Conspicuous Visualizer for Compiler Optimization Fuzzers

2024.04.09.

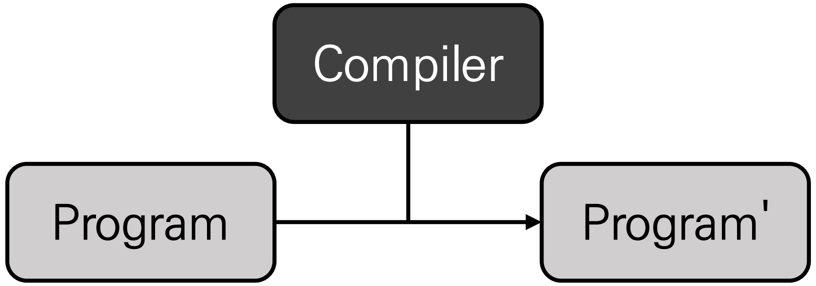
Haejoon Park

Compiler Optimization

- Compilers are hidden heroes of software

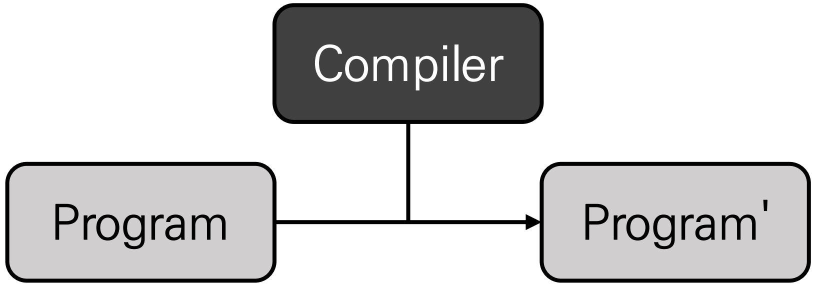
Compiler Optimization

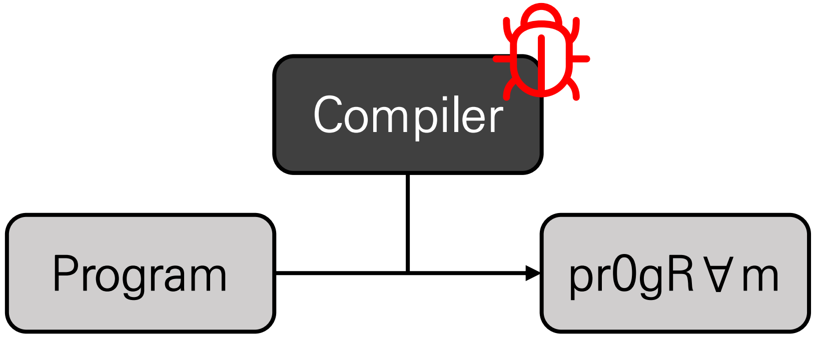
- Compilers are hidden heroes of software



Compiler Optimization

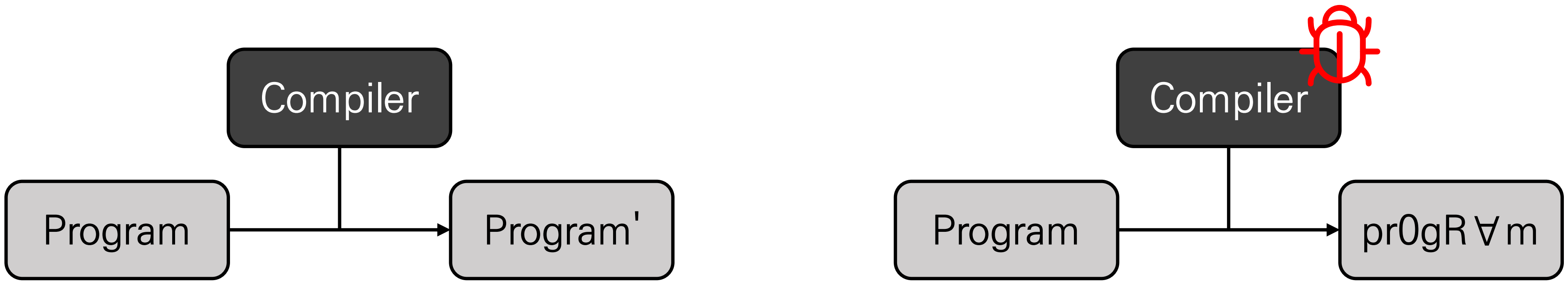
- Compilers are hidden heroes of software
- What if there’s bug?





