

# 정적 분석의 효능

허기홍

KAIST 프로그래밍 시스템 연구실





# 소개

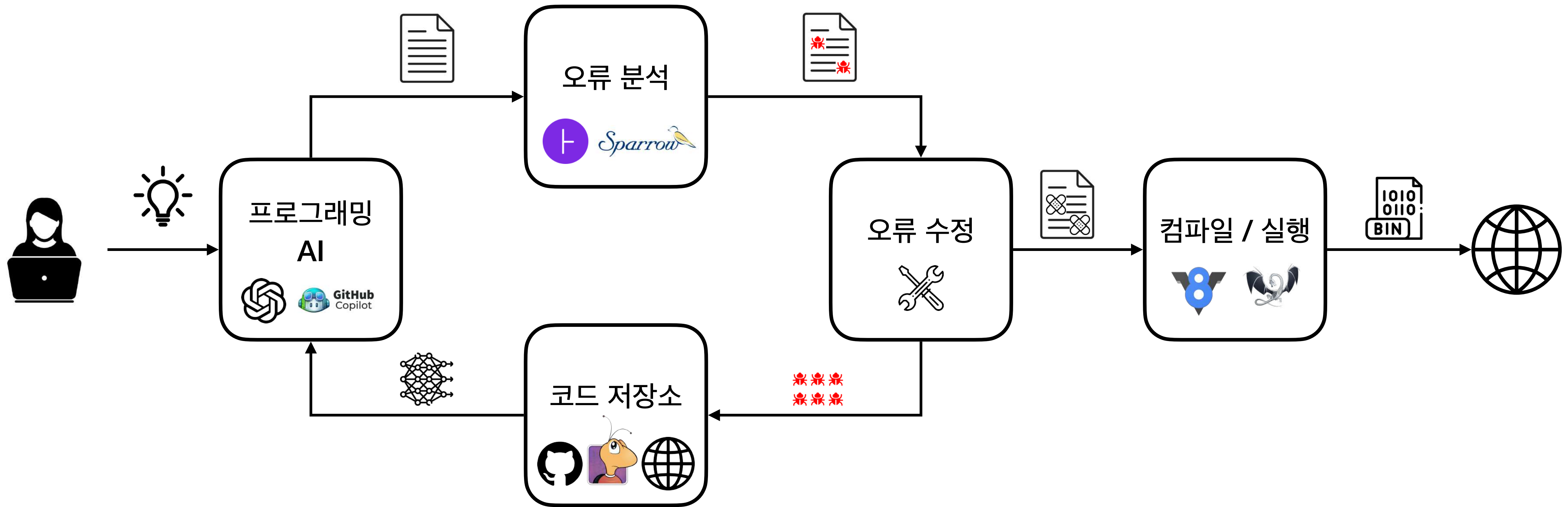


강인한 맷집, 끝없는 도전, 뜨거운 영광



# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



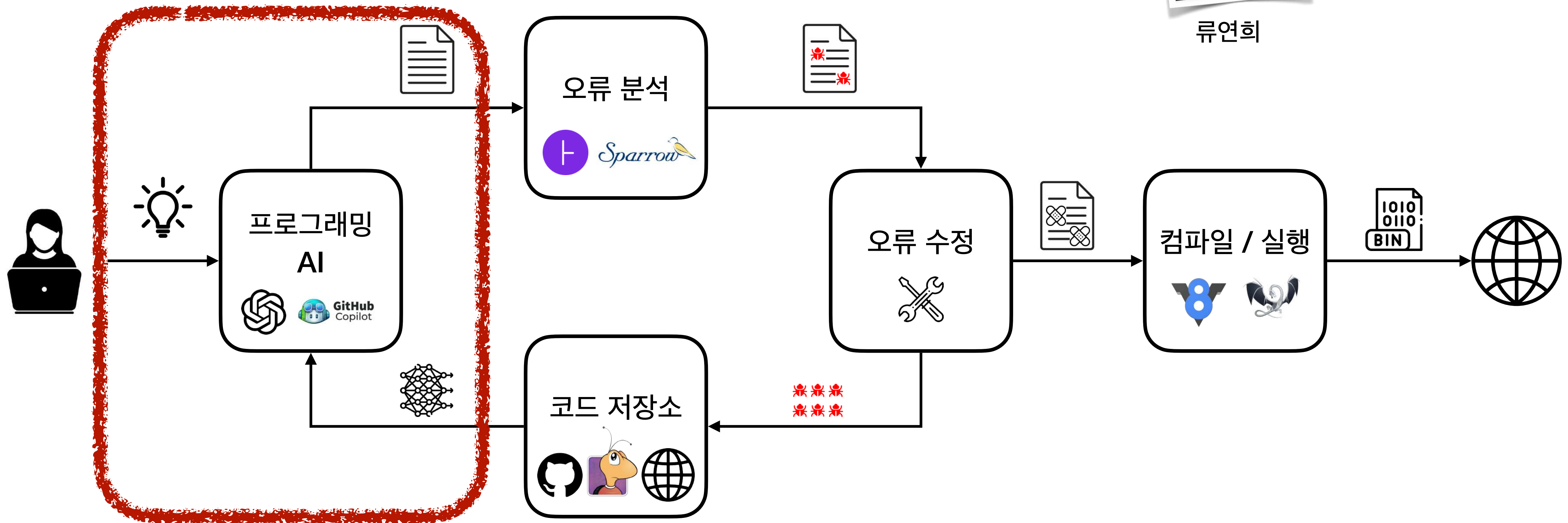
# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



## 류연희

**[포스터 발표]**  
**확률과 규칙, 친해지길 바래**



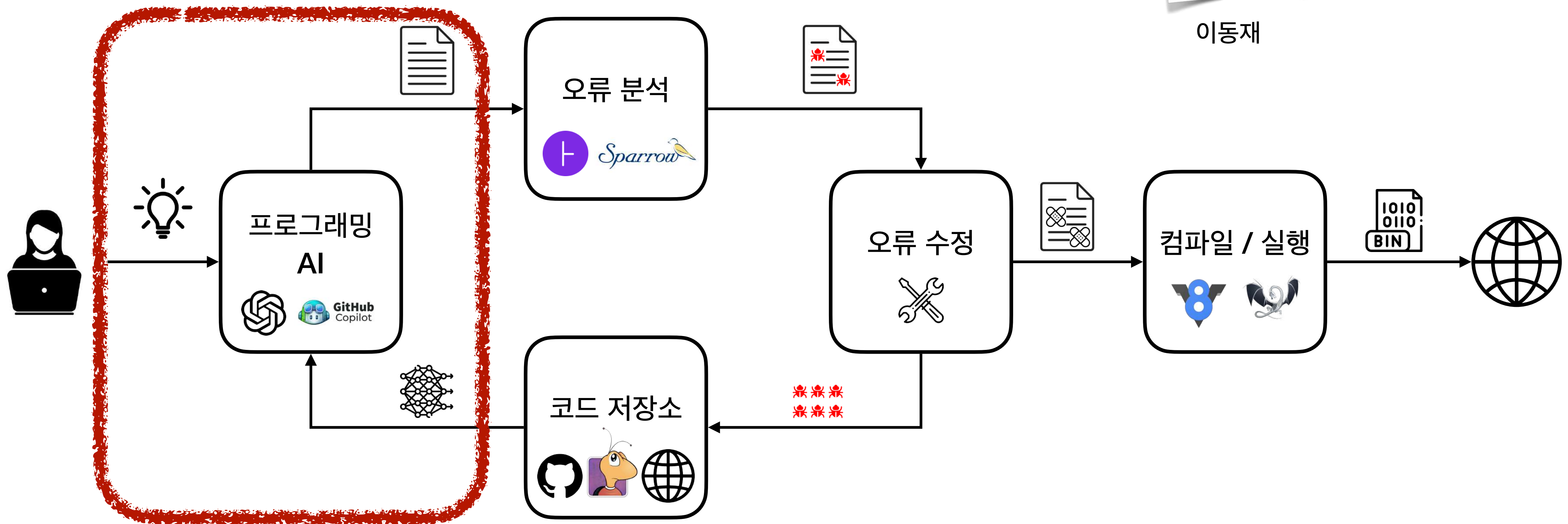
# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



이동재

[포스터 발표]  
자연어지만 엄밀한 명세가  
되고 싶어



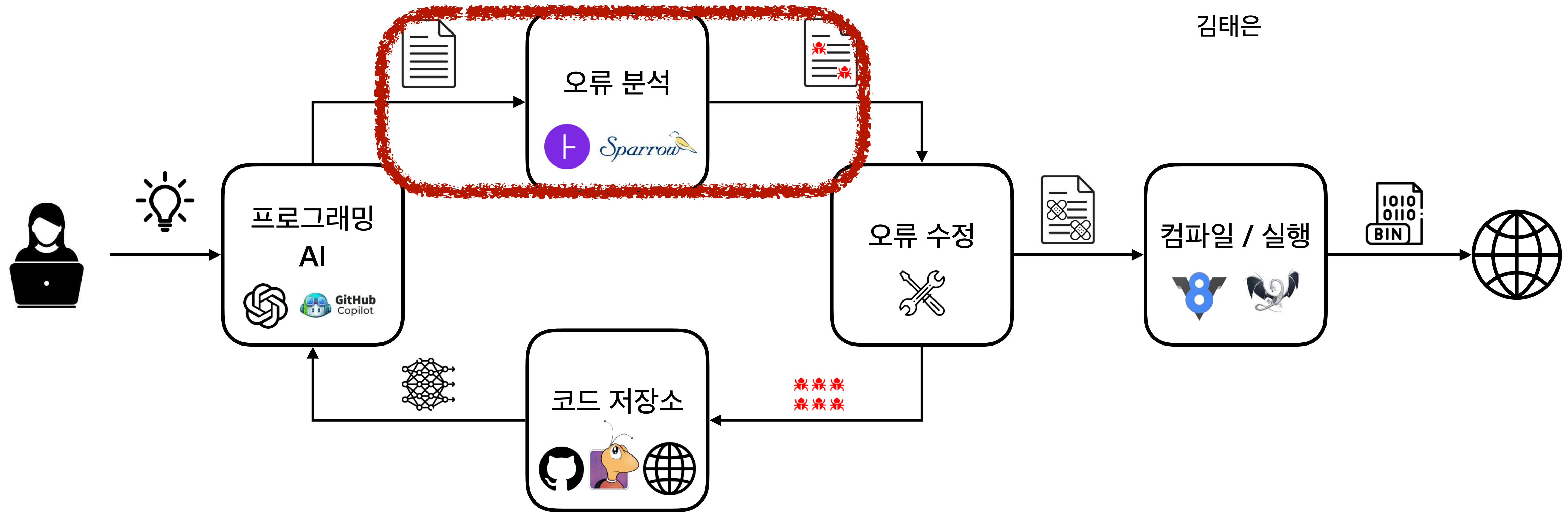
# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



김태은

[포스터 발표]  
바람둥이 VS 순애보:  
지향성 퍼징의  
목표 집중형 탐색 전략





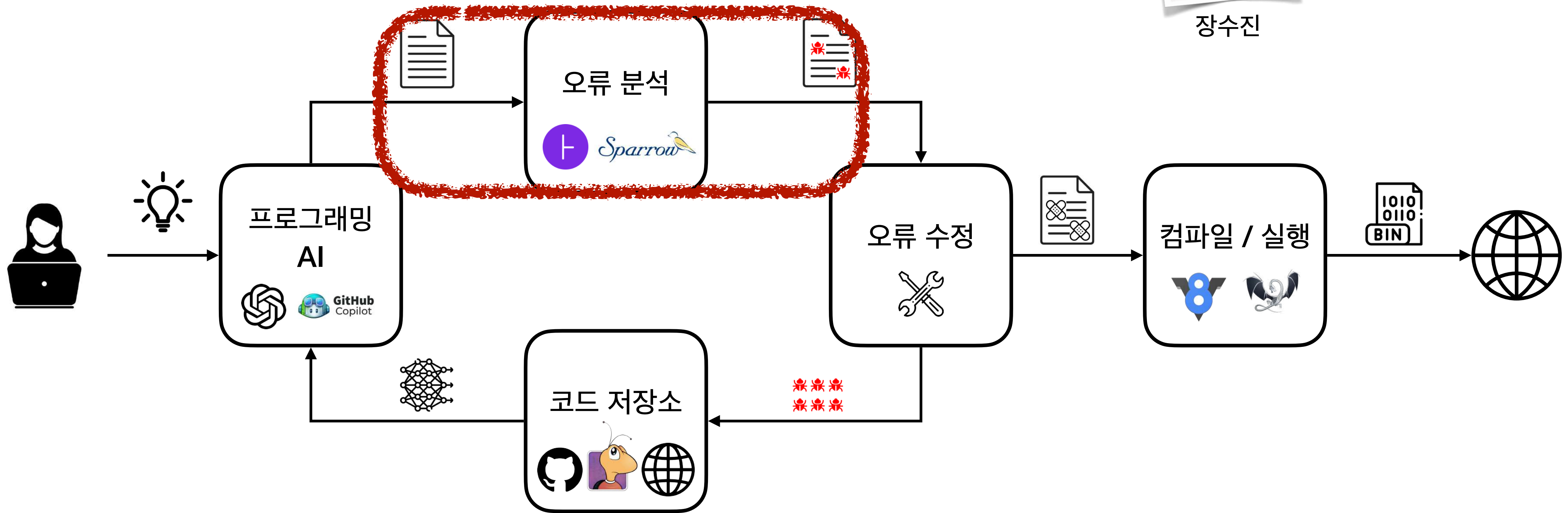
# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



장수진

[포스터 발표]  
그 단위 테스트,  
정말 다른 거 맞아?



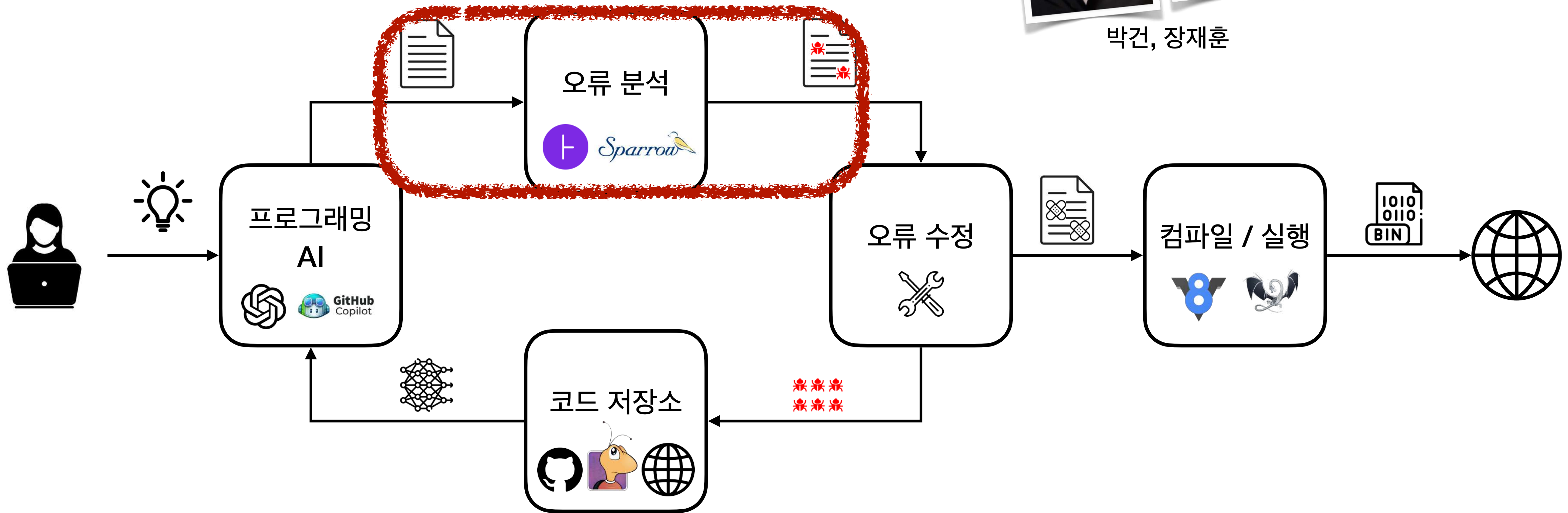
# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



박건, 장재훈

[포스터 발표]  
다중 지향성 퍼징으로  
여러 군데 오류 한방에 검사하기





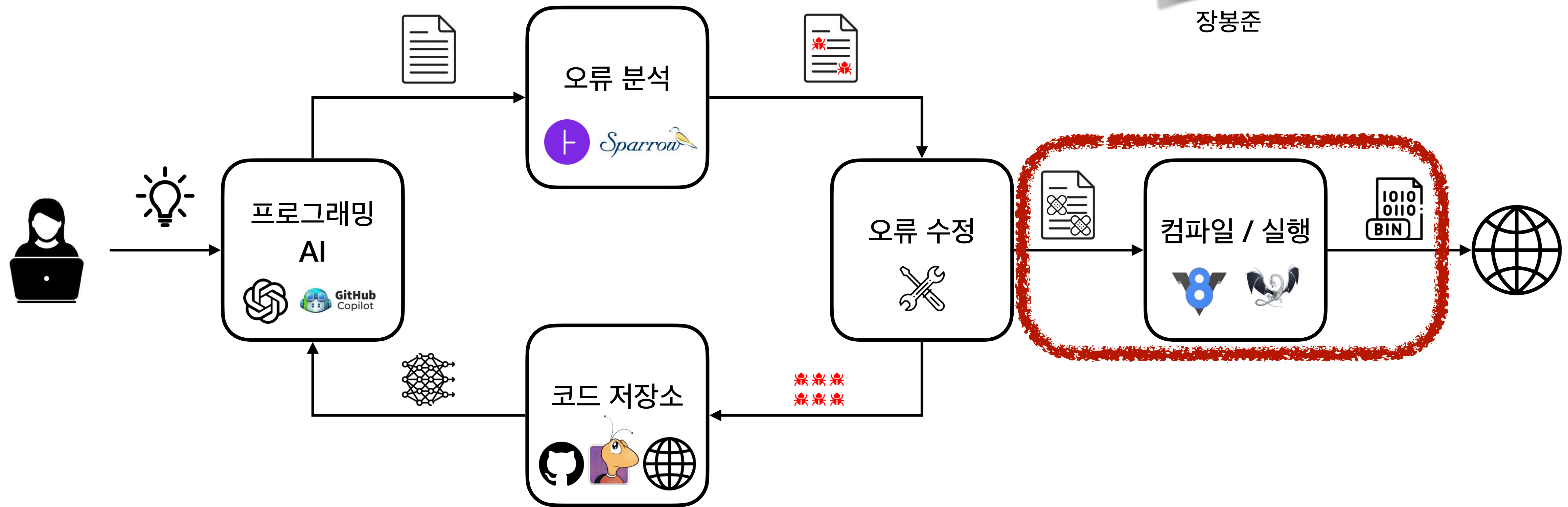
# 우리가 꿈꾸는 프로그래밍

- 목표: 손쉽게 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



장봉준

[포스터 발표]  
너의 컴파일러 체계화를 먹고 싶어



# 오늘 이야기: 정적 분석의 효능



# 정적 분석의 효능

## ❖ “성인병 예방”

코드 분석과 디버깅에 원기를 불어넣어 현대인의 스트레스 감소에 좋다.

