PLDI COPENHAGEN 2024

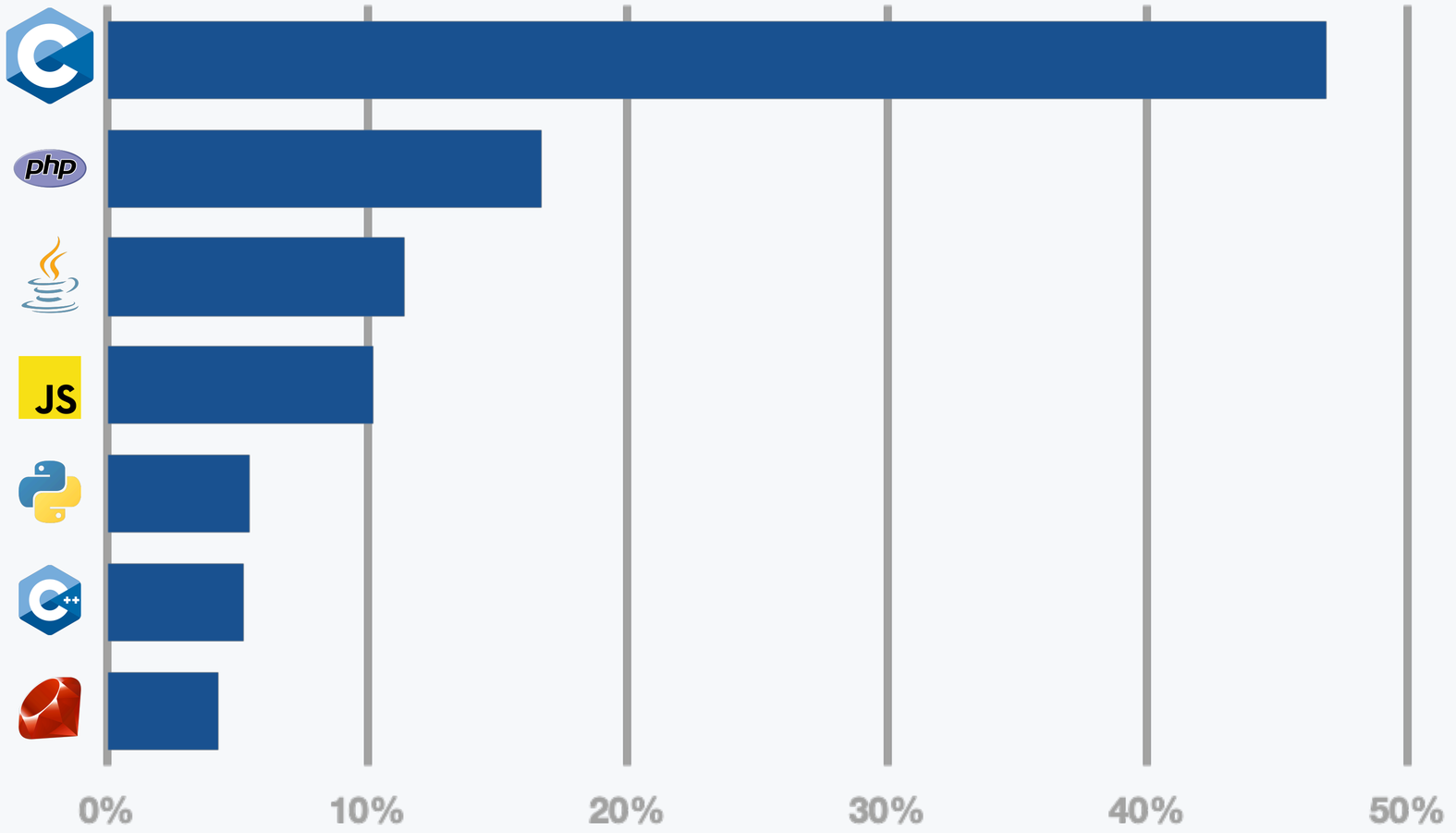
Don’t Write, but Return

Replacing Output Parameters with Algebraic Data Types in C-to-Rust Translation

홍재민, 류석영(KAIST) · {jaemin.hong, sryu.cs}@kaist.ac.kr

PLRG

C의 낮은 안전성

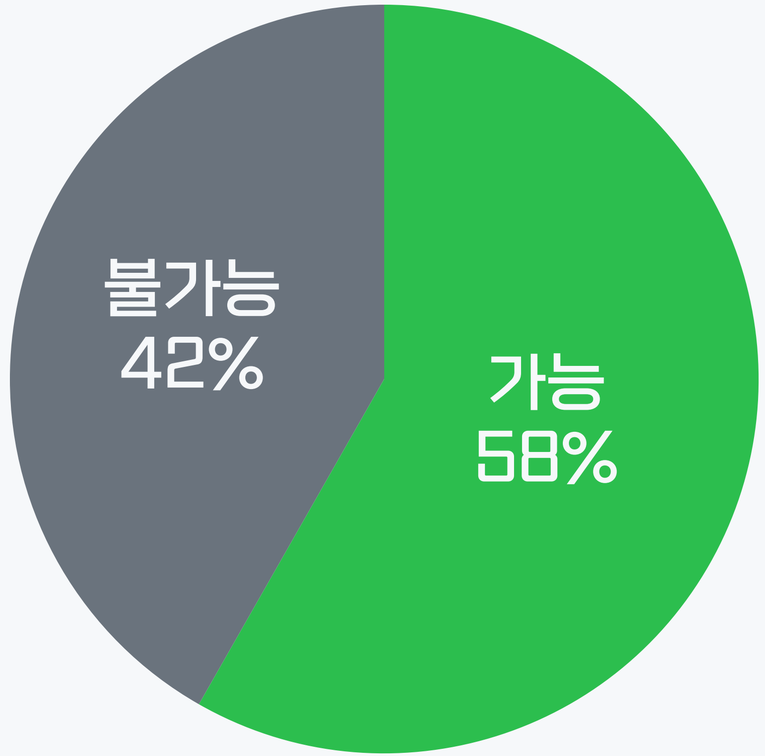


오픈소스 보안 취약점의 언어별 비율

WhiteSource Report: What are the Most Secure Programming Languages

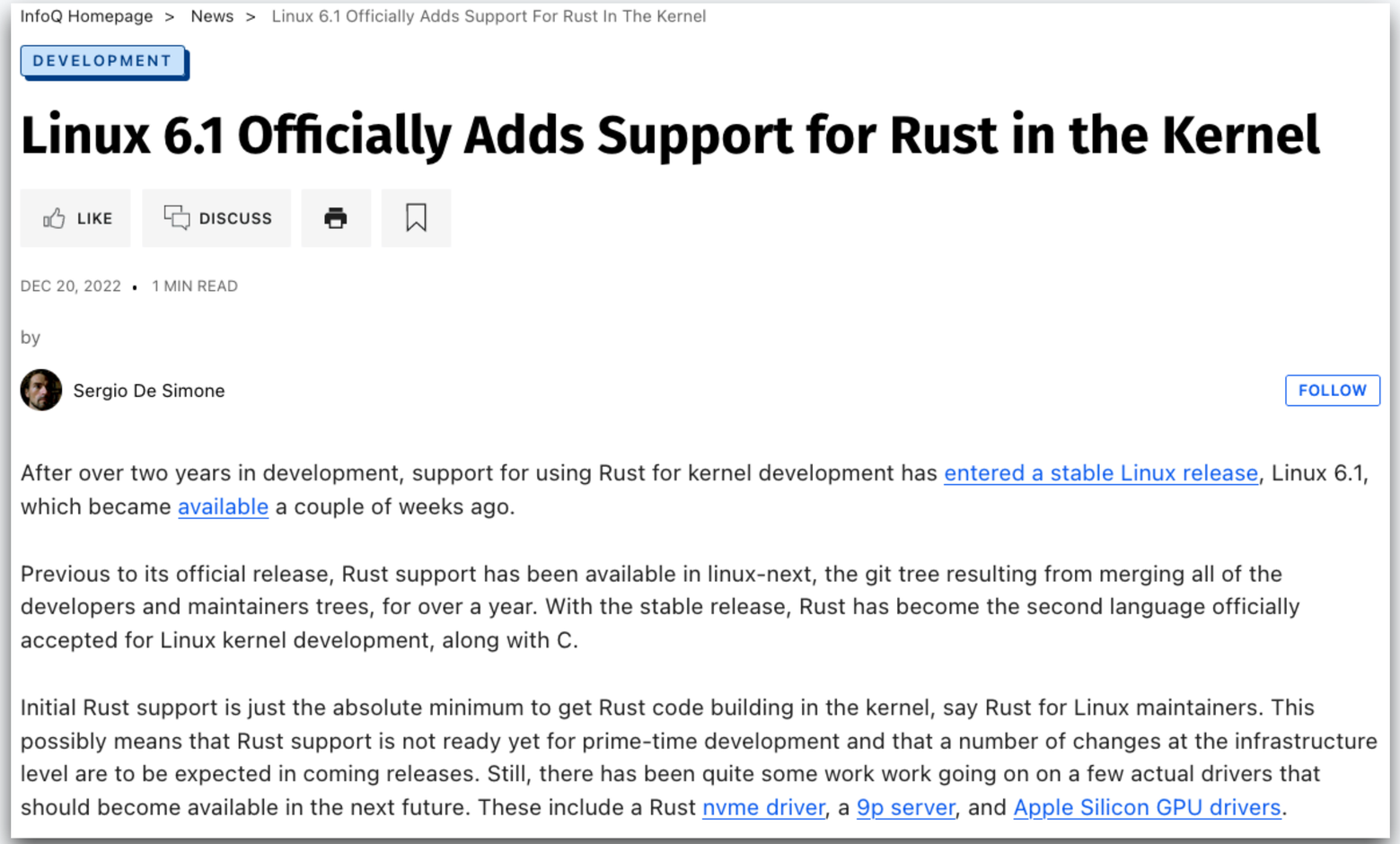
러스트의 높은 안전성

```
error[E0499]: cannot borrow `s` as mutable more than once at a time
 --> b.rs:5:14
  |
4 |     let r1 = &mut s;
  |              ------ first mutable borrow occurs here
5 |     let r2 = &mut s;
  |              ^^^^^^ second mutable borrow occurs here
6 |
7 |     println!("{}, {}", r1, r2);
  |                        -- first borrow later used here

error: aborting due to previous error

For more information about this error, try `rustc --explain E0499`.
```