Compiler Optimization

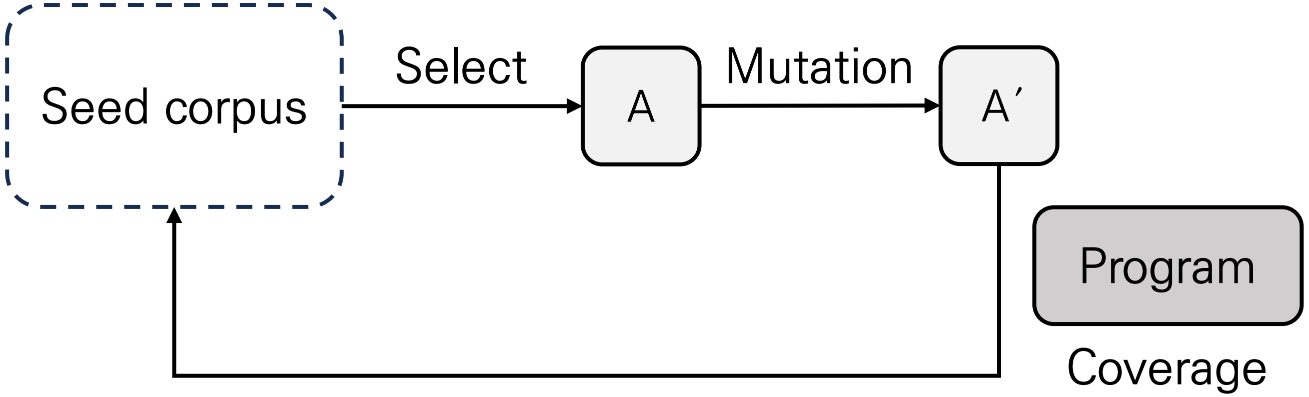
- Compilers are hidden heroes of software
- What if there’s bug?



- ‘Miscompilation’ of optimizing compilers

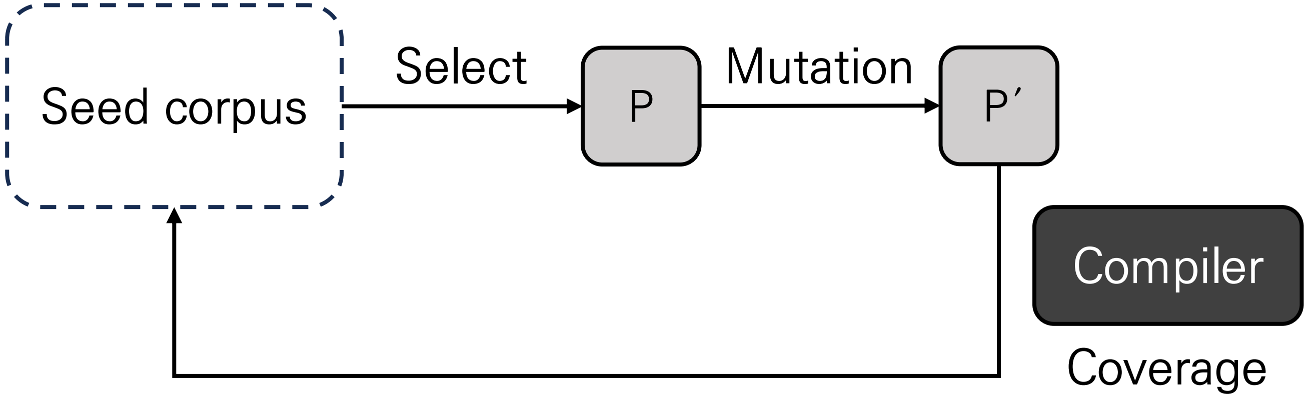
Ensuring Correctness Is IMPORTANT!

- Fuzzing



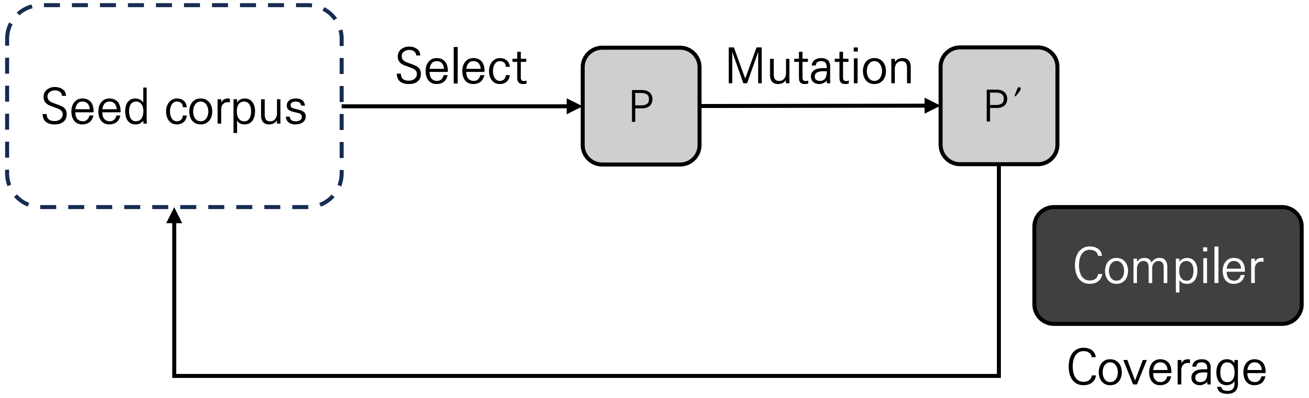
Ensuring Correctness Is IMPORTANT!

