Toward Impactful SE Research by Taking Roads Less Traveled

Zhendong Su

University of California, Davis

What is the **key mission** of

Computer Science?

To help people **turn creative ideas into working systems**

Software research, especially **SE**, is

at the center of this mission

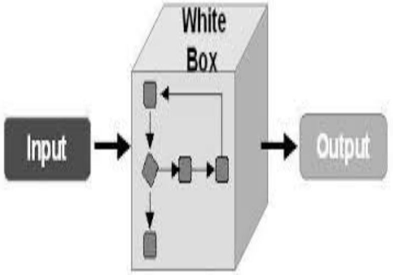
PERSPECTIVE
IS
EVERYTHING

1/2
1/2

Two instances

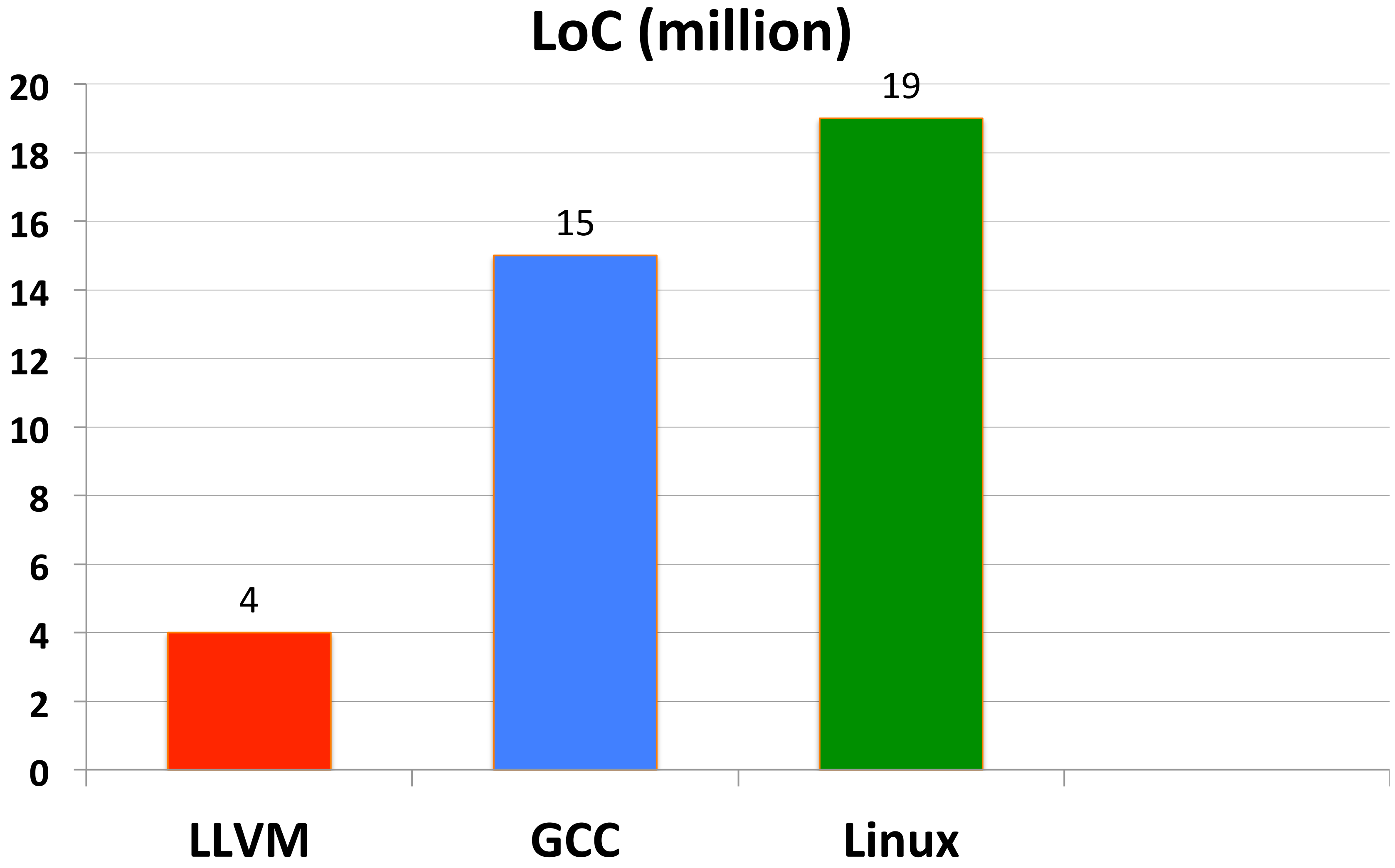
- ❑ **Validate production compilers**
  - ◆ Black-box analysis