cURL의 버그를 러스트가 방지할 수 있었는가?

러스트가 기존 시스템에 도입되는 중



InfoQ Homepage > News > Linux 6.1 Officially Adds Support For Rust In The Kernel

DEVELOPMENT

Linux 6.1 Officially Adds Support for Rust in the Kernel

LIKE DISCUSS

DEC 20, 2022 • 1 MIN READ

by

Sergio De Simone FOLLOW

After over two years in development, support for using Rust for kernel development has entered a stable Linux release, Linux 6.1, which became available a couple of weeks ago.

Previous to its official release, Rust support has been available in linux-next, the git tree resulting from merging all of the developers and maintainers trees, for over a year. With the stable release, Rust has become the second language officially accepted for Linux kernel development, along with C.

Initial Rust support is just the absolute minimum to get Rust code building in the kernel, say Rust for Linux maintainers. This possibly means that Rust support is not ready yet for prime-time development and that a number of changes at the infrastructure level are to be expected in coming releases. Still, there has been quite some work work going on on a few actual drivers that should become available in the next future. These include a Rust nvme driver, a 9p server, and Apple Silicon GPU drivers.

https://www.infoq.com/news/2022/12/linux-6-1-rust/

러스트의 대수적 타입

옵션 타입을 사용해
부분 함수를 표현

```
fn div(n: i32, d: i32) -> Option<i32> {
    if d == 0 {
        None
    } else {
        Some(n / d)
    }
}
```

튜플 타입을 사용해
여러 값을 반환하는
함수를 표현

```
fn div(n: i32, d: i32) -> (i32, i32) {
    (n / d, n % d)
}
```

C의 출력 매개변수

반환값은 성공 여부를
나타내고, 결과는
출력 매개변수에 씀

```c
int div(int n, int d, int *q) {
    if (d == 0) {
        return 1;
    } else {
        *q = n / d; return 0;
    }
}
```

결괏값 하나는 반환,
다른 하나는 출력
매개변수에 씀

```c
int div(int n, int d, int *r) {
    *r = n % d; return n / d;
}
```