- Compiler Fuzzing



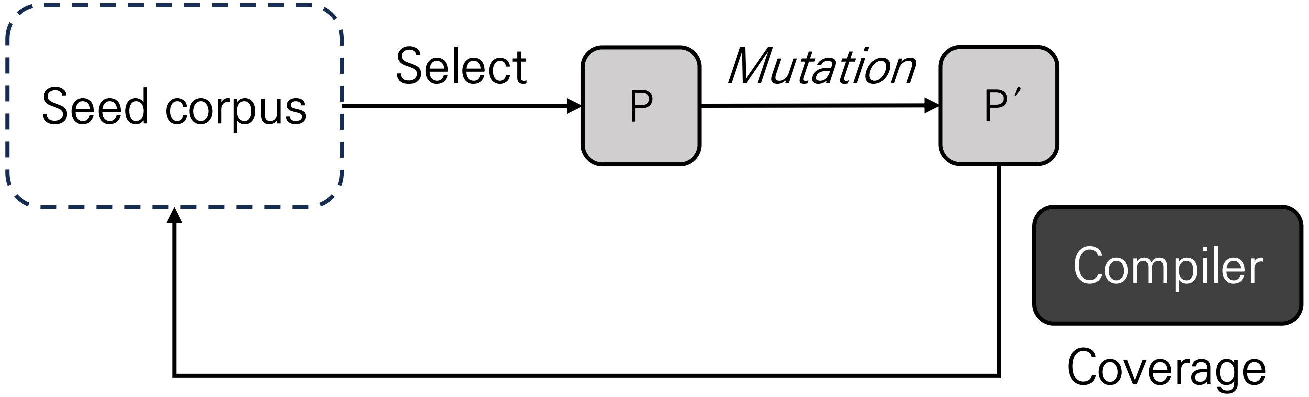
Problems: High-Level Features

- Imagine you are a developer of…
  - the fuzzer
  - the compiler



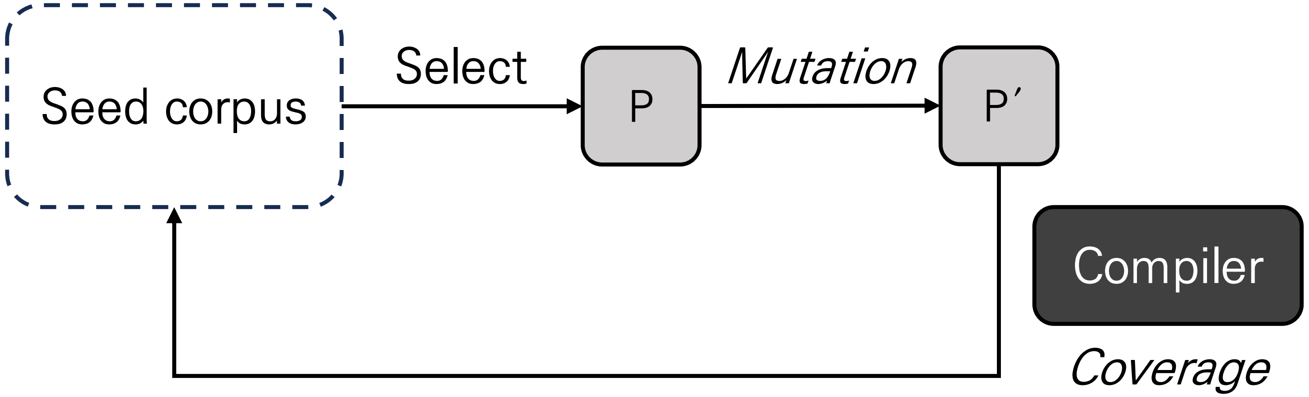
Problems: High-Level Features

- How can mutations occur?



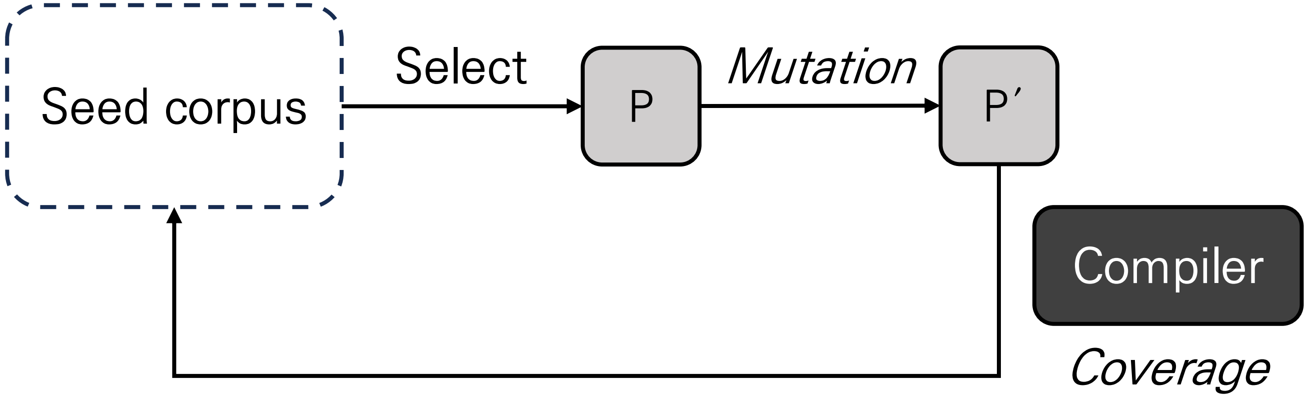
Problems: High-Level Features

- How can mutations occur?
- How can each mutation affect coverage?



