AST
ADVANCED SOFTWARE TECHNOLOGIES

Randomized and Systematic Testing of Software

Zhendong Su

ETH Zurich

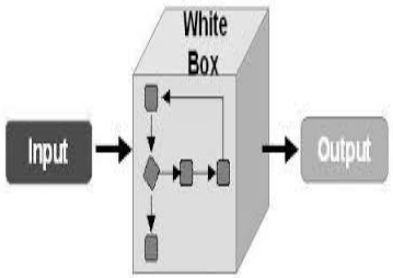
automated testing

- Oracle generation
  - How to define test oracles?
- Test input generation
  - How to synthesize test inputs?

Two instances

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

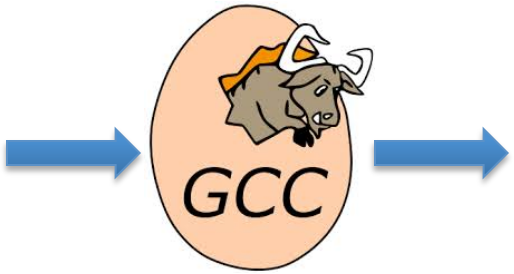
- **Analyze floating-point software**
  - Dynamic analysis



Validate Production Compilers

compilers

```
int main ()
{
  // do something
  …
  return 0;
}
```




```
…
call    puts
movl    $0, %eax
popq    %rbp
ret
```

Developers' **belief**:

Compilers are ***faithful*** translators

let’s reflect on this trust

How trustworthy are compilers?

How they impact security & reliability?

In 2014, GCC 4.9 miscompiled the Linux kernel!

"Ok, so I'm looking at the code generation and your compiler is pure and utter *.***

……

Adding Jakub to the cc, because gcc-4.9.0 seems to be terminally broken."

-- Linus Torvalds

Type #1 bugs

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

Type #2 “bugs”

gcc “bug” 8537

```
#include <string>
using std::string;
#include <memory>

// The specifics of this function are not important for
// demonstrating this bug.
const string getPasswordFromUser() const;

bool isPasswordCorrect() {
   bool isPasswordCorrect = false;
   string Password("password");

   if(Password == getPasswordFromUser()) {
      isPasswordCorrect = true;
   }

   // Removed from the optimized code although it secures
   // the code by wiping the password from memory
   memset(Password, 0, sizeof(Password));
   return isPasswordCorrect;
}
```

debatable …

From: "Joseph D. Wagner" <wagnerjd@prodigy.net>
To: <fw@gcc.gnu.org>, gcc-bugs@gcc.gnu.org, <gcc-prs@gcc.gnu.org>,
nobody@gcc.gnu.org, wagnerjd@prodigy.net,<gcc-gnats@gcc.gnu.org>
Cc:
Subject: RE: optimization/8537: Optimizer Removes Code Necessary for Security
Date: Sun, 17 Nov 2002 08:59:53 -0600

Direct quote from:
http://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Bug-Criteria.html

"If the compiler produces valid assembly code that does not correctly execute the input source code, that is a compiler bug."

So to all you naysayers out there who claim this is a programming error or poor coding, YES, IT IS A BUG!

Type #3 “bugs”

how about this one?

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;

