

정적 분석과 데이터를 활용한 오류 검출: 보편적 오류와 특수한 오류

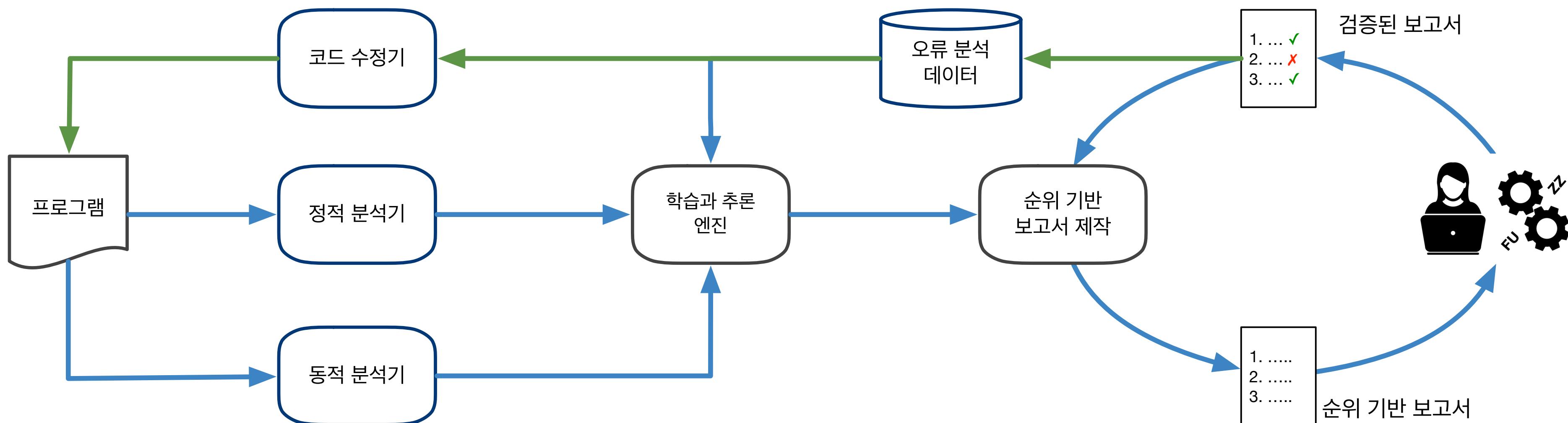
허기홍

KAIST 전산학부

소프트웨어 재난 연구센터 워크샵

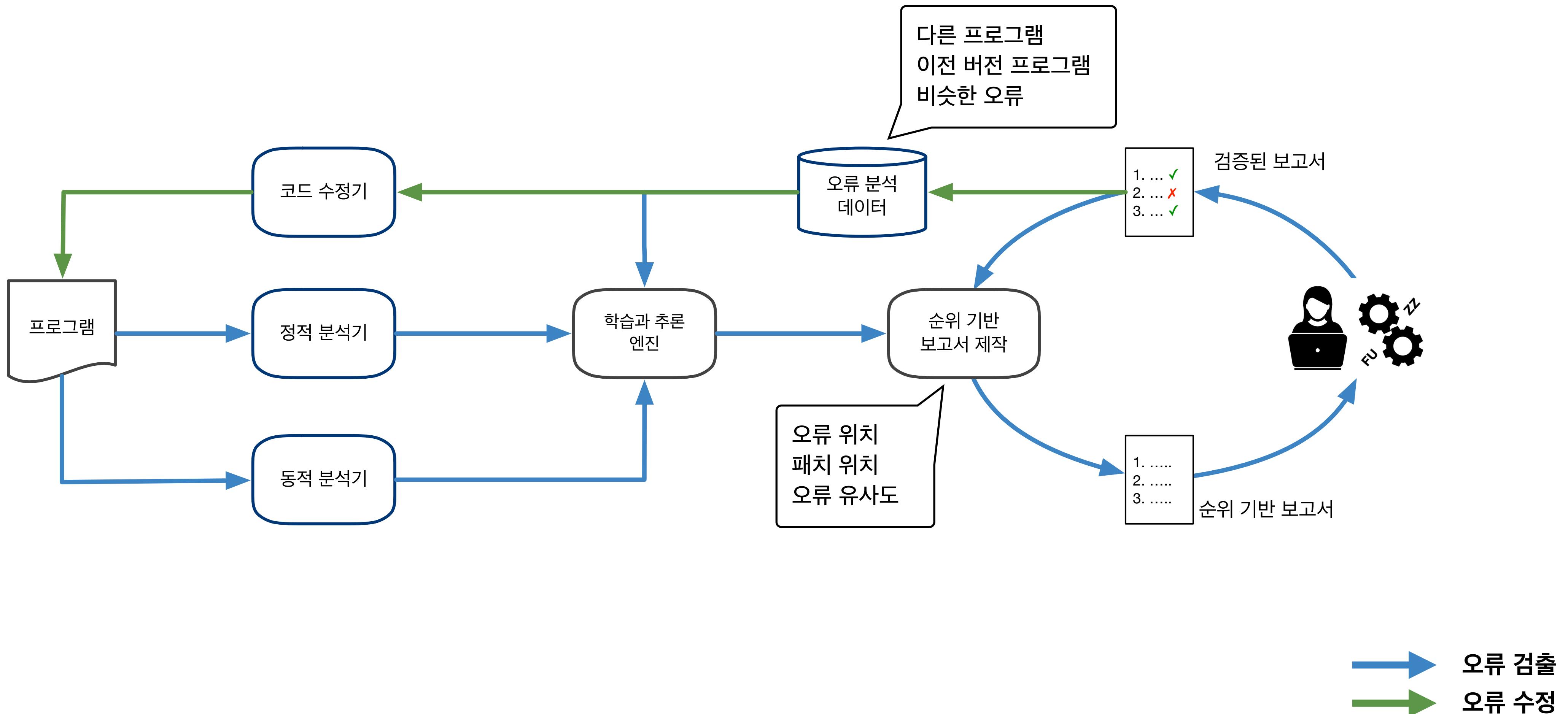