- ❑ **Analyze floating-point software**
  - ◆ Dynamic analysis



Validate Production Compilers

Compiler complexity



LLVM bug 14972

```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

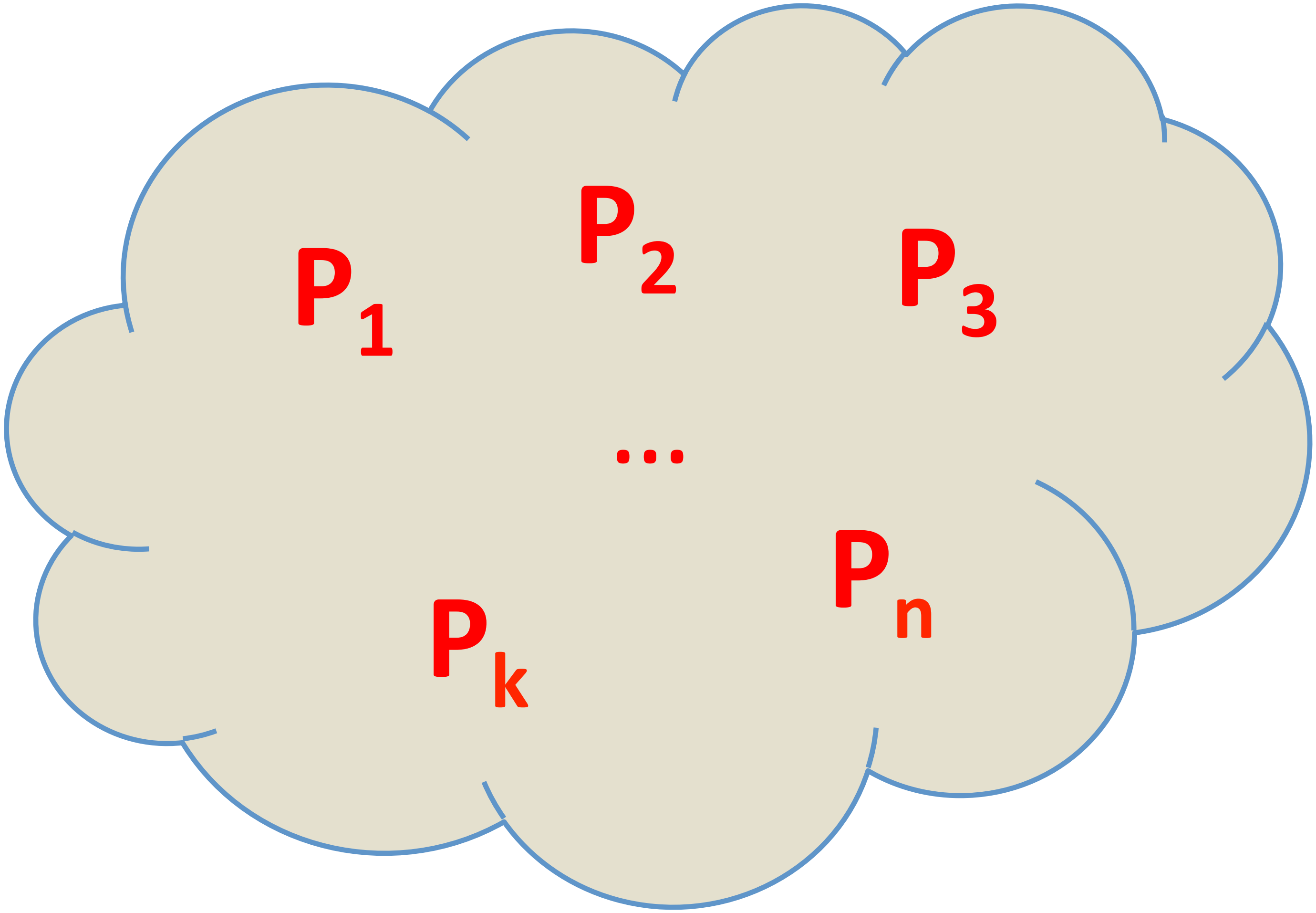
Developer comment

> *“… very, very concerning when I got to the root cause, and very annoying to fix …”*

http://llvm.org/bugs/show_bug.cgi?id=14972

Vision

$P \equiv$



P_1 P_2 P_3

…

P_k P_n

Key challenges

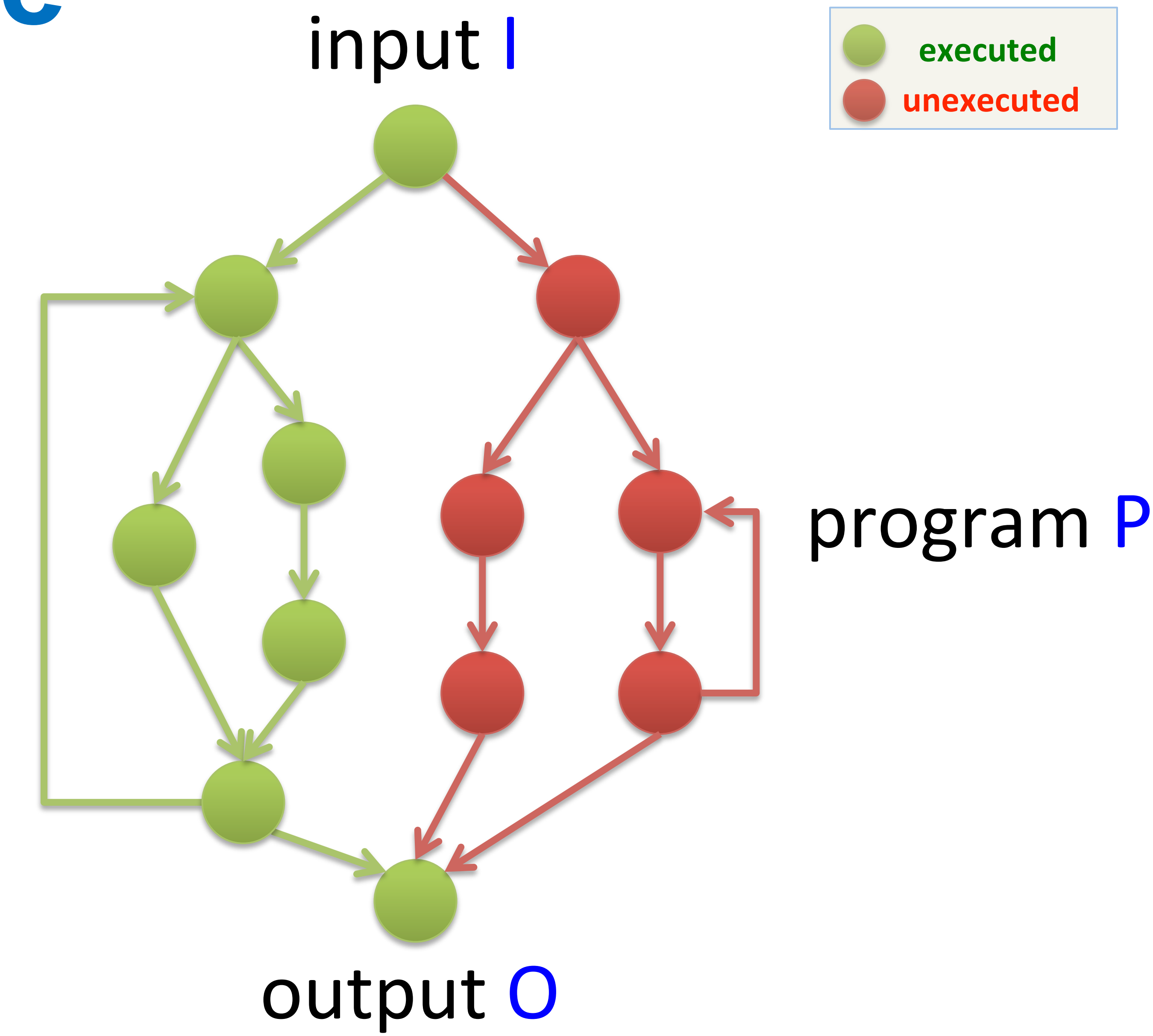
- Generation
  - How to generate **different**, yet **equivalent** tests?
- Validation
  - How to check that tests are **indeed equivalent**?

- Both are long-standing hard issues

Equiv. modulo inputs

- Relax equiv. wrt a **given input i**