if (buf + len >= buf_end)
    return; // len too large

if (buf + len < buf)
    return; //overflow, buf+len wrapped around

// write to buf[0..len-1]
...
```

even more debatable …

- Pointer overflow is **undefined behavior**
  - Compiler assumes code has no undefined behavior
  - Thus compiler assumes: **buf + len** cannot overflow
  - Then compiler infers: **if (buf + len < buf) ⇒ if (0)**
- But this is a **security threat**
  - Use a large **len** to trigger **buffer overflow**

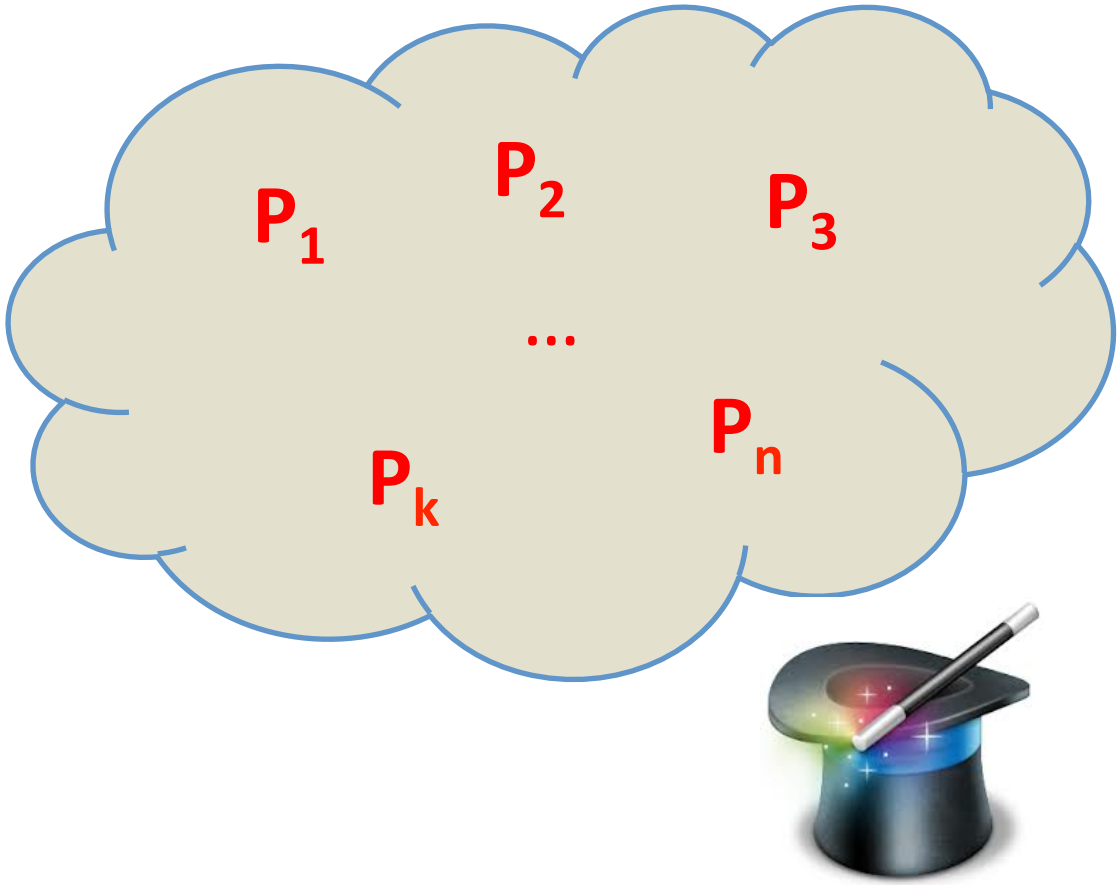
Type #1 bugs: EMI Testing