연구실 행보

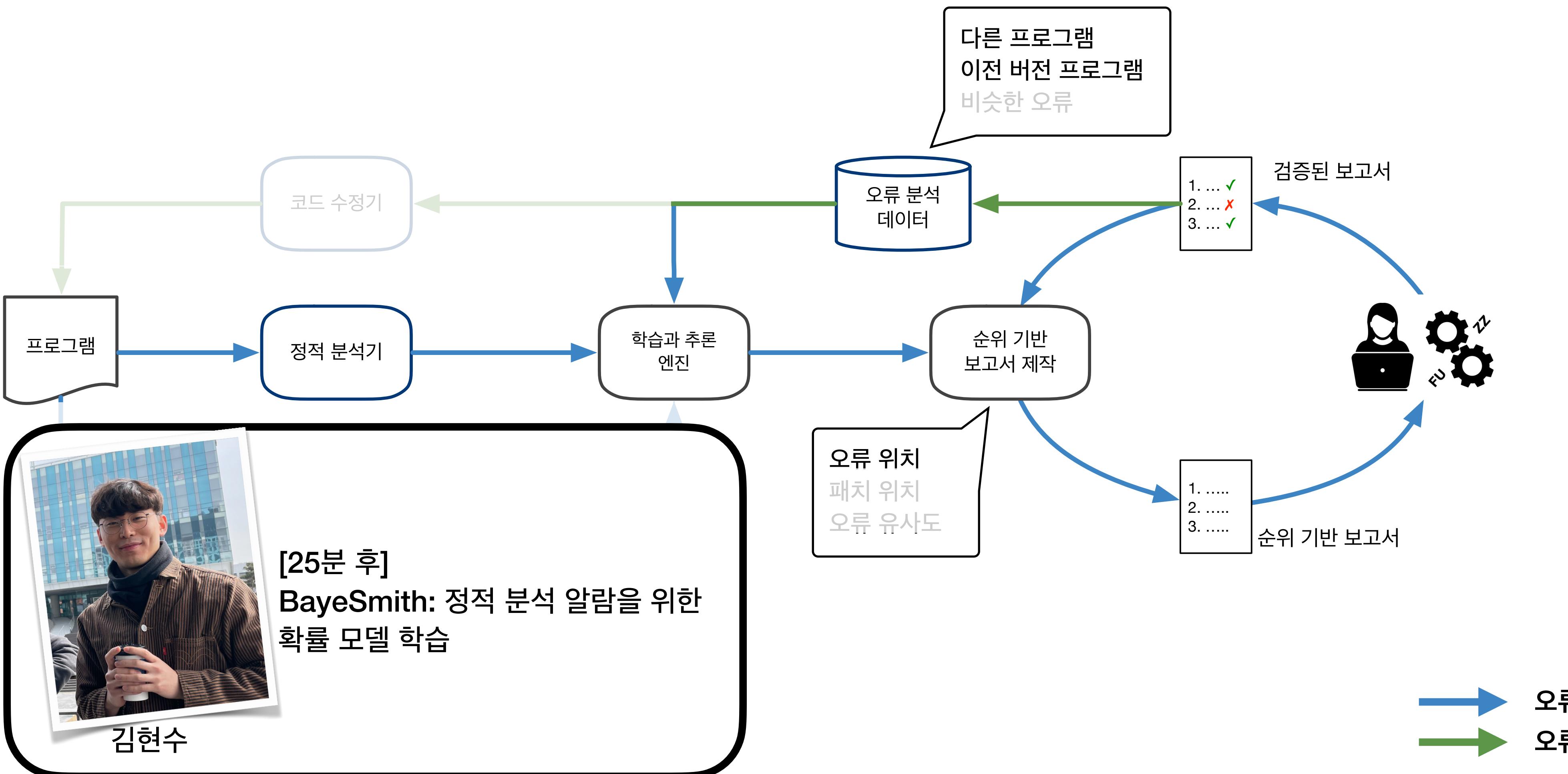


→ 오류 검출
→ 오류 수정

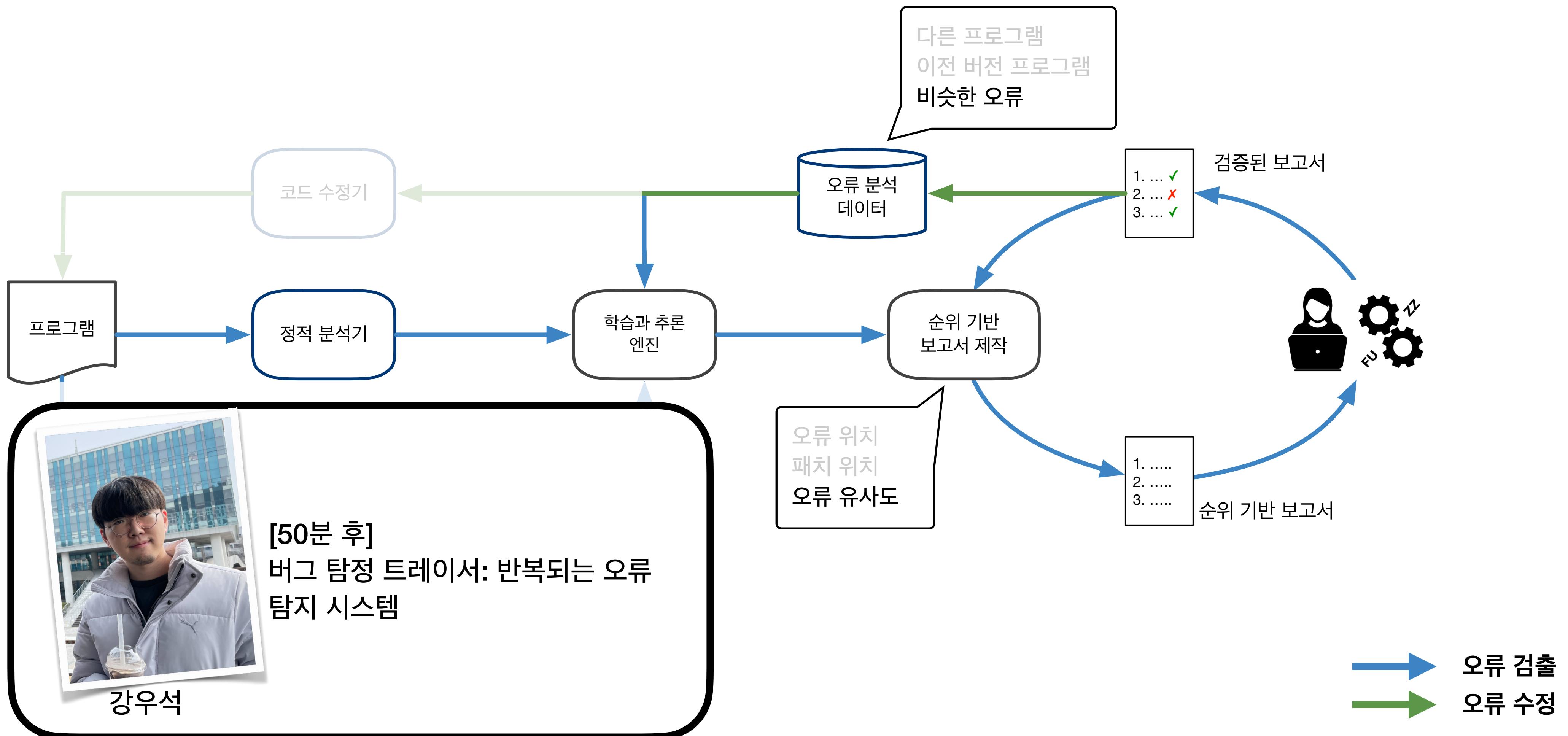
연구실 행보



연구실 행보



연구실 행보

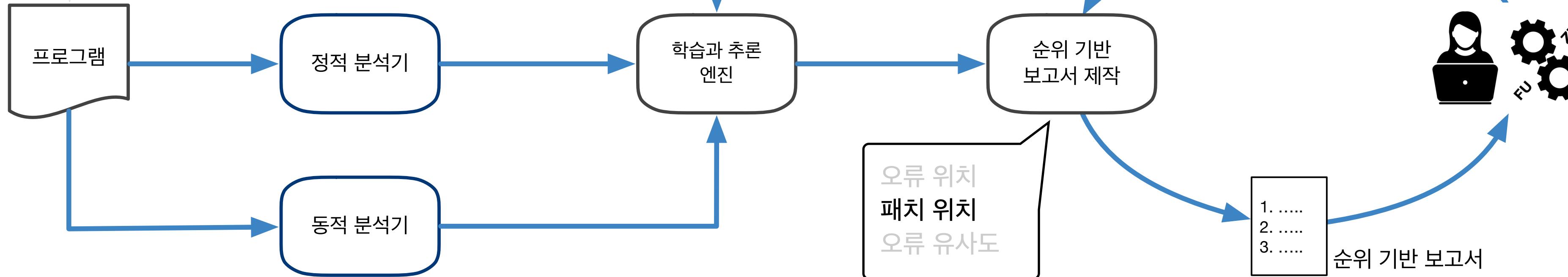