  - **Must**: $P(i) = P_k(i)$ on input **i**
  - **Okay**: $P(j) \neq P_k(j)$ on all input $j \neq i$
- Exploit close **interplay** between
  - **Dynamic** program execution on **some input**
  - **Static** compilation for **all input**

Profile



input I

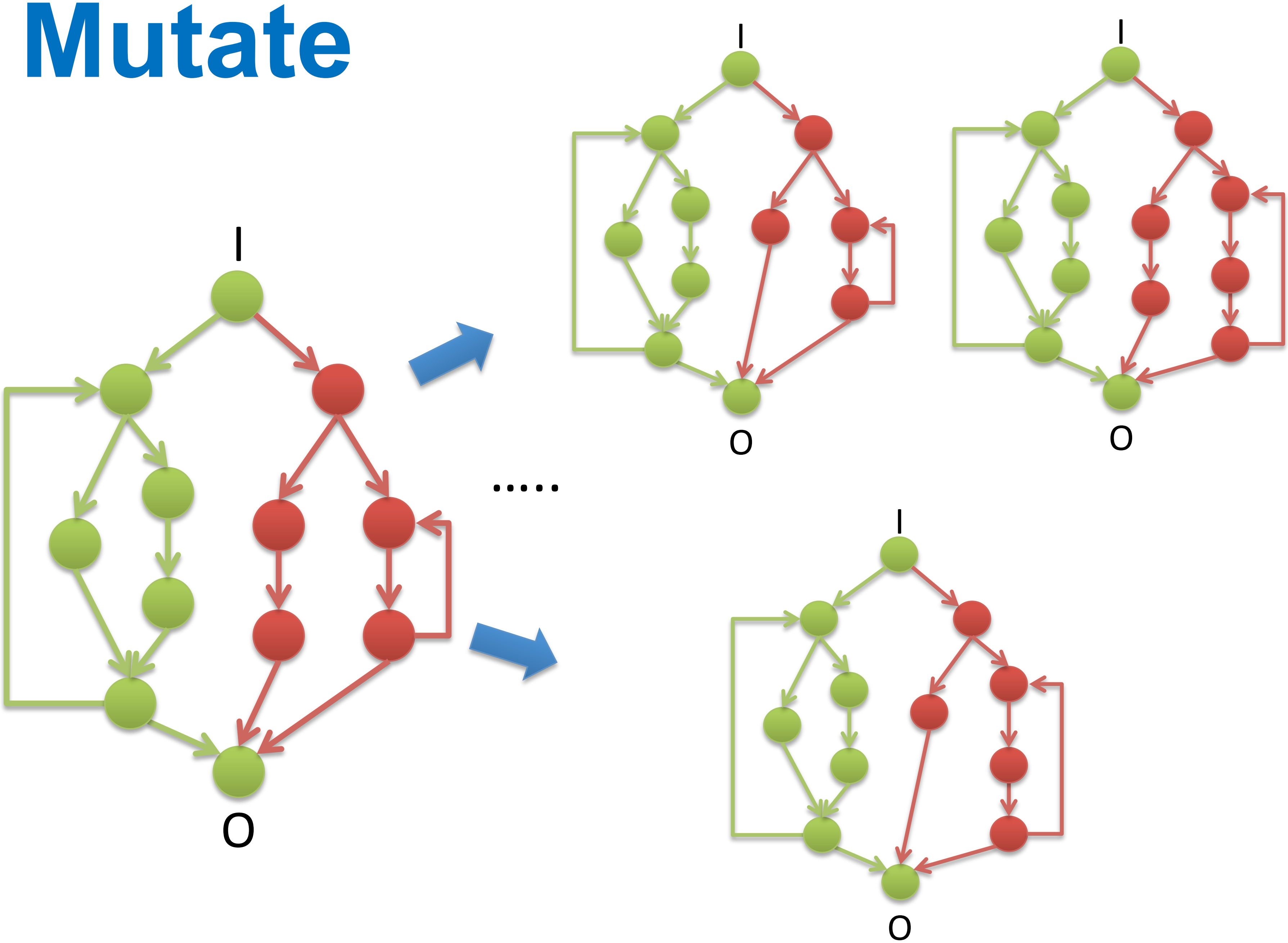
● executed

● unexecuted

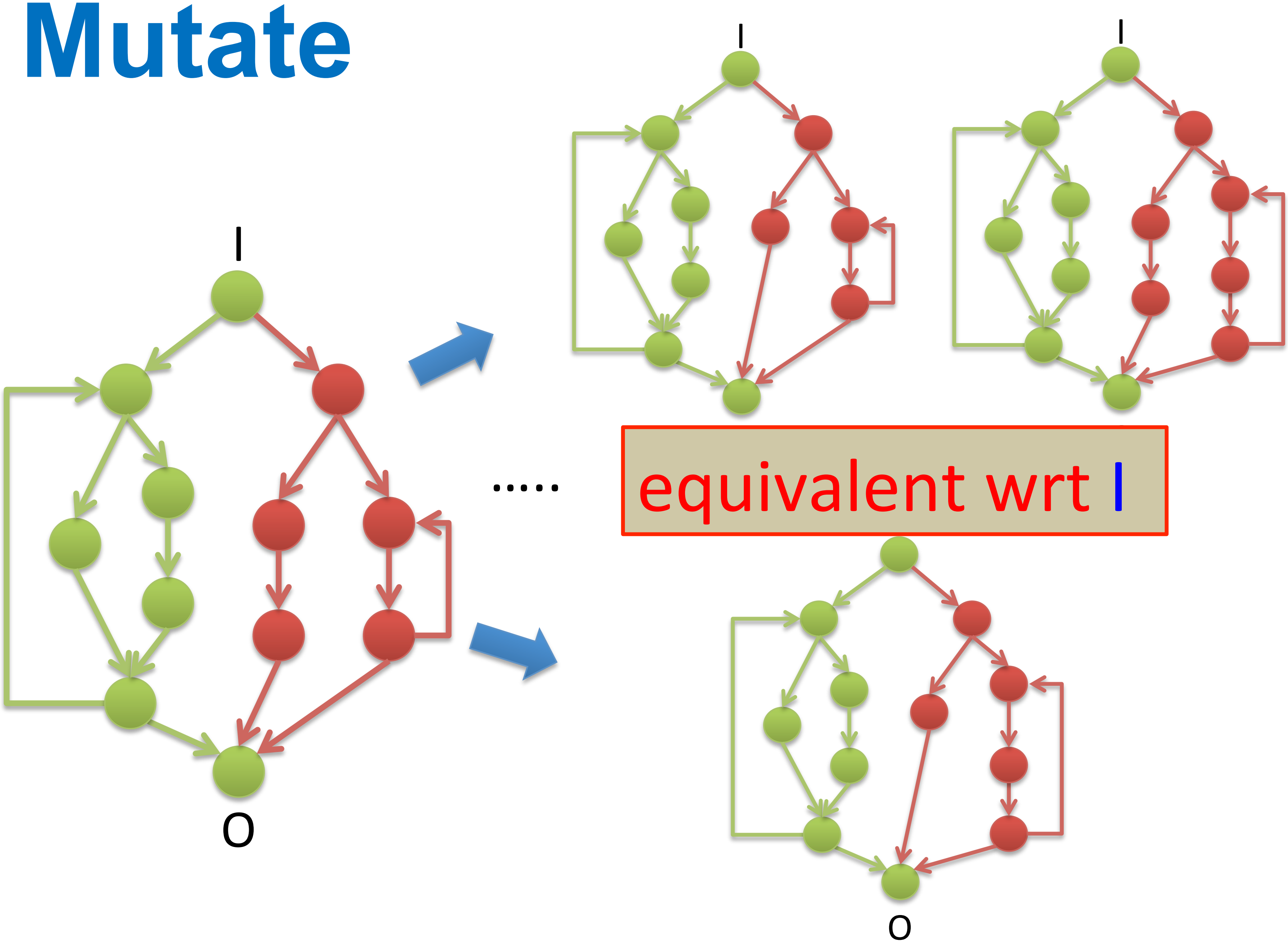
program P

output O

Mutate

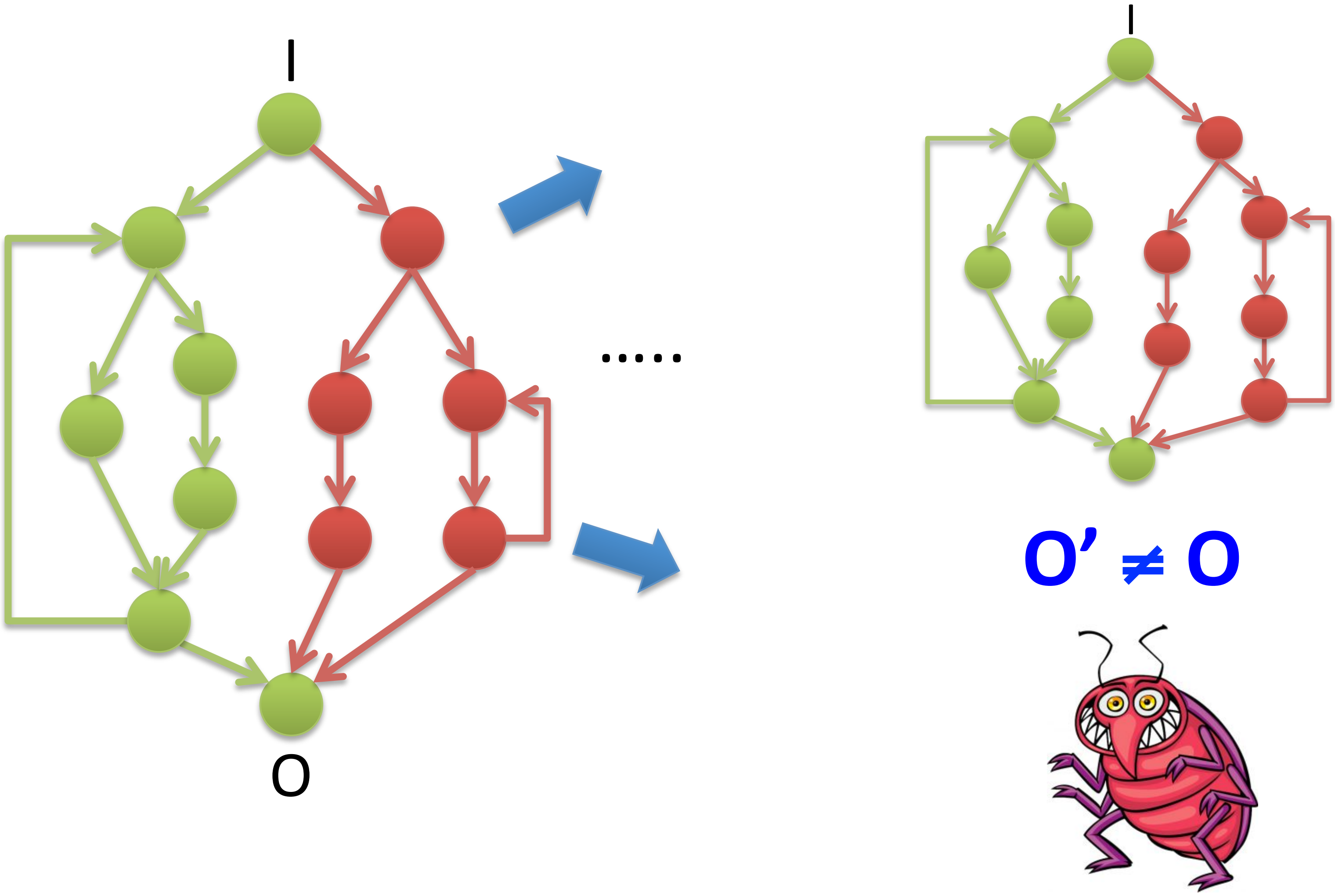


Mutate



equivalent wrt I

Find bugs



O’ ≠ O

Revisit challenges

- Generation (**easy**)
  - How to generate **different**, yet **equivalent** tests?
- Validation (**easy**)
  - How to check that tests are **indeed equivalent**?

- ~~Both are long-standing hard issues~~

LLVM bug 14972