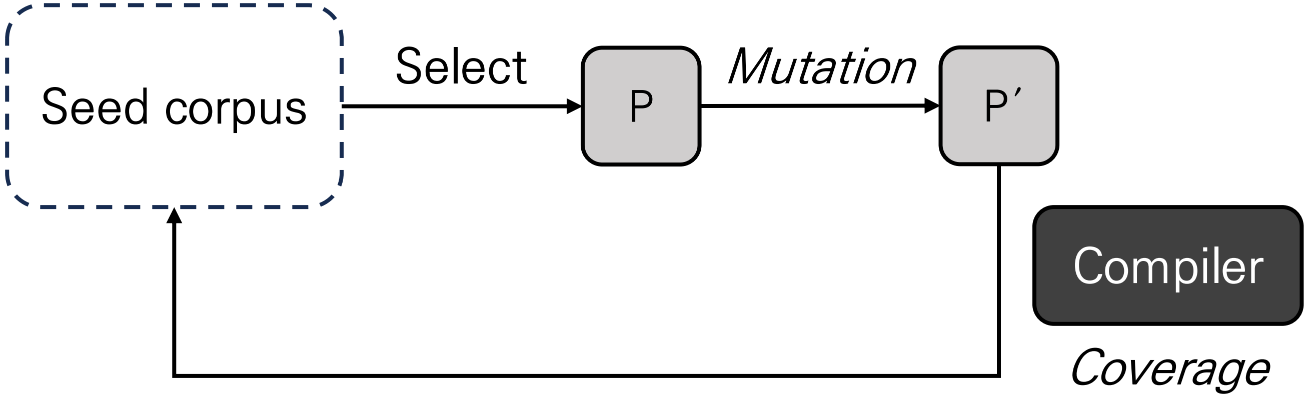
Problems: High-Level Features

- How can mutations occur?
- How can each mutation affect coverage?
- How much coverage has been covered so far?



Problems: High-Level Features

- How can mutations occur?
- How can each mutation affect coverage?
- How much coverage has been covered so far?



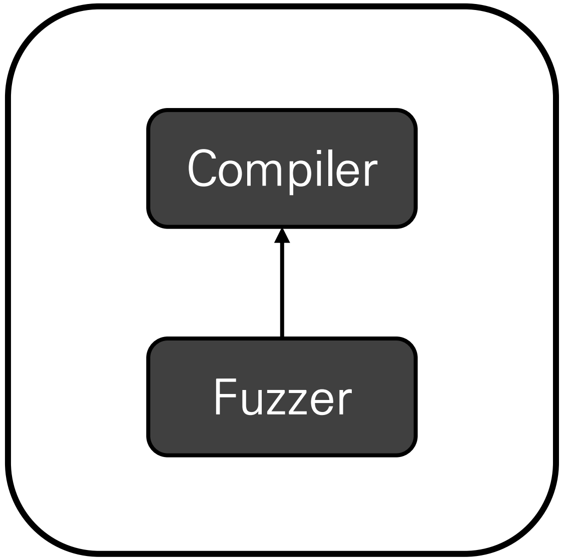
- We need fascinating helpers!