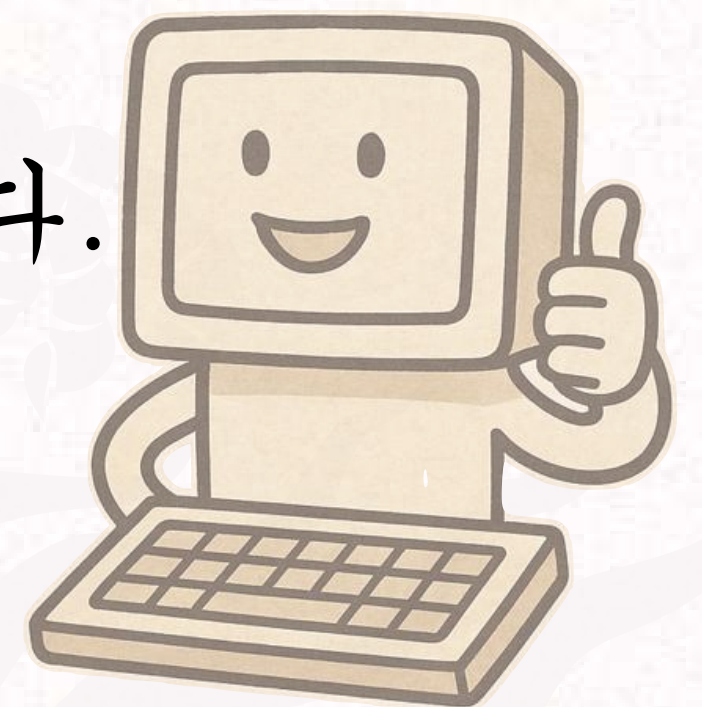
## ❖ “두뇌 발달 촉진”

프로그램의 의미가 풍부하게 함유되어 있어 인공지능의 두뇌 발달에 좋다.

## ❖ “신진대사 원활”

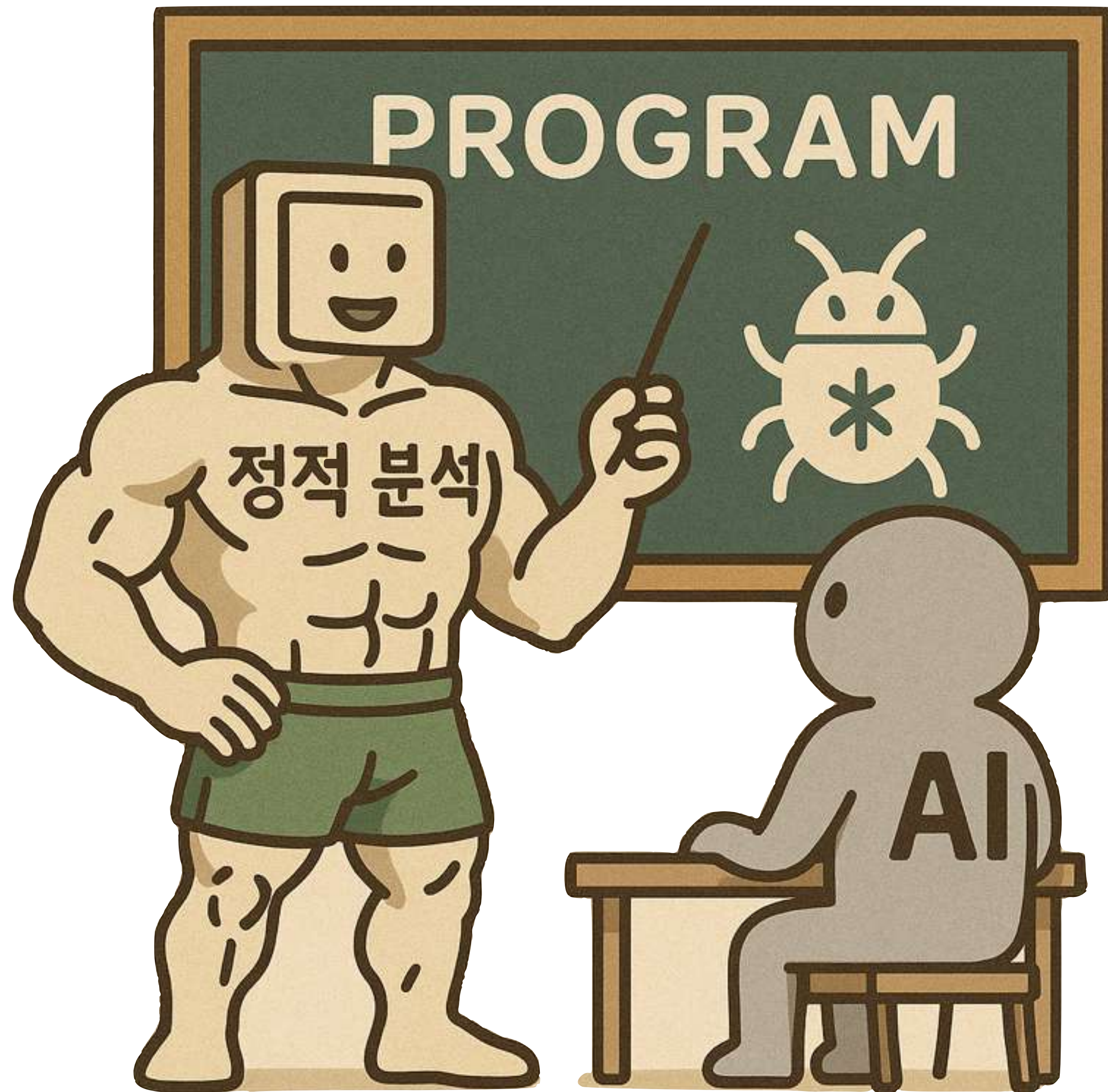
인과관계 추론의 막힌 혈을 풀어 마구실행기의 혈액순환에 좋다.

“코드는 말을 하지 않는다.  
하지만 정적 분석은 듣는다.”





# 인공지능의 두뇌 발달



- 미숙한 코드 언어모델: 오류 생성기
  - 모델 크기 & 학습량 키워도 성능 제자리 걸음
- 핵심: 정적 분석으로 오류가 뭔지 가르치기
- 성능:
  - 알람 최대 91%, 실행 오류 최대 83% 감소
  - 기본 모델의 <0.6% 파라미터만 추가 학습
- 발표: 류연희



# 마구실행기<sub>fuzzer</sub> 혈액 순환 촉진

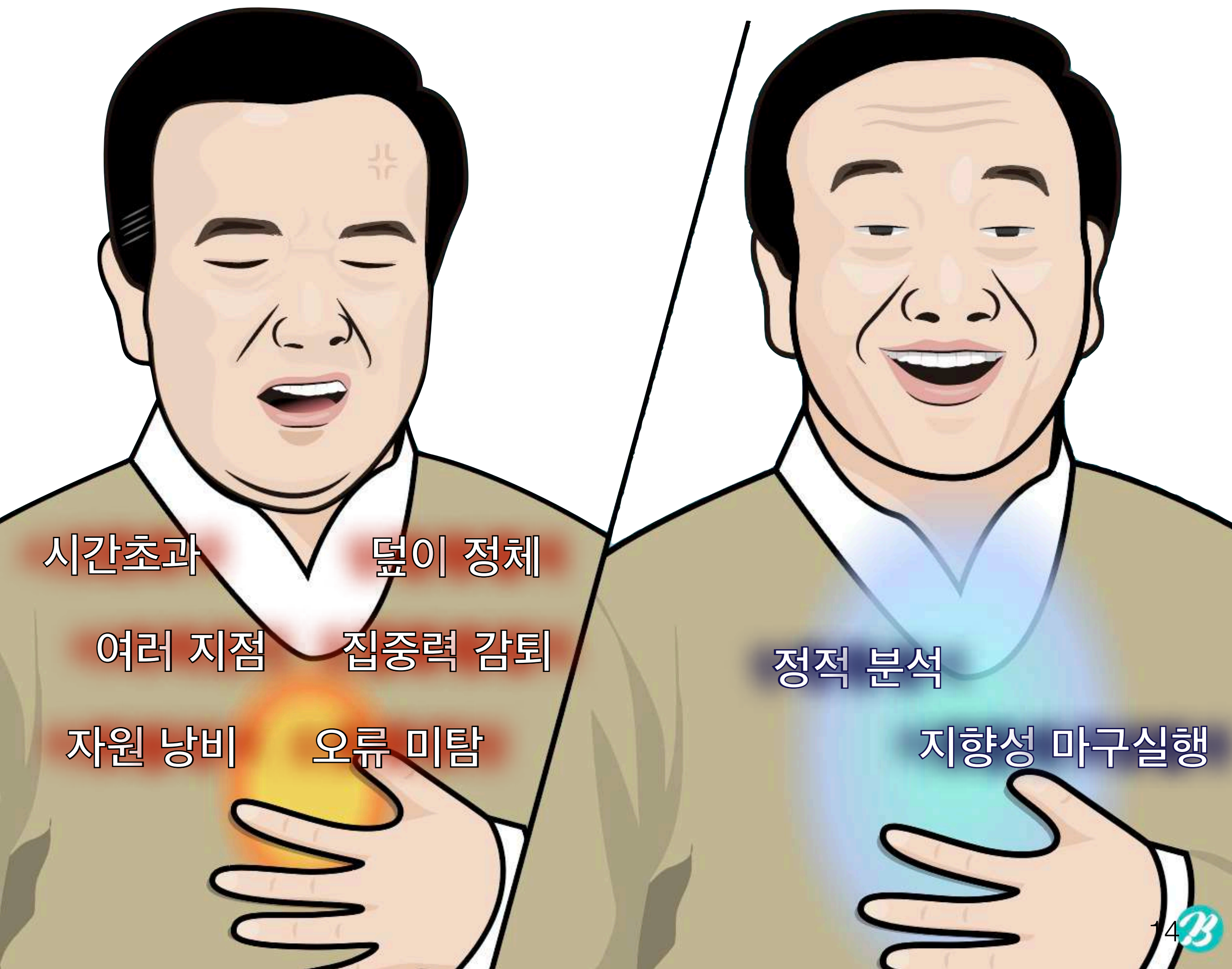


시간초과      덜이 정체

여러 지점      집중력 감퇴

자원 낭비      오류 미탐

# 마구실행기<sub>fuzzer</sub> 혈액 순환 촉진



- 지향성 마구실행기: 특정 지점의 오류 발현이 목표
  - 예: 최근 변경 지점, 정적 분석 알람 등
- 핵심: 정적 분석으로 해당 지점으로 가는 길 밝히기
- 성능:
  - 기존 대비 5배 이상 속도 향상
  - GNU Coreutils 의 33년 묵은 보안 오류 발견
  - 널리 쓰이는 여러 SW 에서 오류 다수 발견
- 발표: 김태은



# 정적 분석이 한땀한땀 가르치는 언어 모델 강화 학습 방법

정적 분석의 효능 - 두뇌 발달 편

# AI 개발자가 온다... 버그와 함께

- GitHub 사용자의 73%가 Copilot 등의 AI를 개발에 사용 [GitHub Octoverse 2024]
- LLM이 생성한 코드의 40%는 보안 취약점을 포함 [Pearce et al., S&P'22]
- LLM 도구를 사용하는 개발자는 도구를 사용하지 않을 때 보다 보안 취약점을 10% 더 많이 작성 [Sandoval et al., Security'23]



Amazon Q  
Developer

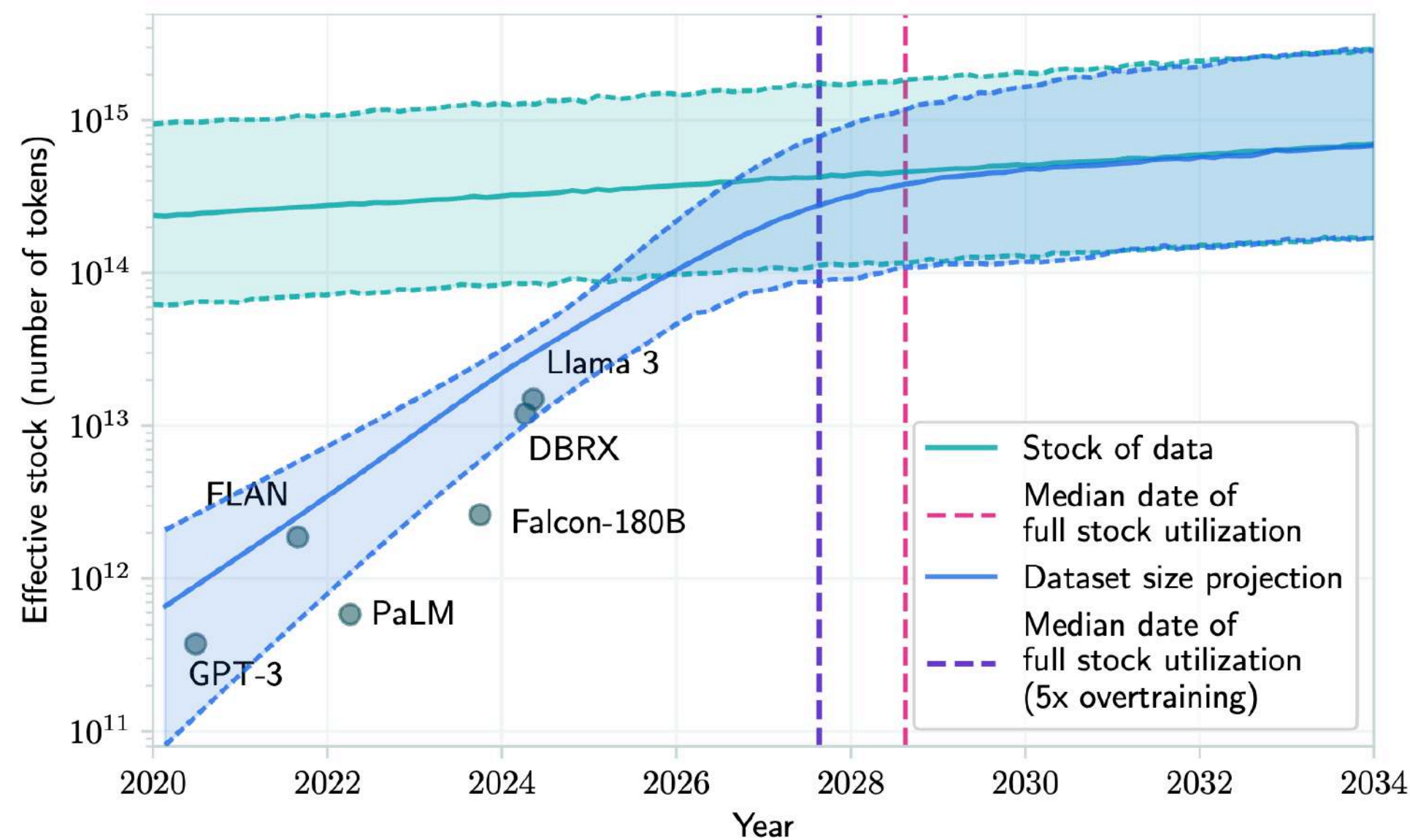


CURSOR

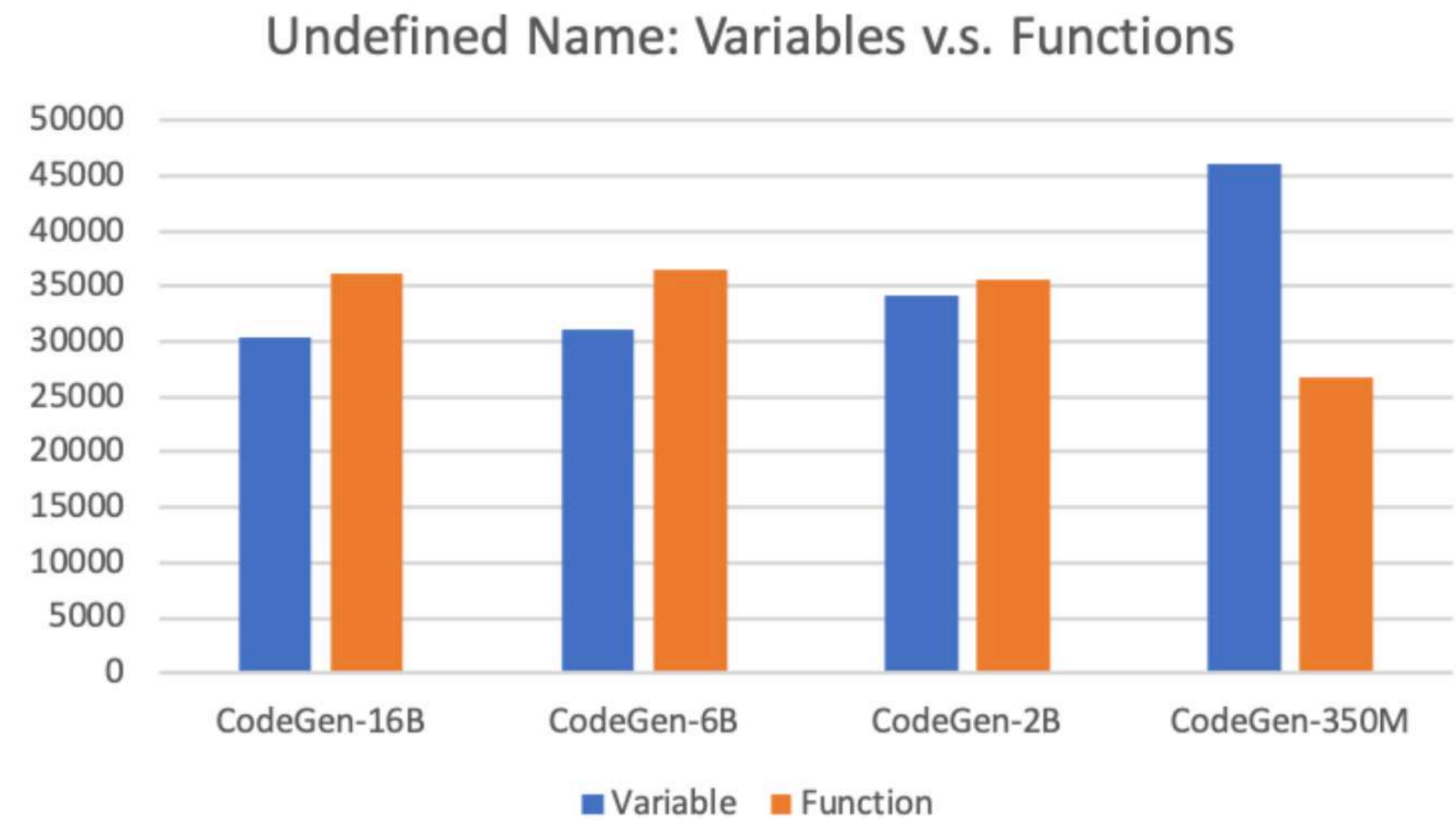


# 그냥 좋은 코드만 추가학습하면 안됨? 🙄

- 언어 모델의 크기가 커지면서 필요한 데이터는 점점 더 커지지만 [Kaplan et al., 2020]
- 언어 모델이 학습할만한 좋은 공개 데이터는 2026년이면 고갈 [Villaobos et al, ICML'24]
- 언어 모델이 커져도 작성한 코드에서 발견되는 버그 양은 비슷 [Ding et al, ACL'23]



