

# Finding and Understanding Bugs in C Compilers

2021 Most Influential PLDI Paper Award

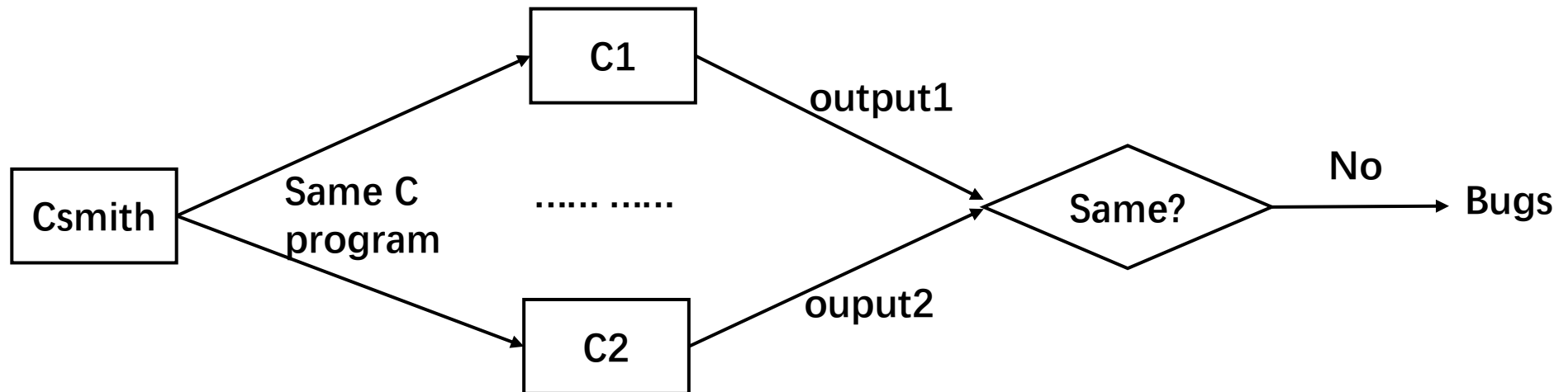
报告人：张灵毓

# Goal

- Finding bugs in mainstream C compilers like GCC and LLVM.

# Method

- Randomly generating C programs.
- Differential testing.



# Contributions

- Generating random programs that are **expressive**.



- Using many C language features.
  - Ensuring every program has one single interpretation.
- 
- A collection of qualitative and quantitative results about the bugs.

# Randomly Generating Programs

Randomly creating  
struct type declarations

```
struct X{  
    int a;  
    float b;  
};  
  
struct Y{  
    struct X x;  
    int c[2];  
};
```

ExtDef → Specifier FunDec CompSt (recurse)

Specifier → TYPE (random)  
| StructSpecifier

FunDec → ID LP RP (random)

.....

Exp → ID LP RP (ramdom)

```
void foo(){  
    .....  
    bar();  
}
```

foo is suspended until bar is finished.

# Randomly Generating Programs

```
struct X{  
    int a;  
    float b;  
};  
  
struct Y{  
    struct X x;  
    int c[2];  
};
```

Output main

Call first generated function

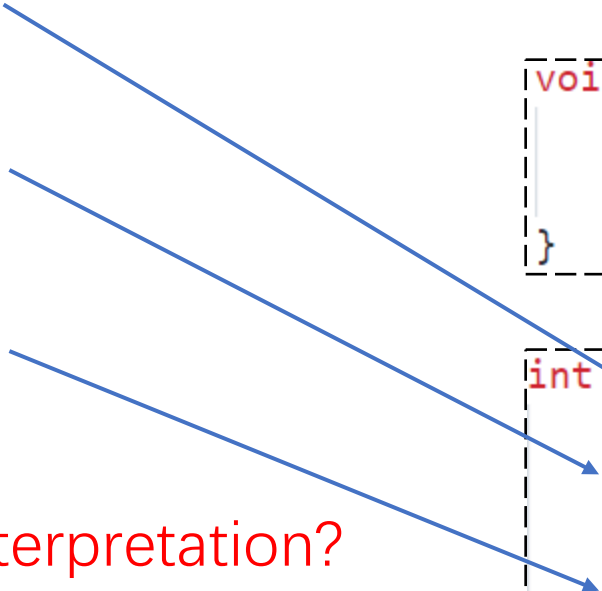
Output checksum

How to ensure one single interpretation?