연구실 행보



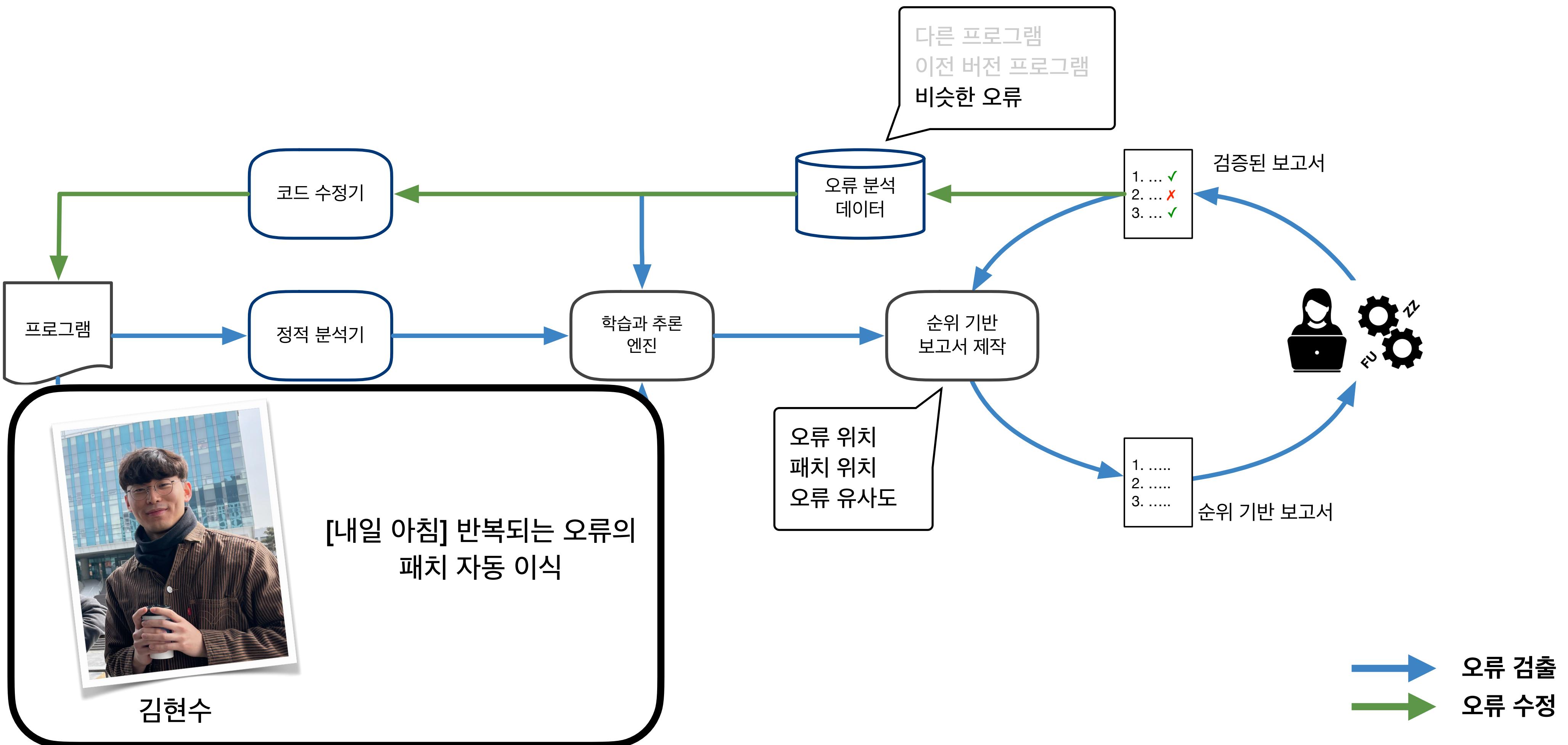
박종찬, 김재호, 김태은

[오늘 저녁] 코드의 변신은 유죄! 코드 변화 분석 기반 결함 위치 추정

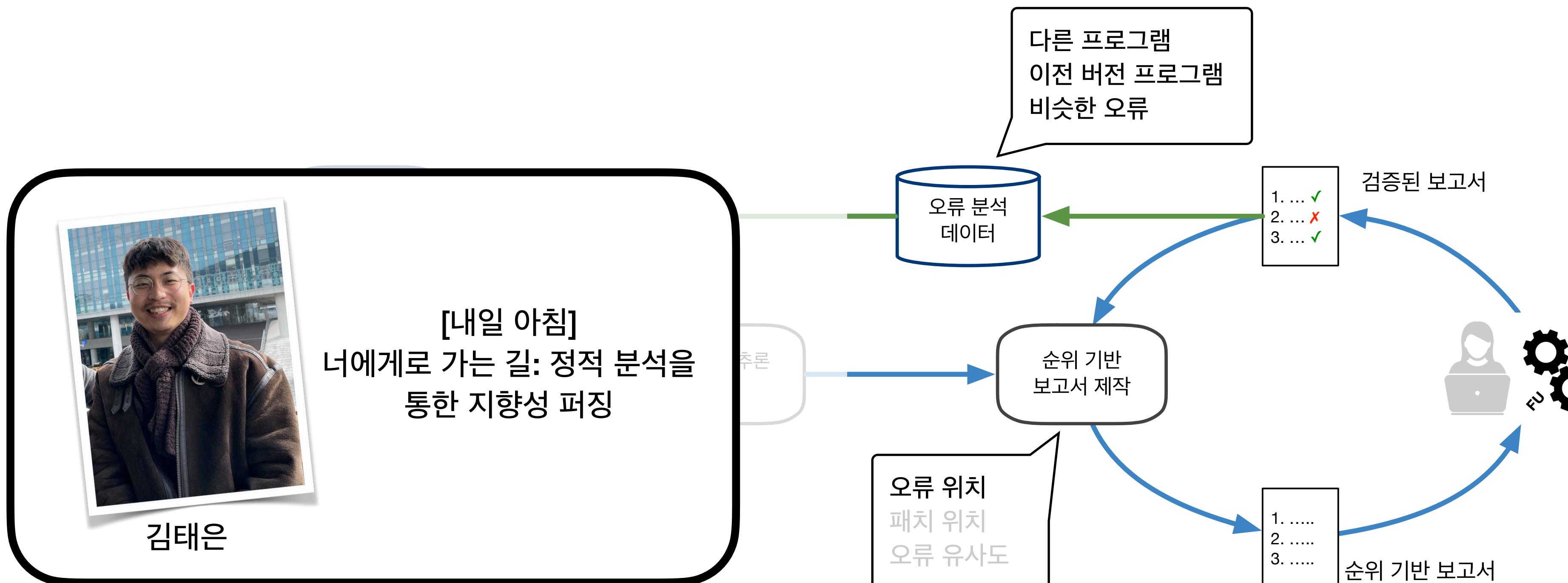