Goal: Coverage Visualization

- Coverage visualizer for optimizing compilers and their fuzzers

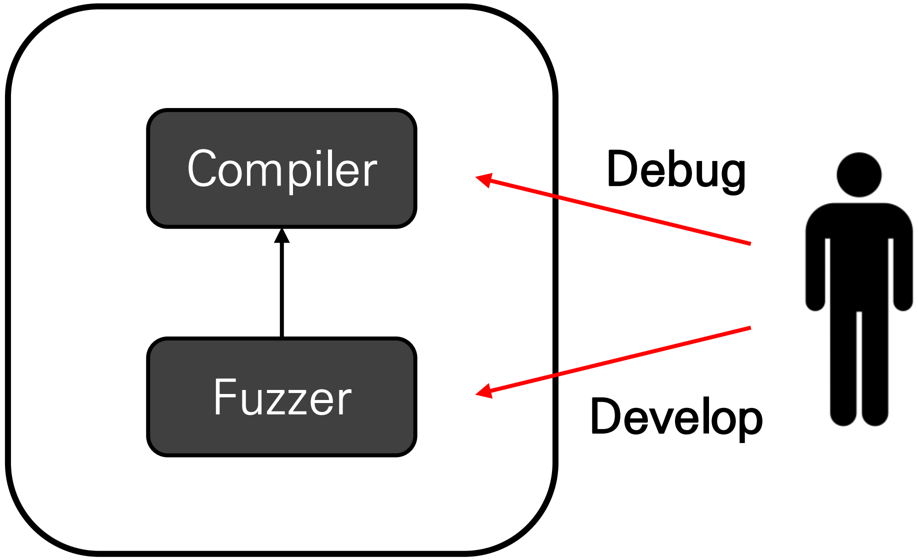
Goal: Coverage Visualization

- Coverage visualizer for optimizing compilers and their fuzzers



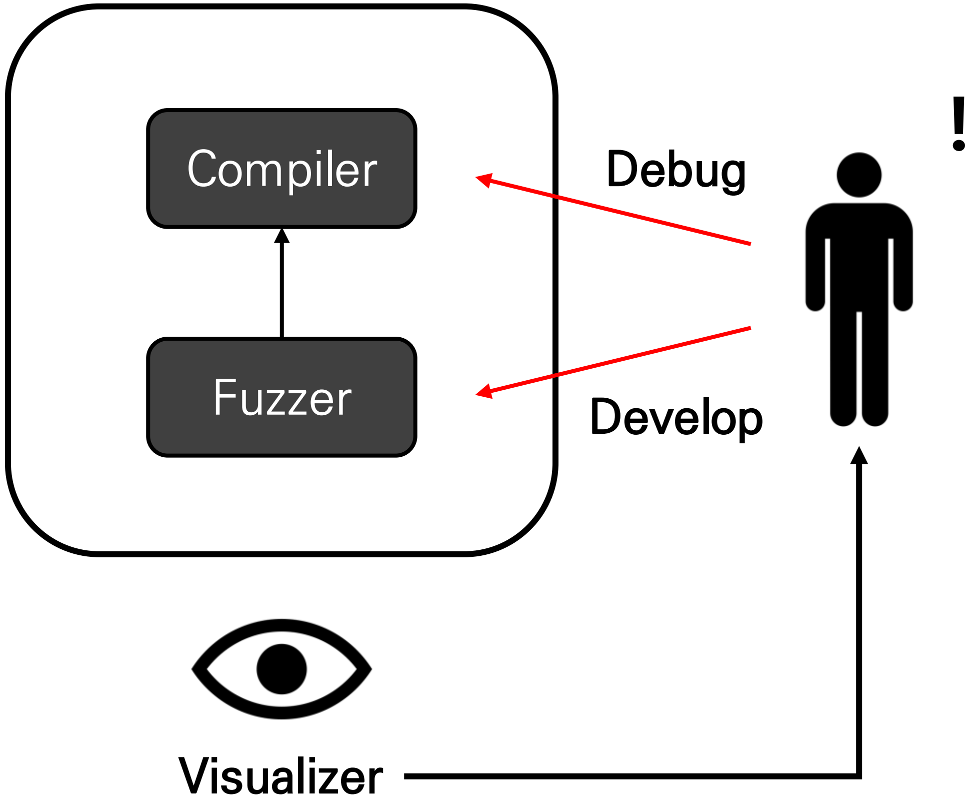
Goal: Coverage Visualization

- Coverage visualizer for optimizing compilers and their fuzzers



Goal: Coverage Visualization

- Coverage visualizer for optimizing compilers and their fuzzers



Goal: Coverage Visualization

- Bridge the gap between the fuzzing process and human comprehension
  - Diagnose bugs effectively
  - Develop more fuzzing strategies
  - Accelerate compiler optimization fuzzing research

Sample Compiler & Fuzzer

- LLVM IR
- Supports simple mutation operators

```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, 1
  return i32 %z
}
```

- **Ideas are universal**;
  can apply to other fuzzers for optimizing compilers

Visualization of Compiler Fuzzer

- Four visualization strategy for optimizing compiler fuzzers

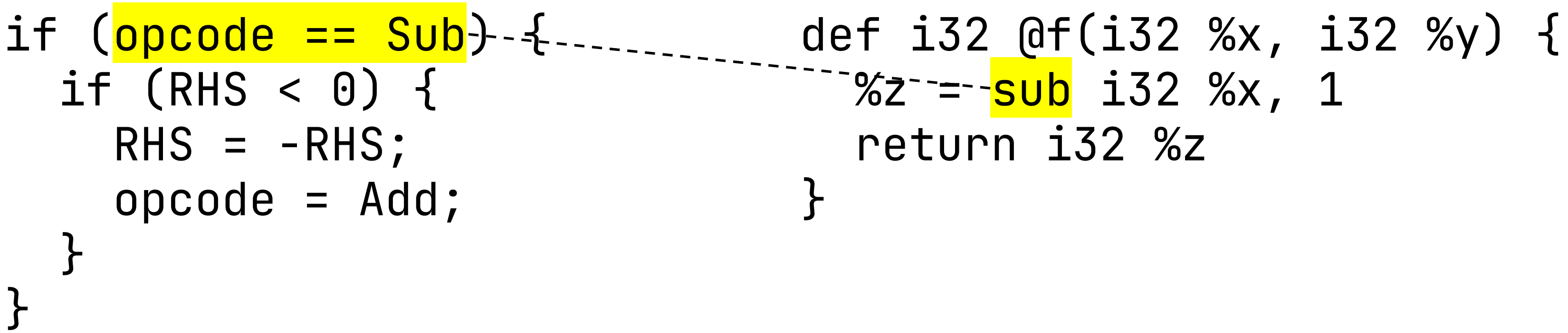
```
if (opcode == Sub) {
  if (RHS < 0) {
    RHS = -RHS;
    opcode = Add;
  }
}
```

```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, 1
  return i32 %z
}
```

