Visualize Optimization-directed Fuzzing for Effective Optimization Testing

Jaeseong Kwon

IS661

Compiler’s Optimization

- Compiler optimizes the program to have lower cost
  - The optimized program must have the same semantics as the original

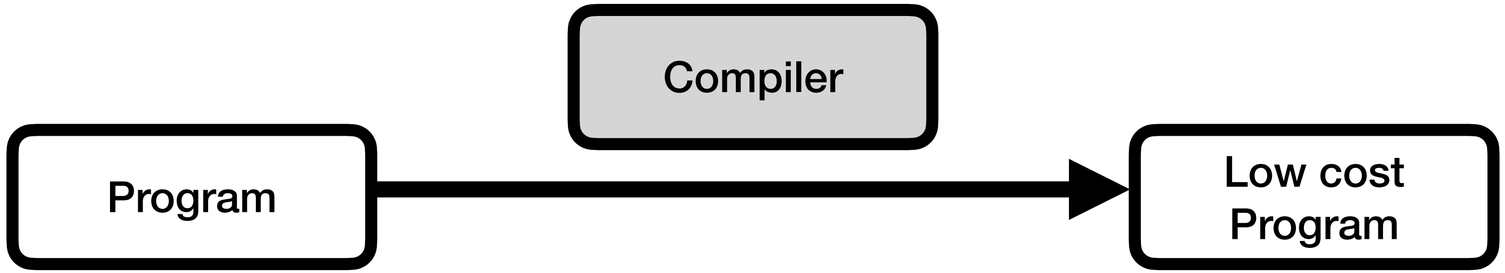
```
int foo1 (int a) {
  int temp = a + 1;
  return temp + 2;
}
```

```
int foo1 (int a) {
  return a + 3;
}
```

```
int foo2 (int a) {
  int temp = a * 4;
  return temp;
}
```

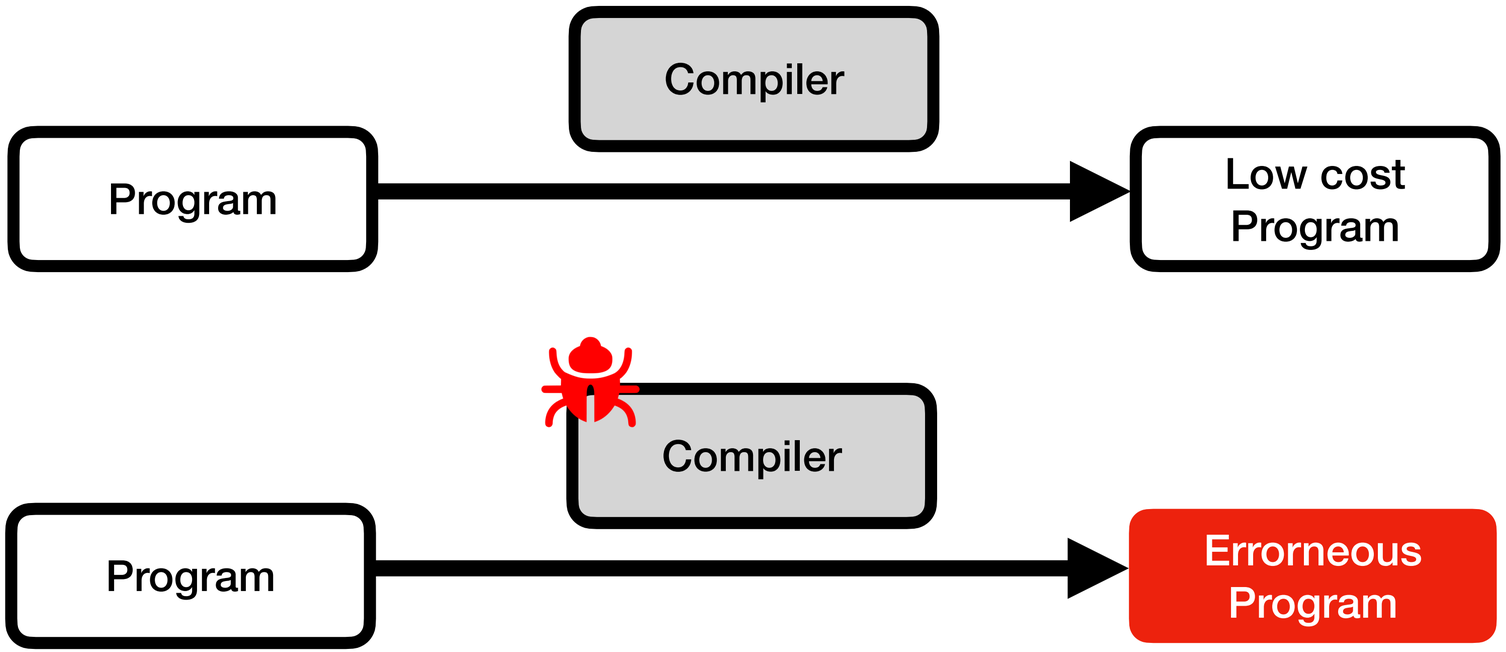
```
int foo2 (int a) {
  return a << 2;
}
```