→ 오류 검출
→ 오류 수정

연구실 행보



연구실 행보



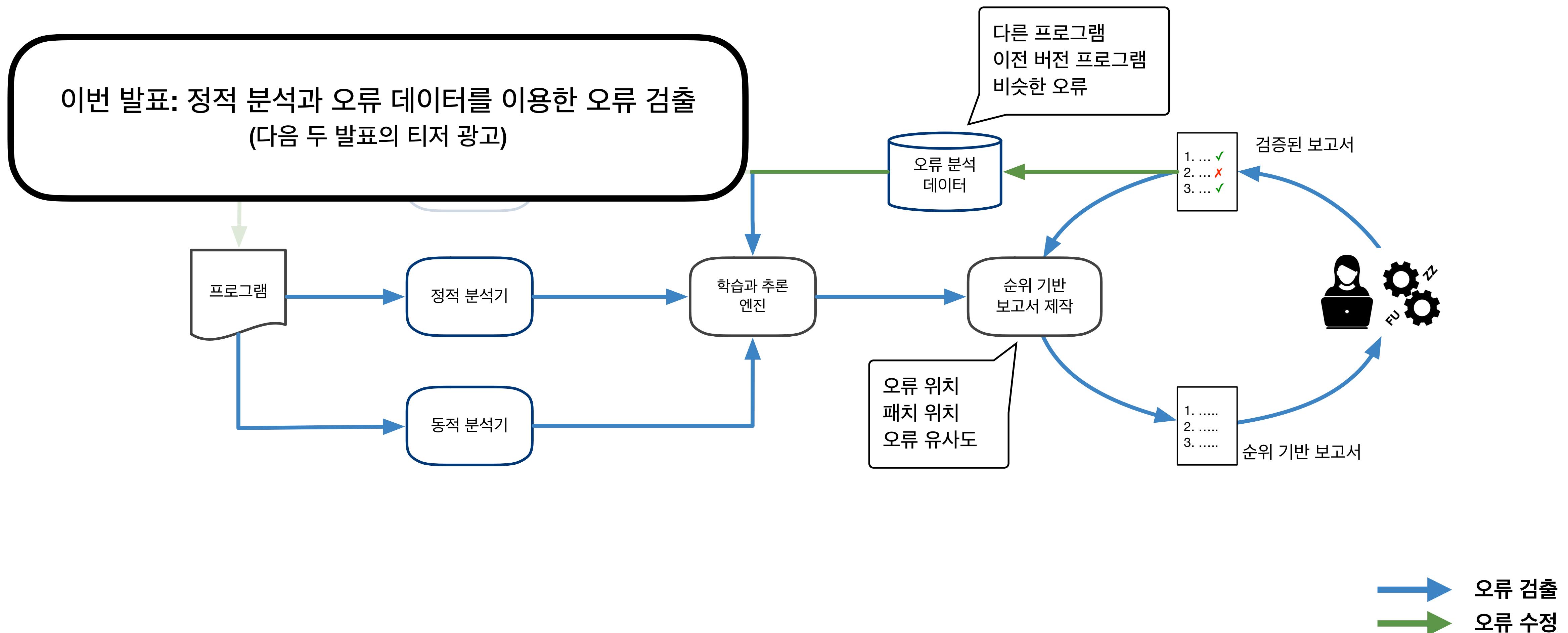
→ 오류 검출
→ 오류 수정

연구실 행보



연구실 행보

이번 발표: 정적 분석과 오류 데이터를 이용한 오류 검출 (다음 두 발표의 티저 광고)