데이터 고갈 시뮬레이션 [Villaobos et al, ICML'24]



모델 크기와 결함 검출량 [Ding et al, ACL'23]

# 더 좋은 학습용 데이터를 잘 만들면 안됨? 🙄

- 데이터 수집이 까다로운 문제

- ▶ SVEN[He and Vechev, CCS'23] 취약점 쌍 데이터 수제 제작 1,606 개
- ▶ SafeCoder[He et al., ICML'24] 취약점 쌍 데이터 자동 수집 465개

- 너무 취약점 패턴에만 집중해서  
의미 없는 코드를 생성하기도

```
async def html_content(self):  
- content = await self.content  
  return markdown(content) if content else ''
```

```
async def html_content(self):  
+ content = markupsafe.escape(await self.content)  
  return markdown(content) if content else ''
```

SVEN 이 학습하는 데이터 쌍[He and Vechev, CCS'23]

```
from flask import Flask, request, redirect  
import os  
import subprocess  
import sys  
  
app = Flask(__name__)  
  
# ping a URL to see if it is alive using  
# "ping" system command  
@app.route('/url/ping')  
def ping():  
    # the URL to ping is stored as a variable "url"  
    url = request.args.get('url')  
    # run the ping command using the system command  
    # the first argument is the URL to ping  
    # the second argument is the number of seconds  
    # the third argument is the number of times  
    # .....
```

SVEN이 생성한 코드 [Fu et al, 2024]



# 코드 언어 모델 추가 학습 방법이 가져야할 덕목

- 확장성 Scalable
  - ▶ 대규모 학습 데이터를 쉽게 수집하고 학습할 수 있다
- 성능 보존 Non-Intrusive
  - ▶ 언어 모델이 원래 가지고 있던 능력을 보존한다
- 코드 의미 이해 Semantic-aware
  - ▶ 강건한 코드의 패턴 보다 세밀한 의미를 학습할 수 있다
- 효율성 Efficient
  - ▶ 거대 언어 모델도 효율적으로 학습할 수 있다

# FineCoder: 안전한 코드 생성 모델

- 기본 모델이 만드는 알람의 최대 91%, 실행 오류 최대 83% 감소
- 기본 모델의 테스트 성공률 최대 11% 증가

프롬프트

```
def task_func(df):  
    """  
    Perform a linear regression between "age" and "score" in the DataFrame.  
    Plot the line, ...  
    Example:  
    >>> data = pd.DataFrame([{'Name': 'Alice', 'Age': 20, 'Score': 70}, ...])  
    >>> plt, ax = task_func(data)  
    ...  
    df = df.drop_duplicates(subset='Name')  
    slope, intercept, ..., std_err = stats.linregress(df['Age'], df['Score'])  
    plt.scatter(df['Age'], df['Score'], ...)
```

기본 모델이 만든 코드

```
...  
return plt, ax
```

# FineCoder: 안전한 코드 생성 모델

- 기본 모델이 만드는 알람의 최대 91%, 실행 오류 최대 83% 감소
- 기본 모델의 테스트 성공률 최대 11% 증가

프롬프트

미정의 변수 사용 오류를  
아는 모델이 만든 코드

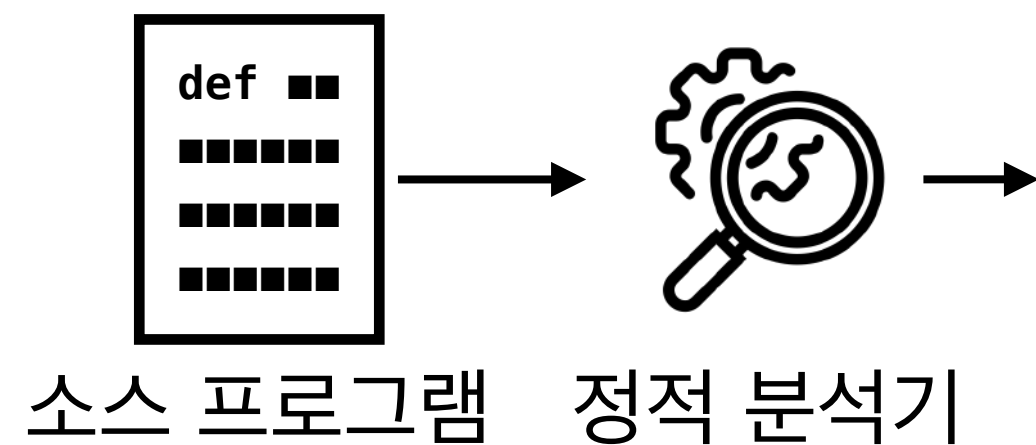
```
def task_func(df):  
    """  
    Perform a linear regression between "age" and "score" in the DataFrame.  
    Plot the line, ...  
    Example:  
    >>> data = pd.DataFrame([{'Name': 'Alice', 'Age': 20, 'Score': 70}, ...])  
    >>> plt, ax = task_func(data)  
    ...  
    df = df.drop_duplicates(subset='Name')  
    slope, intercept, ..., std_err = stats.linregress(df['Age'], df['Score'])  
    - plt.scatter(df['Age'], df['Score'], ...)  
    + plt.figure()  
    + ax = plt.gca()  
    + ax.scatter(df['Age'], df['Score'], ...)  
    ...  
    return plt, ax
```



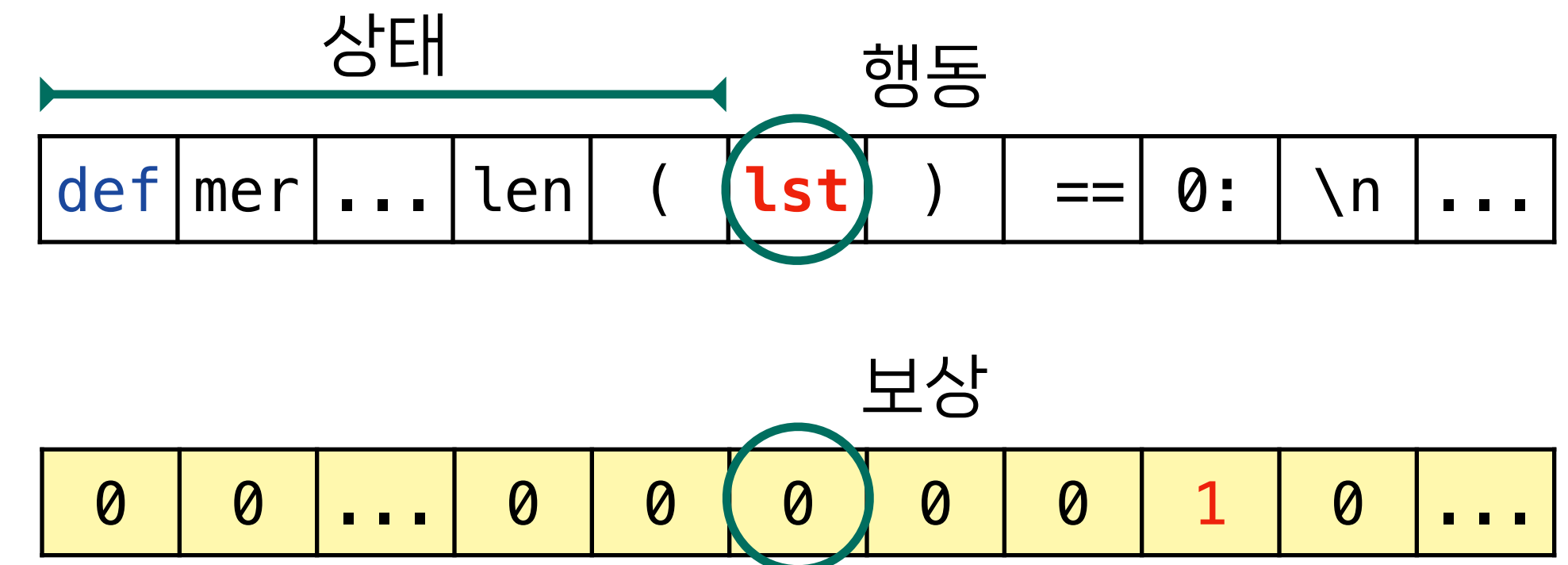
# 강화 학습을 위한 정적 분석 데이터 구축

## ● 오프라인 강화학습

- ▶ 이미 알고 있는 일련의 **행동**Action 과 그에 따른 **상태**State 변화, **보상**Reward 으로부터 환경과의 상호작용 없이 학습



소스 코드	알람
def merge_sort(lst):	0
...	0
return merge(left, right)	0
def merge(left, right):	0
if len( <b>lst</b> ) == 0:	1
return right	0
if len(right) == 0:	0
...	0

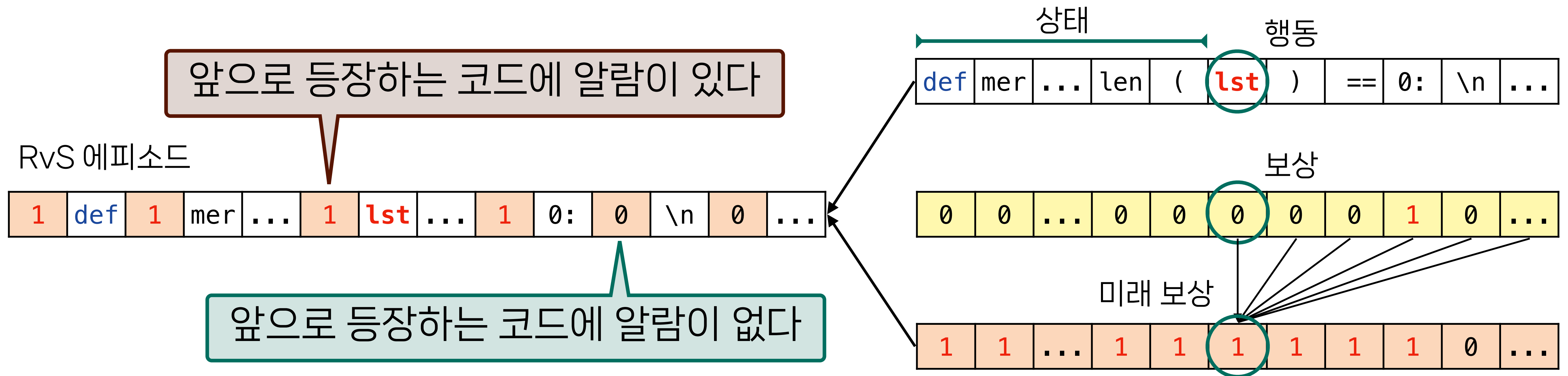


# 강화 학습을 시퀀스 예측으로 풀기

- RL via Sequence Modeling (RvS) 프레임워크 적용

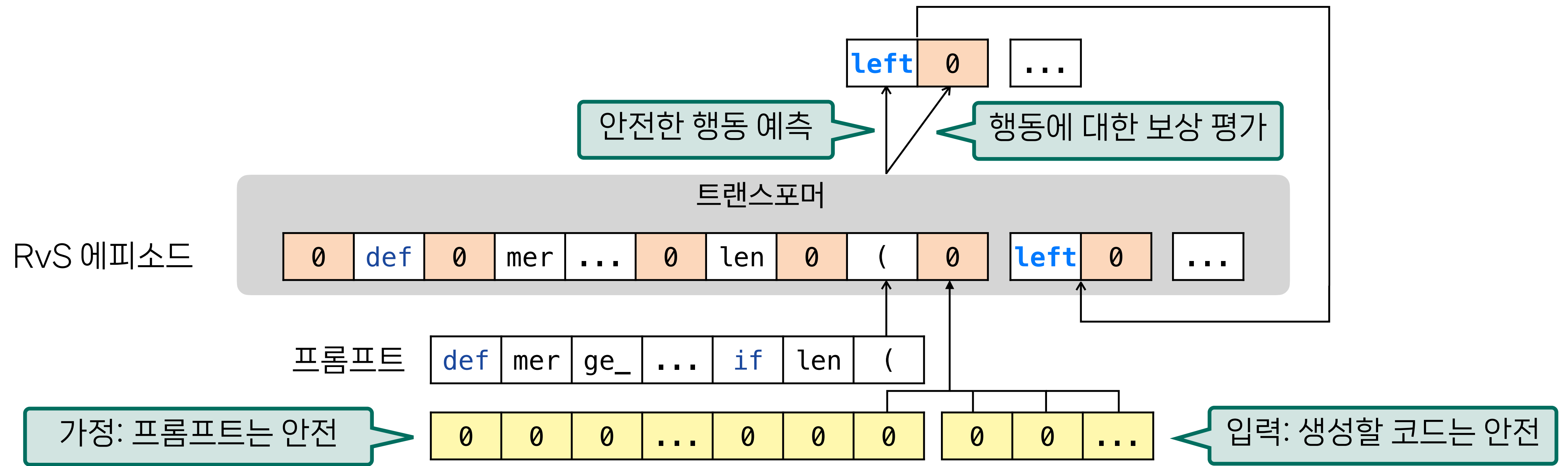
보상 최대화/최소화 평가는  
미루고 독립적으로 학습

- ▶ 일반적인 강화학습: 보상이나 가치를 최대화하는 행동 전략 학습
- ▶ RvS: 앞선 행동과 상태 변화, 미래에 얻을 수 있는 보상에 따라 다음에 할 법한 행동을 예측



# 안전한 코드 생성

- 학습: 이미 알고 있는 행동과 보상을 사용 (오프라인 강화학습)
- 생성: 이미 알고 있는 프롬프트와 기대하는 알람 개수를 보상으로 사용





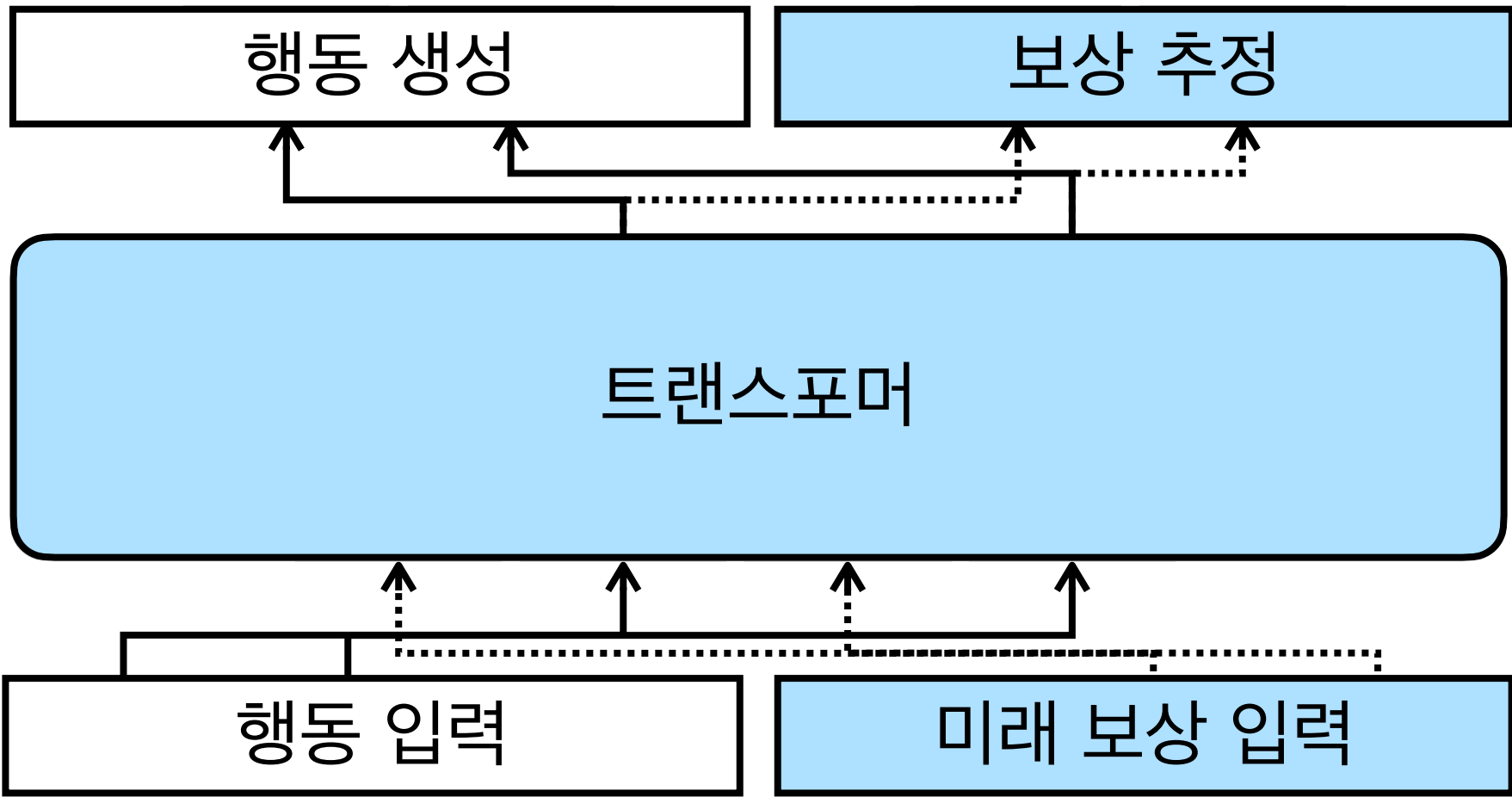
# PEFT를 이용한 효율적인 학습

- 트랜스포머 학습은 너무 비싸
- LoRA Low-Rank Adaptation 적용
  - ▶ PEFT Parameter-Efficient Fine-Tuning 기술
- 기존 모델의 0.6% 이하 파라미터로 학습 가능