Compiler’s Optimization Bugs



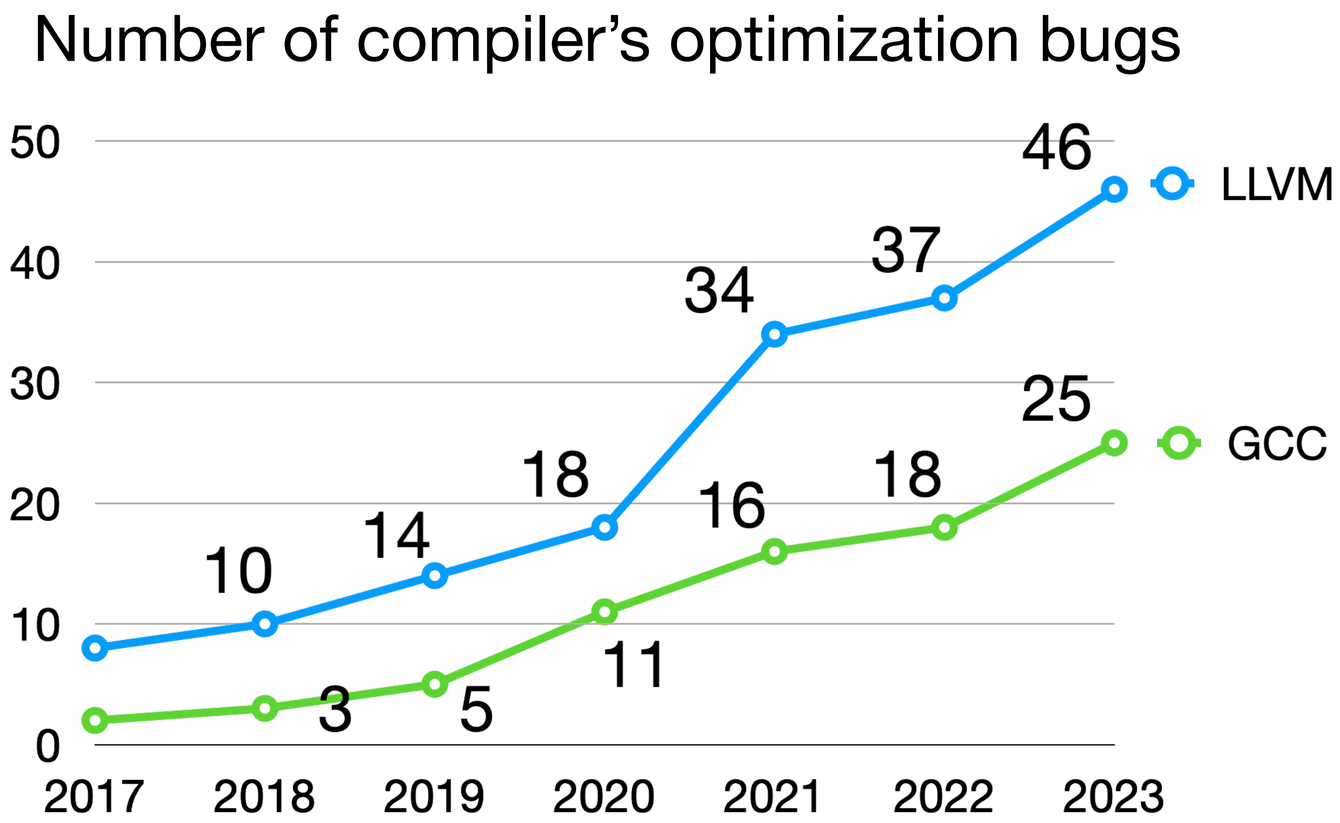
Compiler’s Optimization Bugs

- Incorrect optimization can produce erroneous programs



Compiler’s Optimization Bugs

- Compiler optimization bugs are increasing



Challenge: Compiler Optimization's Complexity

- Highly complex logic in Compiler's Optimization
  - Hard to verify optimization logic is correct
  - e.g Optimize $(X \,\&\, Y) == C \;?\; X \,|\, Y \;:\; (X \oplus Y) \,|\, C$ into $(X \oplus Y) \,|\, C : X \oplus Y$

Challenge: Compiler Optimization's Complexity

- Highly complex logic in Compiler's Optimization
  - Hard to verify optimization logic is correct
  - e.g Optimize $(X \,\&\, Y) == C \;?\; X \,|\, Y \;:\; (X \oplus Y) \,|\, C$ into $(X \oplus Y) \,|\, C : X \oplus Y$
- large codebase
  - 175K LoC in one module of LLVM's optimization

Our Solution: Directed Fuzzing

- Testing a program with randomly generated input
- Aims to reach the given target location (In this case, compiler optimization)

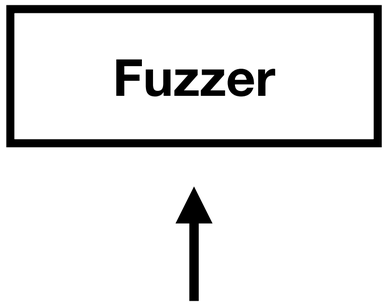
Our Solution: Directed Fuzzing

- Testing a program with randomly generated input
- Aims to reach the given target location (In this case, compiler optimization)

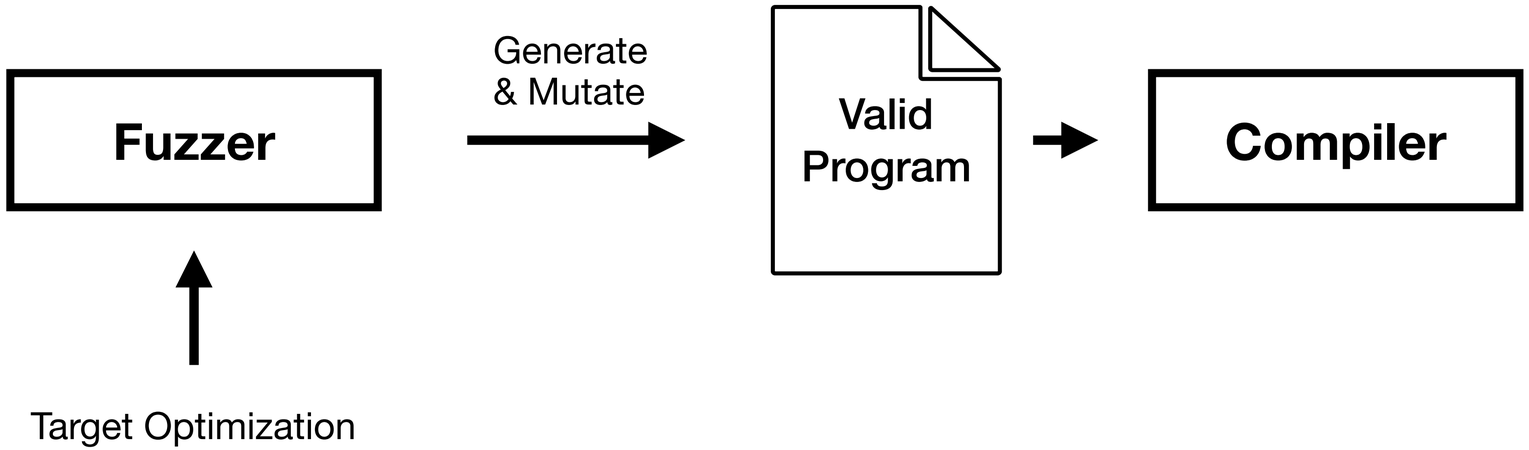
Compiler Optimization Directed Fuzzing

- Generates valid program as a compiler's input
- Appropriate guide strategy for compiler optimization