  - Step-by-step
  - Interactive (mouse-over, cursor)

Visualization of Compiler Fuzzer

- ⓐ General coverage visualization



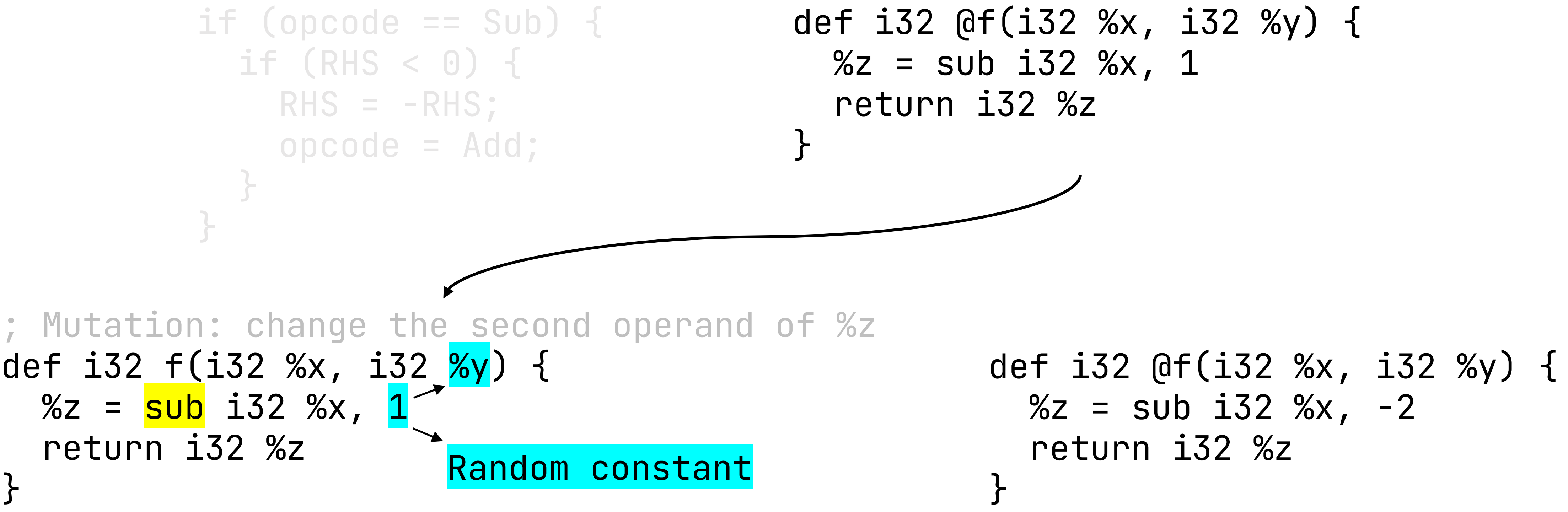
```
if (opcode == Sub) {
  if (RHS < 0) {
    RHS = -RHS;
    opcode = Add;
  }
}
```

```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, 1
  return i32 %z
}
```

  - Common and generic feature

Visualization of Compiler Fuzzer

- ⓑ How each mutation occurs?



```
if (opcode == Sub) {
  if (RHS < 0) {
    RHS = -RHS;
    opcode = Add;
  }
}
```

```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, 1
  return i32 %z
}
```

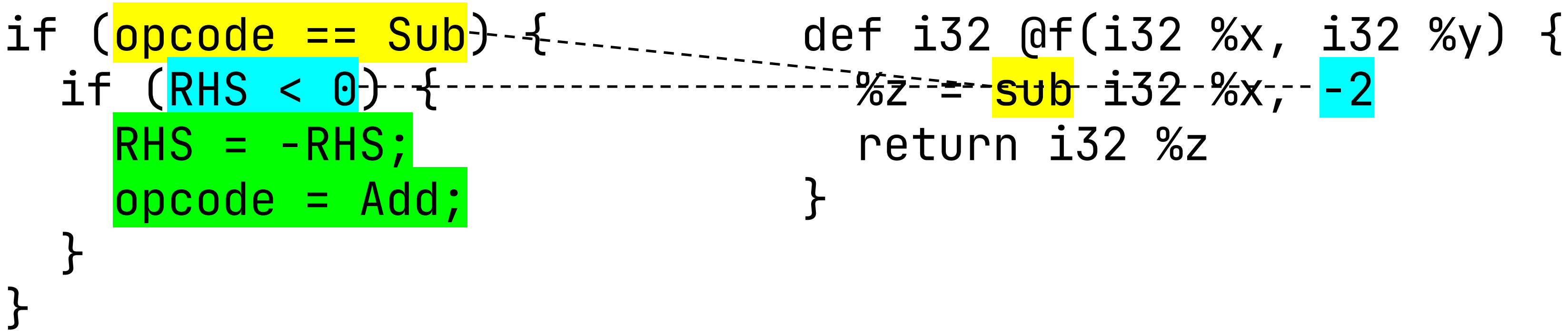
```
; Mutation: change the second operand of %z
def i32 f(i32 %x, i32 %y) {
  %z = sub i32 %x, 1
  return i32 %z
}
```