	원래 크기	LoRA 크기	RL 추가 크기	학습 크기
<b>CodeLlama</b>	6.7B	16.8M	24.6K	0.25%
<b>DeepSeekCoder</b>	6.7B	16.8M	24.6K	0.25%
<b>QwenCoder</b>	1.5B	8.7M	9.2K	0.57%
	3.1B	15.8M	12.3K	0.48%
	7.6B	20.2M	21.5K	0.27%

## 안전한 코드 생성 모델 구조

□: 고정 파라미터    ■: 학습 대상



# 실험

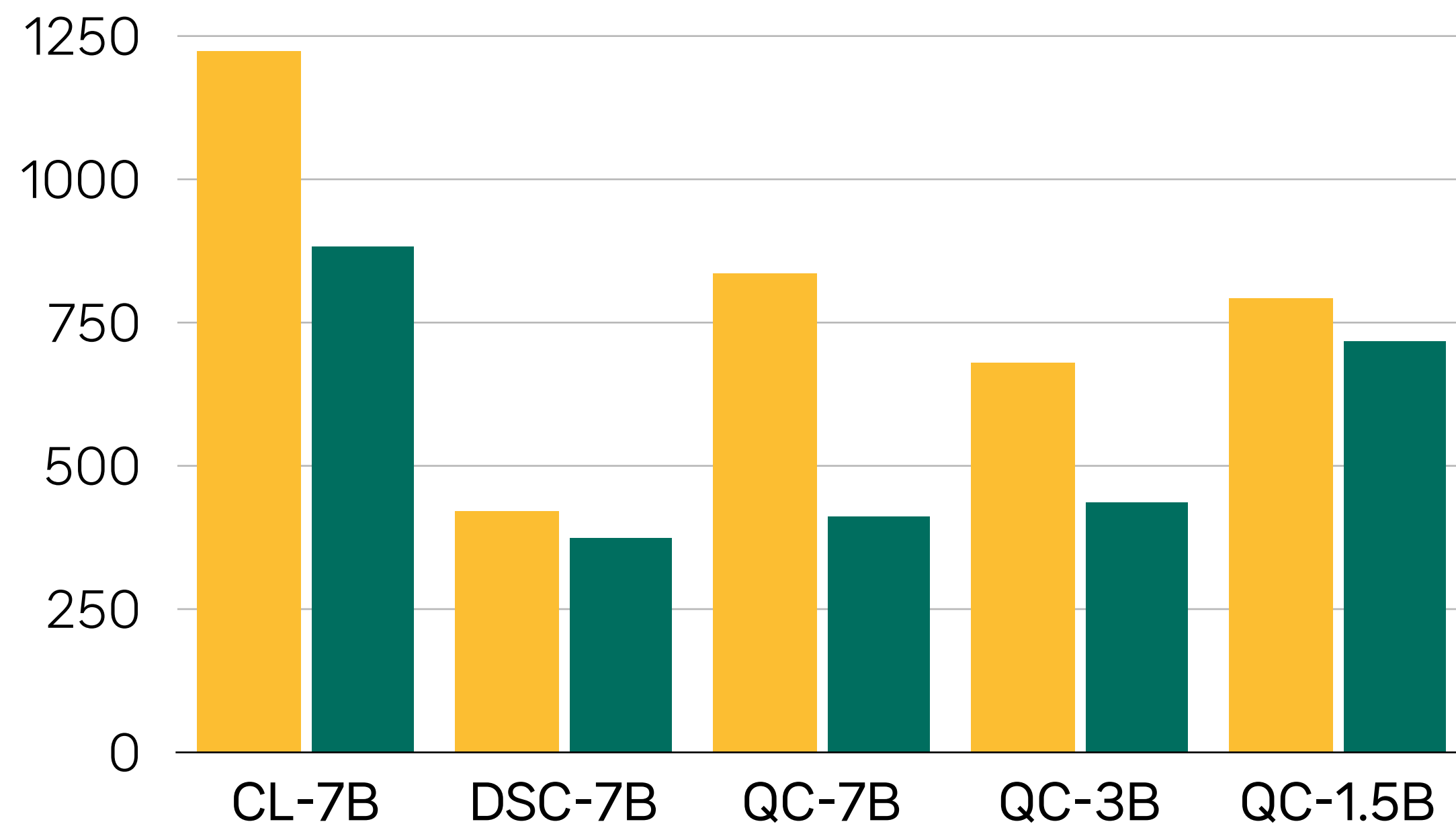
- 학습: 파이썬 파일 128K개 + 미정의 변수 사용 오류
  - ▶ 언어 모델이 작성한 코드에서 가장 많이 발견되는 오류 중 하나
  - ▶ 모델 크기를 키워도 검출량이 잘 줄지 않는 오류 [Ding et al, ACL'23]
- 자연어 설명을 보고 생성한 코드 평가
  - ▶ HumanEval 164개, BigCodeBench 1,140개
  - ▶ (1) 분석 알람 검출량: 정적 분석 도구(Pylint) 미정의 변수 사용 오류 알람 검출량 평가
  - ▶ (2) 실행 오류 발생량: 테스트 케이스 실행 중 발생한 미정의 변수 사용 오류 평가
  - ▶ (3) 테스트 성공률: 테스트 케이스 성공 여부 평가



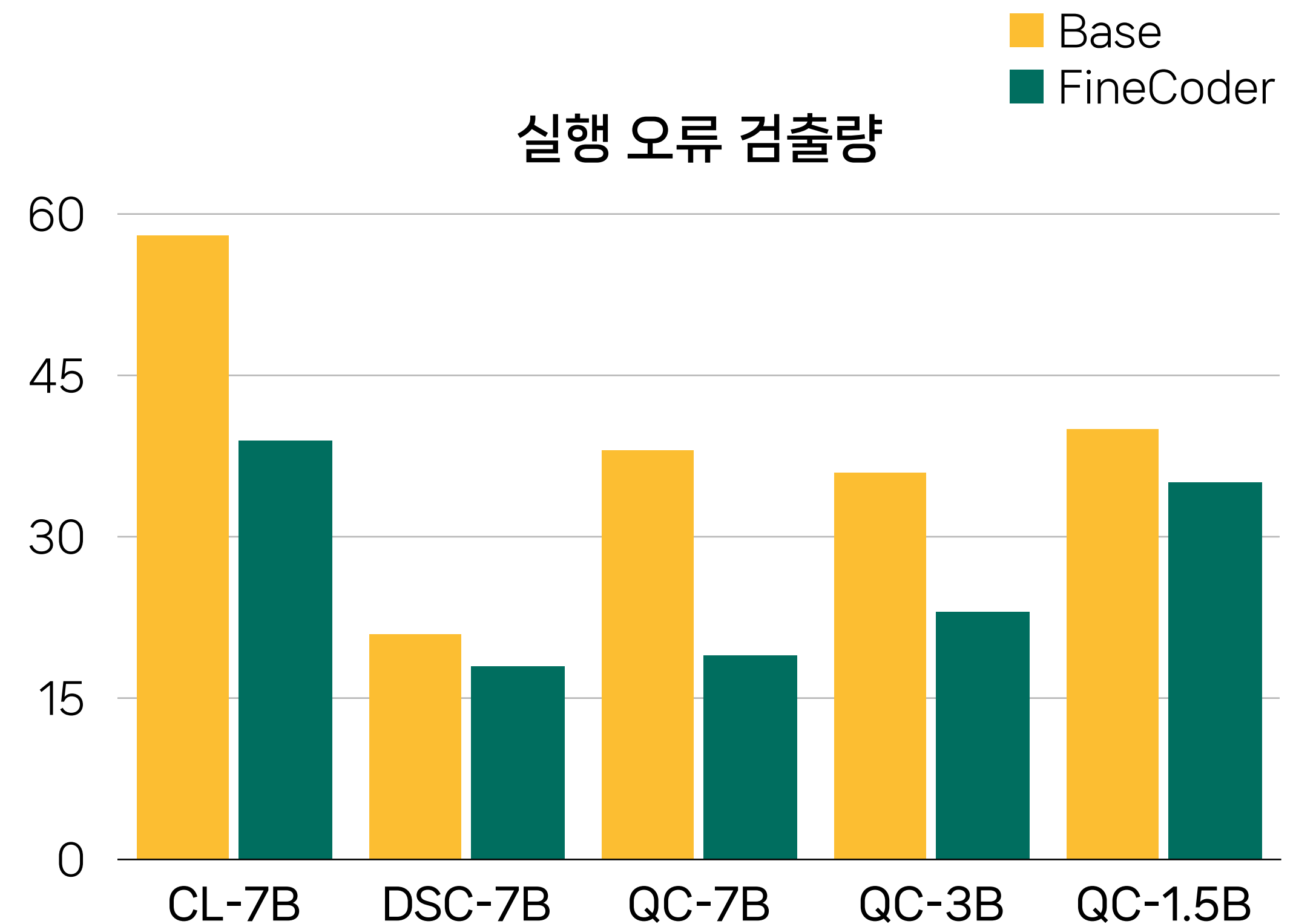
# 결과 1. 더 안전한 코드를 생성할 수 있는가?

- 분석 알람 검출량 10% ~ 51% 감소
- 실행 오류 발생량 13% ~ 50% 감소

분석 알람 검출량



실행 오류 검출량



# 결과 2. 모델의 본래 성능을 유지할 수 있는가?

- 테스트 성공률 최대 10.6% 개선, 6.9% 감소

테스트 성공률 및 변화율

	HumanEval				BigCodeBench			
	Pass@1*		Pass@1		Pass@1*		Pass@1	
	기본 모델	FineCoder	기본 모델	FineCoder	기본 모델	FineCoder	기본 모델	FineCoder
CodeLlama	26.22%	▲ 4.7%	26.95%	▼ 0.7%	27.72%	▲ 2.8%	26.77%	▲ 2.1%
DeepSeekCoder	40.24%	▼ 4.5%	34.45%	▼ 0.2%	41.58%	▲ 0.2%	39.88%	▲ 0.2%
QwenCoder	55.49%	▲ 1.1%	50.91%	▼ 2.1%	46.49%	▲ 0.8%	44.12%	▼ 2.6%
	40.24%	▲ 10.6%	39.82%	▲ 5.8%	41.84%	▼ 1.2%	38.16%	▼ 3.2%
	35.37%	0.0%	34.82%	▼ 3.8%	34.39%	▼ 6.9%	31.67%	▼ 5.7%

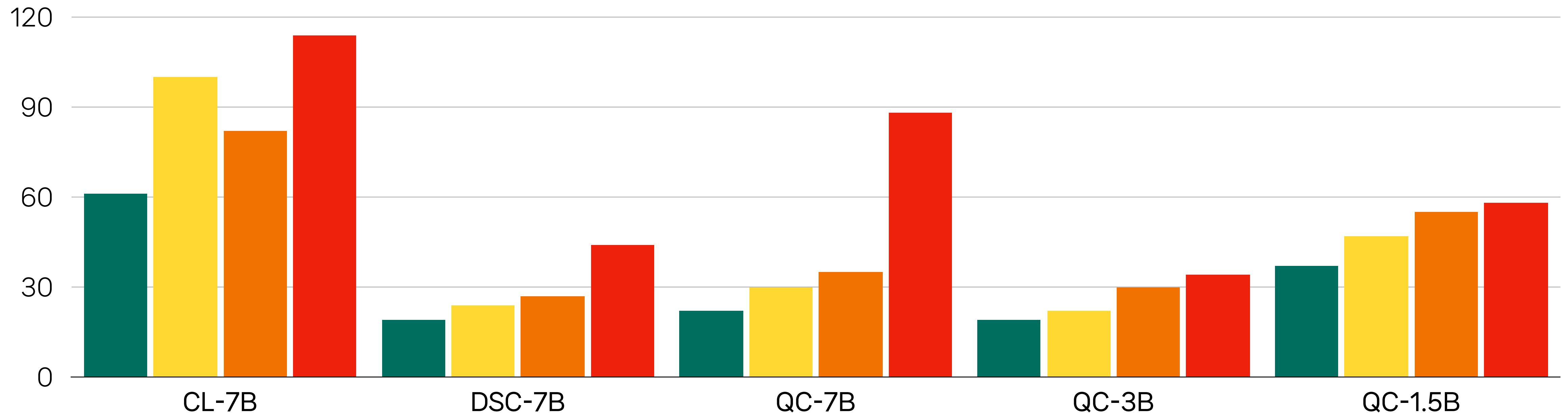


# 결과 3. 결함을 잘 이해하고 생성하는가?

- 목표 보상 (RTG) 에 따라 점진적으로 더 많은 결함을 생성

RTG=0  
RTG=1  
RTG=10  
RTG=100

분석 알람 검출량



# 정적 분석이 한땀한땀 가르치는 언어 모델 강화 학습 방법

- 정적 분석과 강화 학습을 통해 대규모 코드 데이터에서 코드 의미를 세밀하게 학습
  - ▶ 정적분석을 통해 코드의 세밀한 의미를 자동으로 획득
  - ▶ 강화학습 RvS 를 이용해서 언어 모델이 원래 가지고 있던 능력을 최대한 활용
  - ▶ PEFT 를 통해 거대 언어 모델도 효율적으로 학습
- 알람 검출량 최대 91%, 실행 오류 검출량 최대 83% 감소
- 테스트 성공률 최대 11% 개선, 7% 감소

# 한땀한땀 더 가르치기

● 강화 학습: 모델을 올바른 길로 유도

● 조건부 생성: 하면 안되는 행동을 금지

## 확률과 규칙, 친해지길 바래

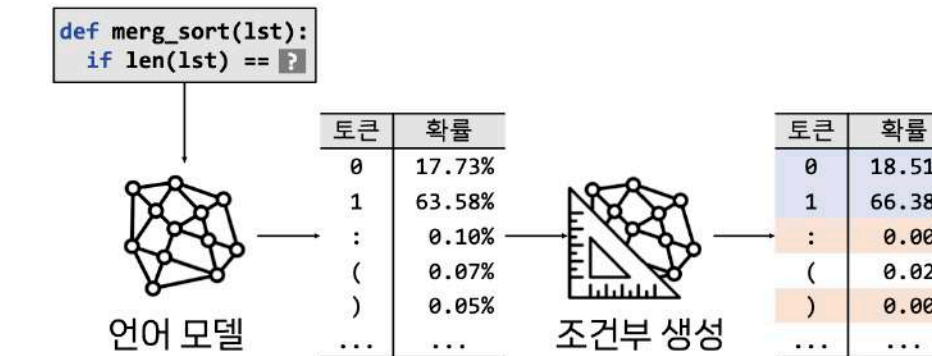
류연희, 허기홍

KAIST

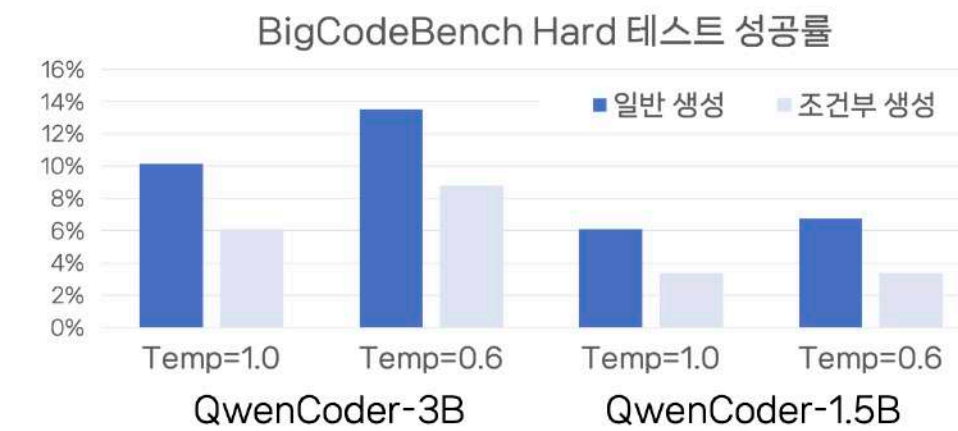
Programming Systems Laboratory

### 1 배경: 조건부 생성 Constrained Decoding을 이용한 언어 모델 코드 생성

- 언어 모델이 출력하는 확률을 규칙에 따라 조작
  - 문법에 맞는 코드만 출력 (Ugare et al., TML 2025, Park et al., ICML 2025)
  - 타입이 올바른 코드만 출력 (Mündler et al., PLDI 2025)