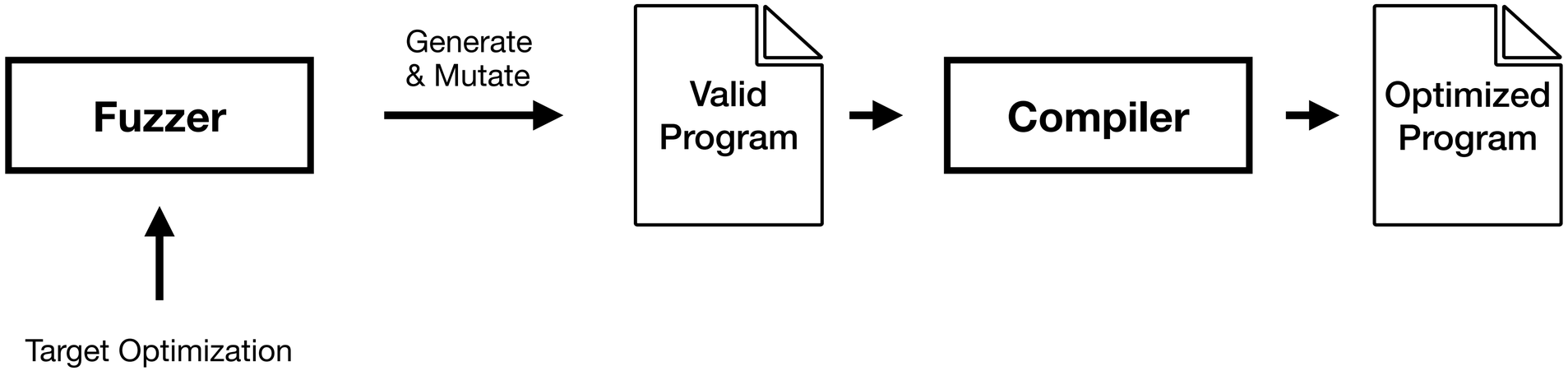
Compiler Optimization directed Fuzzer



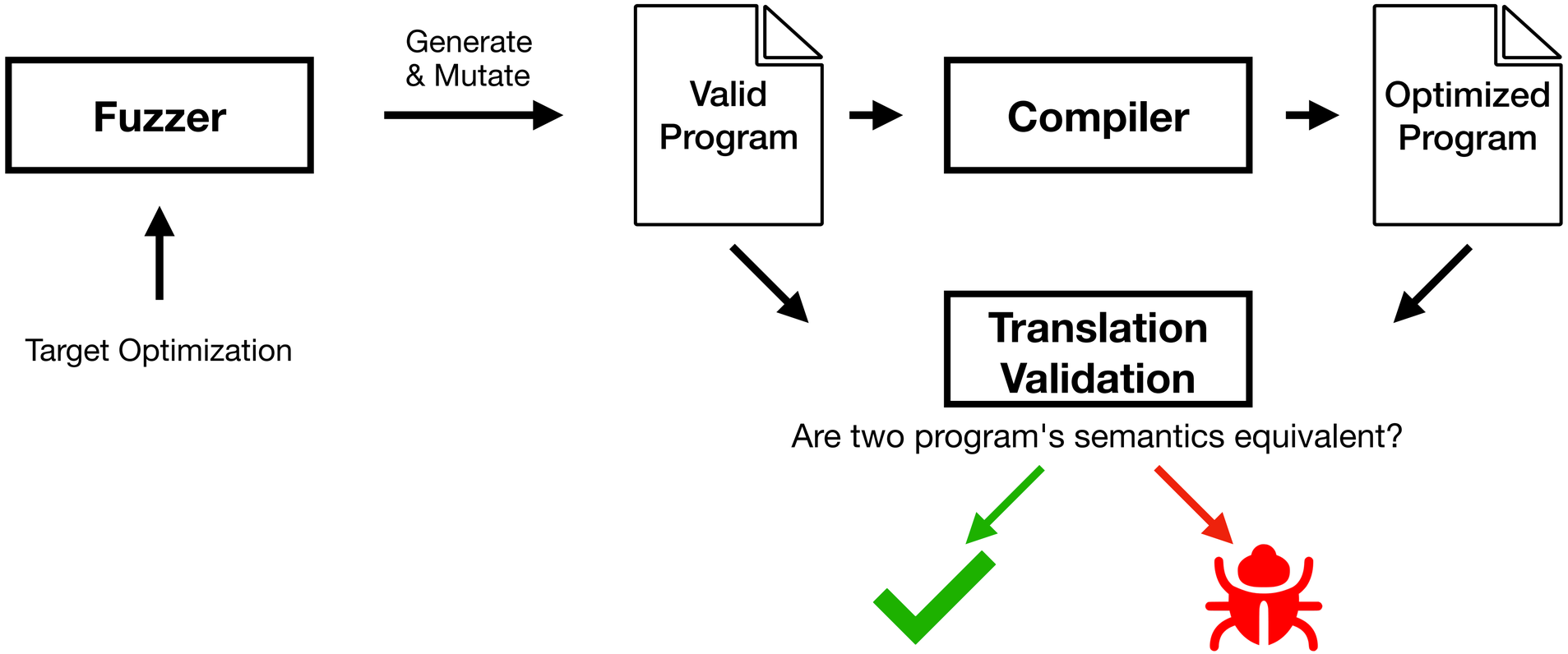
Compiler Optimization directed Fuzzer



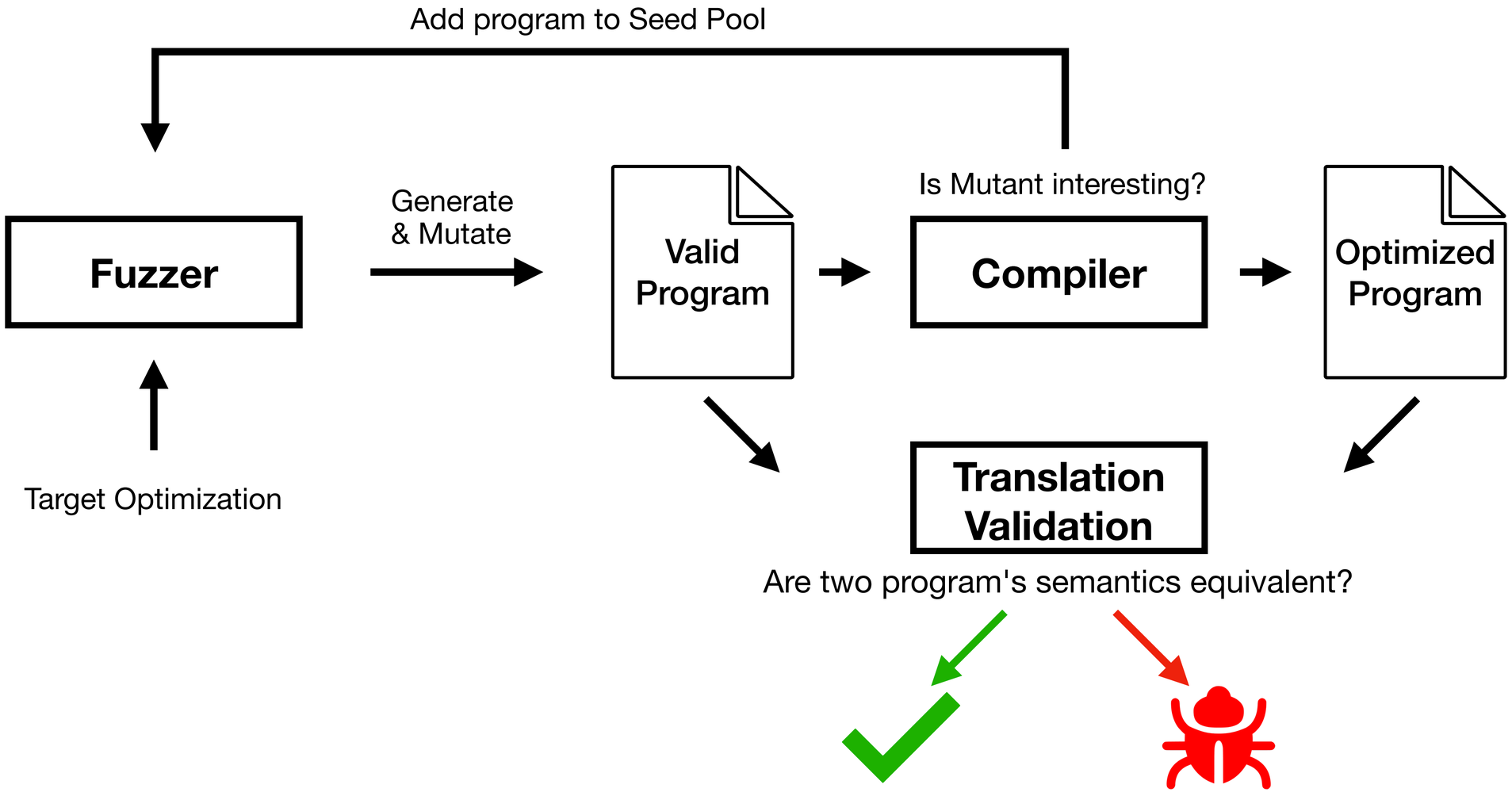
Compiler Optimization directed Fuzzer



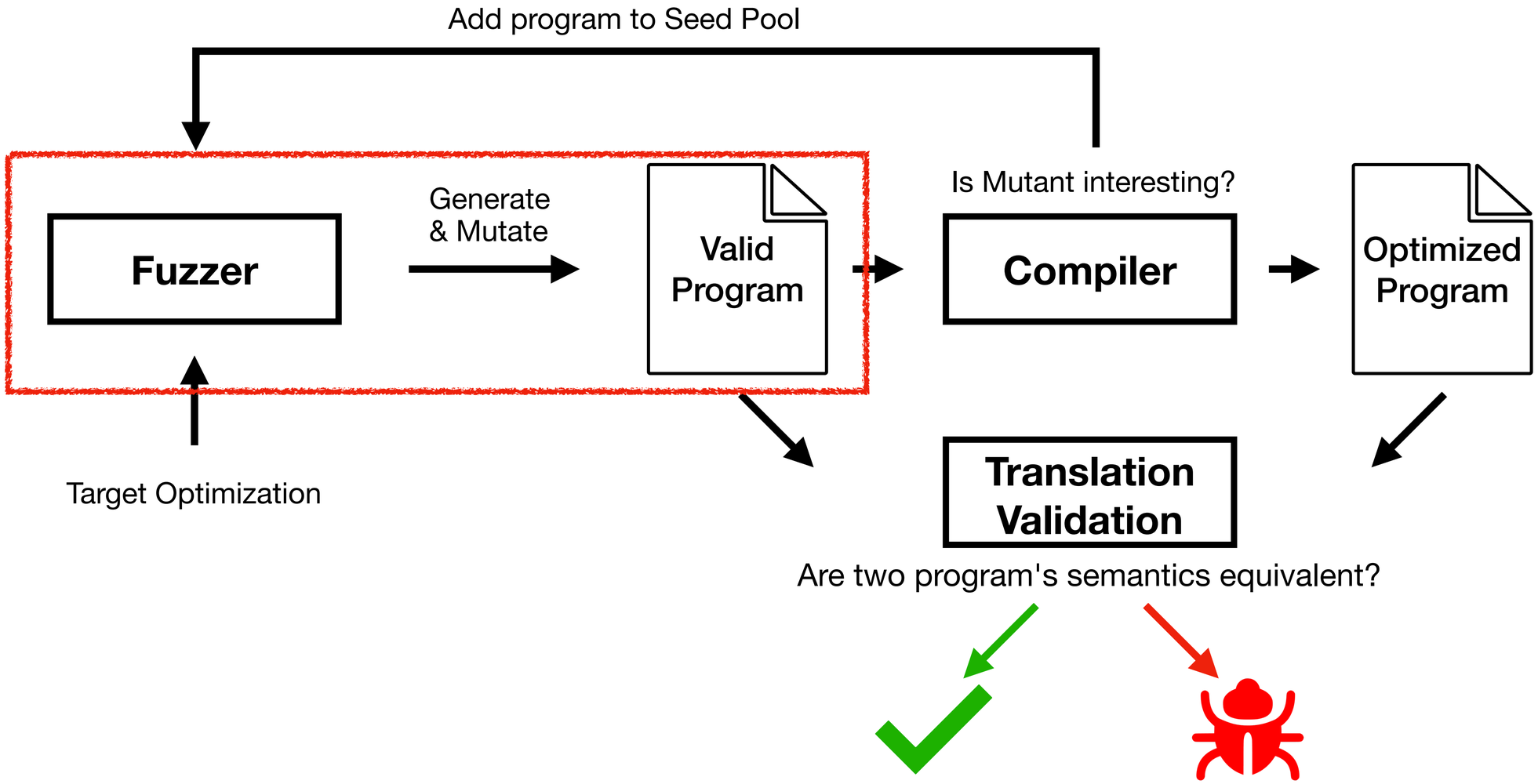
Compiler Optimization directed Fuzzer



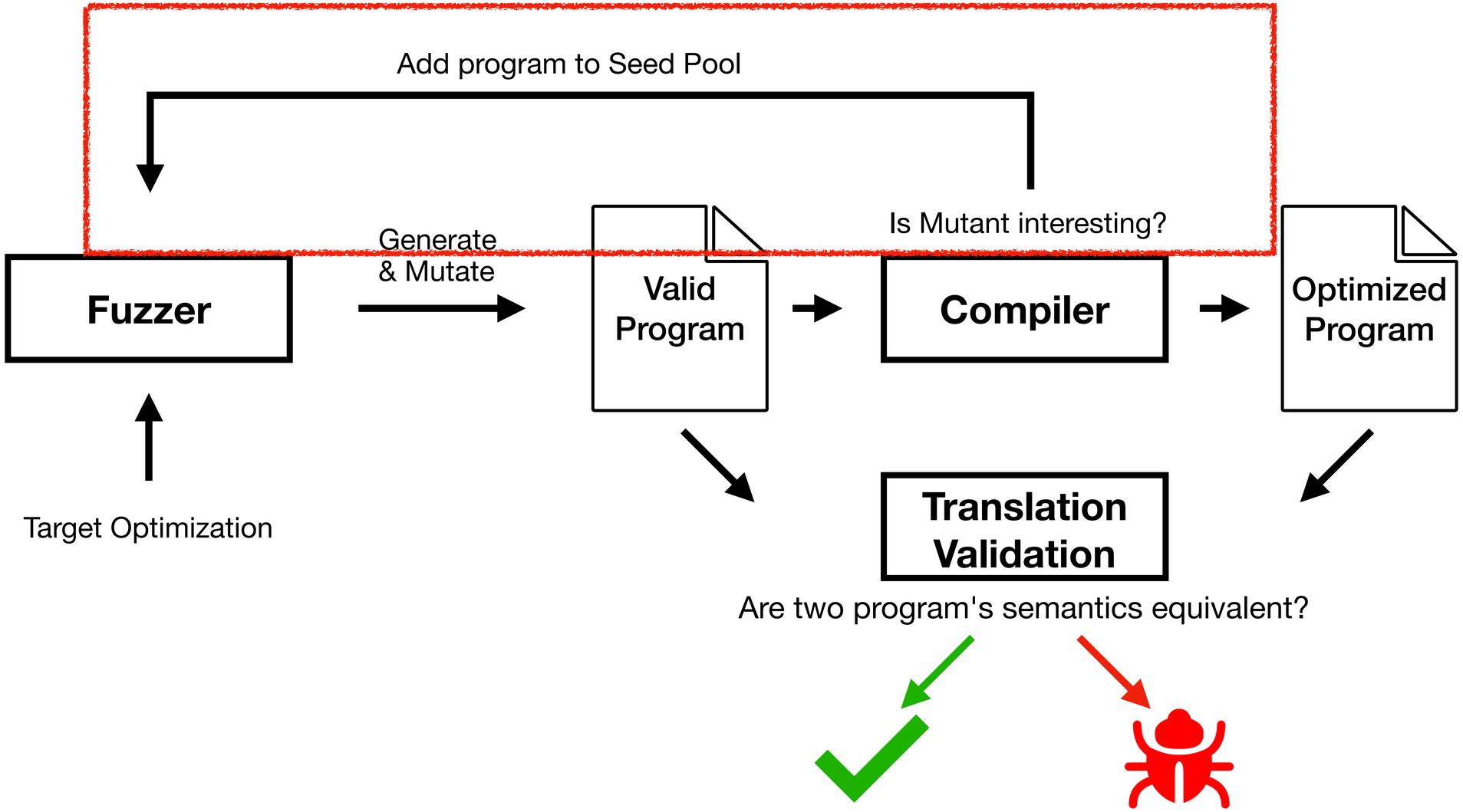
Compiler Optimization directed Fuzzer



Compiler Optimization directed Fuzzer



Compiler Optimization directed Fuzzer

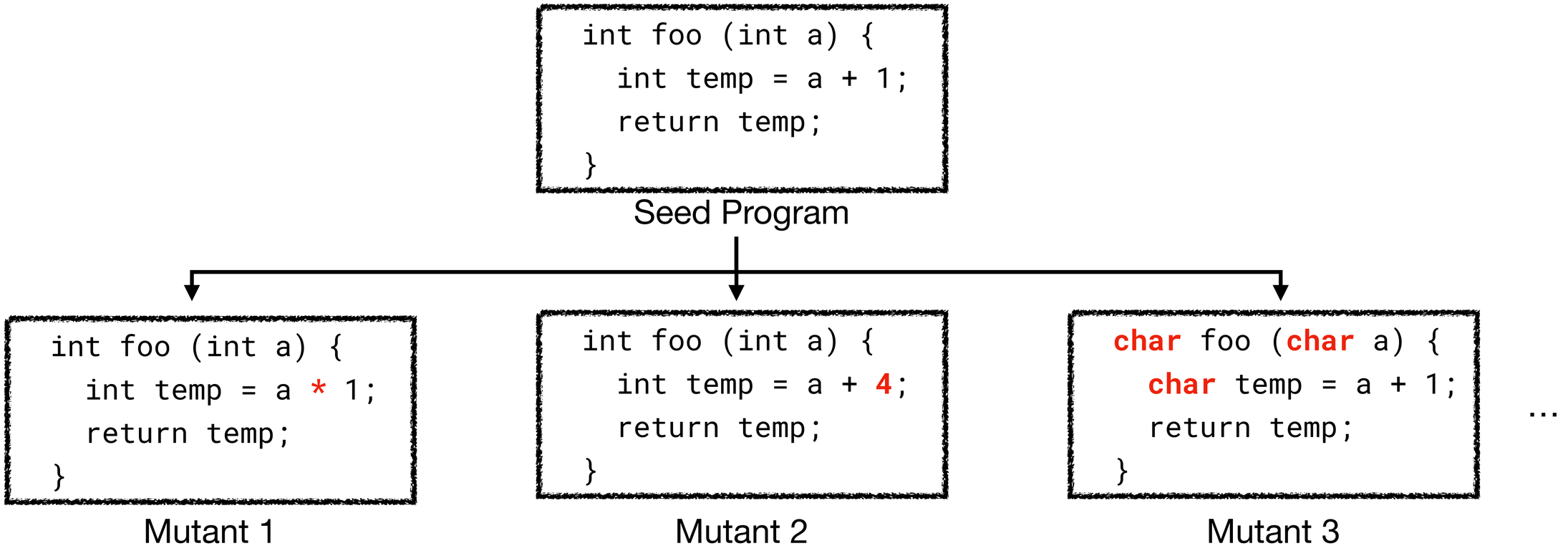


Generate Valid Program

- Mutation based input generation
  - Should make valid input (program)
  - We have several mutation strategy

Generate Valid Program

- Mutation based input generation
  - Should make valid input (program)