SW 오류 검출

- 보편적 오류 검출
 - 대상: 어디에나 존재하는 오류
- 특수한 오류
 - 대상: 특정한 상황에 존재하는 오류

```
void f() {  
    char buf[16];  
    int i = 0;  
    while (i < 16) i++;  
    // buffer overrun  
    buf[i] = 0;  
}
```

```
void g() {  
    char buf[16];  
    char *p = input();  
    // buffer overrun  
    strcpy(buf, p);  
}
```

```
void read_bmp(File *f) {  
    int w = read_bmp_width(f);  
    int h = read_bmp_height(f);  
    // integer overflow  
    int area = w * h;  
    char *buf = malloc(area);  
    ...  
}
```

```
void read_jpg(File *f) {  
    int w = read_jpg_width(f);  
    int h = read_jpg_height(f);  
    // integer overflow  
    int area = w * h;  
    char *buf = malloc(area);  
    ...  
}
```

SW 오류 검출

- 보편적 오류 검출
 - 대상: 어디에나 존재하는 오류
 - 방법: 보편적인 논리 성질을 검사
- 특수한 오류
 - 대상: 특정한 상황에 존재하는 오류
 - 방법: 해당 오류 패턴을 검사

`idx < size?`

`area < INT_MAX?`

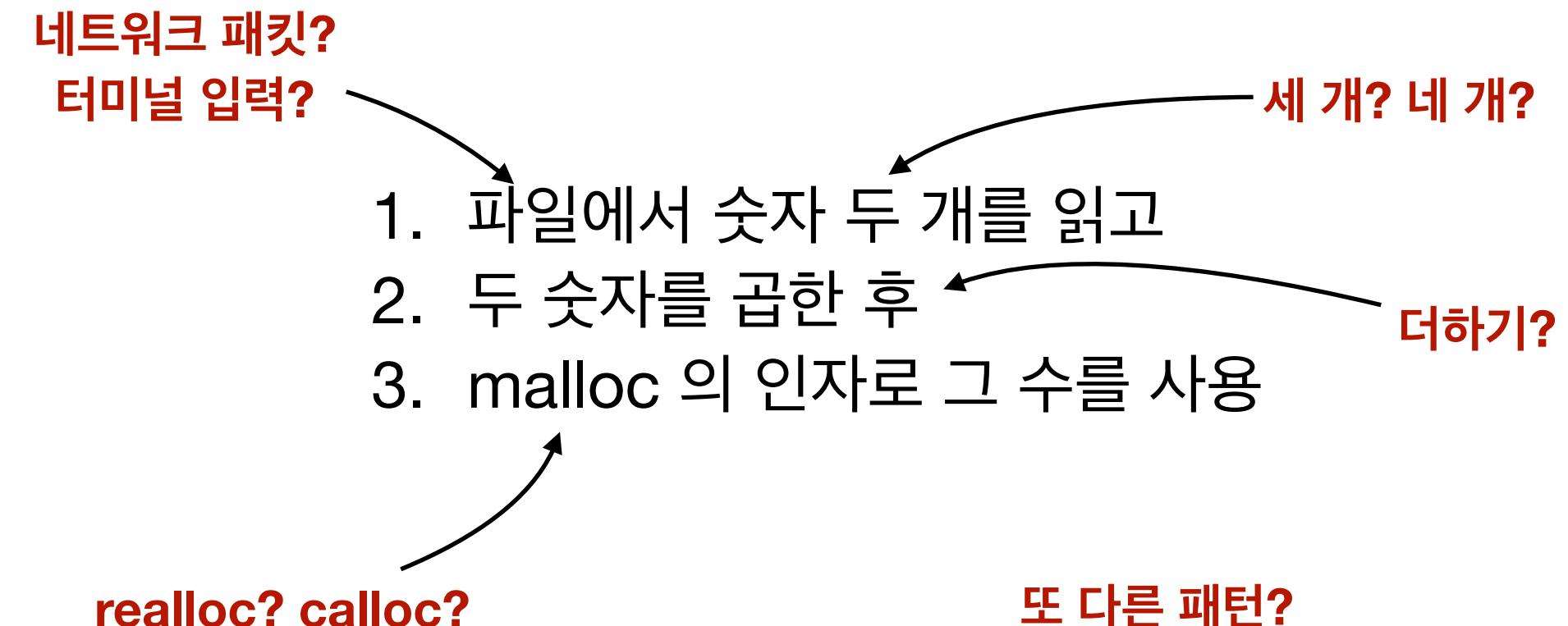
`p != NULL?`

1. 파일에서 숫자 두 개를 읽고
2. 두 숫자를 곱한 후
3. malloc 의 인자로 그 수를 사용

SW 오류 검출

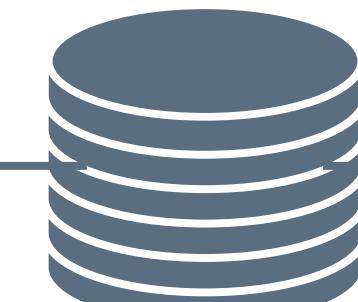
- 보편적 오류 검출
 - 대상: 어디에나 존재하는 오류
 - 방법: 보편적인 논리 성질을 검사
 - 문제점: 많은 오탐
- 특수한 오류
 - 대상: 특정한 상황에 존재하는 오류
 - 방법: 해당 오류 패턴을 검사
 - 문제점: 많은 미탐

포인터?
순환문?
외부 라이브러리?
함수 호출?



SW 오류 검출

- 보편적 오류 검출
 - 대상: 어디에나 존재하는 오류
 - 방법: 보편적인 논리 성질을 검사
 - 문제점: 많은 오탐
 - 해결책: 정적 분석 + 여러 환경과 상호작용
- 특수한 오류
 - 대상: 특정한 상황에 존재하는 오류
 - 방법: 해당 오류 패턴을 검사
 - 문제점: 많은 미탐
 - 해결책: 정적 분석 + 알려진 오류와 유사도 비교