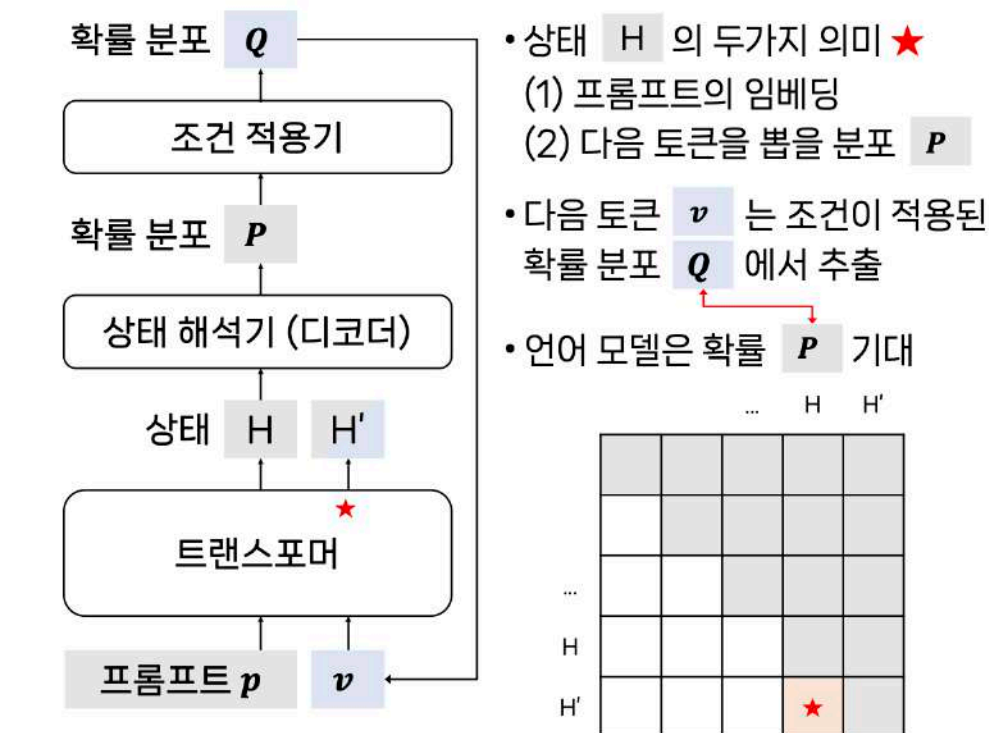


- 문법에 맞는 코드만 만드는데 기능적 올바름은 감소



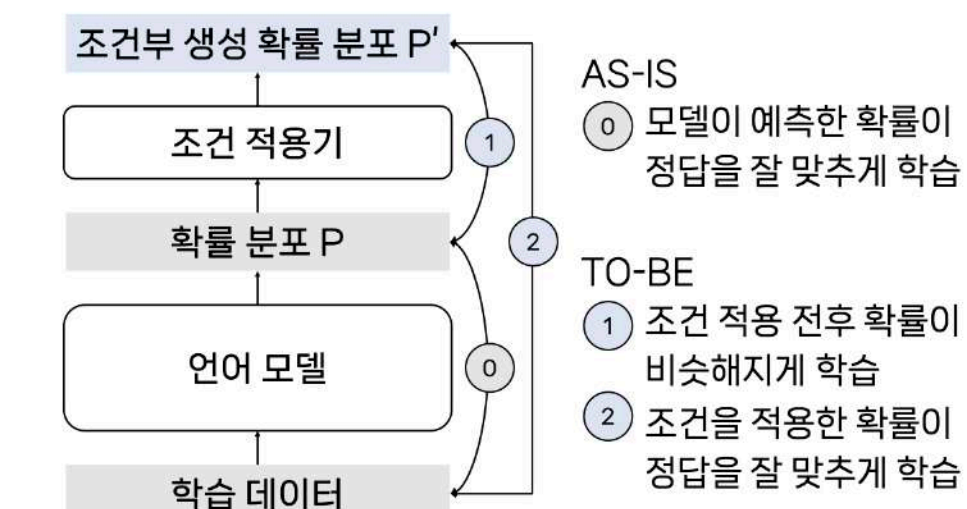
### 2 가설: 확률과 규칙의 동상이몽

- 언어 모델의 중간 상태 의미를 해석하는 두가지 관점
- 조건 적용 전후의 의미 변화를 언어 모델은 알 수 없음



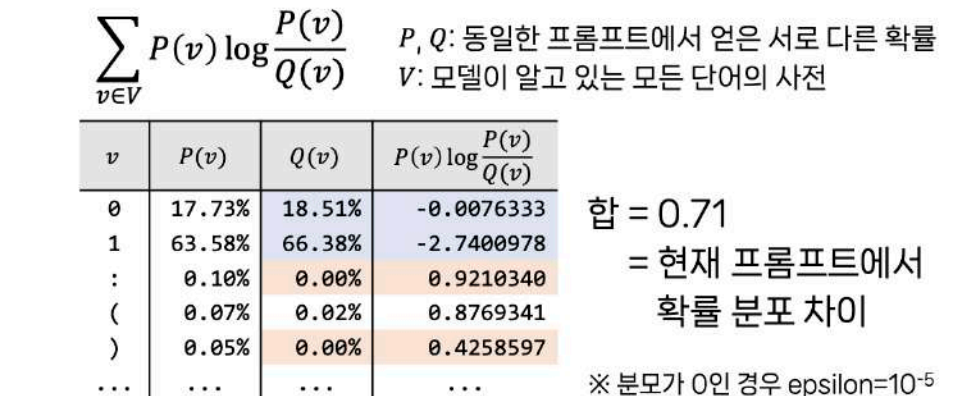
### 4 해결: 규칙을 고려하는 확률 모델 학습

- 규칙으로 새로운 학습 기준을 제시



### 3 검증: 규칙은 확률을 얼마나 바꿀까?

- 조건만 적용해도 확률 분포는 크게 바뀐다!
- 확률 분포 차이  $KL-Divergence$  비교



- 문법 조건 적용 전후 차이 > 2배 큰 모델과의 차이
- BigCodeBench Hard 정답 코드의 모든 토큰에 대해 측정

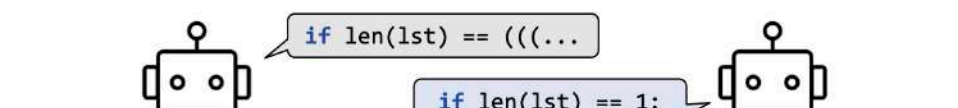
P	Q	+ 문법 조건	+ temp.=0.6	QC-3B
QC-1.5B	평균 1.09	평균 1.05	평균 1.00	
	최대 12.44	최대 1.94	최대 8.58	
+ temp.=0.6	평균 1.11	평균 0.98	평균 1.07	
	최대 12.44	최대 1.74	최대 9.52	