  - We have several mutation strategy

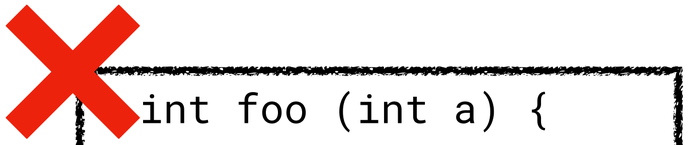


Generate Valid Pro

- Mutation not always generate valid program
  - We should handle invalid situation

```
int foo (int a) {
  int temp = a + 1;
  return temp;
}
```

Seed Program



```
int foo (int a) {
  //invalid type
  int temp = a + "1";
  return temp;
}
```

Invalid Program 1

```
int foo (int a) {
  //undefined variable
  int temp = a + b;
  return temp;
}
```

Invalid Program 2

Input Guide Strategy

- To testing target optimization, Fuzzer should generate program occurring target optimization
- The inputs generated by the fuzzer must be well-guided towards the target

Which mutant is more interesting?

How Compiler Optimization Works?

- Nested condition statement + return optimized program

How Compiler Optimization Works?

- Nested condition statement + return optimized program

Target Optimization: $X + 0 \rightarrow X$

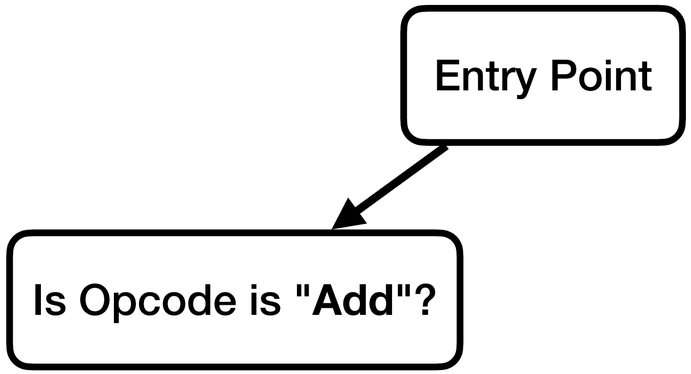
How Compiler Optimization Works?

- Nested condition statement + return optimized program

Target Optimization: $X + 0 \rightarrow X$

```
int foo (int X) {
  int temp = X + 0;
  return temp;
}
```

Input Program



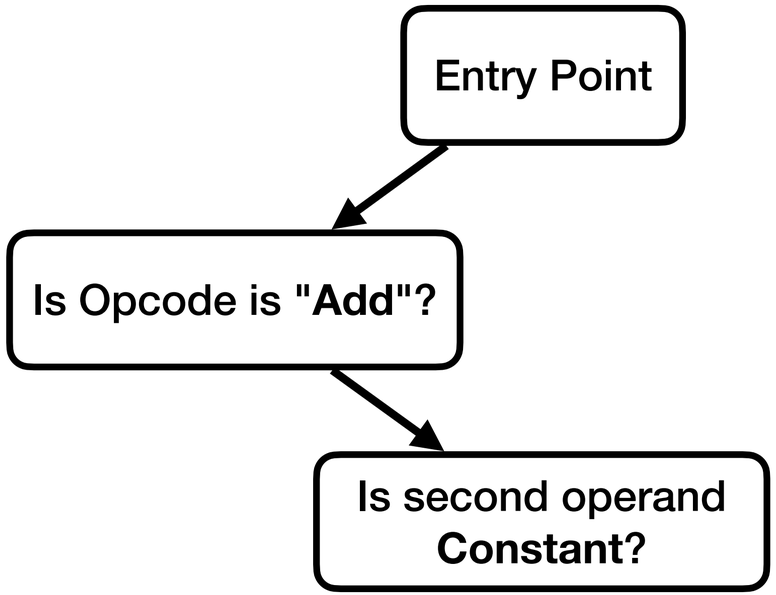
How Compiler Optimization Works?

- Nested condition statement + return optimized program

Target Optimization: $X + 0 \rightarrow X$

```
int foo (int X) {
  int temp = X + 0;
  return temp;
}
```

Input Program



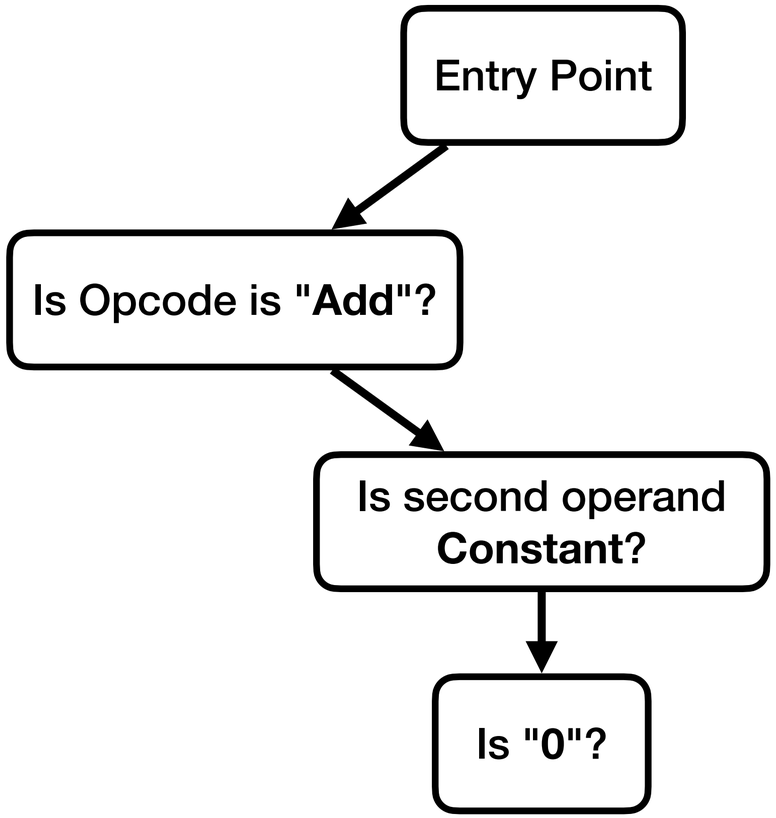
How Compiler Optimization Works?

