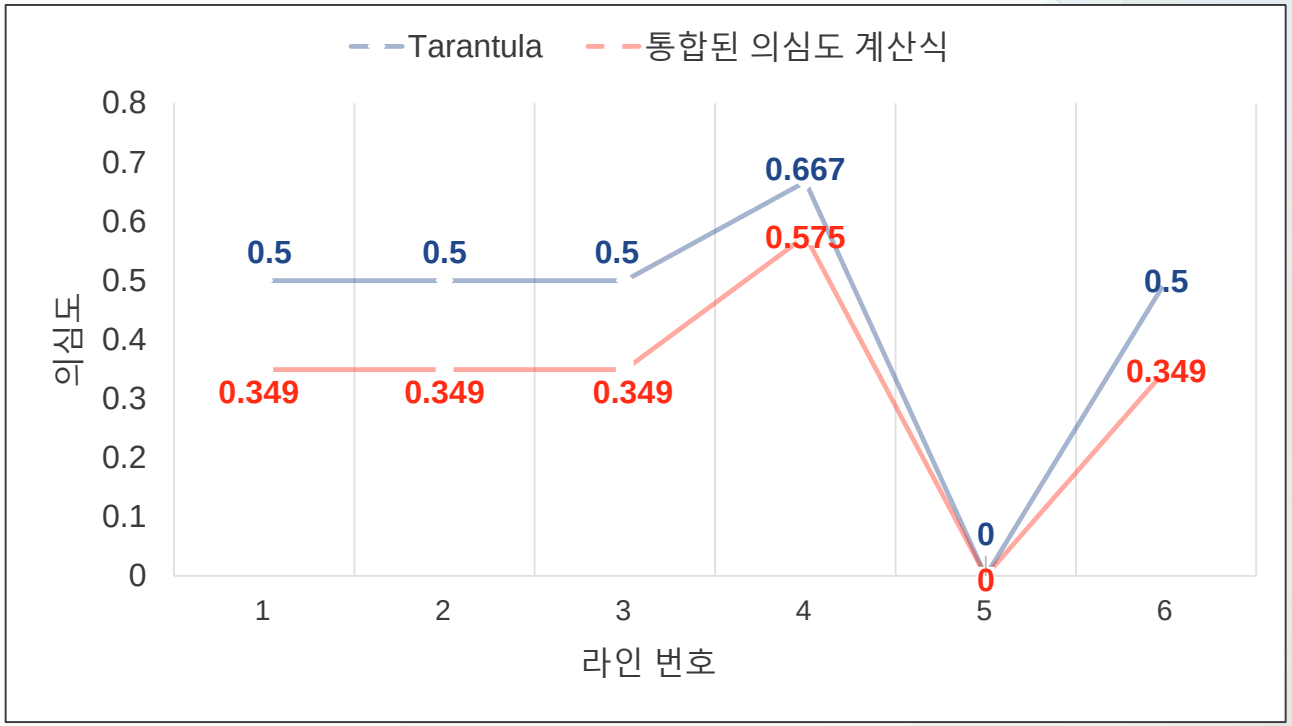


그림 10. GCD 알고리즘에 의심도 계산식 비교

결함 위치 식별기 성능 향상 결과

- 오류 코드가 아닌 코드들에 대한 의심도 값이 감소
- 오류 코드 의심도 값 증가

```
1 def gcd(x, y):
2     while(y != 0):
3         if (x > y):
4             x = x - y
5         else:
6             y = y - x
7     return x
```