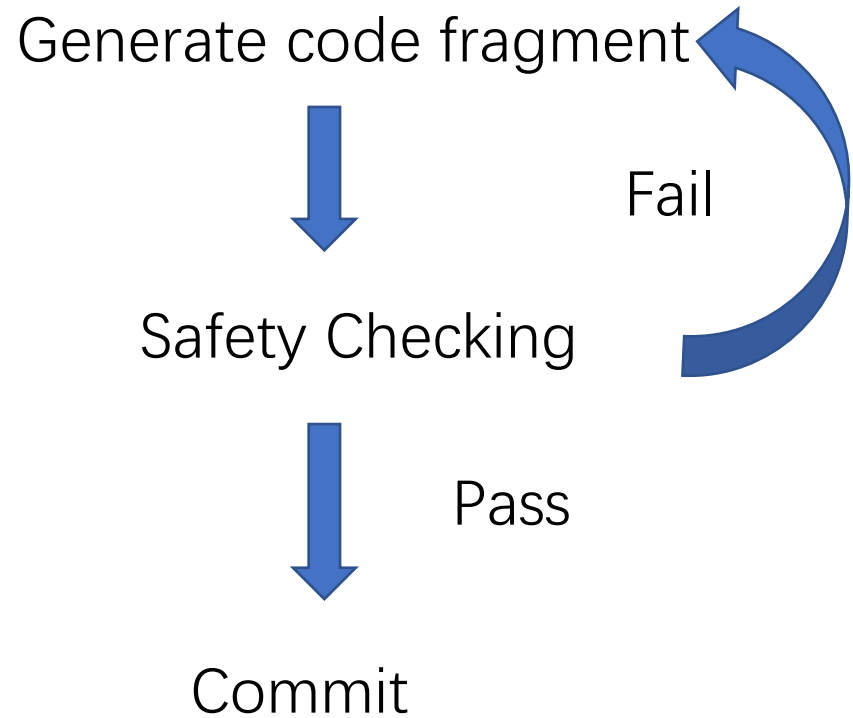
Static Analysis and Run-time Checking

```
void foo(){  
    .....  
    bar();  
}
```

```
int main(){  
    .....  
    foo();  
    .....  
    print(checkSum);  
    return 0;  
}
```



# Safety Mechanisms



- Integer Safety
- Type Safety
- Pointer Safety
- Effect Safety
- Array Safety
- Initializer Safety

# Integer safety

- Signed Overflow

$$x + 1 > x \rightarrow 1$$
$$\text{INT\_MAX} + 1 \rightarrow \text{INT\_MIN}$$

Wrapper Functions

- Shift-past-bitwidth

$1 \ll 31$  is illegal in C99 with 32-bit ints

# Type safety

- Qualifier Safety

```
// object of const-qualified type
const int n = 1;
int* p = (int*)&n;
// undefined behavior
*p = 2;
```

Static Analysis



# Pointer safety

- Null-pointer Dereference.
- Invalid-pointer Dereference.

```
int* p;
```

```
int foo(){  
    p = 0;  
    *p = 1; // null pointer.  
    .....  
    int a = 3;  
    p = &a;  
}
```

```
int bar(){  
    int x = *p; //invalid pointer.  
}
```

Pointer Analysis

*Pts = {locs, null, invalid}*

No Heap

# Effect safety

- Unspecified Order

```
func(a(), b());
```

```
int a = i++ + ++i;
```

Pointer Analysis

*Effect = {Set<sub>read</sub>, Set<sub>written</sub>}*

- Read/Write Conflict between Sequence Points

```
int a = p + func();
```

# Array safety

- Indices out of bounds.

## For Loop

```
for(int i = 0; i < arr.size(); ++i){  
    //not modify i  
    .....  
}
```

## Modulo Operator

```
arr[i % arr.size()];
```

# Initializer safety

- Uninitialized Function-scoped Variable

```
int foo(){  
    int a;//a is uninitialized  
    int x = a + 233;  
}
```

Structurally Ensure Initializing

---

```
int foo(){  
    int a;  
    goto LABEL; //span initializer  
    a = 1; //initialized here  
LABEL:  
    int x = a + 233;  
}
```