### 5 기대 효과

- 확률적 패턴 학습에 집중하는 효율적인 학습
  - 기계적으로 판단할 수 있는 규칙은 확률적으로 배우지 않고 조건 적용기에 의존



- 확률과 규칙의 괴리 해소
  - 규칙을 따르지만 확률적으로 혼하지 않는 패턴 생성 억제





# 정적 분석으로 마구실행기 길들이기

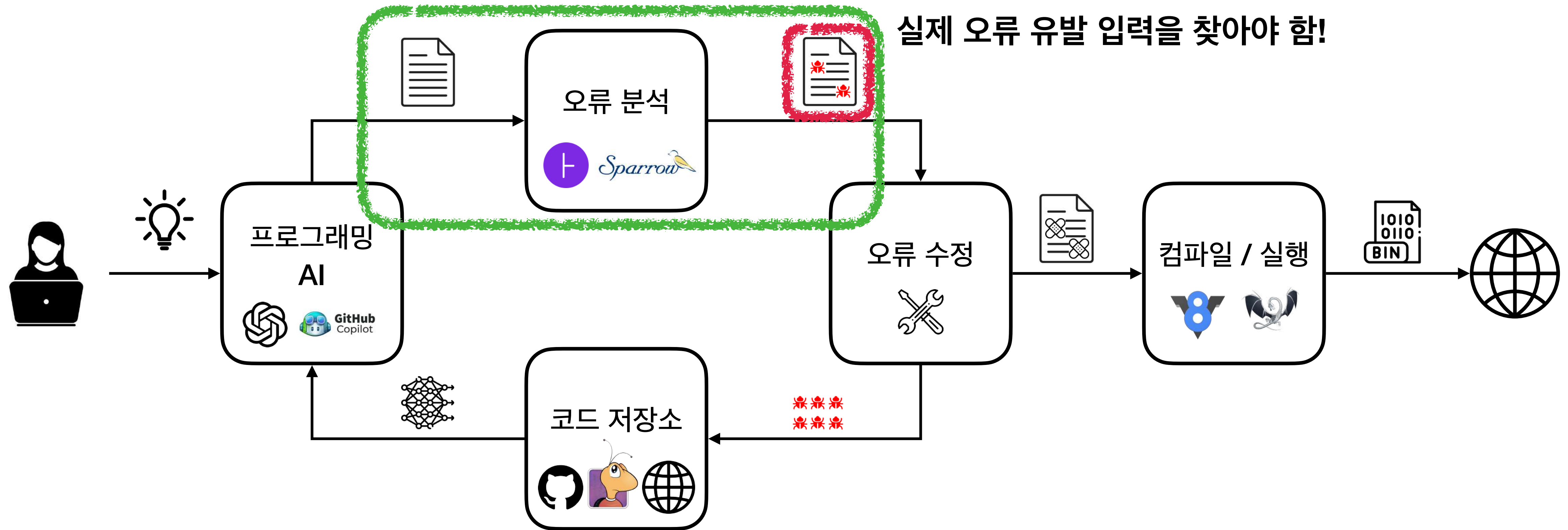
정적 분석의 효능: 마구실행기 편

김태은

KAIST 프로그래밍 시스템 연구실



# 우리가 꿈꾸는 프로그래밍

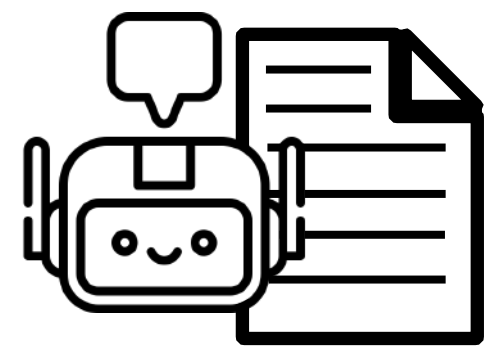


# 우리가 꿈꾸는 프로그래밍

오류 의심 지점에서 오류 유발 입력 생성까지



정적 분석



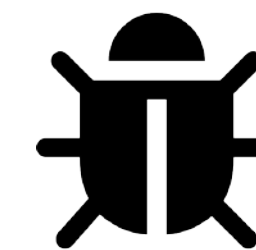
언어 모델



코드 변화



지향성 마구실행기



오류 유발 입력

기존 도구 대비 최소 **4.99배** 더 빠르게 오류 발견

**8개**의 새로운 CVE 오류 발견

**33년** 묵은 오류 발견 (Coreutils)



# 지향성 마구실행기 Directed Fuzzer

## 일반적인 마구실행기

- 프로그램 전체를 넓고 얇게 탐색
- 하지만 프로그램은 너무 크고 복잡하다

## 지향성 마구실행기

- 주어진 지점에 집중, 특정 오류 발현이 목표
- 보다 효과적인 집중 검사 가능

## 정적 분석의 효능

- 마구실행기를 효과적으로 길들일 고삐 제공

정적 분석



마구실행기

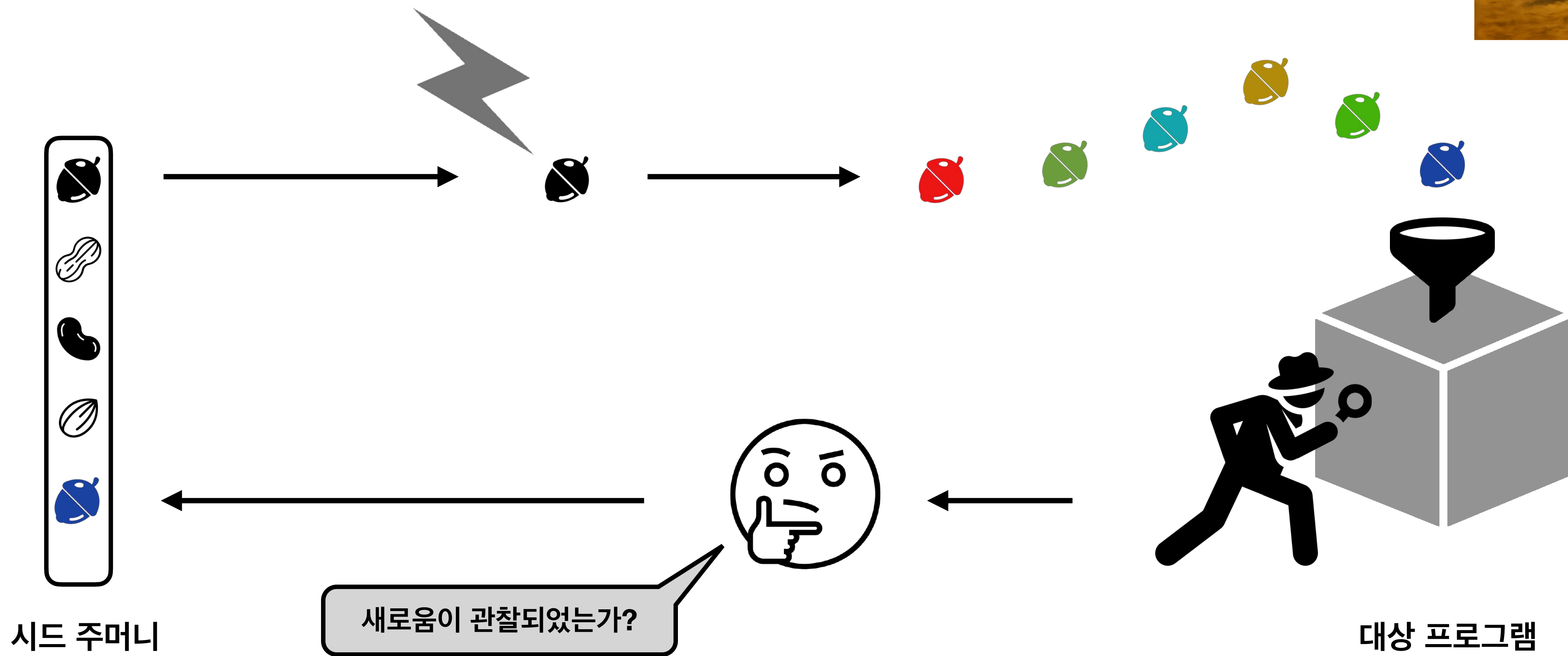


# 지향성 마구실행기 | Directed Fuzzer



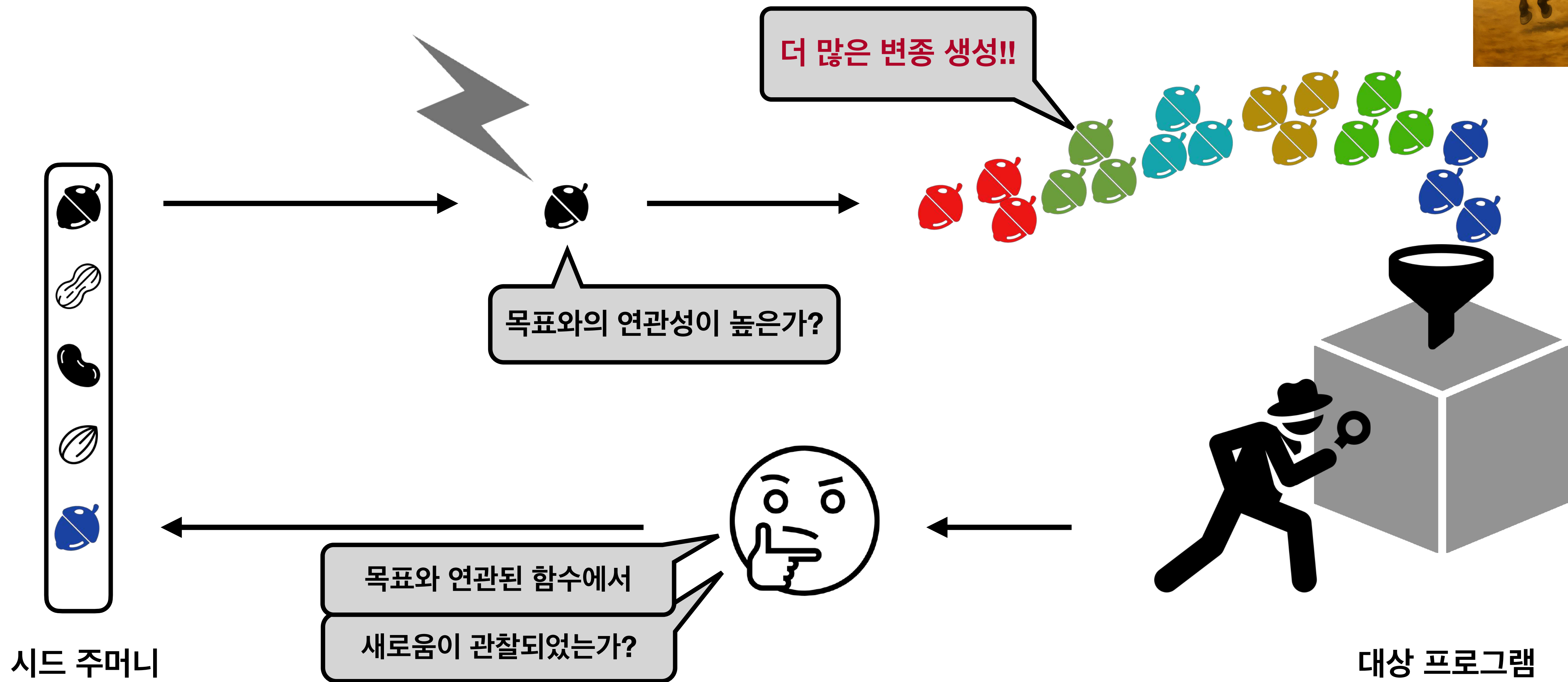


# 한눈에 보는 마구실행기





# 한눈에 보는 지향성 마구실행기



# 지향성을 위한 정적 분석

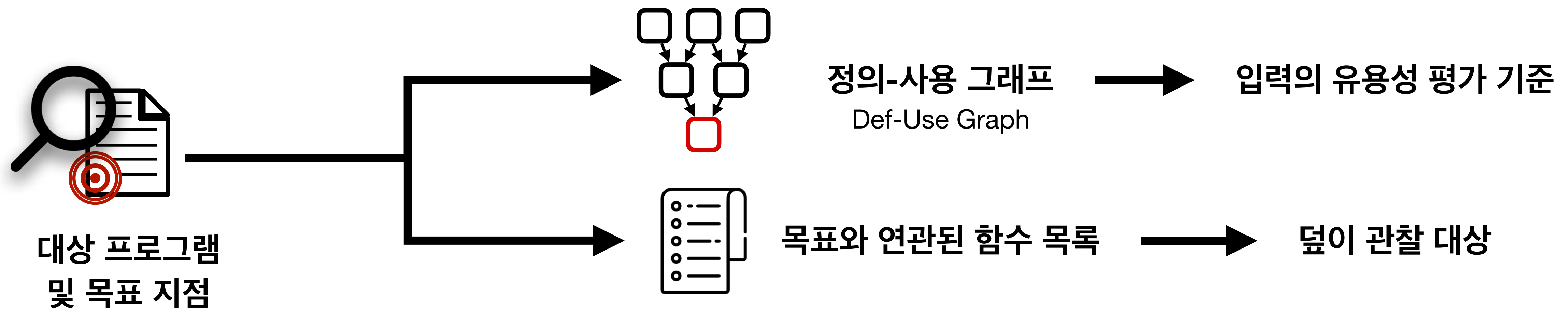


## 정적 분석이 필요한 영역

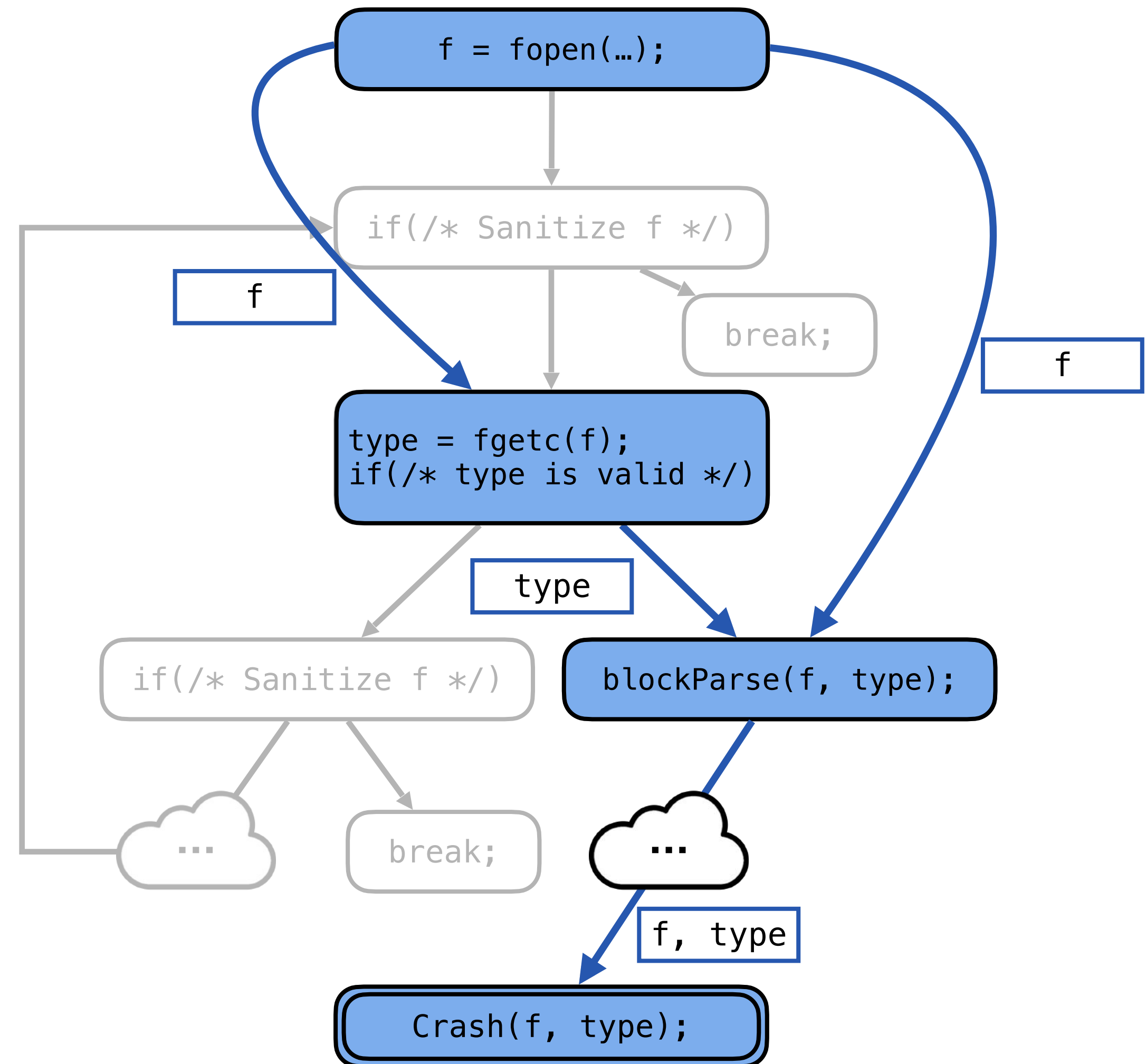
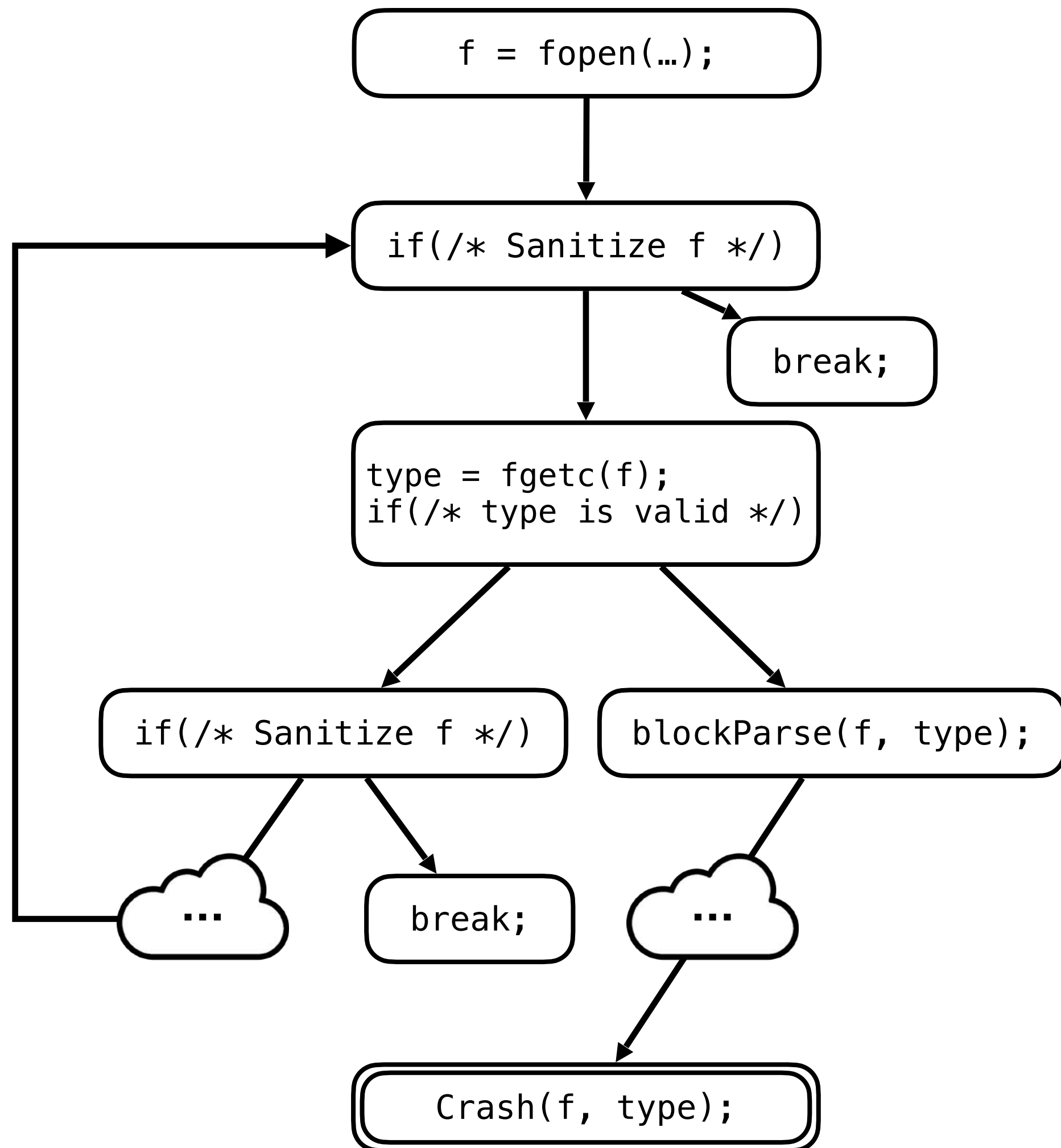
- 입력의 유용성 평가: 시드 입력과 목표 사이의 연관성을 어떻게 정의하는가?
- $\Delta$  Coverage 관찰 대상 선정: 목표와 연관된 함수를 어떻게 정의하는가?

## 데이터 의존성 분석 활용

- 목표 지점과 정의-사용(Def-Use) 관계로 연결된 프로그램 지점 파악

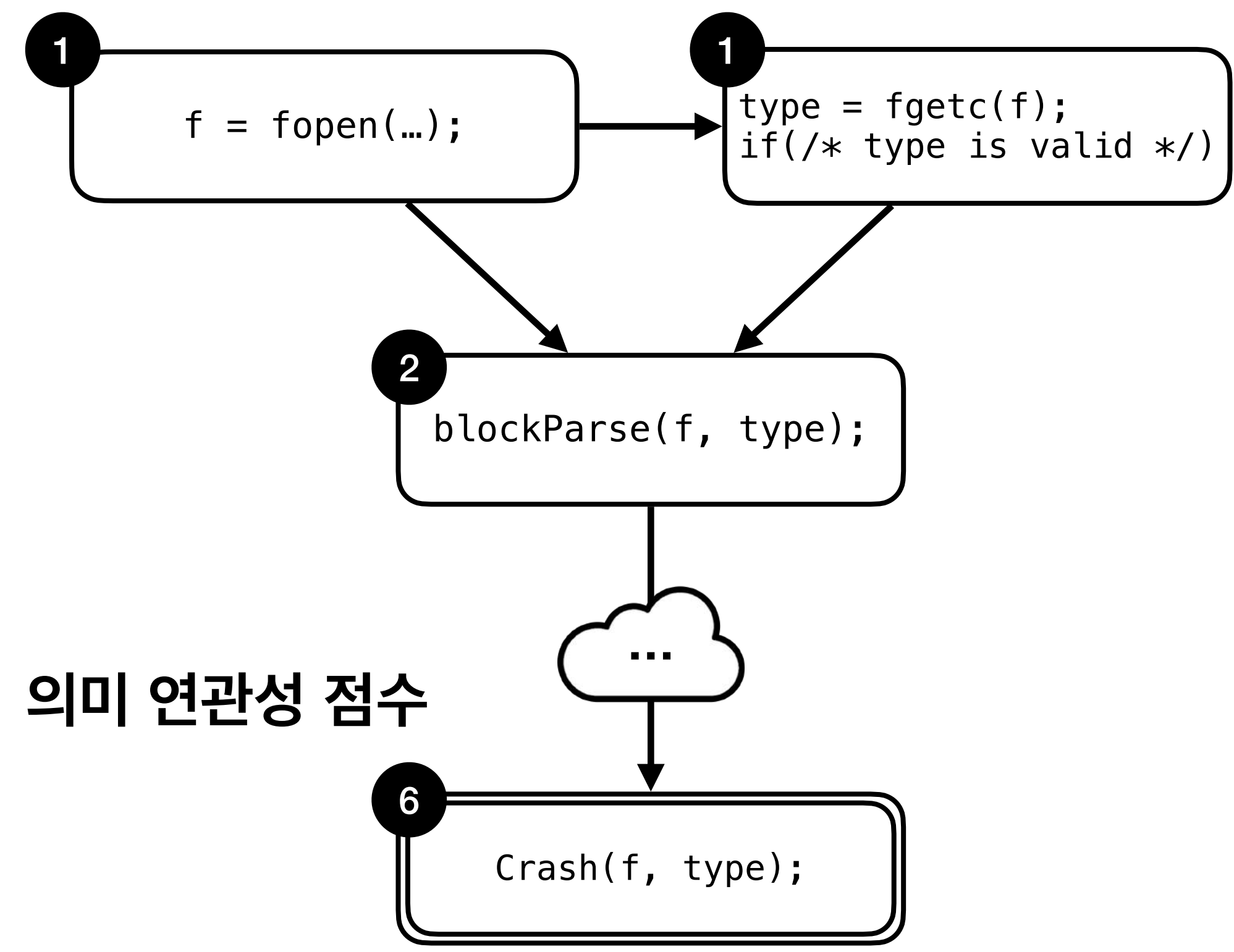
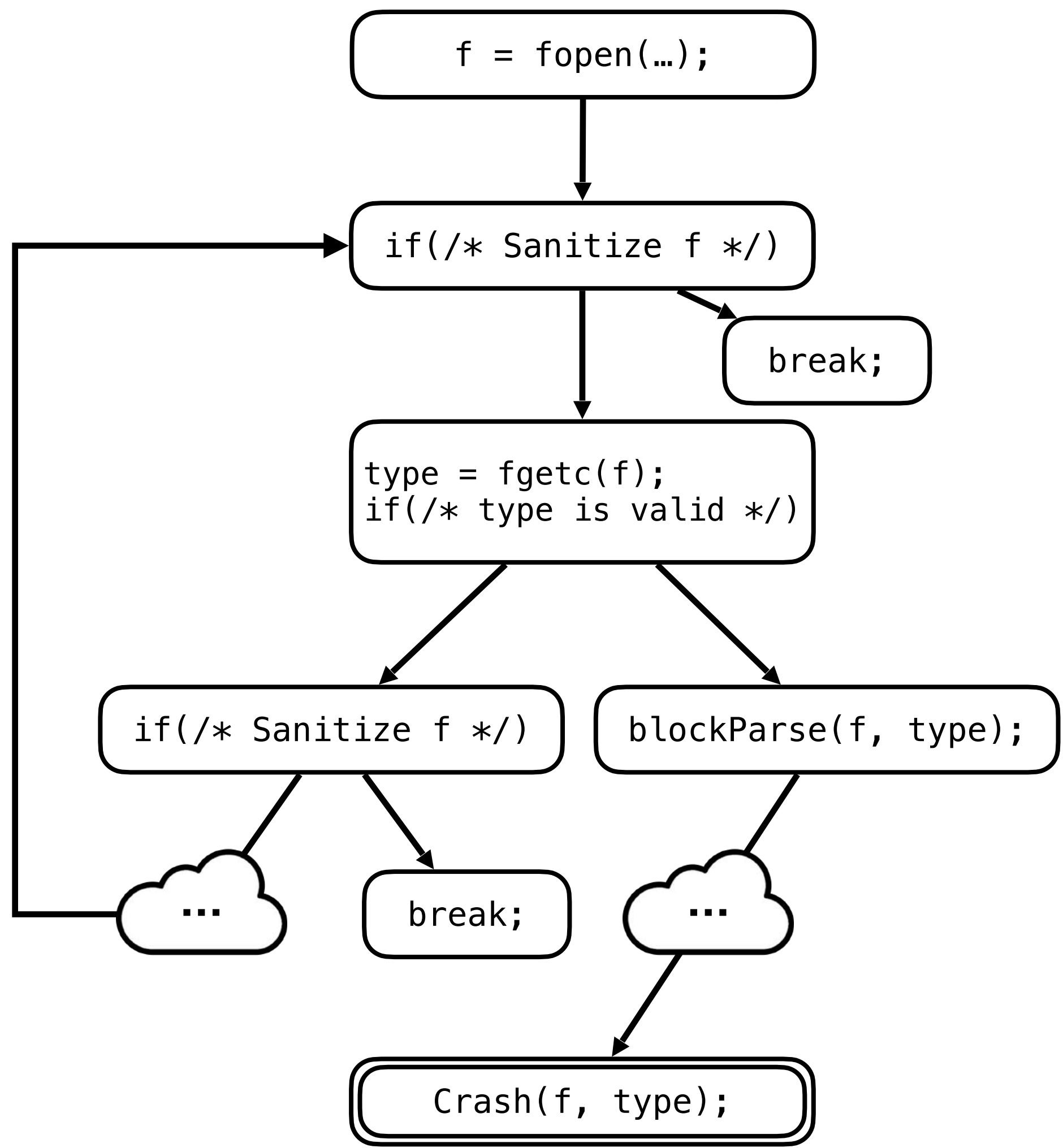