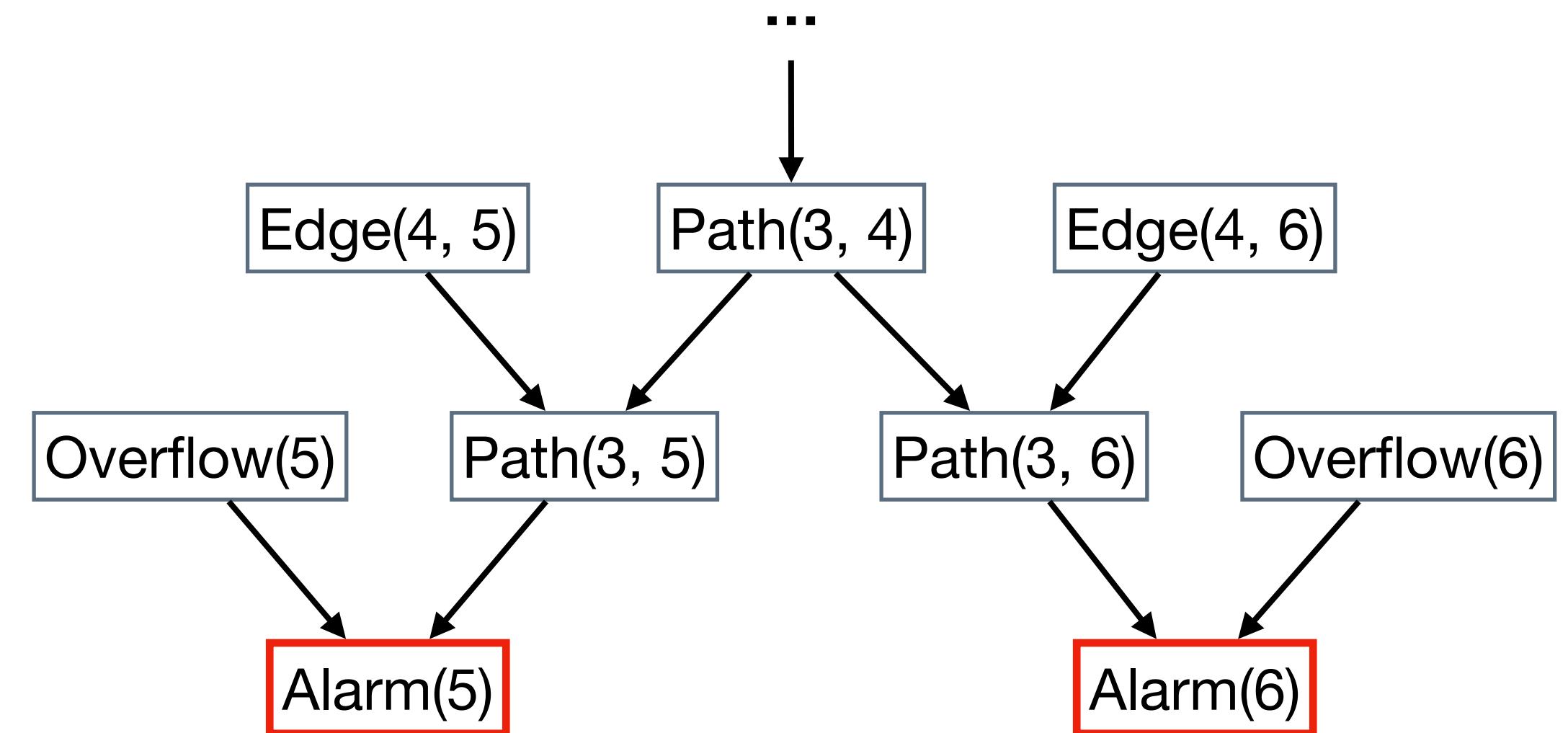
오류 분석 데이터

정적 분석을 위한 베이지안 추론

1. 정적 분석이 경보를 낸 논리 추론 단계를 그래프로 기록

예: 요약 의미를 정의-사용 관계로 함축 표현한 그래프

```
1: char *line = input();
2: char *tok = strtok(line, " ");
3: char *p = tok + strlen(tok);
4: while (p > tok) {
5:     if (is_digit(*p)) break; // false alarm
6:     if (is_alpha(*p)) break; // false alarm
7:     p--;
8: }
```

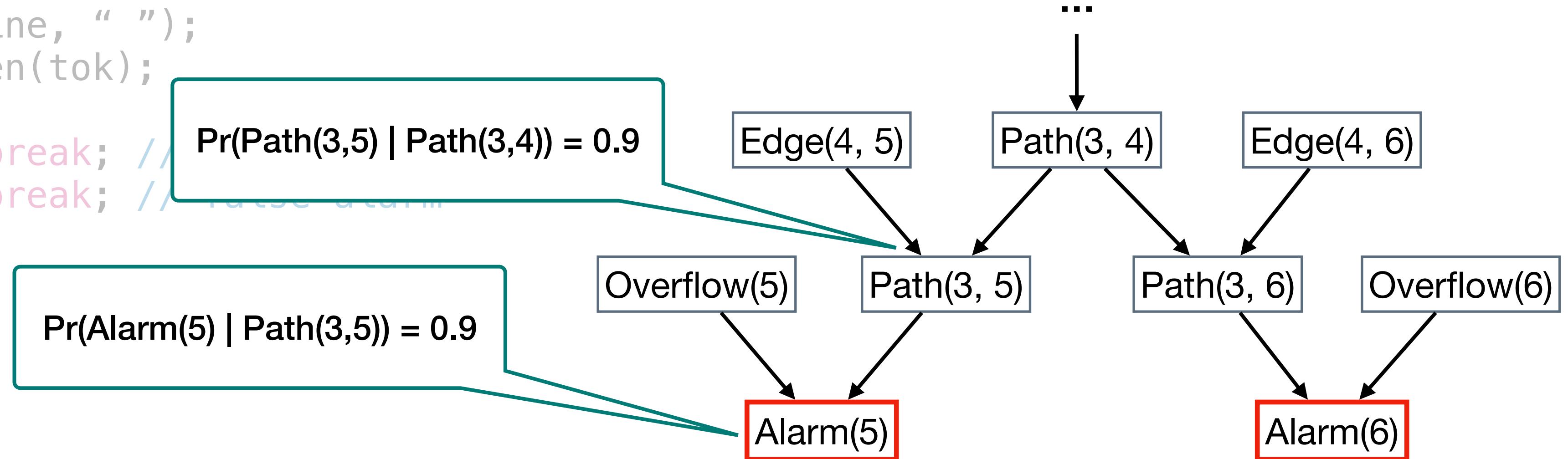


정적 분석을 위한 베이지안 추론

2. 논리 추론 그래프를 베이지안 네트워크로 변환

정적 분석 결과와 실제 실행 결과의 불일치를 정량화

```
1: char *line = input();
2: char *tok = strtok(line, " ");
3: char *p = tok + strlen(tok);
4: while (p > tok) {
5:   if (is_digit(*p)) break; // false alarm
6:   if (is_alpha(*p)) break; // false alarm
7:   p--;
8: }
```