```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

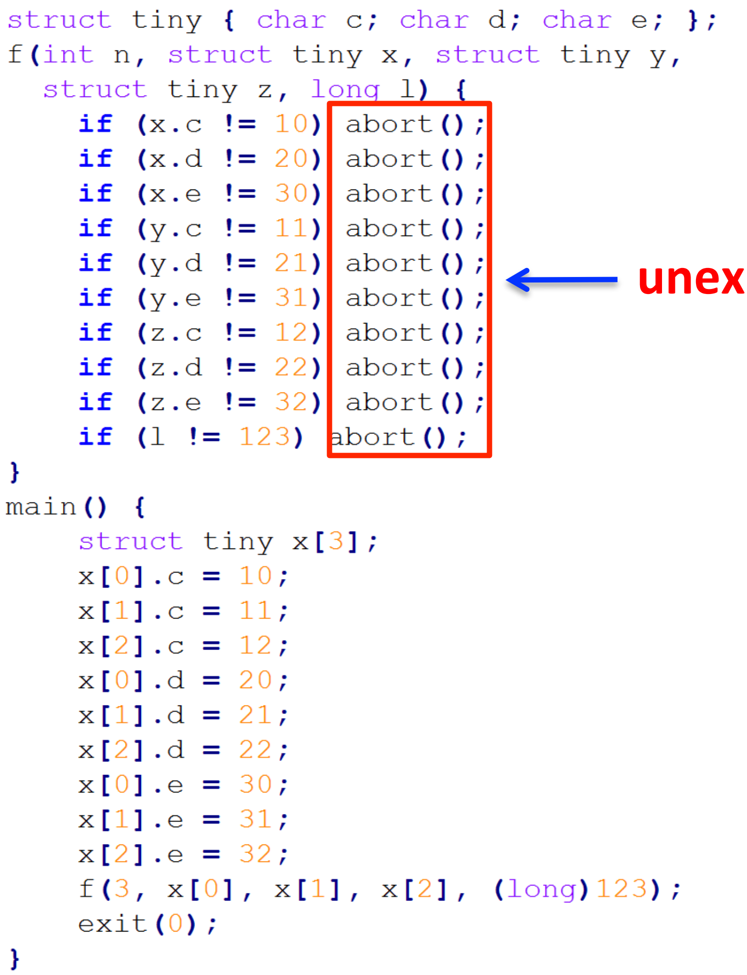
```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

Seed file

```
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) abort();
    if (x.d != 20) abort();
    if (x.e != 30) abort();
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) abort();
    if (z.c != 12) abort();
    if (z.d != 22) abort();
    if (z.e != 32) abort();
    if (l != 123) abort();
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
```

Seed file



```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) abort();
    if (x.d != 20) abort();
    if (x.e != 30) abort();
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) abort();
    if (z.c != 12) abort();
    if (z.d != 22) abort();
    if (z.e != 32) abort();
    if (l != 123) abort();
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

unexecuted

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
```

Transformed file

```
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) /* deleted */;
    if (x.d != 20) abort();
    if (x.e != 30) /* deleted */;
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) /* deleted */;
    if (z.c != 12) abort();
    if (z.d != 22) /* deleted */;
    if (z.e != 32) abort();
    if (l != 123) /* deleted */;
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

Reduced file

```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

LLVM bug autopsy

```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

GVN: load struct using 32-bit load

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

LLVM bug autopsy

```
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

GVN: load struct using 32-bit load

SRoA: read past the struct’s end

➔

undefined behavior

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

LLVM bug autopsy

```c
struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
```

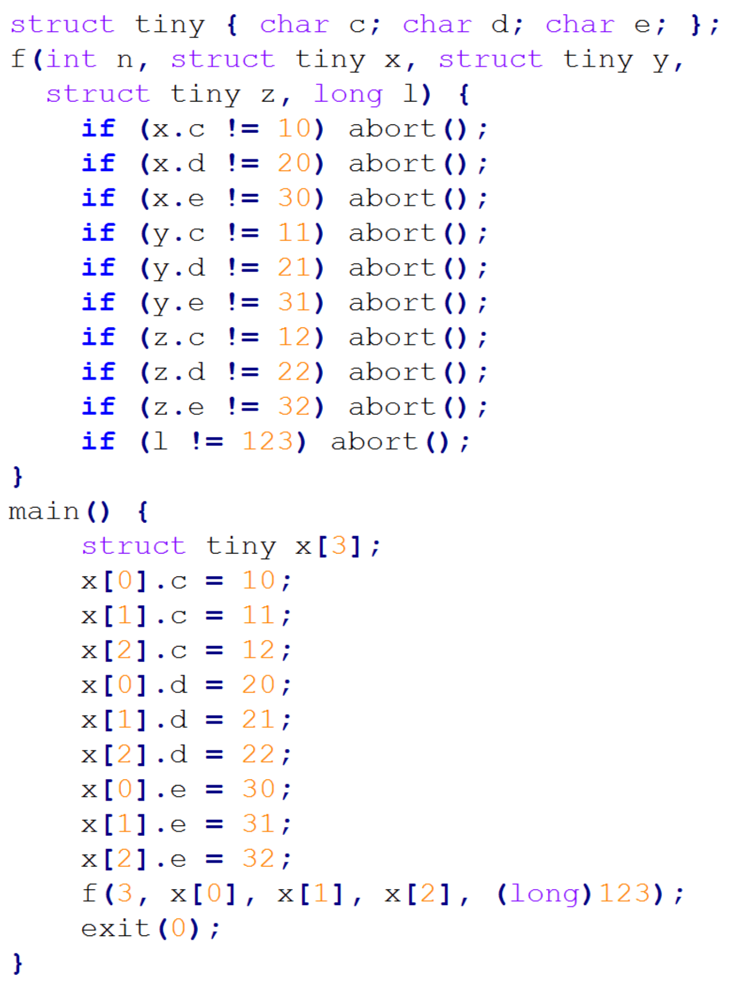
GVN: load struct using 32-bit load

SRoA: read past the struct's end → undefined behavior

remove

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

Seed file



```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) abort();
    if (x.d != 20) abort();
    if (x.e != 30) abort();
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) abort();
    if (z.c != 12) abort();
    if (z.d != 22) abort();
    if (z.e != 32) abort();
    if (l != 123) abort();
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