C의 출력 매개변수

반환값은 성공 여부를
나타내고, 결과는
출력 매개변수에 씀

```
int div(int n, int d, int *q) {
    if (d == 0) {
        return 1;
    } else {
        *q = n / d; return 0;
    }
}

if div(n, d, q) == 0 {
    foo(*q);
}
```

C의 출력 매개변수

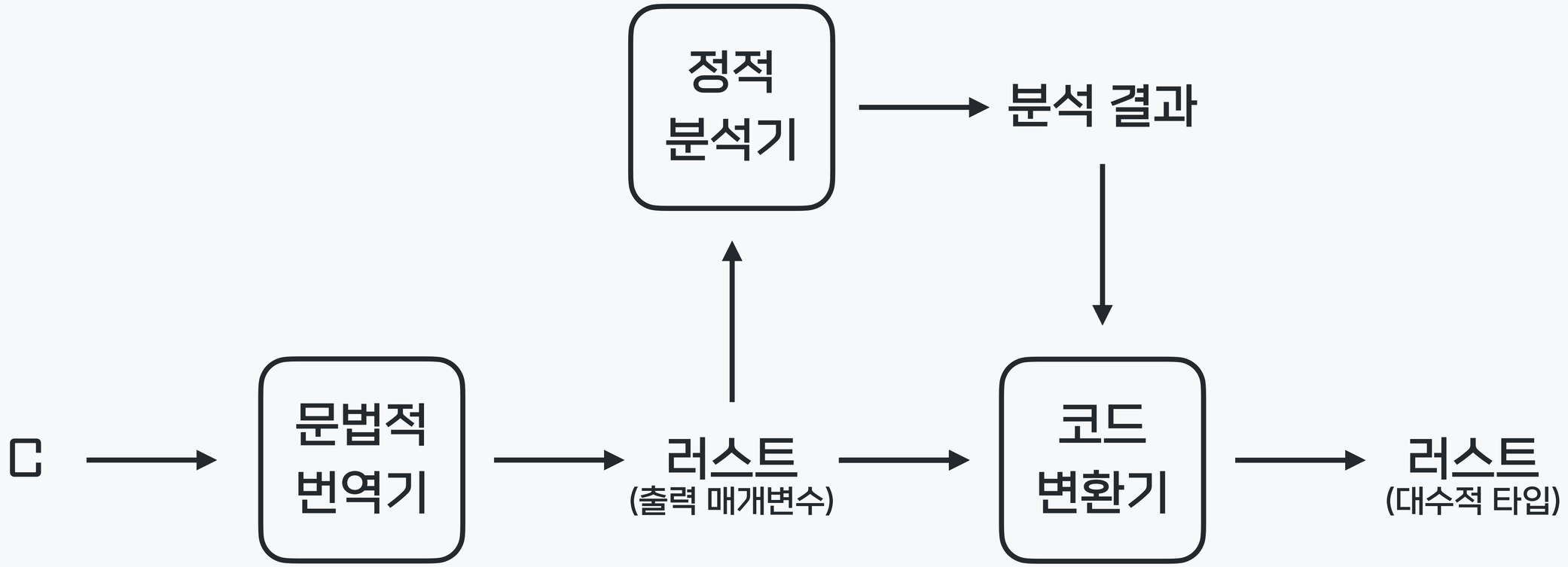
반환값은 성공 여부를
나타내고, 결과는
출력 매개변수에 씀

```
int div(int n, int d, int *q) {
    if (d == 0) {
        return 1;
    } else {
        *q = n / d; return 0;
    }
}

div(n, d, q);
foo(*q);
```

C2Rust의 문법적 번역

```
fn div(n: i32, d: i32, q: *mut i32) -> i32 {
    if d == 0 {
        return 1;
    } else {
        *q = n / d; return 0;
    }
}

fn div(n: i32, d: i32, r: *mut i32) -> i32 {
    *r = n % d; return n / d;
}
```