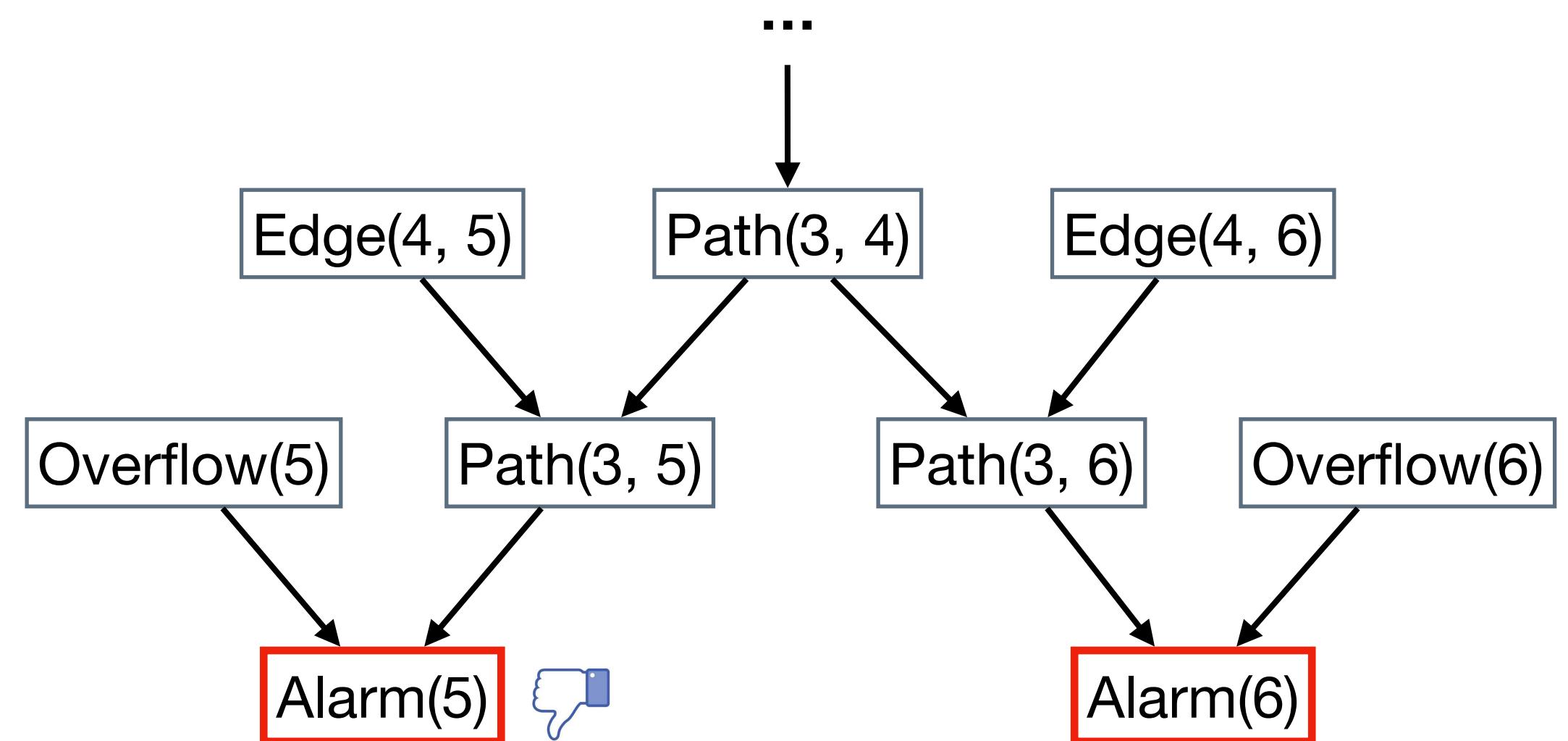


정적 분석을 위한 베이지안 추론

3. 경보를 확률 순으로 정렬 + 사후 관찰을 통한 상호작용

Ranking	a	Pr(a)
1	Alarm(5)	0.95
2	Alarm(6)	0.95
...		
Ranking	a	Pr(a \neg Alarm(5))
...		
54	Alarm(6)	0.43
-	Alarm(5)	0

사후 확률 계산 (Bayes rule)



프로그래밍 환경과 상호작용하는 정적 분석

$$\Pr(a | e)$$

상호작용 대상	e	오류 검출에 드는 노력
사용자 [PLDI'18]	사용자의 검산 결과	44% 감소
이전 버전 [PLDI'19]	이전 버전의 분석 결과	65% 감소
동적 분석 [FSE'21]	프로그램의 실행 결과	35% 감소
...

Q. 많은 오류 데이터를 이용하여 더 좋은 확률 모델을 어떻게 학습할 수 있을까?

김현수, BayeSmith: 정적 분석 알람을 위한 확률 모델 학습 [ICSE'22]

SW 오류 검출

- 보편적 오류 검출
 - 대상: 어디에나 존재하는 오류
 - 방법: 보편적인 논리 성질을 검사
 - 문제점: 많은 오탐
 - 해결책: 정적 분석 + 여러 환경과 상호작용
- 특수한 오류
 - 대상: 특정한 상황에 존재하는 오류
 - 방법: 해당 오류 패턴을 검사
 - 문제점: 많은 미탐
 - 해결책: 정적 분석 + 알려진 오류와 유사도 비교



반복되는 오류

```
long ToL (char *pbuffer) { return (pbuffer[0] | pbuffer[1]<<8 | pbuffer[2]<<16 | pbuffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(pbuffer[0] | pbuffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image_ID = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    buffer = malloc(rowbytes); // malloc with overflowed size
    ...
}
```

gimp-2.6.7 (CVE-2009-1570)