- Nested condition statement + return optimized program

Target Optimization: $X + 0 \rightarrow X$

```
int foo (int X) {
  int temp = X + 0;
  return temp;
}
```

Input Program



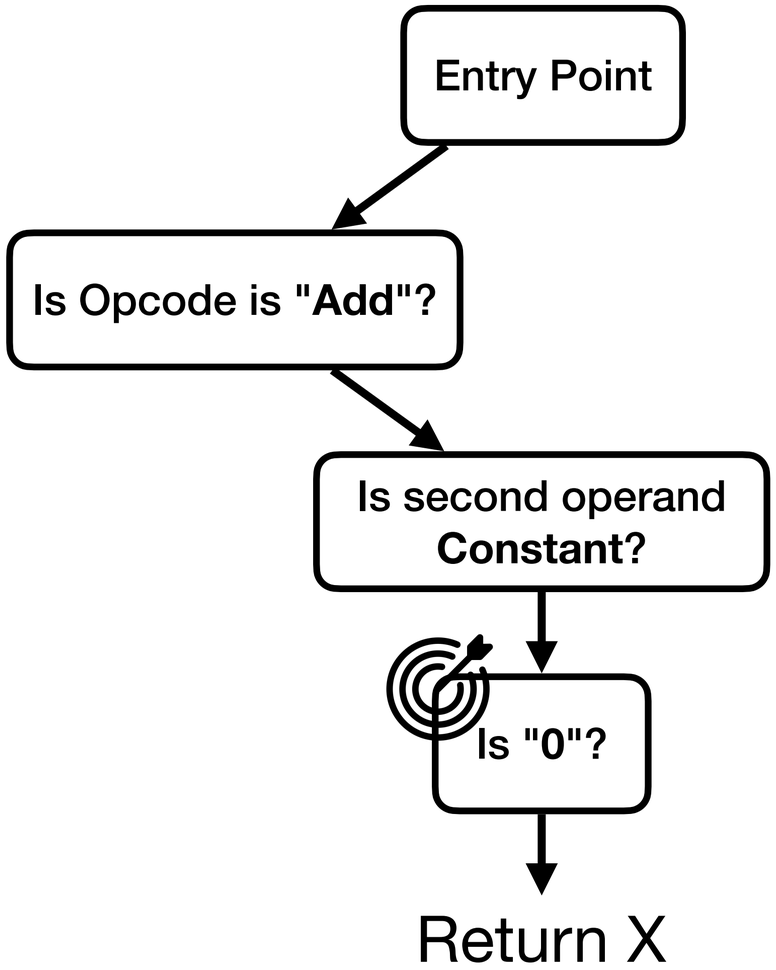
How Compiler Optimization Works?

- Nested condition statement + return optimized program

Target Optimization: $X + 0 \rightarrow X$

```
int foo (int X) {
  int temp = X + 0;
  return temp;
}
```

Input Program



How Compiler Optimization Works?

- Nested condition statement + return optimized program

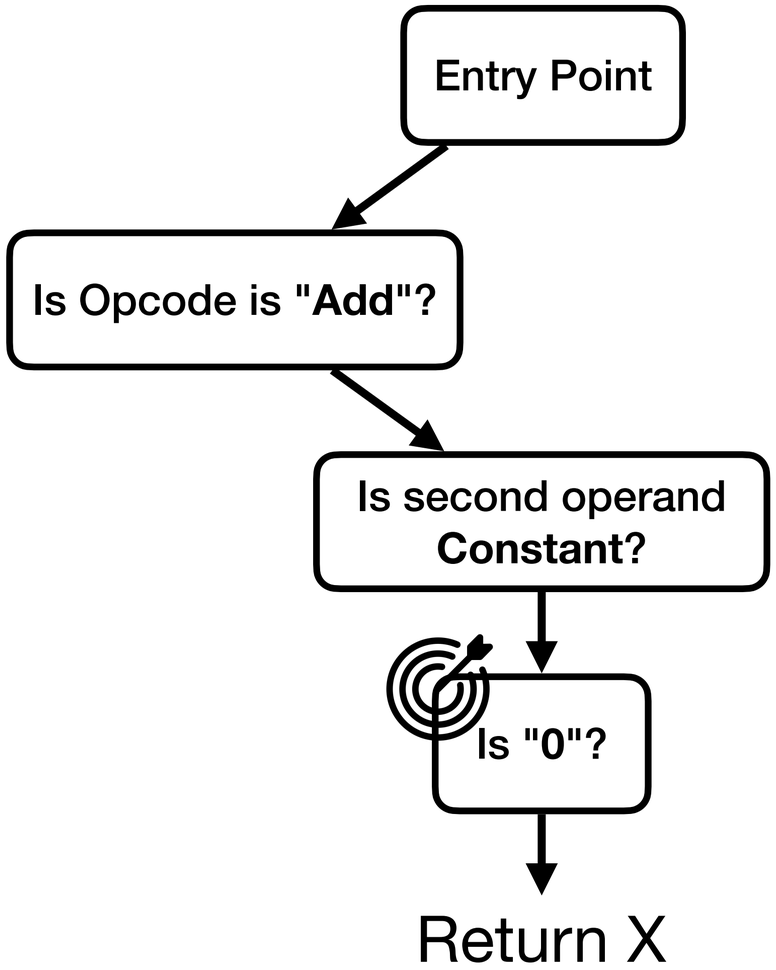
Target Optimization: $X + 0 \rightarrow X$

```
int foo (int X) {
  int temp = X + 0;
  return temp;
}
```

Input Program

```
int foo (int X) {
  int temp = X;
  return temp;
}
```

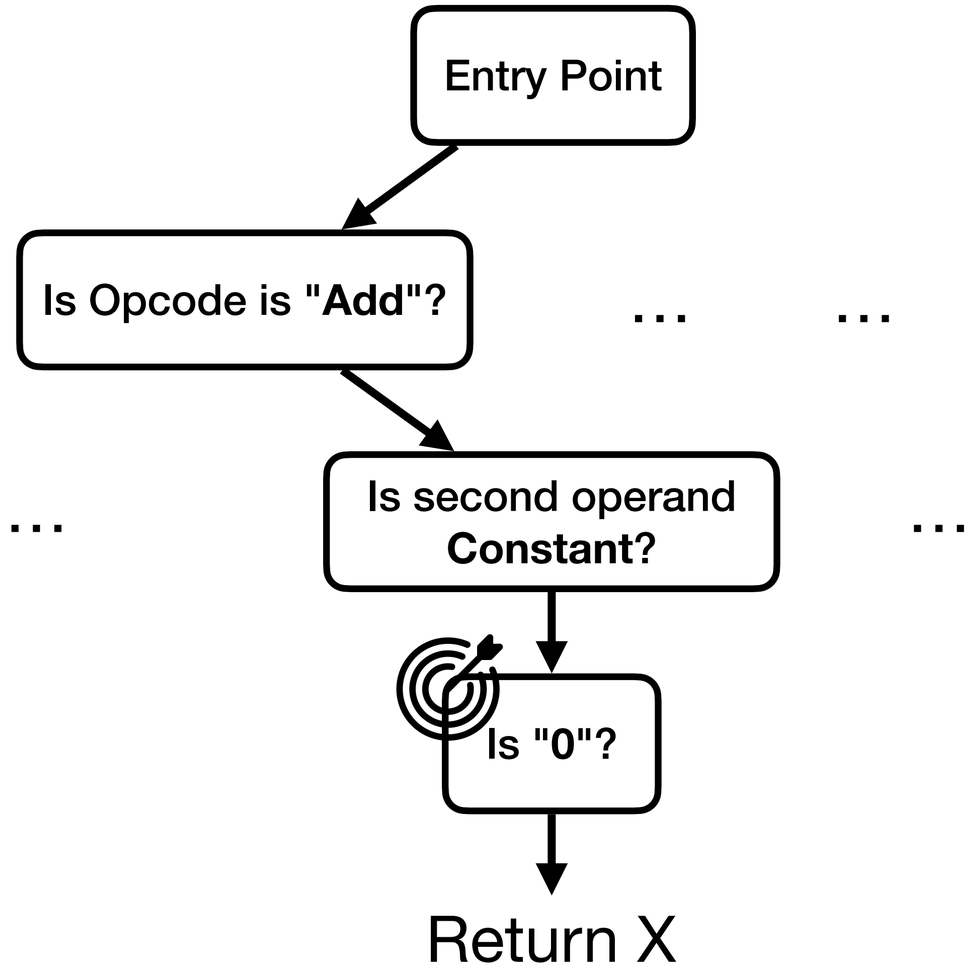
Optimized Program



Our Guide Strategy

```
int foo (int X, int Y) {
  int temp = X + Y;
  return temp;
}
```