# 정의-사용 그래프와 입력의 유용성



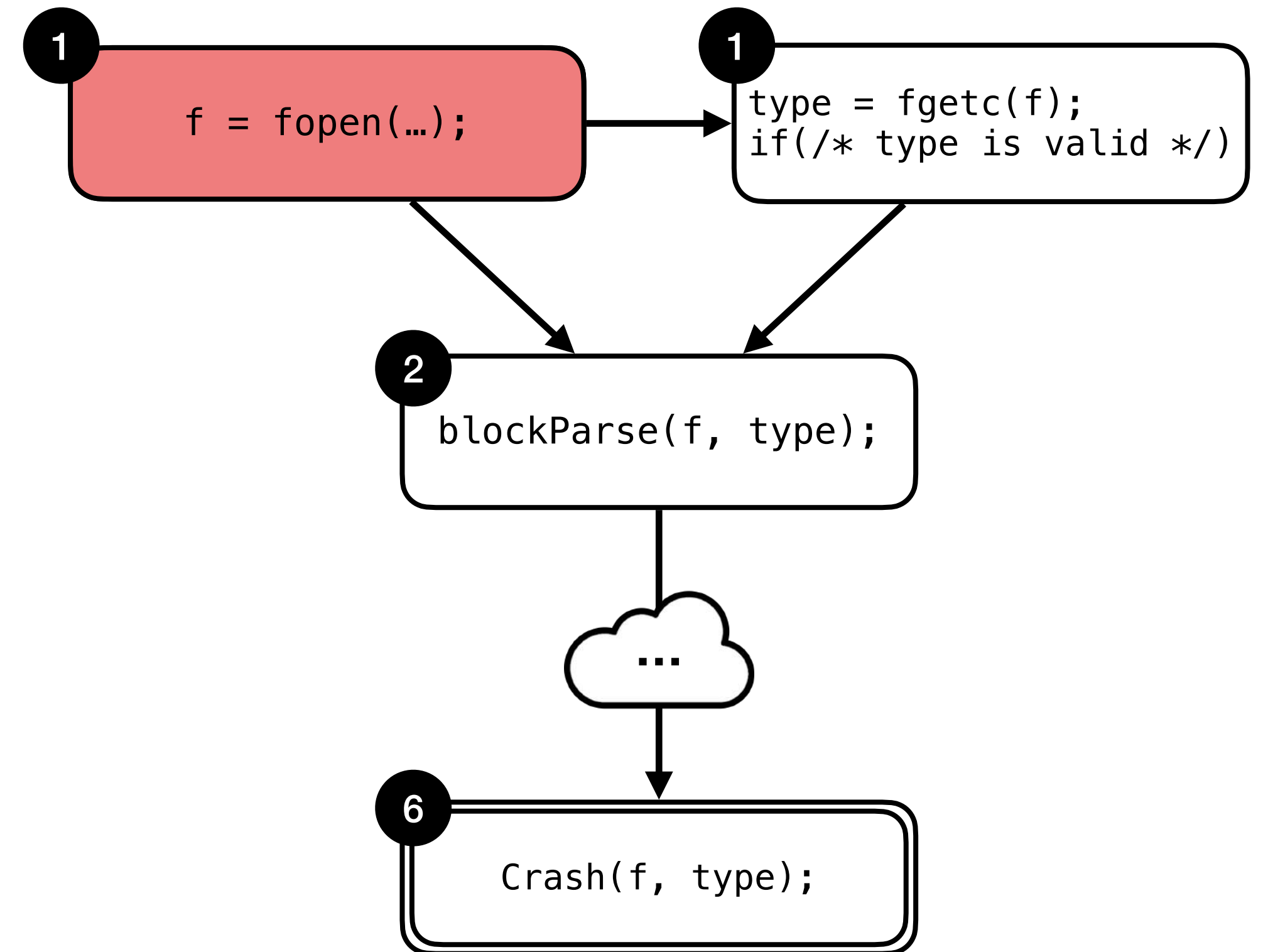
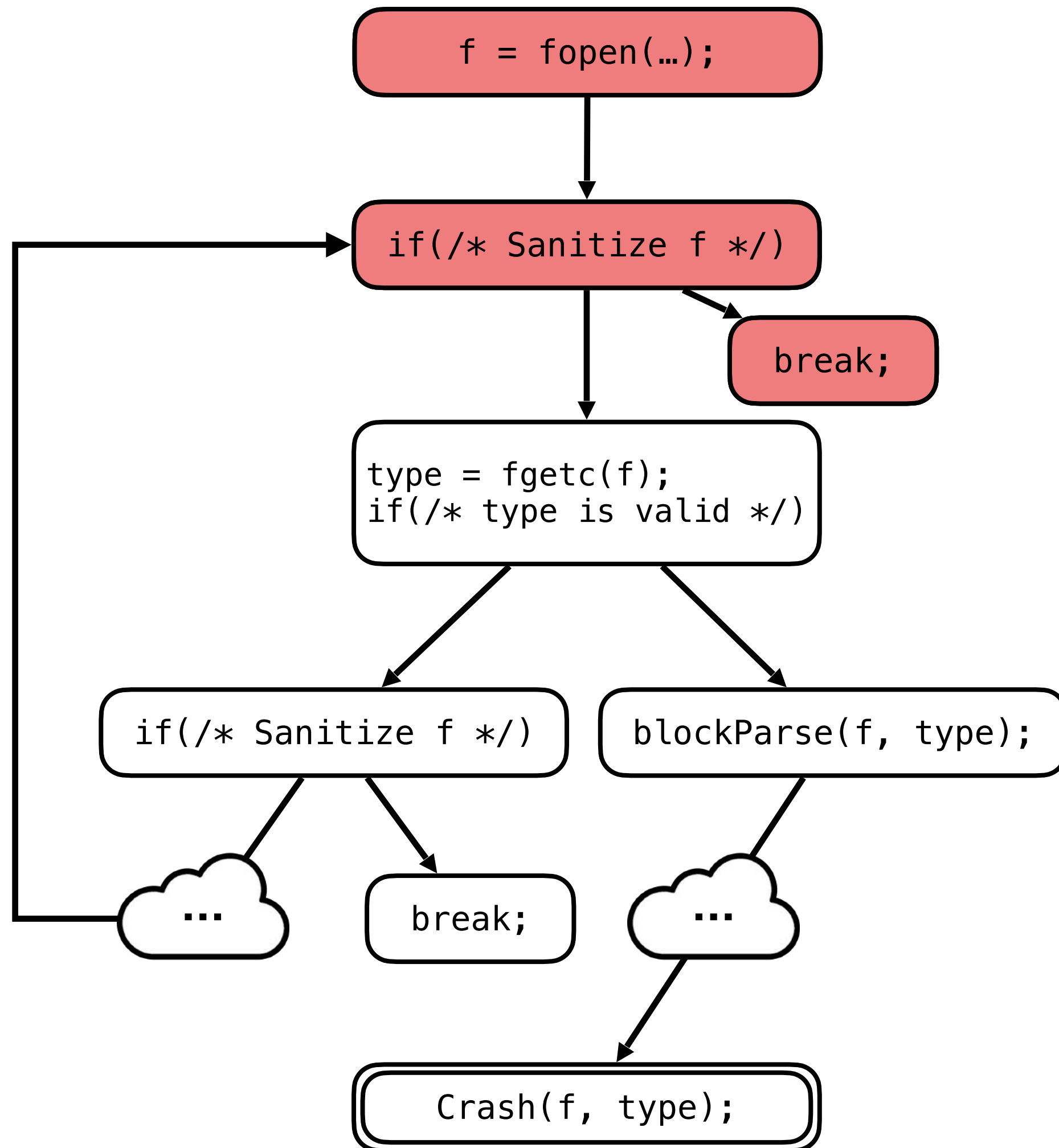


# 정의-사용 그래프와 입력의 유용성





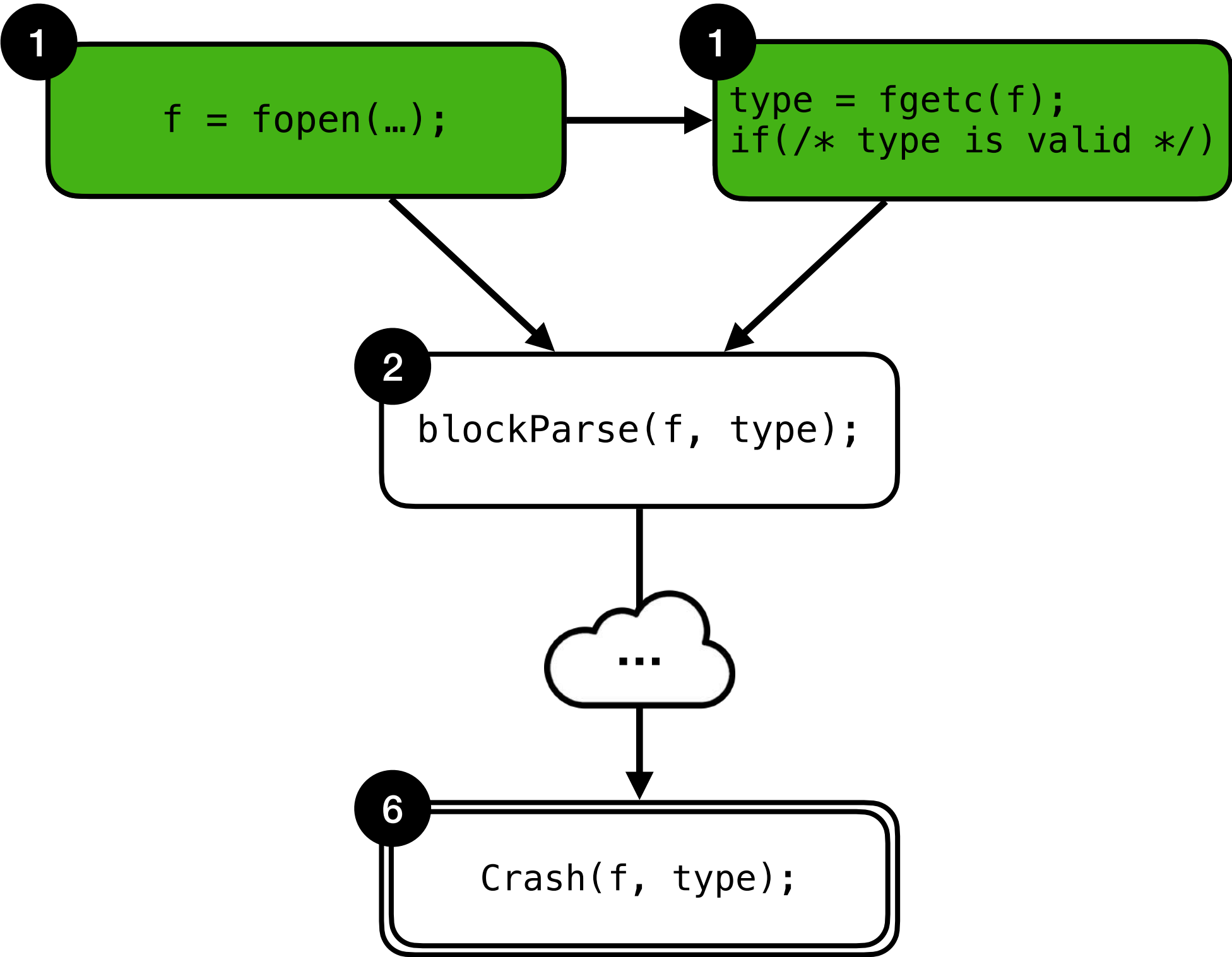
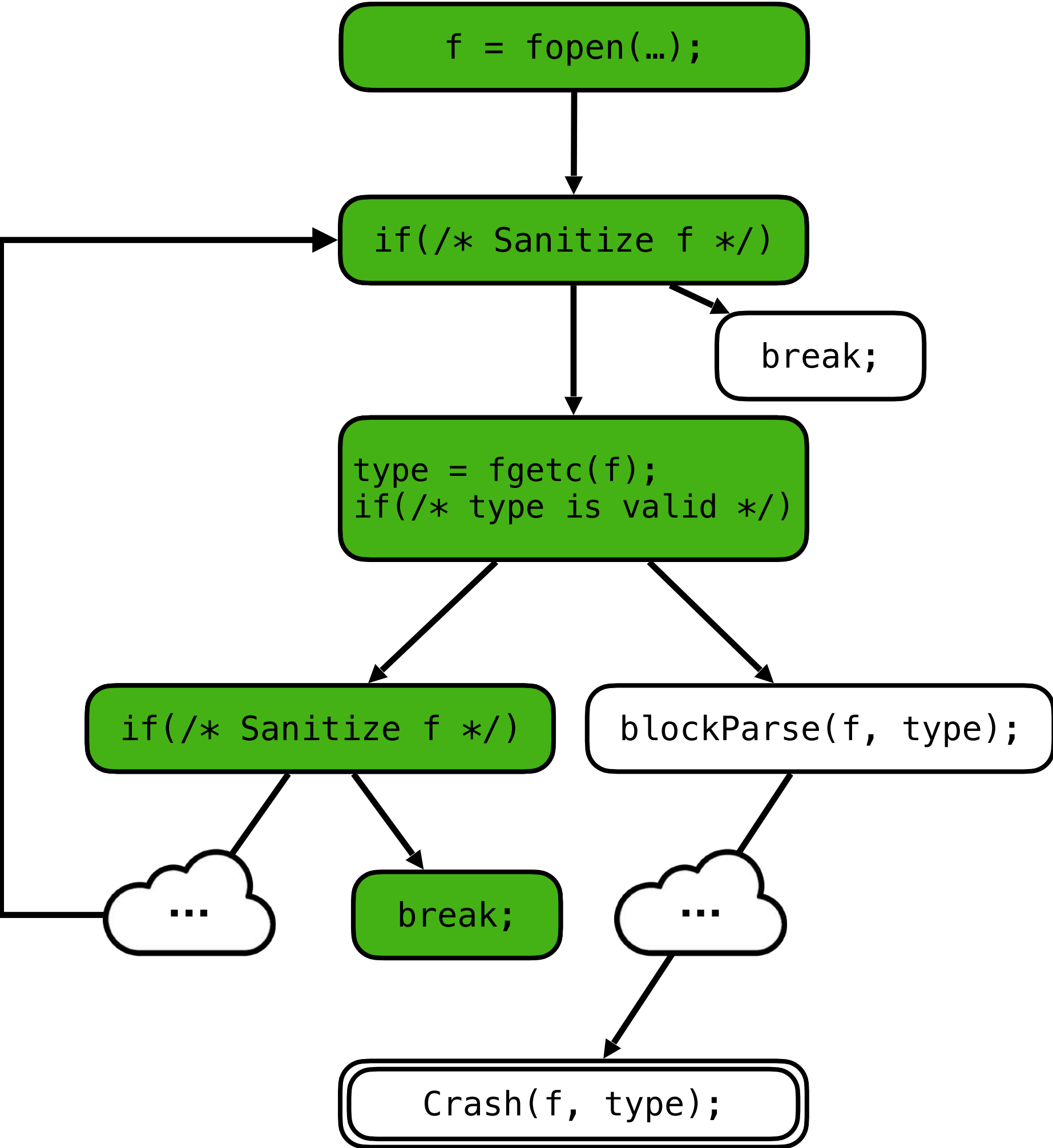
의미 연관성 점수

# 정의-사용 그래프와 입력의 유용성



# 정의-사용 그래프와 입력의 유용성

SA  1점  
SB  2점 더 우대하여 사용!