Random constant

```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, -2
  return i32 %z
}
```

Visualization of Compiler Fuzzer

- ⓒ How did the mutation increase the coverage?



```
if (opcode == Sub) {
  if (RHS < 0) {
    RHS = -RHS;
    opcode = Add;
  }
}
```

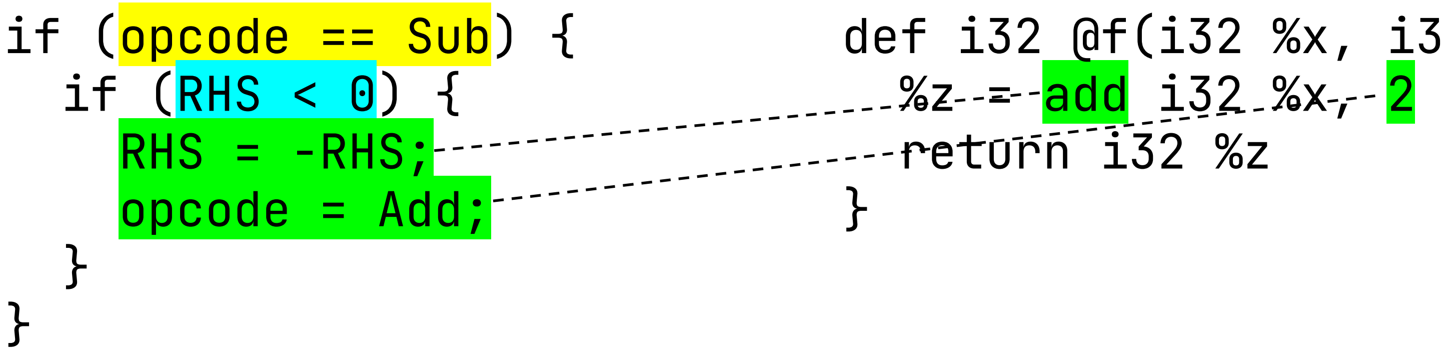
```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, -2
  return i32 %z
}
```

각각이 무슨
효과가 있는지도
PPT에 추가하기

- Some mutations lead to meaningful coverage increases
- Capturing the ‘change’

Visualization of Compiler Fuzzer

- ⓓ How did the optimization changed the input program?



```
if (opcode == Sub) {
  if (RHS < 0) {
    RHS = -RHS;
    opcode = Add;
  }
}
```

```
def i32 @f(i32 %x, i32 %y) {
  %z = add i32 %x, 2
  return i32 %z
}
```

  - Can be used to match intention to actual result

Related Works

- `gcov` (coverage measurement)
    - Tool in GCC to test code coverage in programs

Related Works

- `gcov` (coverage measurement)
  - Tool in GCC to test code coverage in programs

```
if (a != b)
  c = 1;
else
  c = 0;
```

```
100:   12:if (a != b)
100:   13:  c = 1;
100:   14:else
100:   15:  c = 0;
```

Related Works

- **gcov** (coverage measurement)
  - Tool in GCC to test code coverage in programs

```
if (opcode == Sub) {
  if (RHS < 0) {
    RHS = -RHS;
    opcode = Add;
  }
}
```