번역 방법 개요



거짓 양성 없이 출력 매개변수를 찾는 것이 정적 분석의 목표

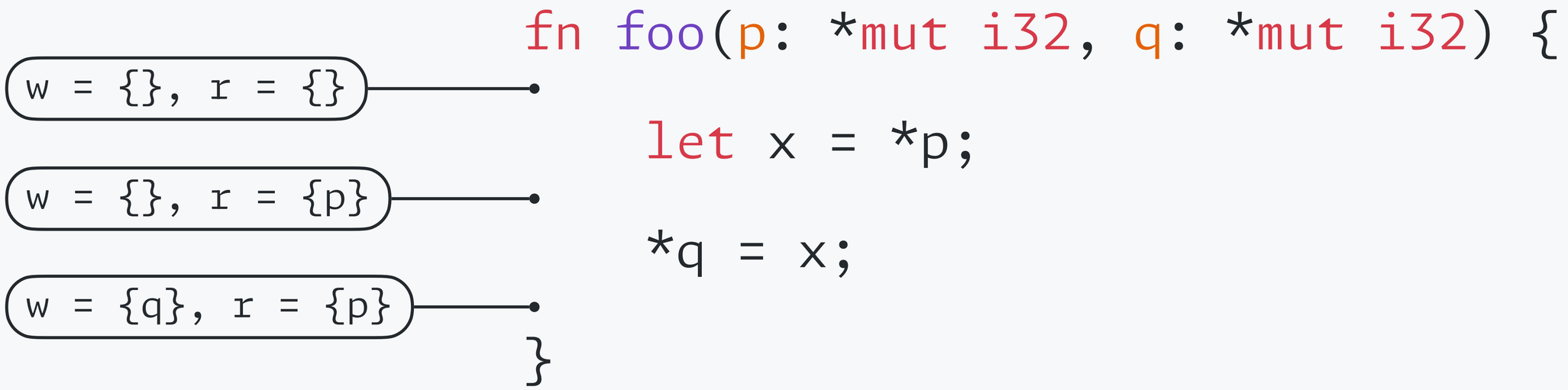
정적 분석_읽기 집합과 쓰기 집합

매개변수	$x \in X$ =	모든 포인터 타입 매개변수의 집합
읽기 집합	$r \in R$ =	$(P(X), \subseteq)$
쓰기 집합	$w \in W$ =	$(P(X), \supseteq)$

$x \in r$ 이면, 아마도 x에 쓰기 전에 x를 읽었다.

$x \in w$ 이면, 반드시 x를 읽기 전에 x에 썼다.