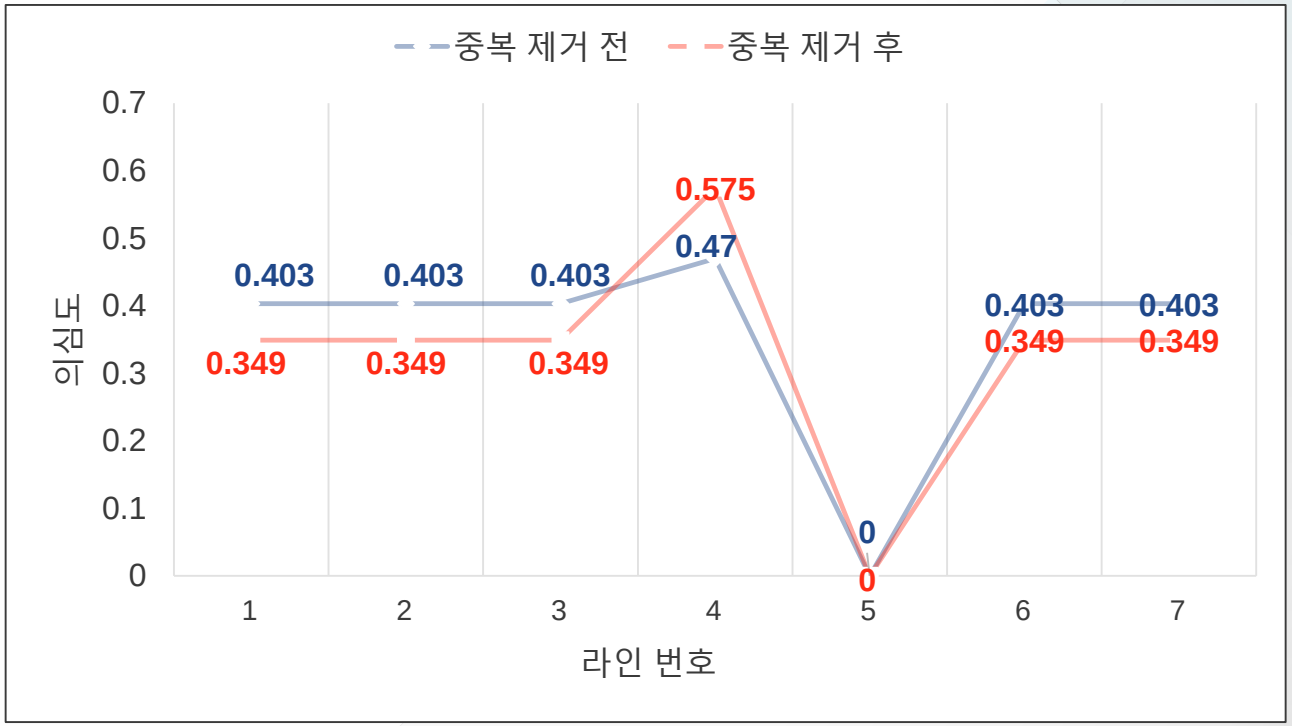


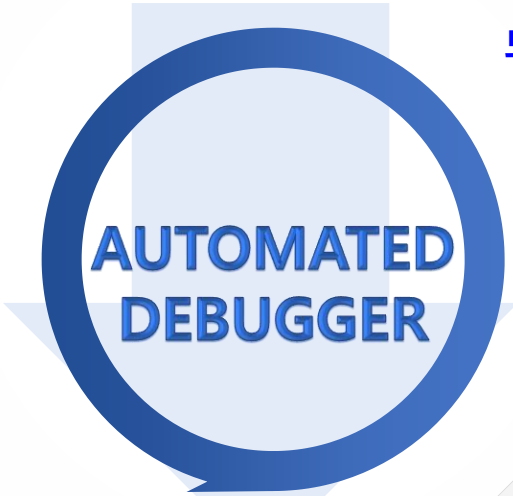
그림 11. GCD 알고리즘에 대한 중복된 커버리지 제거 전 후 비교

4 결론

기대효과

교육 분야

- COVID-19 사태로 인해 언택트 문화의 확산에 따른 **비대면 학습 수요가 증가**하고 있음
- 비전공자의 프로그래밍 학습에 대한 수요가 꾸준히 증가하는 추세임



- 자동 디버거를 통해 미숙한 프로그래머가 스스로 피드백을 받고 학습할 수 있는 **자가학습 시스템 제공**
- 1:多 관계에 있는 교육 현장에서 교사와 학생들 모두에게 **원활한 교육 환경 제공**

소프트웨어 공학 분야

- 소프트웨어 개발 주기에서 유지 보수가 차지하는 부분이 상당함
- 소프트웨어 산업에서 **디버깅에 소요되는 시간과 비용은 막대함**

- **디버깅 및 유지 보수에 대한 비용을 절감**

Q & A

감사합니다.