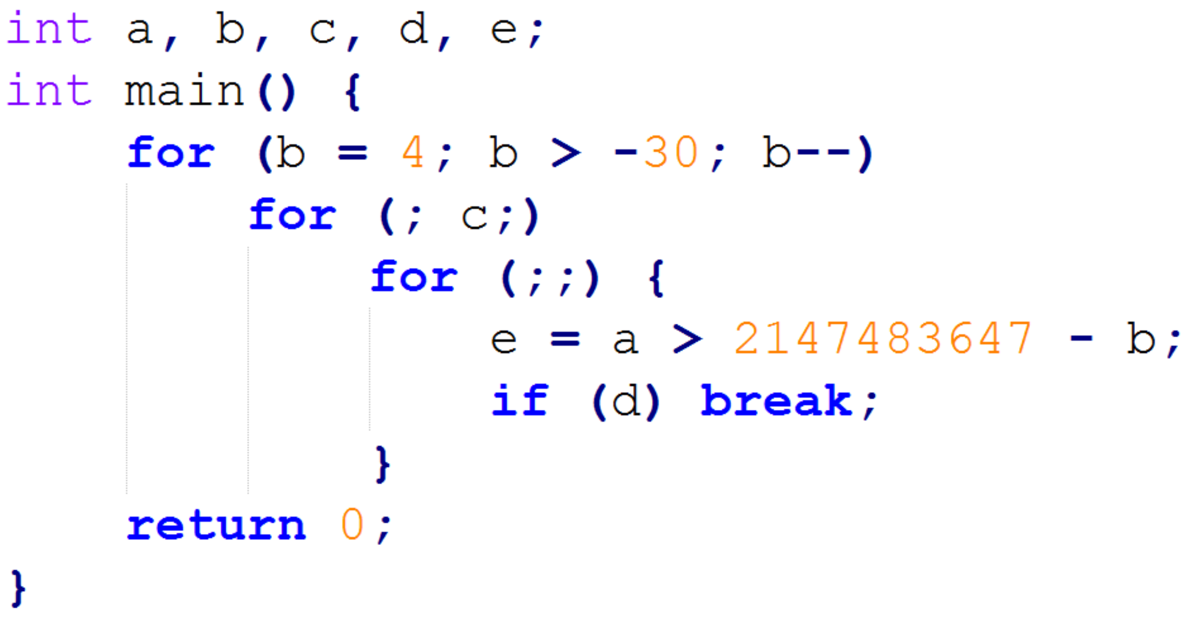
```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
```

Transformed file

```
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y,
  struct tiny z, long l) {
    if (x.c != 10) /* deleted */;
    if (x.d != 20) abort();
    if (x.e != 30) /* deleted */;
    if (y.c != 11) abort();
    if (y.d != 21) abort();
    if (y.e != 31) /* deleted */;
    if (z.c != 12) abort();
    if (z.d != 22) /* deleted */;
    if (z.e != 32) abort();
    if (l != 123) /* deleted */;
}
main() {
    struct tiny x[3];
    x[0].c = 10;
    x[1].c = 11;
    x[2].c = 12;
    x[0].d = 20;
    x[1].d = 21;
    x[2].d = 22;
    x[0].e = 30;
    x[1].e = 31;
    x[2].e = 32;
    f(3, x[0], x[1], x[2], (long)123);
    exit(0);
}
```

```
$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
```

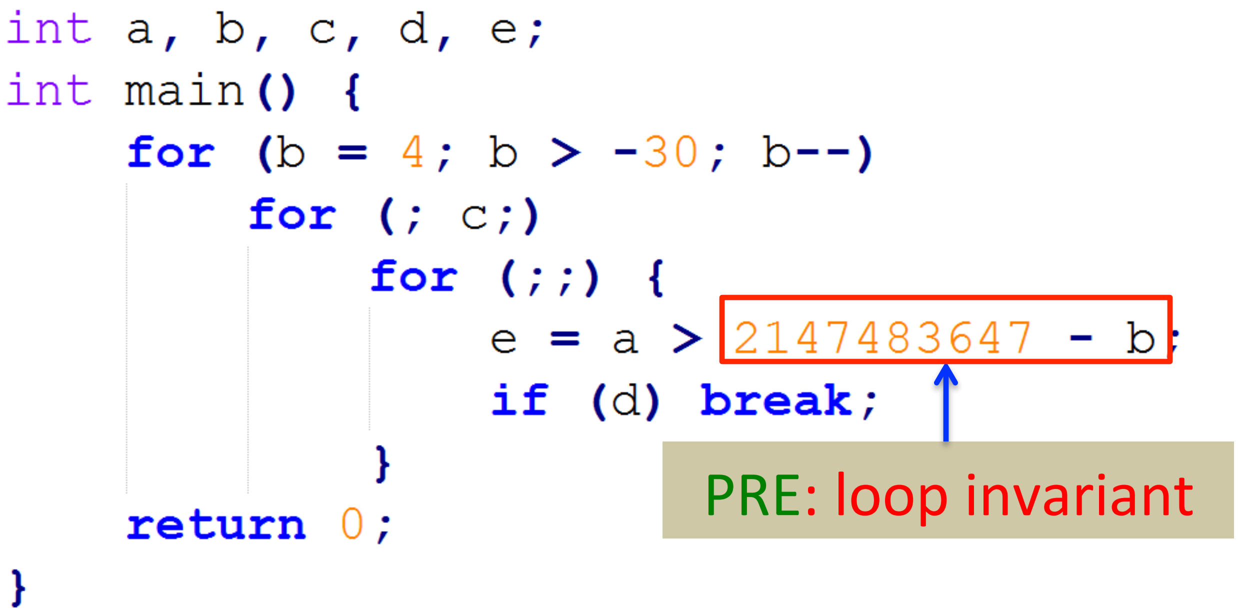
GCC bug 58731



```c
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        for (; c;)
            for (;;) {
                e = a > 2147483647 - b;
                if (d) break;
            }
    return 0;
}
```

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O3 test.c ; ./a.out
^C
```

GCC bug autopsy

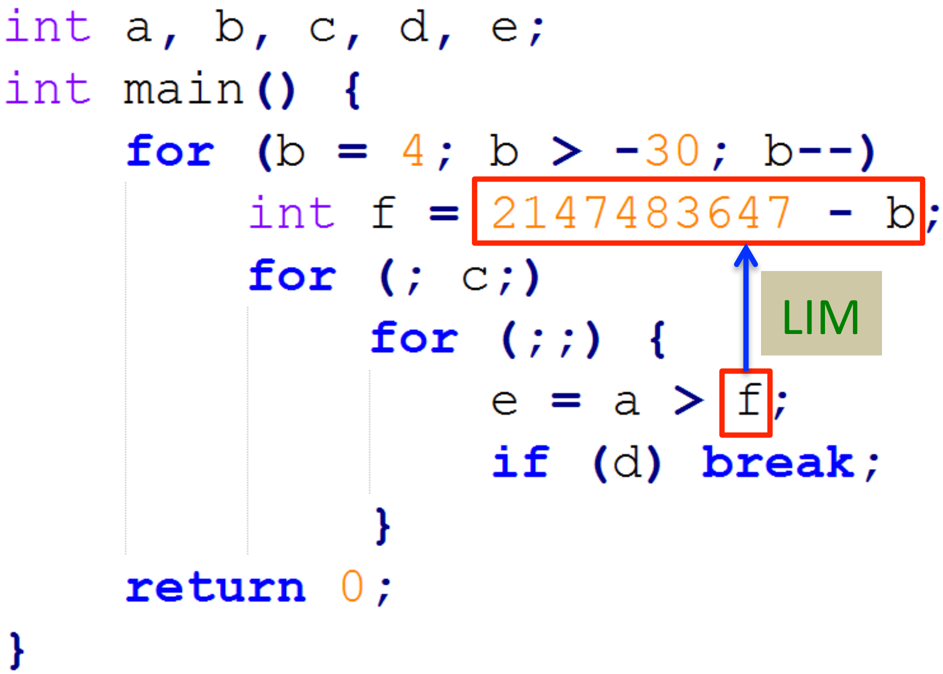


```c
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        for (; c;)
            for (;;) {
                e = a > 2147483647 - b;
                if (d) break;
            }
    return 0;
}
```

PRE: loop invariant

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O3 test.c ; ./a.out
^C
```