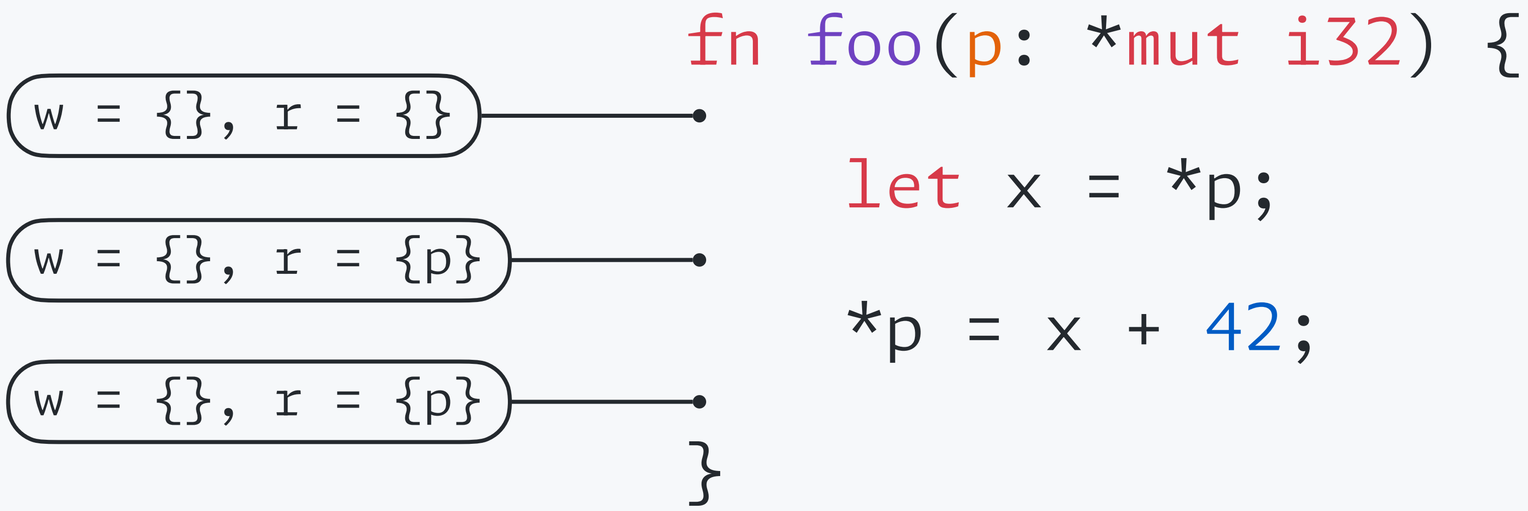
정적 분석_읽기 집합과 쓰기 집합



x ∈ r이면, 아마도 x에 쓰기 전에 x를 읽었다.

x ∈ w이면, 반드시 x를 읽기 전에 x에 썼다.

정적 분석_읽기 집합과 쓰기 집합

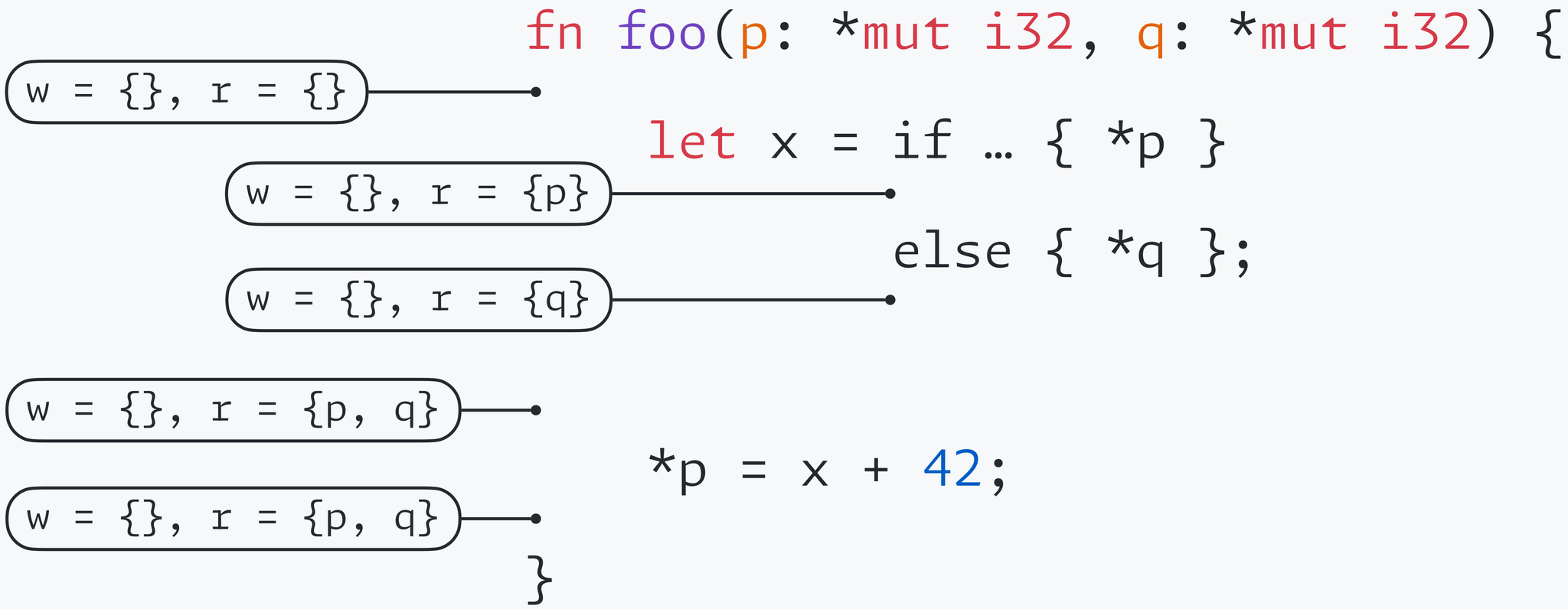



```
fn foo(p: *mut i32) {
    let x = *p;
    *p = x + 42;
}
```

x ∈ r 이면, 아마도 x에 쓰기 전에 x를 읽었다.