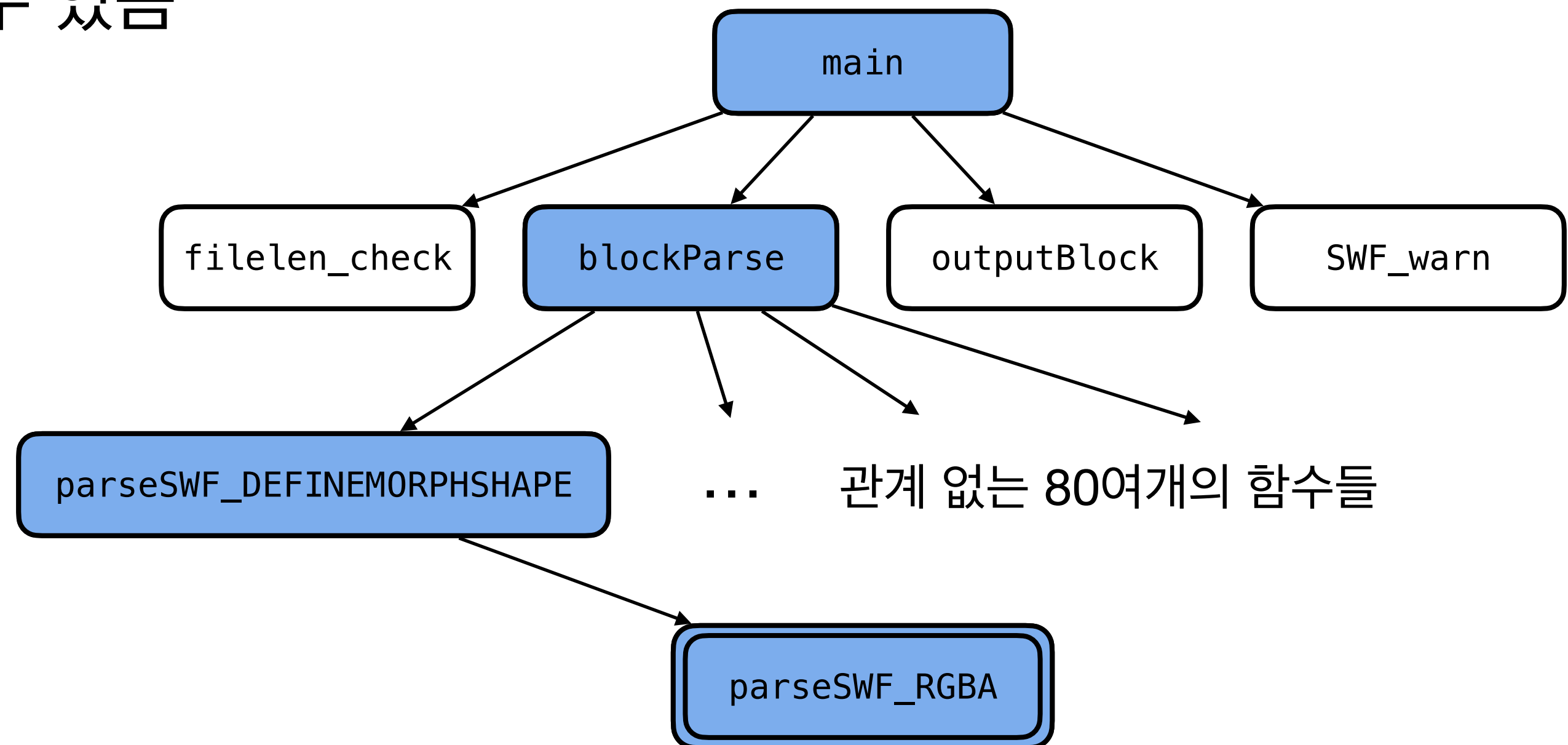




# 정의-사용 그래프와 덮이 관찰

## 선별적 덮이 관찰

- 목표에 대한 정의-사용 그래프가 지나는 함수를 선정
- 선정된 함수로부터만 덮이 관찰
- 원하는 영역에만 탐색을 집중할 수 있음

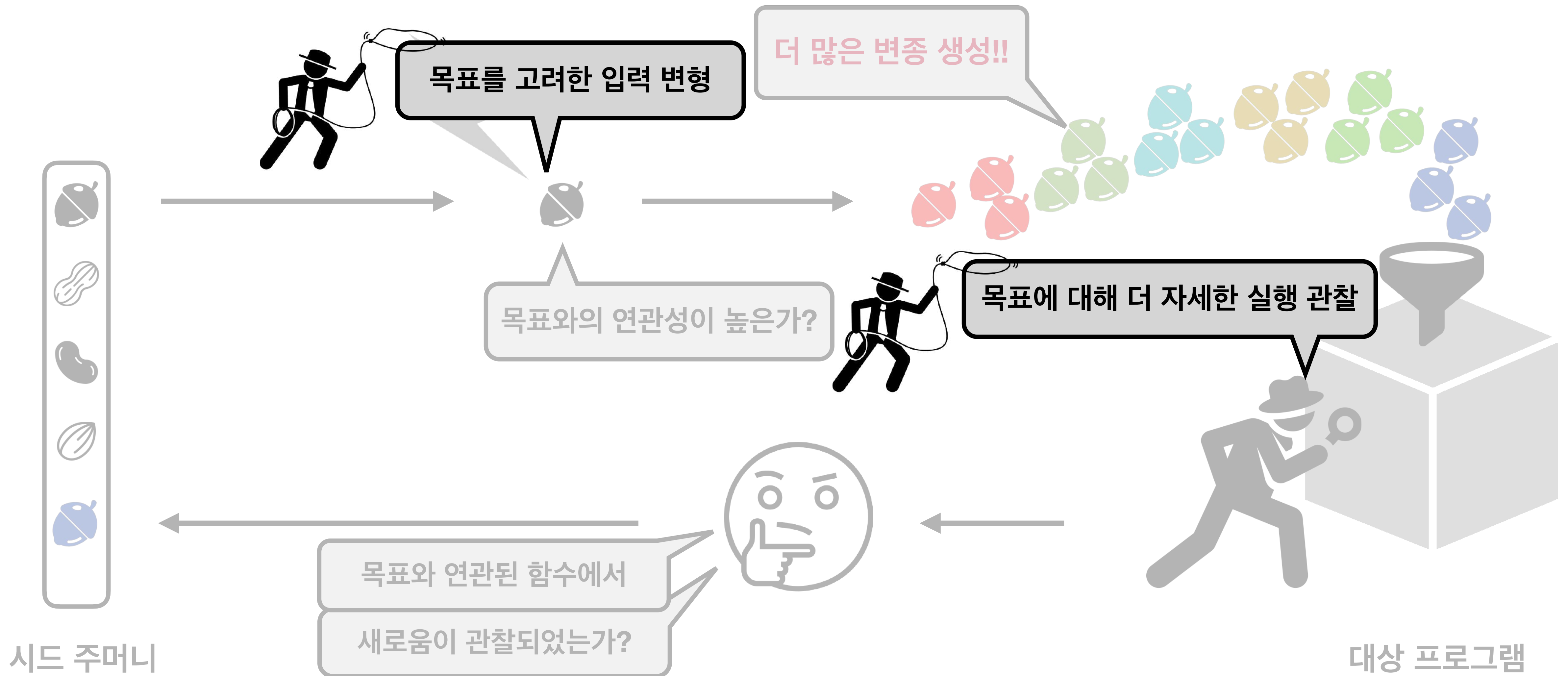


# 정적 분석으로 길들인 지향성 마구실행기



# 보다 목표 집중적인 지향성 마구실행기

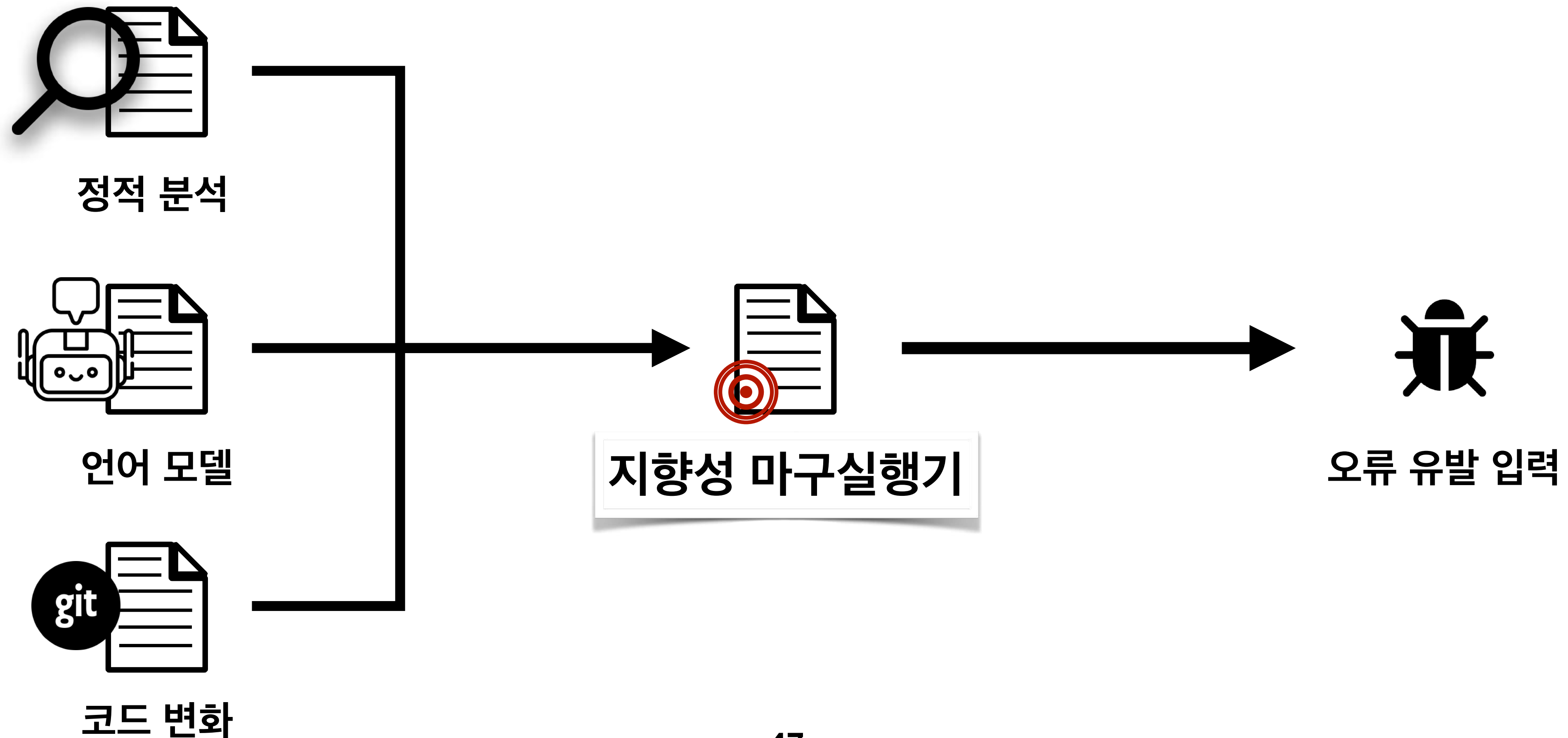
[포스터 발표]  
바람둥이 VS 순애보:  
지향성 마구실행기의 목표 집중 탐색 전략





# 오류 탐색에 참 좋은 정적 분석

사례 1: 오류 의심 지점에서 오류 유발 입력 생성까지



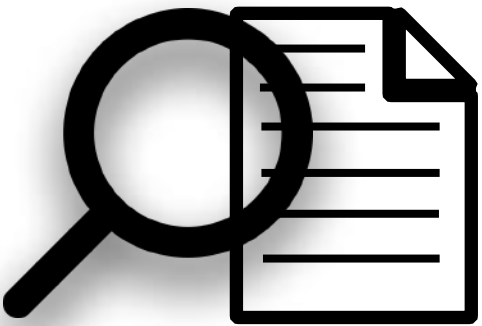
# 오류 탐색에 참 좋은 정적 분석

사례 2: 여러 오류 의심 지점에서 오류 유발 입력 생성까지

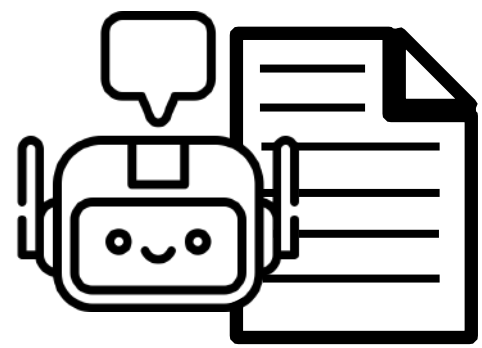


[박건, 장재훈]

다중 지향성 퍼징으로  
여러 군데 오류 한방에 검사하기



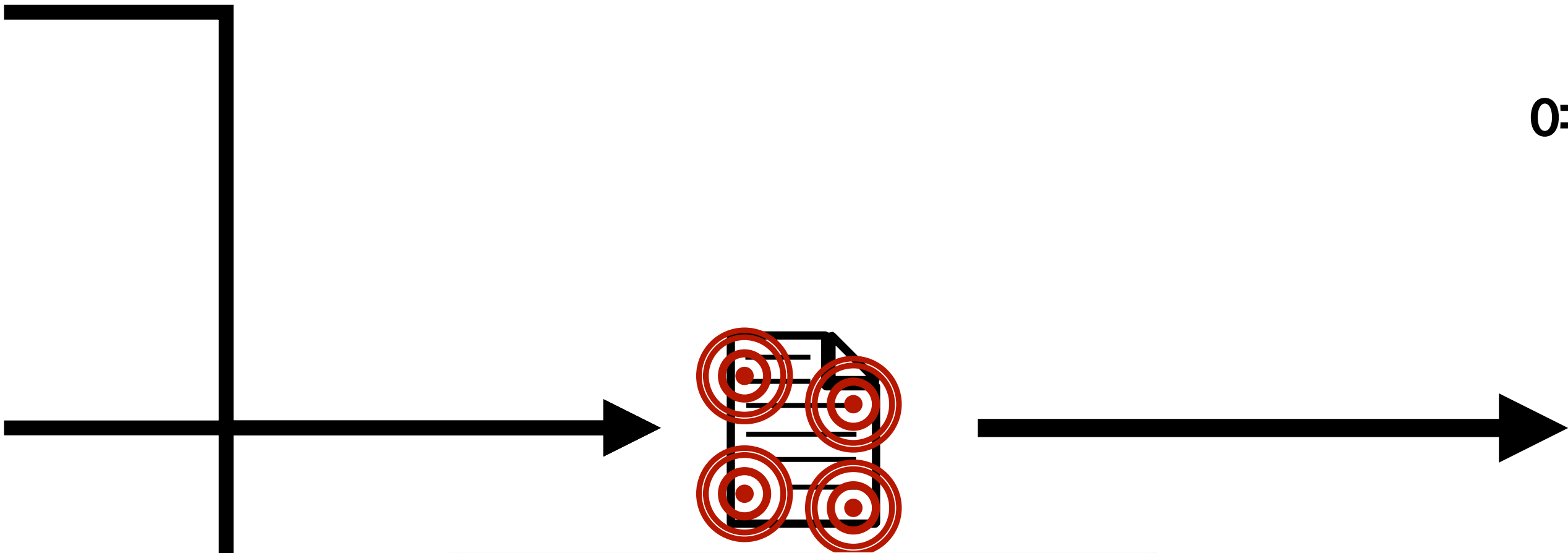
정적 분석



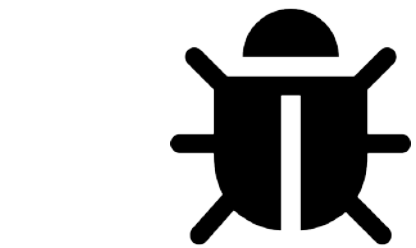
언어 모델



코드 변화



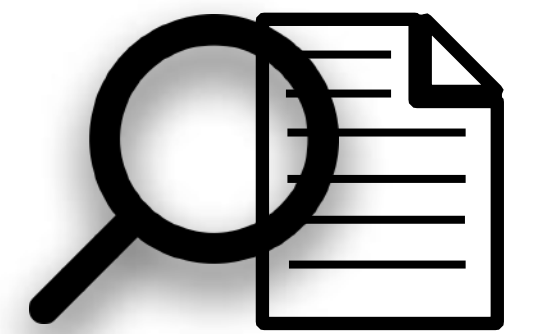
다중지향성 마구실행기



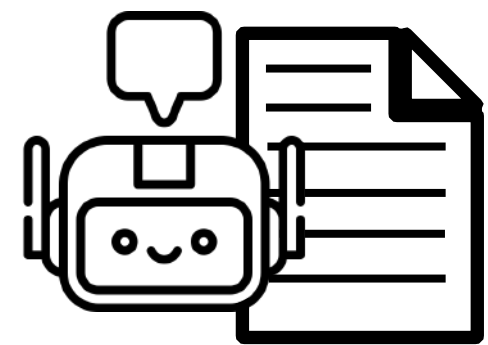
오류 유발 입력

# 오류 탐색에 참 좋은 정적 분석

사례 3: 라이브러리 프로그램을 위한 오류 유발 프로그램 생성



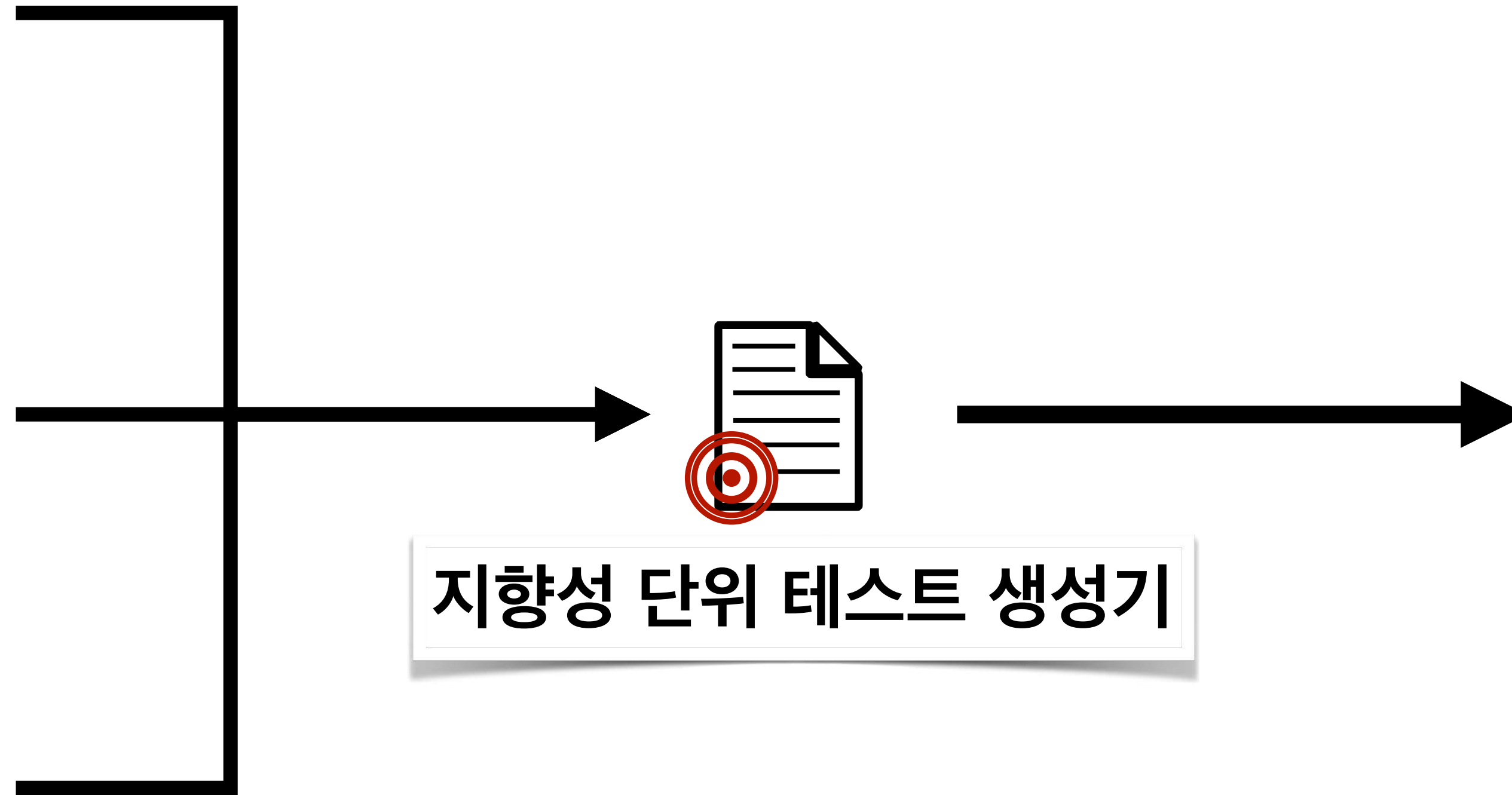
정적 분석



언어 모델



코드 변화



지향성 단위 테스트 생성기

오류 유발 프로그램



[장수진]

그 단위 테스트,  
정말 다른 거 맞아?



# 정리

## 정적 분석의 효능 1: AI의 두뇌 발달

- 오류 분석 결과를 활용하여 효과적으로 AI 강화 학습
- AI로 만든 코드의 오류 비율 대폭 감소

## 정적 분석의 효능 2: 마구실행기/프로그램합성기의 혈액 순환 촉진

- 오류의 인과관계 정보를 활용하여 효과적으로 오류 입력 탐색
- 오류 유발 입력 생성 속도 대폭 향상

“코드는 말을 하지 않는다.  
하지만 정적 분석은 듣는다.”