반복되는 오류

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #1");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);
    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image.bitmap = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    unsigned char *buffer = (unsigned char*) new char[rowbytes]; // malloc with overflowed size
    ...
}
```

sam2p-0.49.4 (CVE-2017-1570)

반복되는 오류

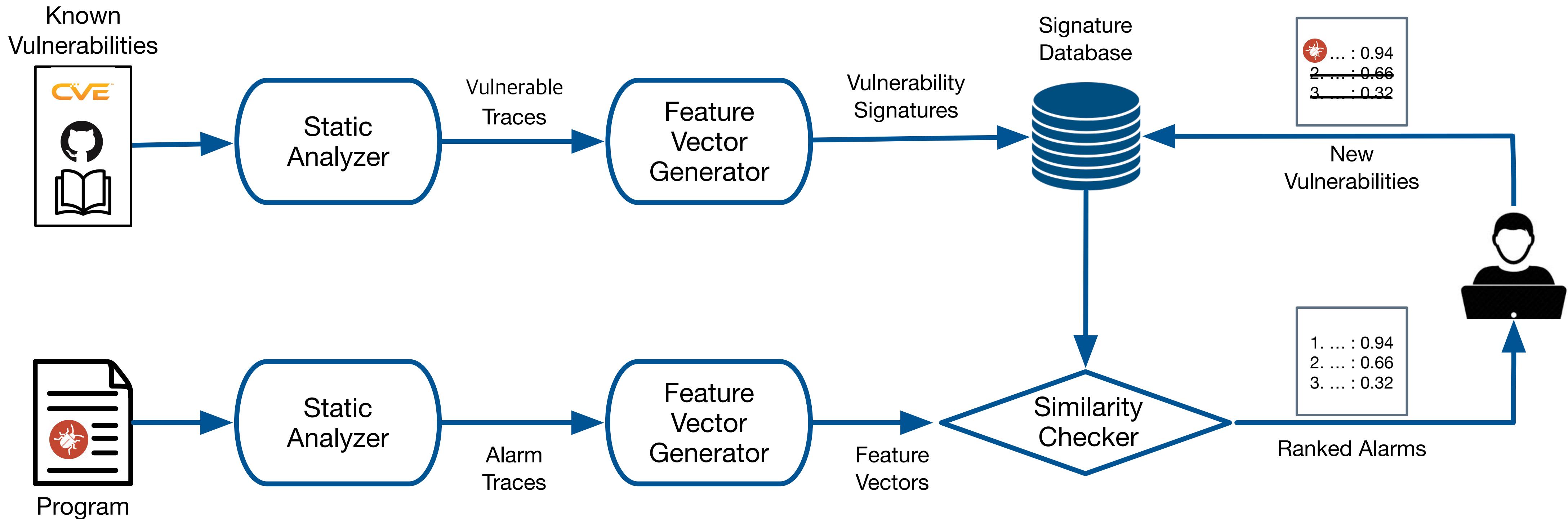
```
long ToL (char *pbuffer) { return (pbuffer[0] | pbuffer[1]<<8 | pbuffer[2]<<16 | pbuffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(pbuffer[0] | pbuffer[1]<<8)); }

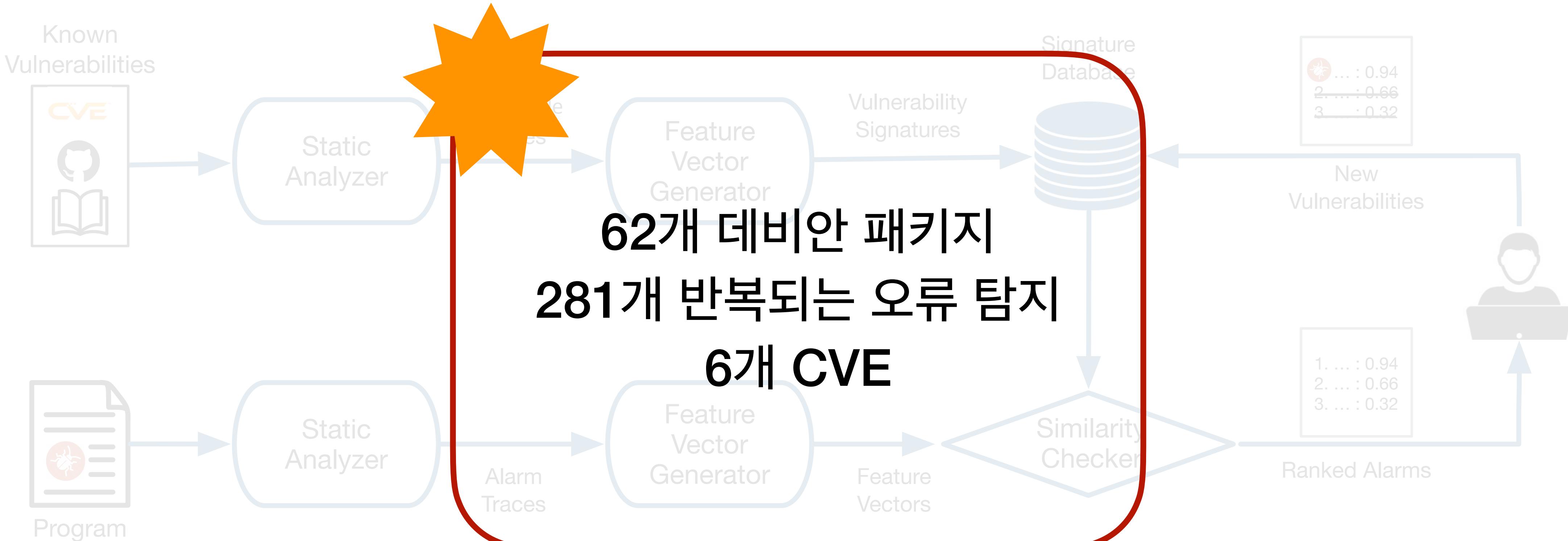
gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head_size, 1, file)) {
        FATALP ("Bitmap_Head.biWidth >= 0");
        long ToL (char *pbuffer);
        static XcursorBool _XcursorReadUInt (XcursorFile *file, XcursorUInt *u) {
            unsigned char bytes[4];
            if ((*file->read)(file, bytes, 4) != 4) return XcursorFalse;
            *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) | (bytes[3] << 24));
            return XcursorTrue;
        }
        short ToS (char *pbuffer);
        _XcursorReadImage (XcursorFile *file, XcursorFileHeader *fileHeader, int toc) {
            XcursorChunkHeader chunkHeader;
            XcursorImage head;
            ...
            if (!_XcursorReadUInt (file, &head.width))
                return NULL;
            if (!_XcursorReadUInt (file, &head.height))
                return NULL;
            image = XcursorImageCreate(head.width, head.height);
            ...
        }
        XcursorImage *XcursorImageCreate (int width, int height) {
            image = malloc (sizeof (XcursorImage) + width * height * sizeof (XcursorPixel));
            ...
        }
    }
}
```

libXcursor-1.1.14 (CVE-2017-16612)

시그니처 기반 오류 분석



시그니처 기반 오류 분석



마무리

- 저희 연구실의 목표: 정적 분석 + 오류 데이터를 이용한 정확한 오류 검출과 수정
 - 일반적 오류: 정적 분석기 주변의 다양하고 유용한 데이터 (e.g., 동적 분석, VCS, 사용자 반응 등)
 - 특수한 오류: 여러 상황에서 발생한 오류 데이터 (e.g., 버그 리포트, 시큐어 코딩 가이드라인 등)