Forbid gotos from spanning  
initializer

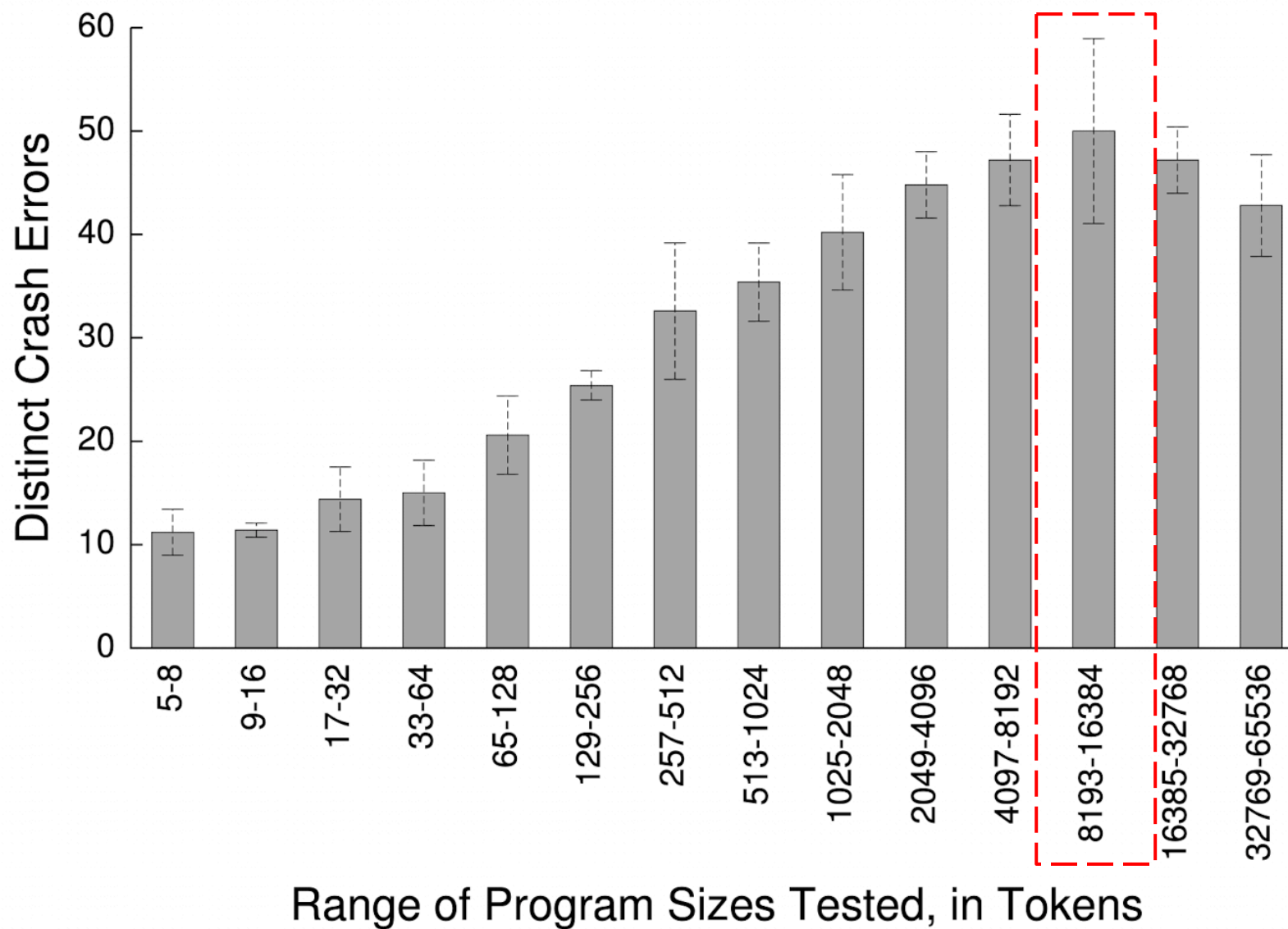
# Results

	<b>GCC</b>	<b>LLVM</b>
Crash	2	10
Wrong code	2	9
Total	4	19

---

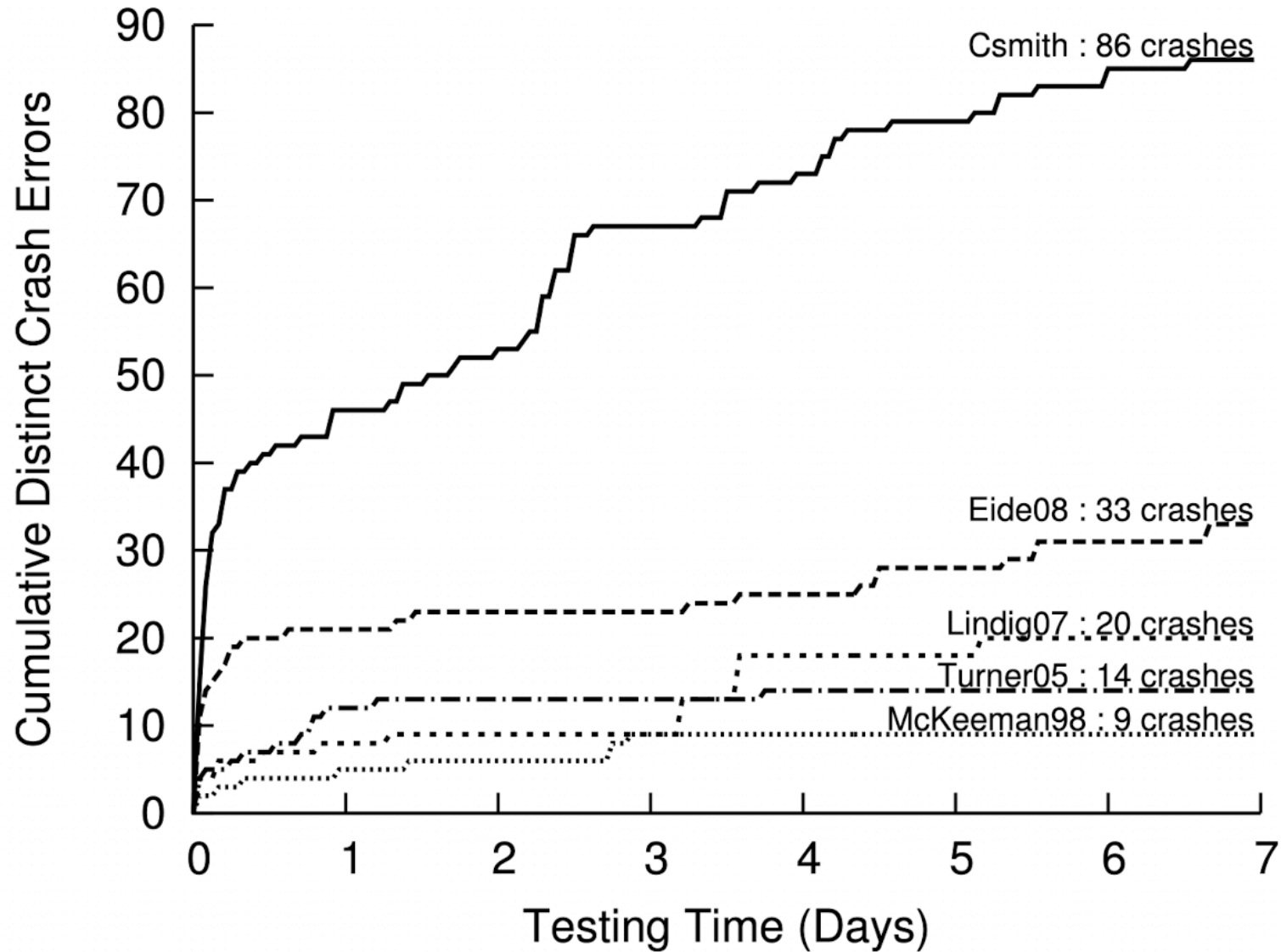
**Table 2.** Crash and wrong-code bugs found by Csmith that manifest when compiler optimizations are disabled (i.e., when the `-O0` command-line option is used)

# Results



**Figure 4.** Number of distinct crash errors found in 24 hours of testing with Csmith-generated programs in a given size range

# Results



**Figure 5.** Comparison of the ability of five random program generators to find distinct crash errors

# Results

		<b>Line Coverage</b>	<b>Function Coverage</b>	<b>Branch Coverage</b>
GCC	make check-c	75.13%	82.23%	46.26%
	make check-c & random	75.58%	82.41%	47.11%
	% change	+0.45%	+0.13%	+0.85%
	absolute change	+1,482	+33	+4,471
Clang	make test	74.54%	72.90%	59.22%
	make test & random	74.69%	72.95%	59.48%
	% change	+0.15%	+0.05%	+0.26%
	absolute change	+655	+74	+926

**Table 3.** Augmenting the GCC and LLVM test suites with 10,000 randomly generated programs did not improve code coverage much

Guess: these metrics are **too shallow to** capture Csmith's effects



Thank you!