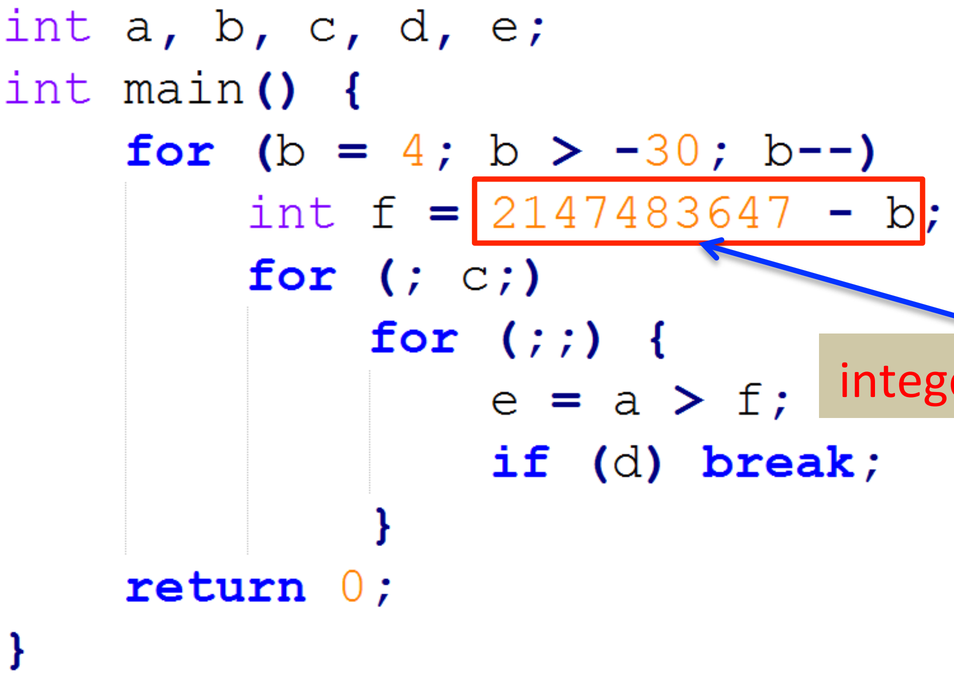
GCC bug autopsy



```c
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        int f = 2147483647 - b;
        for (; c;)
            for (;;) {
                e = a > f;
                if (d) break;
            }
    return 0;
}
```

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O3 test.c ; ./a.out
^C
```

GCC bug autopsy

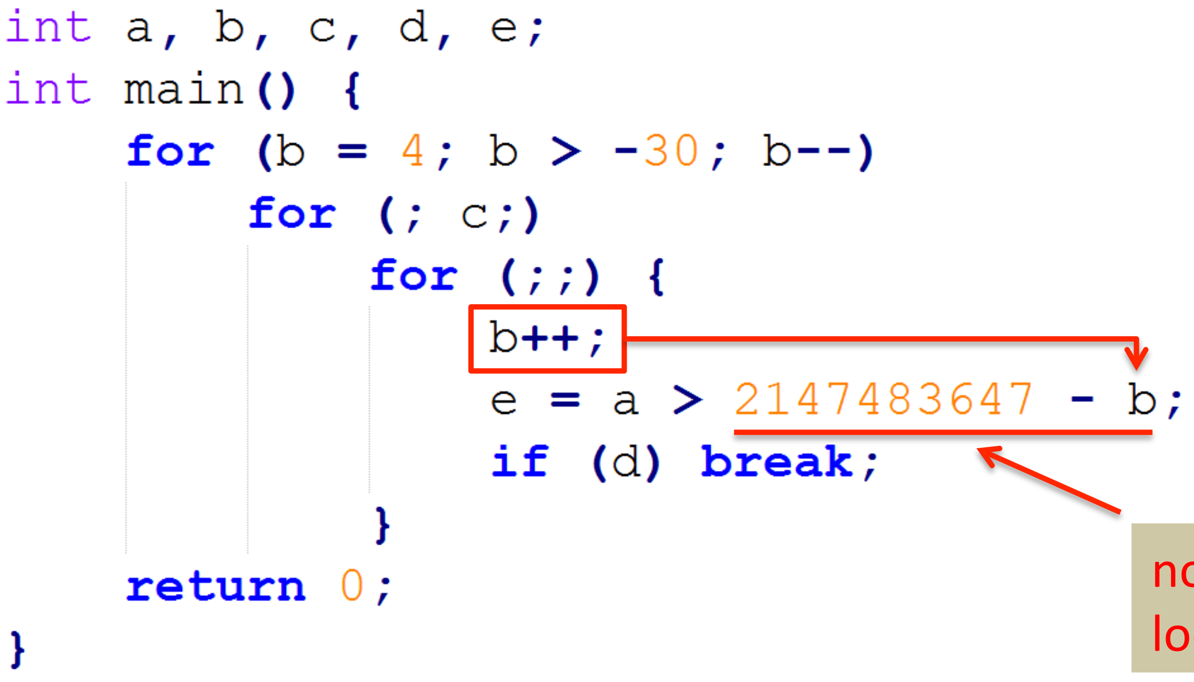


```c
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        int f = 2147483647 - b;
        for (; c;)
            for (;;) {
                e = a > f;
                if (d) break;
            }
    return 0;
}
```

integer overflow

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O3 test.c ; ./a.out
^C
```

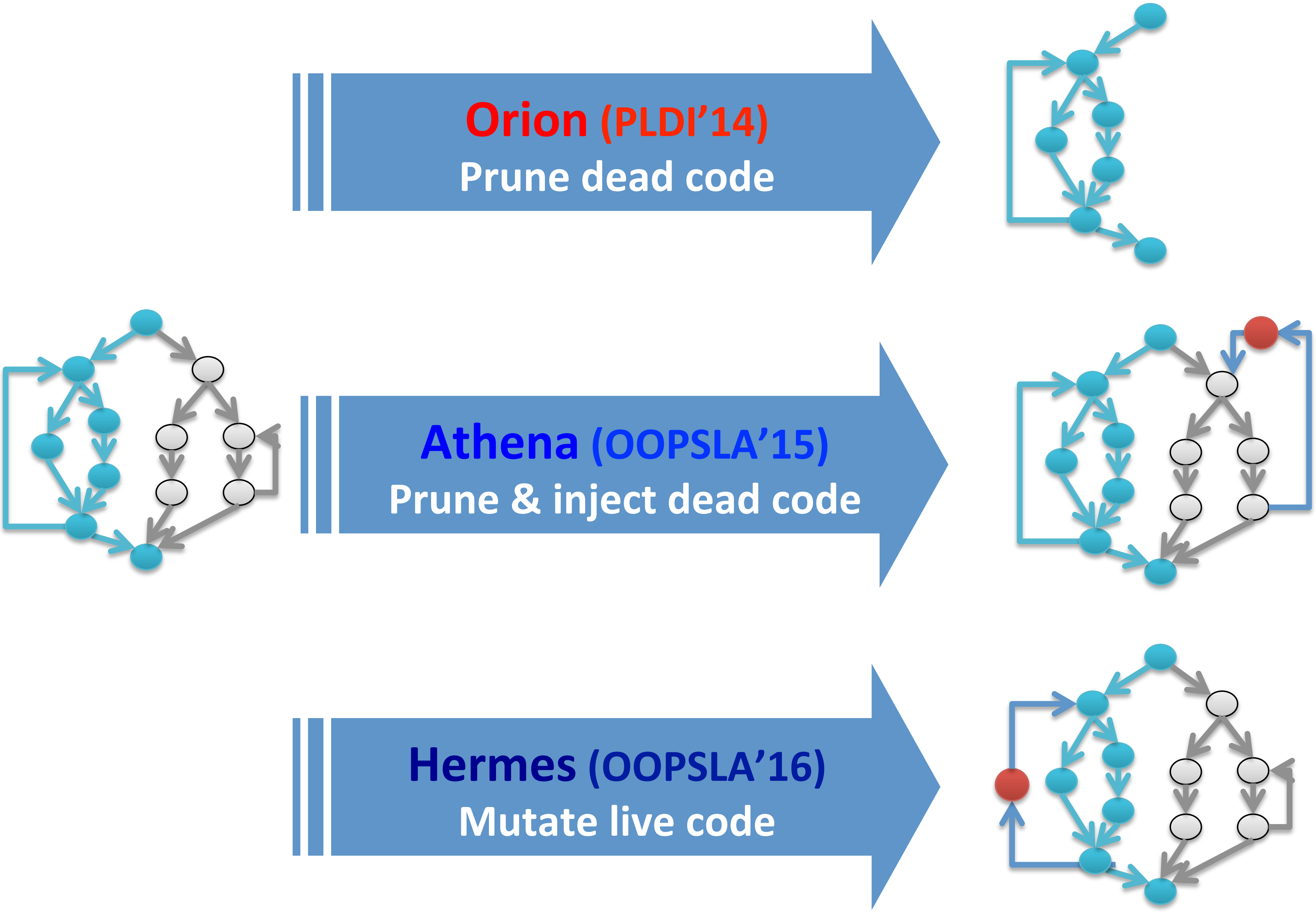
Seed program



```c
int a, b, c, d, e;
int main() {
    for (b = 4; b > -30; b--)
        for (; c;)
            for (;;) {
                b++;
                e = a > 2147483647 - b;
                if (d) break;
            }
    return 0;
}
```