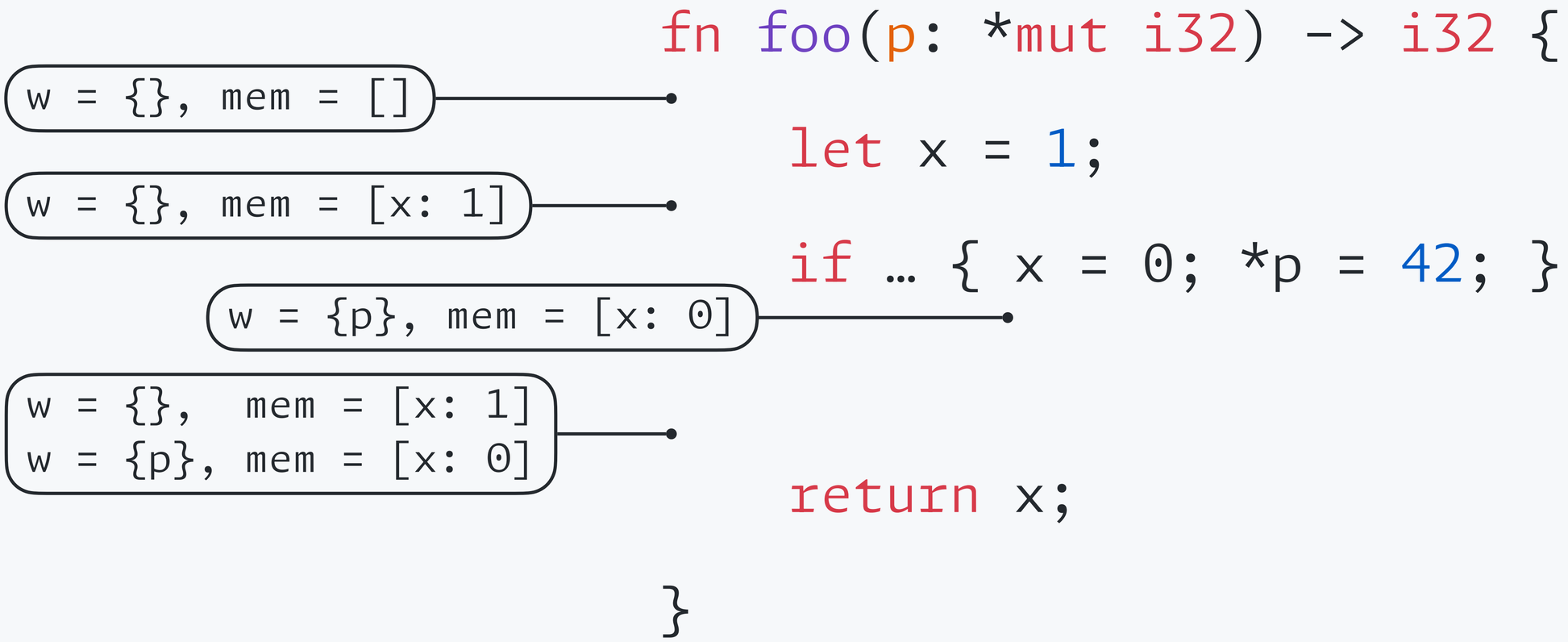
x ∈ w 이면, 반드시 x를 읽기 전에 x에 썼다.

정적 분석_실행 흐름이 합쳐질 때



실행 흐름이 합쳐질 때, 읽기 집합을 합집합함

정적 분석_실행 흐름이 합쳐질 때



실행 흐름이 합쳐질 때, 쓰기 집합이 다르면 합치지 않고 정보를 분리해서 유지함

정적 분석_결과 해석

Must 출력 매개변수:
모든 쓰기 집합에 포함

```
w = {p}
```

May 출력 매개변수:
일부 쓰기 집합에만 포함

```
w = {},  mem = [return value: 1]
w = {p}, mem = [return value: 0]
```

코드 변환_Must 출력 매개변수

```rust
fn foo(p: *mut i32) -> i32 {


    *p = 42;
    return 37;
}

let y = foo(x);
```

변환 전

```rust
fn foo() -> (i32, i32) {
    let pv = 0;
    let p = &mut pv;
    *p = 42;
    return (37, pv);
}

let (y, v) = foo();
*x = v;
```