How to find 1600+ GCC/LLVM bugs in 5 years?

real

vision

$P \equiv$



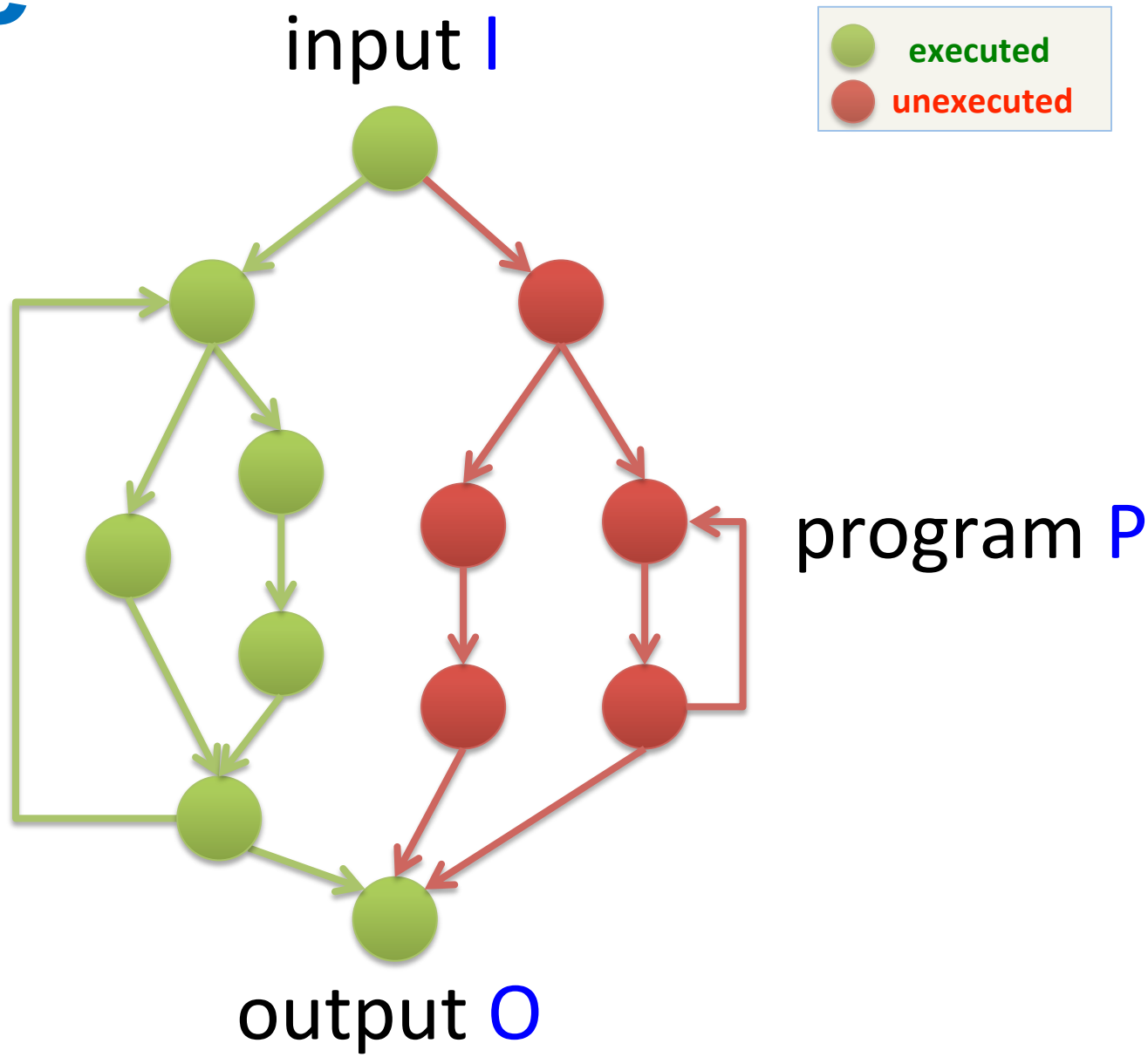
key challenges

- Generation
  - How to generate **different**, yet **equivalent** tests?
- Validation
  - How to check that tests are **indeed equivalent**?