no longer a loop invariant

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O3 test.c ; ./a.out
```



Orion (PLDI'14)

Prune dead code

Athena (OOPSLA'15)

Prune & inject dead code

Hermes (OOPSLA'16)

Mutate live code

Bug counts

	GCC	LLVM	TOTAL
Reported	710	594	**1304**
Fixed	467	309	**776**

- **ISSTA'15: Stress-testing link-time optimization**
- **ICSE'16: Analyzing compilers' diagnostic support**
- **PLDI'17: Skeletal program enumeration**

LLVM 3.9 & 4.0 Release Notes

"… thanks to **Zhendong Su and his team** whose fuzz testing **prevented many bugs** going into the release …"

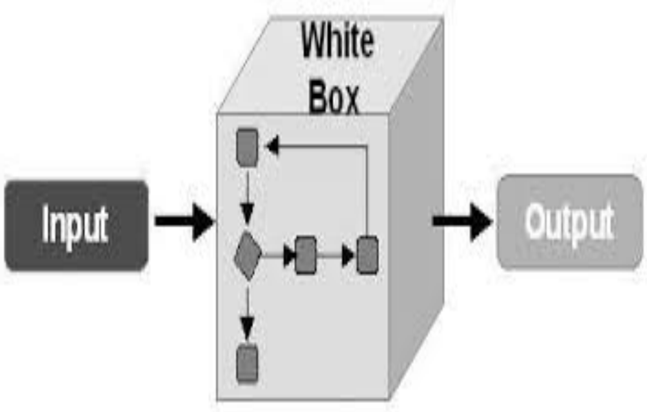
EMI

- **General**, **widely applicable**
  - Can test compilers, analysis, transformation tools
  - Finds real-world tests and requires no reference compilers
- **Very effective** realizations
  - Uncovered **1300+ bugs** in GCC and LLVM in 3 years
  - Many **miscompilations** and **long latent** bugs

	GCC	LLVM	TOTAL
Reported	710	594	**1304**
Fixed	467	309	**776**

Take-away*: Not only do good research, but be its* ***loyal****,* ***continuous*** *user!*

Analyze Floating-Point Software



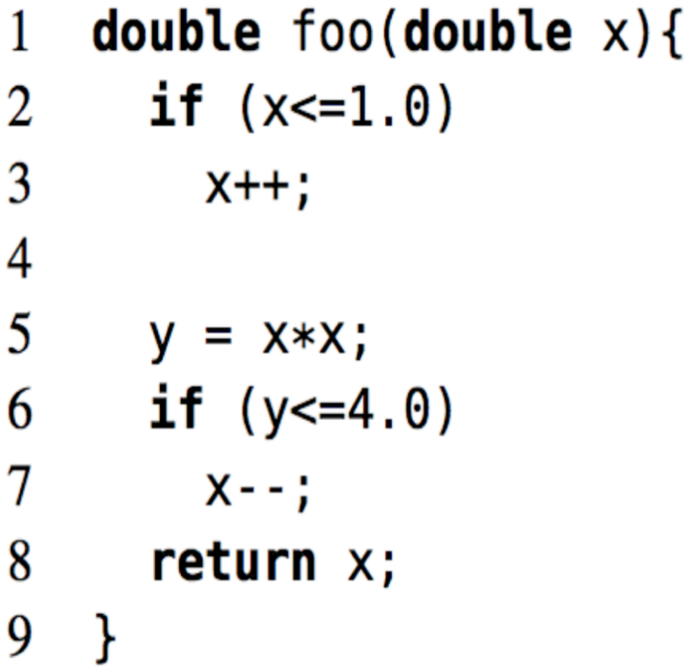
Floating-point code

- **Important:** bugs can lead to disasters
- **Challenging:** hard to get right

Why difficult?

- **FP Math ≠ Real Math**
- **Non-linear** relations
- **Transcendental** functions

  sin, log, exp, …



```
double foo(double x){
  if (x<=1.0)
    x++;