변환 후

코드 변환_May 출력 매개변수

```
fn foo(p: *mut i32) -> i32 {




    let x = 1;
    if … { x = 0; *p = 42; }

    return x;

}

if foo(x) == 0 { bar(*x); }
else { baz(); }
```

변환 전

```
fn foo() -> Option<i32> {
    let pv = 0;
    let p = &mut pv;
    let pw = false;
    let x = 1;
    if … { x = 0; *p = 42;
           pw = true; }
    return if pw { Some(pv) }
           else { None };
}
match foo() {
    Some(v) => { *x = v; bar(*x); }
    None => { baz(); }
}
```

변환 후

평가_유용성 및 올바름

55개의 C 프로그램(기존 벤치마크 20개, GNU 패키지 35개)을 사용하여 평가

효율성

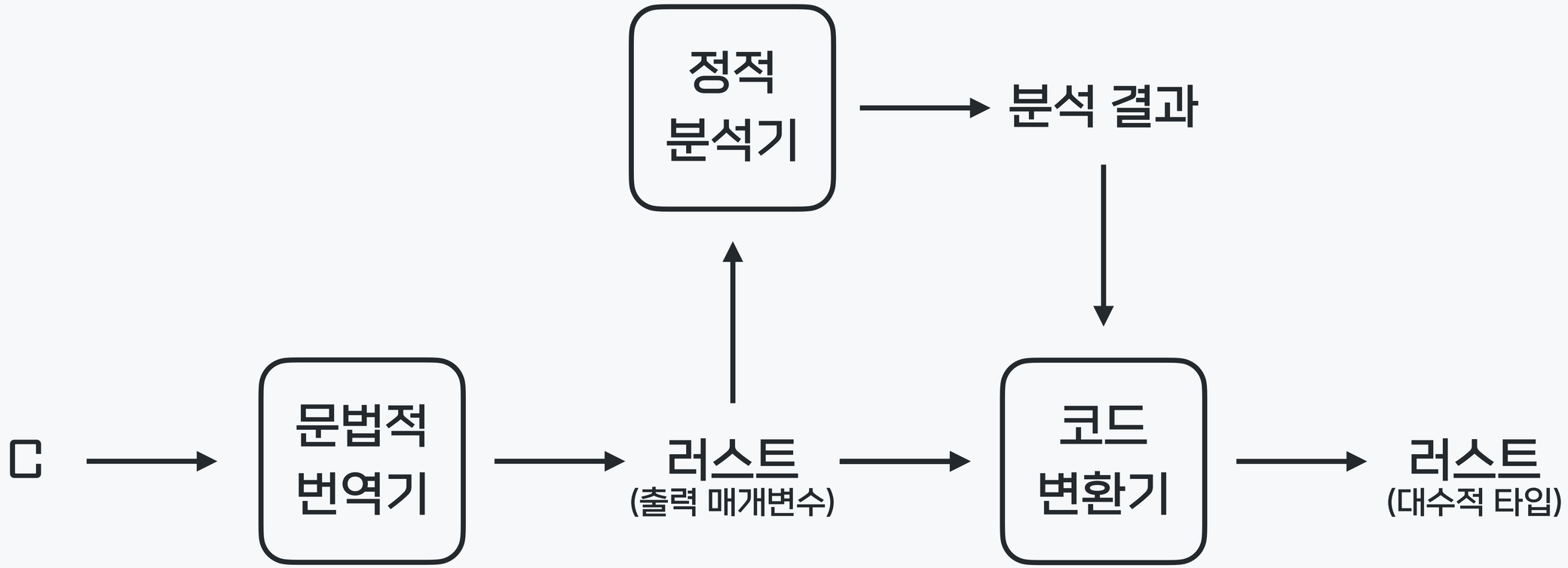
90,000줄의 코드를 213초 안에 분석 및 변환 가능

유용성

프로그램당 평균 24.4개의 함수에서 30.4개의 출력 매개변수 탐지

올바름

테스트 케이스가 있는 26개의 프로그램 중 25개가 변환 후에도 테스트 모두 통과



Don’t Write, but Return

Replacing Output Parameters with Algebraic Data Types in C-to-Rust Translation