Seed Program

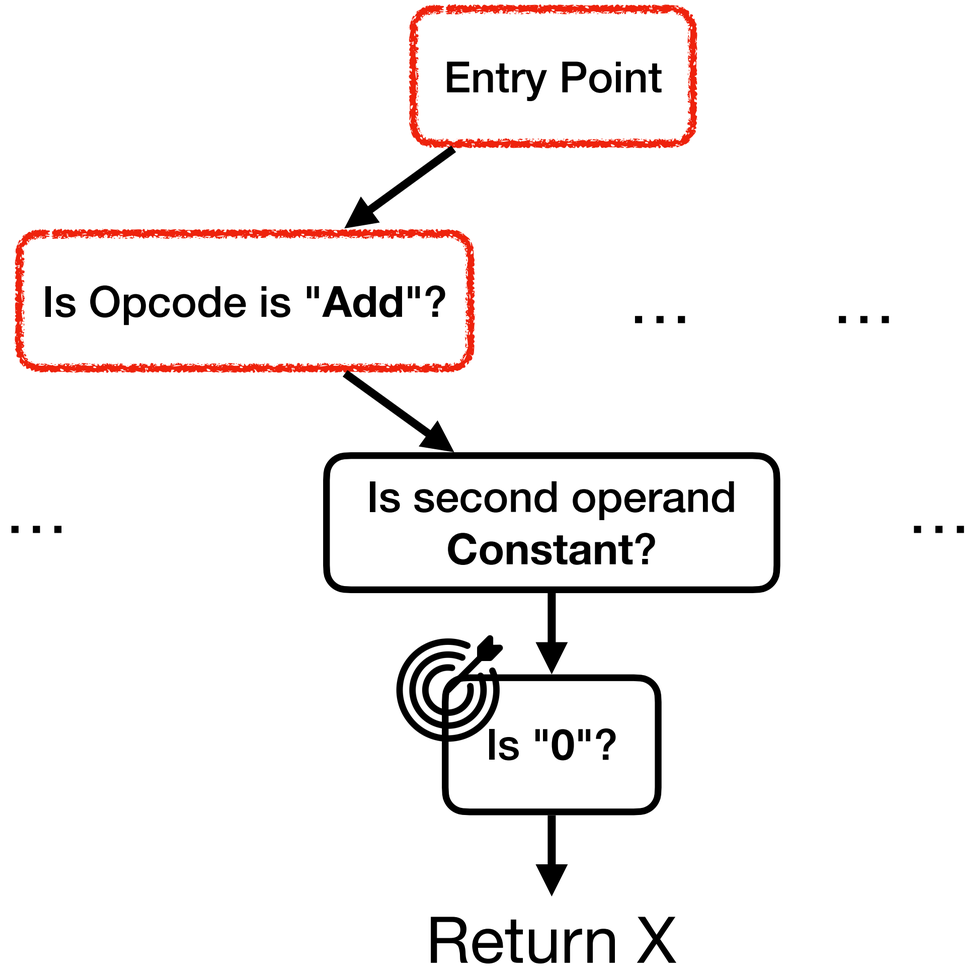


Our Guide Strategy

Covered Condition

```
int foo (int X, int Y) {
  int temp = X + Y;
  return temp;
}
```

Seed Program



Our Guide Strategy

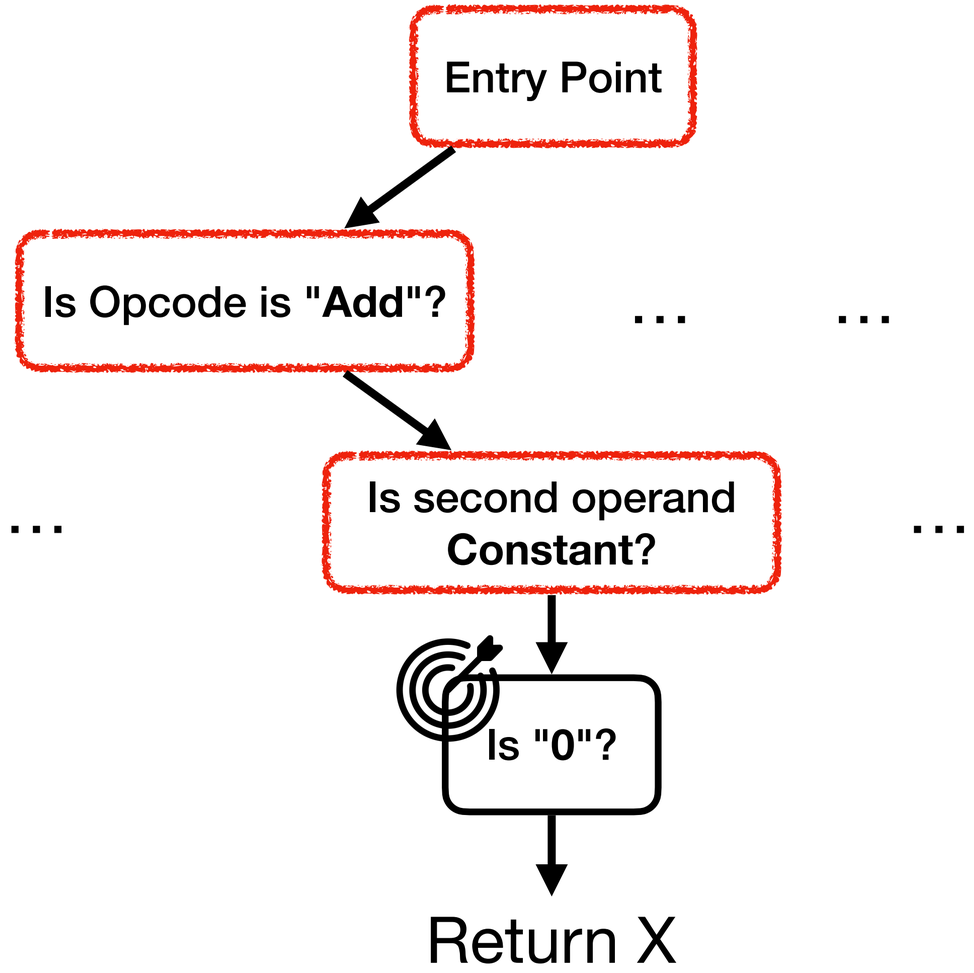
Covered Condition

```
int foo (int X, int Y) {
  int temp = X + Y;
  return temp;
}
```

Seed Program

```
int foo (int X, int Y) {
  int temp = X + 3;
  return temp;
}
```

Mutant (Interesting)



Our Guide Strategy

```
int foo (int X, int Y) {
  int temp = X + Y;
  return temp;
}
```