  y = x*x;
  if (y<=4.0)
    x--;
  return x;
}
```

Challenging for all known approaches

New perspective: ME

(p, ϕ)

Analyzing numerical programs

- Coverage-based testing
- Boundary value analysis
- Numerical exception detection

Floating-point constraint solving

$\Downarrow$ **Mathematical Execution (ME)**

r

Mathematical optimization (MO)

input **x** drives **p** to satisfy ϕ $\leftrightarrow$ **x** minimizes **r**

FP constraints

Solving the floating-point constraint $\boldsymbol{\pi}$

$$(SIN(x) == x) \wedge (x \geq 10^{-10})$$

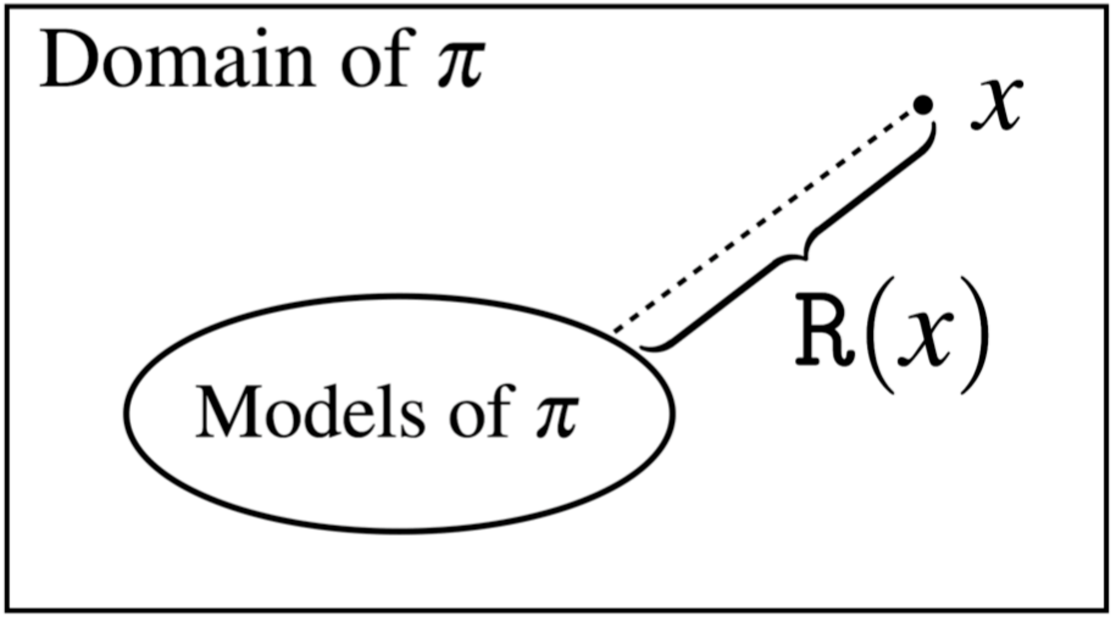
- Satisfiable if x is floating-point

$$\text{For } x \in \mathbb{F}, SIN(x) = x \Leftrightarrow x \simeq 0$$

- Unsatisfiable if x is real

$$\text{For } x \in \mathbb{R}, SIN(x) = x \Leftrightarrow x = 0$$

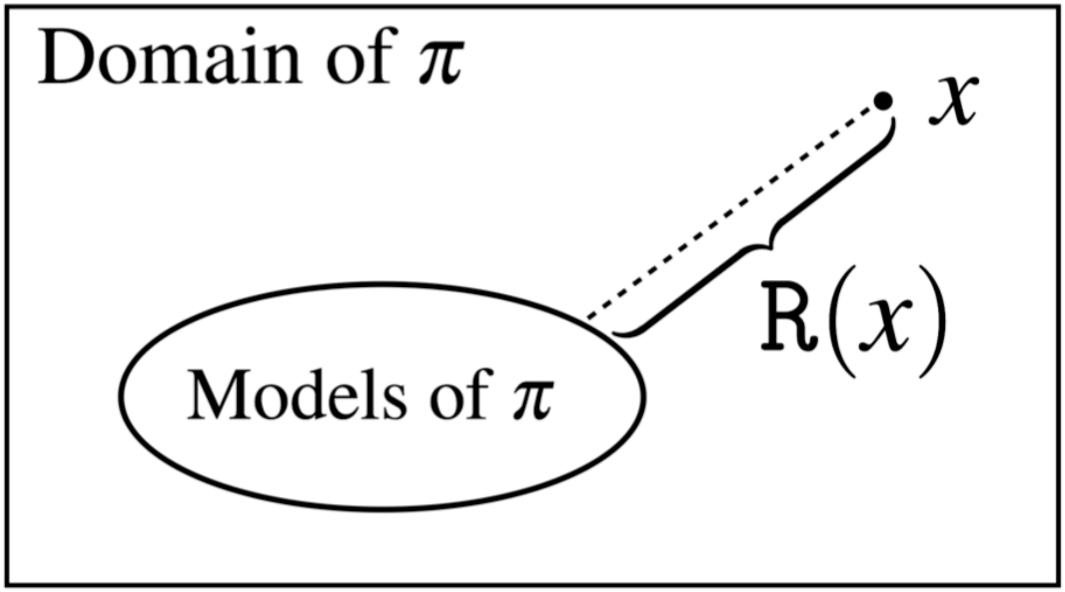
Step 1



Simulate π with a floating-point program R

- $\mathrm{R}(x) \geq 0$ for all x
- $\mathrm{R}(x) = 0 \Leftrightarrow x \models \pi$

Step 2



Minimize R as if it is a mathematical function

- Let x^* be the minimum point

$$\pi \text{ satisfiable} \Leftrightarrow \mathsf{R}(x^*) = 0$$

Construct R

Necessary Conditions to meet :

1. $\mathsf{R}(x) \geq 0$ for all x
2. $\mathsf{R}(x) = 0 \Leftrightarrow x \models \pi$

How?

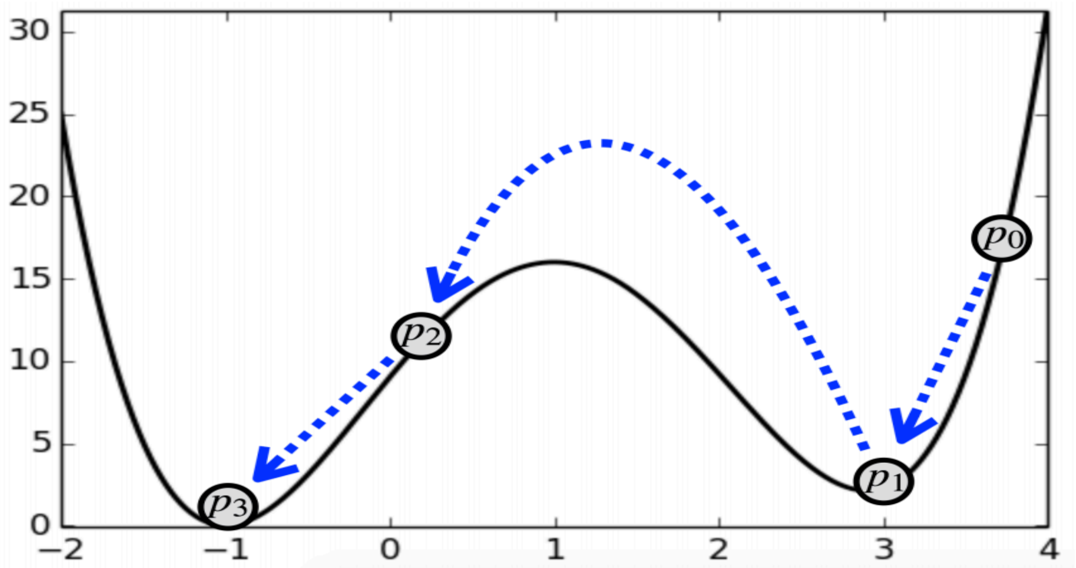
Constraint π	Program R
$x == y$	$(x - y)^2$
$x \leq y$	$x \leq y \ ?\ 0 : (x - y)^2$
$\pi_1 \wedge \pi_2$	$\mathsf{R}_1 + \mathsf{R}_2$
$\pi_1 \vee \pi_2$	$\mathsf{R}_1 * \mathsf{R}_2$

R can be constructed from a CNF form

Minimize R

Unconstrained programming techniques:

- Local optimization
- Monte Carlo Markov Chain (MCMC)
- We use them as black-box
- Do not analyze π; execute **R**



Theoretical guarantees

Let R satisfy (1) $\mathsf{R}(x) \geq 0$, and (2) $\mathsf{R}(x) = 0 \Leftrightarrow x \models \pi$, and x^* be a minimum point of R. Then

$$\pi \text{ satisfiable} \Leftrightarrow \mathsf{R}(x^*) = 0.$$

Threats

- Floating-point inaccuracy when calculating with R
- Sub-optimal x^*

Example

$$(SIN(x) == x) \wedge (x \geq 10^{-10})$$

$$\Downarrow$$

$$(SIN(x) - x)^2 + \begin{cases} 0 & \text{if } x \geq 10^{-10} \\ (x - 10^{-10})^2 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

$$x^* = 9.0 * 10^{-9} \text{ (can be others)}$$

XSat & results

- Developed the ME-based XSat tool
- Evaluated against MathSat and Z3
- Used SMT-Comp 2015 FP benchmarks
- Result summary
  - 100% consistent results
  - 700+X faster than MathSat
  - 800+X faster than Z3

Take-away: *Don’t be afraid of difficult problems, look at them from* ***new perspectives***, *even for “damned” problems!*

Generalizations

- Coverage-based testing of FP code
- Boundary value analysis
- FP exception detection
- Path divergence detection

ME in the long run

- Offers a new general analysis paradigm
- Complements existing approaches
  - Random concrete execution (CE)
  - Symbolic execution (SE)
  - Abstract execution (AE)

Additional Reflections

Pursue “**less-crowded**” directions

Much work on **applying** ML/DL/AI

but much less on

- ❑ **Formalizing** ML/DL/AI systems
- ❑ **Testing/verifying/explaining** them
- ❑ **Making** them more **usable** & **effective**

Pursue “**less-crowded**” directions

❑ **Intelligent programming assistants**

❑ **Compilers for developers** (not machines)

❑ **Knowledge capture & reuse**

❑ Anything **leading closer to key mission**

Many exciting

opportunities,

challenges,

uncertainties