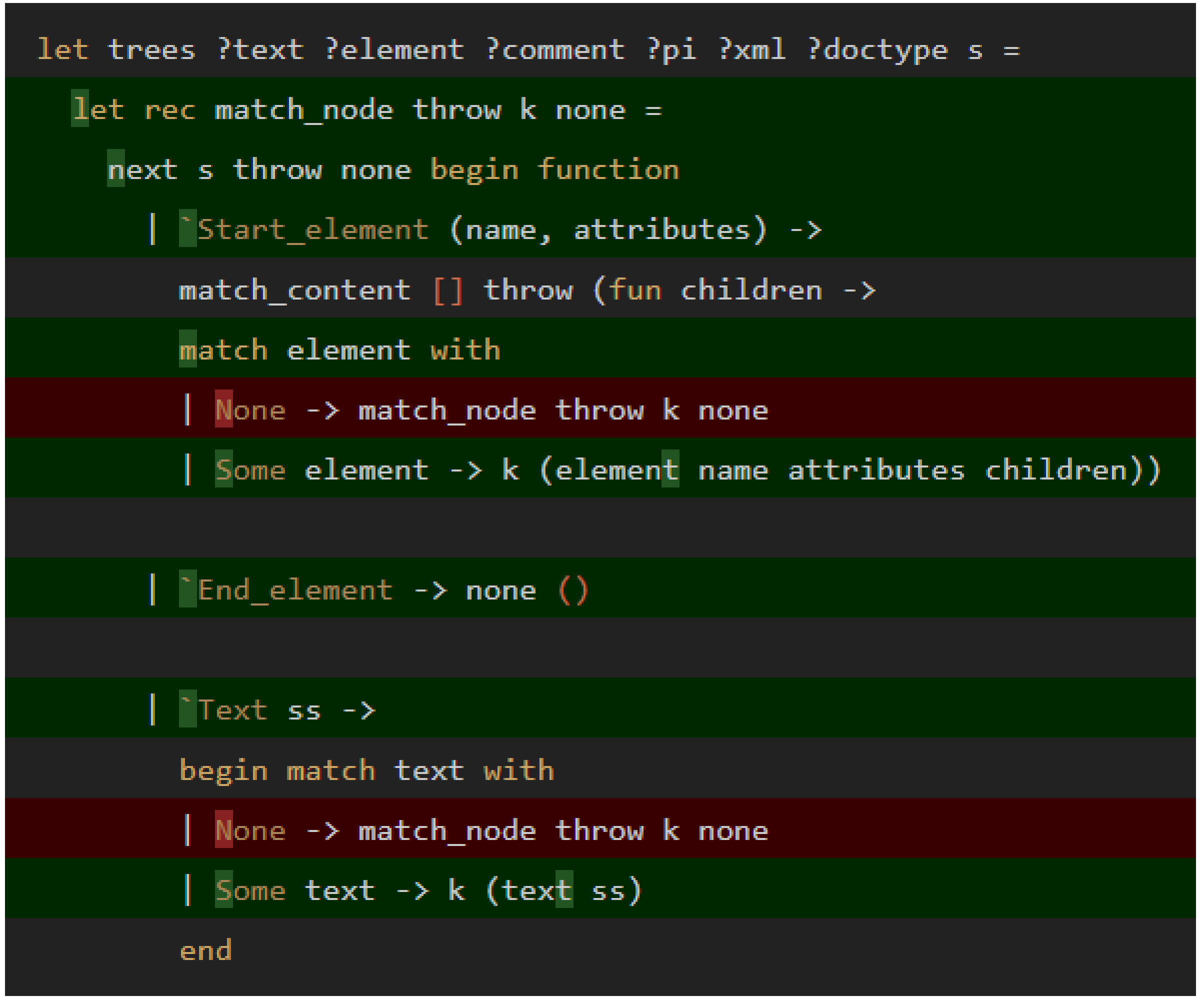
```
def i32 @f(i32 %x, i32 %y) {
  %z = sub i32 %x, 1
  return i32 %z
}
```

  - Implementation: parse **gcov** coverage output

Related Works

- Coverage visualization
  - Many examples historically

- To our project…
  - Show compiler coverage
  - Apply to highlighting tokens in input programs



```
let trees ?text ?element ?comment ?pi ?xml ?doctype s =
  let rec match_node throw k none =
    next s throw none begin function
      | `Start_element (name, attributes) ->
        match_content [] throw (fun children ->
        match element with
        | None -> match_node throw k none
        | Some element -> k (element name attributes children))

      | `End_element -> none ()

      | `Text ss ->
        begin match text with
        | None -> match_node throw k none
        | Some text -> k (text ss)
        end
```

https://github.com/aantron/bisect_ppx

Evaluation

- How can the efficacy of the visualizer be evaluated?

Evaluation

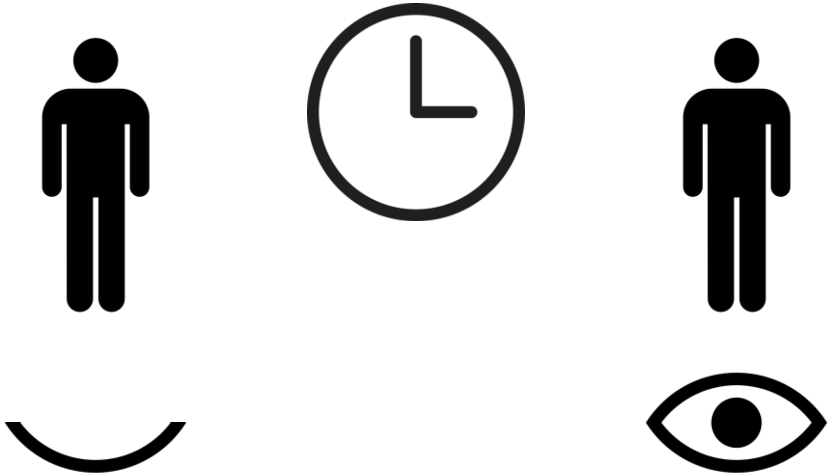
- How can the efficacy of the visualizer be evaluated?
- Solution 1: time measurement

Evaluation

- How can the efficacy of the visualizer be evaluated?
- Solution 1: time measurement

Evaluation

- How can the efficacy of the visualizer be evaluated?
- Solution 1: time measurement



Evaluation

- How can the efficacy of the visualizer be evaluated?
- Solution 2: survey

Statement	Very satisfied	Satisfied	Neutral	Dissatisfied
Overall satisfaction		O		
Visualizing Coverage		O		
Ease of Use	O			
Design and Readability			O	
...				

CV

- KAIST School of Computing Bachelor’s degree
- KAIST School of Computing Master student
  - Programming Systems Laboratory

- GitHub Repository
  - https://github.com/p-has-done

Summary

- It is important to check whether compilers are correct
  - Fuzzers can help it
- Visualizer for optimizing compiler fuzzers
  - Will bridge the gap between the fuzzing process and human comprehension

- Four functions of visualization (with sample compiler & fuzzer)
  - Each function can aid development for fuzzer and compiler
- Evalutaion: time & survey