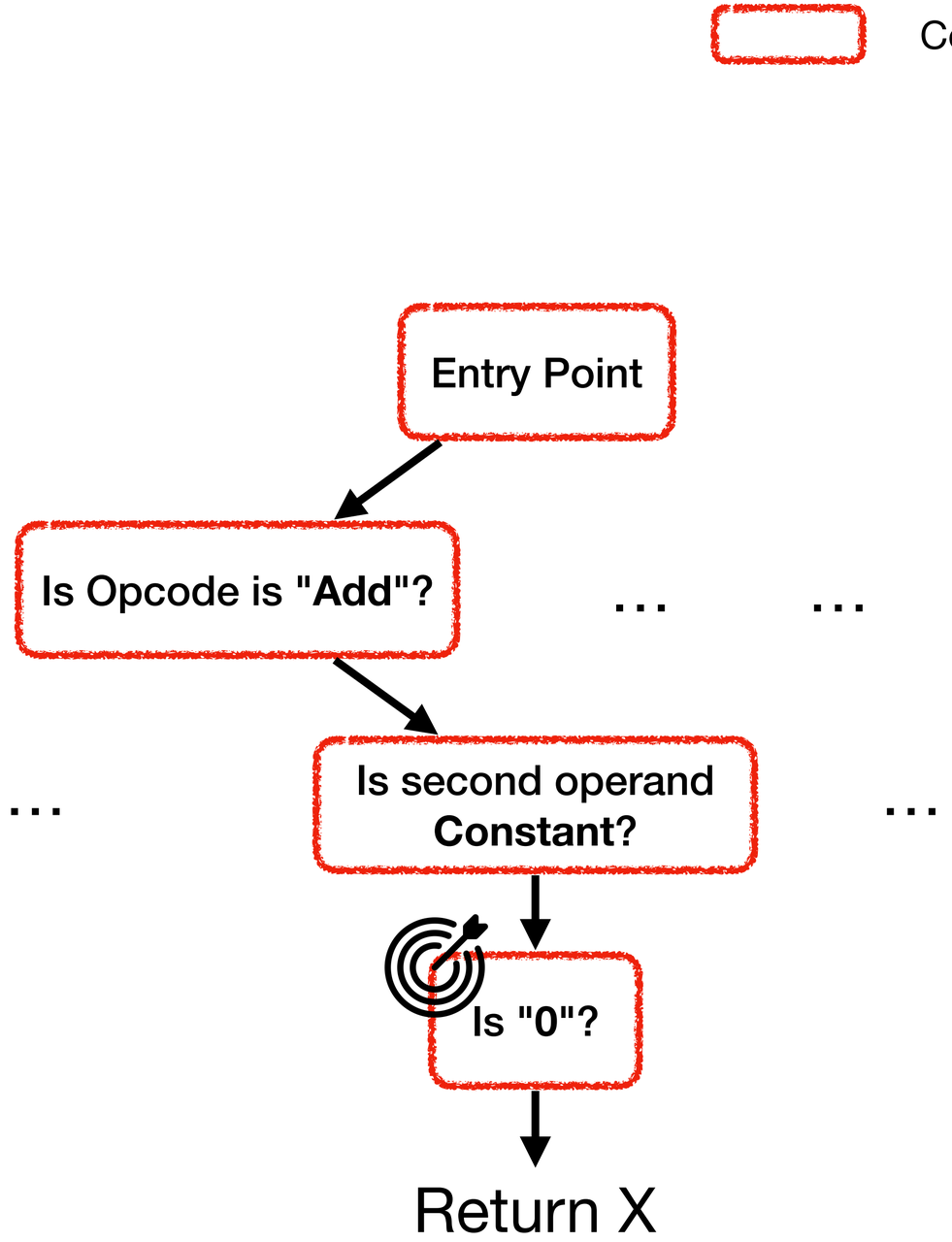
Seed Program

```
int foo (int X, int Y) {
  int temp = X + 3;
  return temp;
}
```

New Seed

```
int foo (int X, int Y) {
  int temp = X + 0;
  return temp;
}
```

Mutant (Interesting)



How to Improve Our Fuzzing Process?

- **Better Mutation**
  - All mutation should create a valid compiler IR
  - Each mutation should success more than the intended success rate

How to Improve Our Fuzzing Process?

- **Better Mutation**
  - All mutation should create a valid compiler IR
  - Each mutation should success more than the intended success rate

- **Effective Guide Performance**
  - Fuzzer should efficiently make IR that raise target optimization
  - Fuzzer should guide mutations towards target optimization well

Research Goal: Visualize Our Fuzzer

- **Visualize**
  - "A computer should make both calculations and **graphs**"*

*F. J. Anscombe. 1973. Graphs in Statistical Analysis

Research Goal: Visualize Our Fuzzer

- **Visualize**
  - "A computer should make both calculations and **graphs**"*
- **Mutation Statistics**
  - Graphically represent the attempt rates and success rates of each mutation

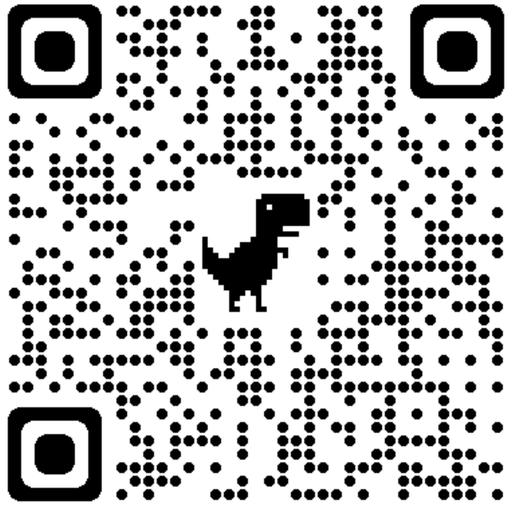
*F. J. Anscombe. 1973. Graphs in Statistical Analysis

Research Goal: Visualize Our Fuzzer

- **Visualize**
  - "A computer should make both calculations and **graphs**"*
- **Mutation Statistics**
  - Graphically represent the attempt rates and success rates of each mutation
- **Conditional Statement Tree Heat Map**
  - Measuring guide performance by displaying the coverage in a heatmap

*F. J. Anscombe. 1973. Graphs in Statistical Analysis

Demo



http://143.248.41.84:8000/index.html

Use School WiFi or KVPN

Mobile Available

Details

- Node toggling due to too many nodes (Almost 10,000 nodes)

Details

- Node toggling due to too many nodes (Almost 10,000 nodes)
- Log scale due to large variance in the number of times nodes covered

Details

- Node toggling due to too many nodes (Almost 10,000 nodes)
- Log scale due to large variance in the number of times nodes covered
- Search function for desired nodes

Details

- Node toggling due to too many nodes (Almost 10,000 nodes)
- Log scale due to large variance in the number of times nodes covered
- Search function for desired nodes
- Code snippet of the node (conditional statement)

Impacts

Easy Understanding

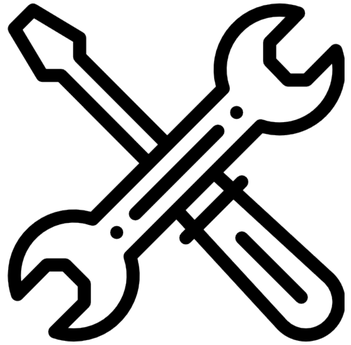
Entire visualization on **one** screen

Impacts

Easy Understanding

Entire visualization on **one** screen

Maintenance



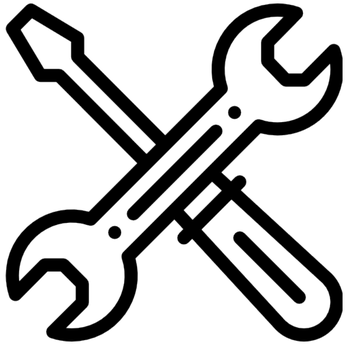
Found **3** bugs in our Fuzzer Implementation

Impacts

Easy Understanding

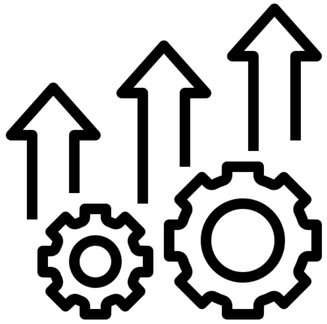
Entire visualization on **one** screen

Maintenance



Found **3** bugs in our Fuzzer Implementation

Improvement



Easy understanding of the current performance of our fuzzer

Future Works

- Display the seeds that covered the target node
- Naturally rendering the heatmap changes over time
- More useful features when improving the fuzzer