- Both are long-standing hard issues

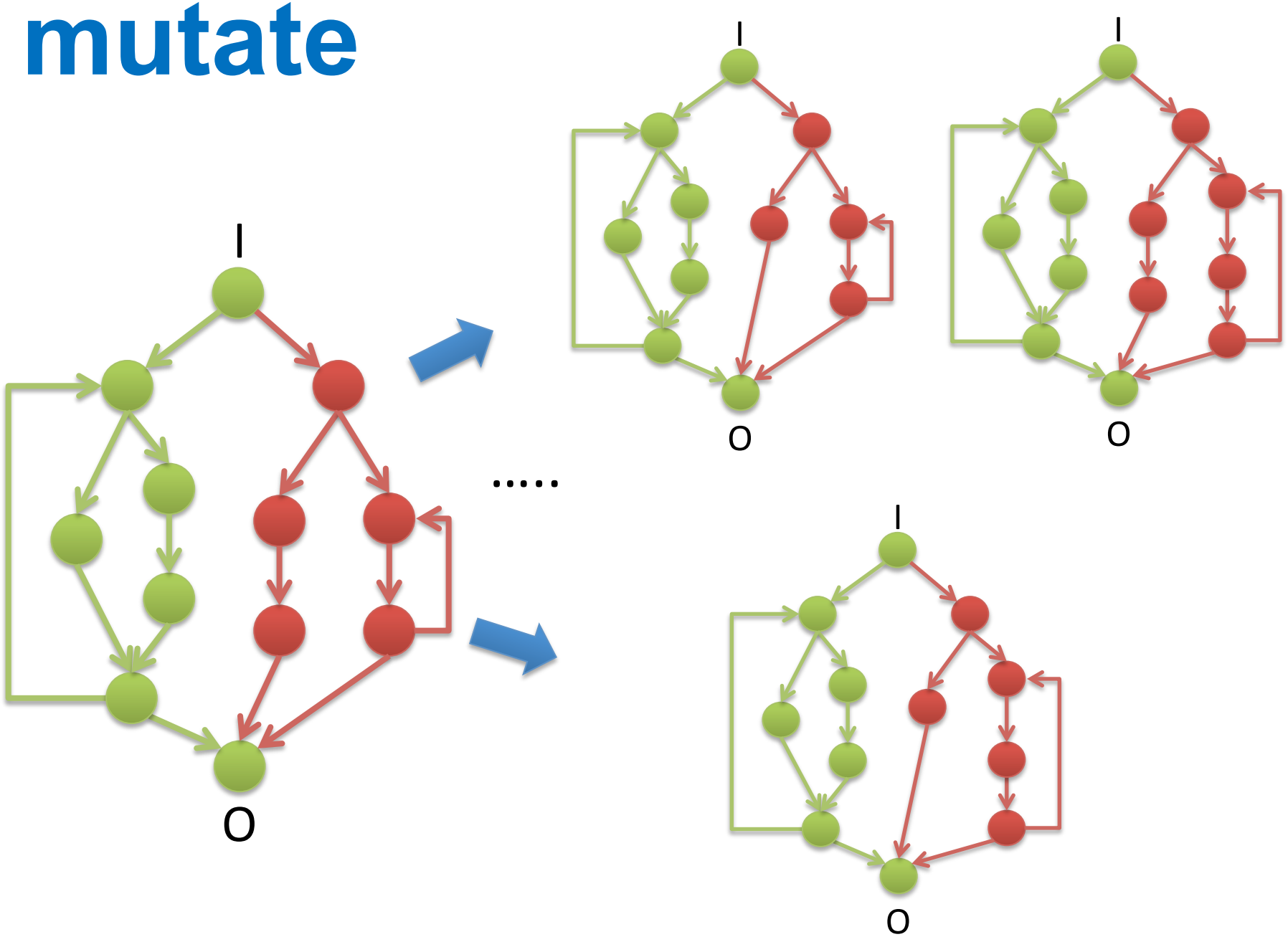
equiv. modulo inputs

- 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**

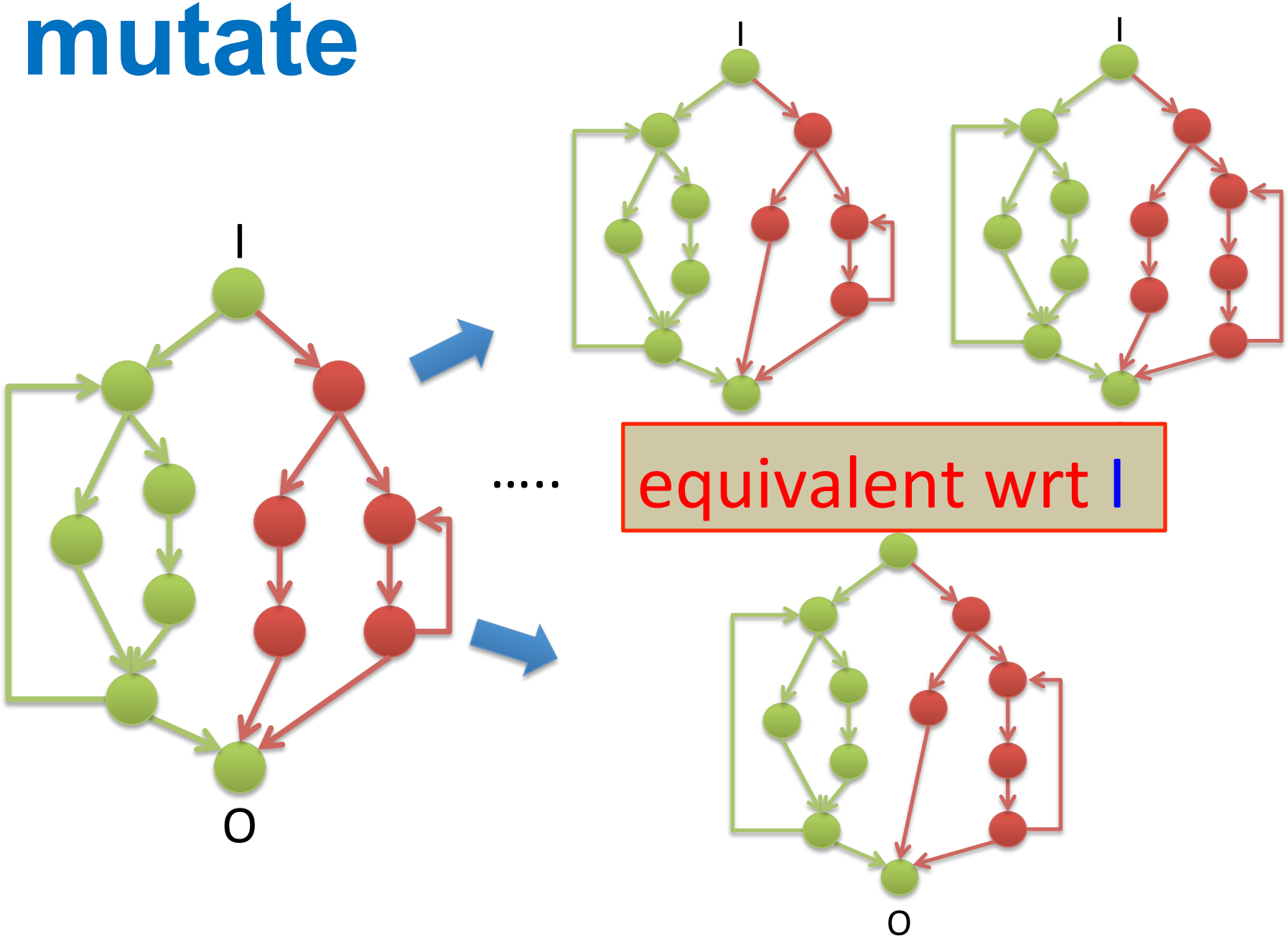
profile



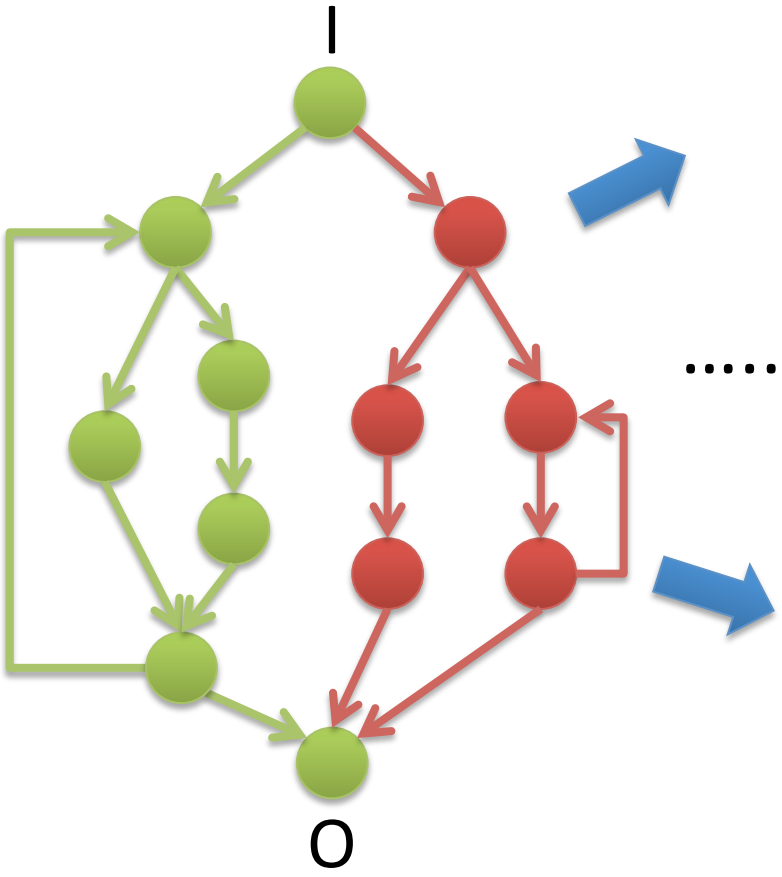
mutate

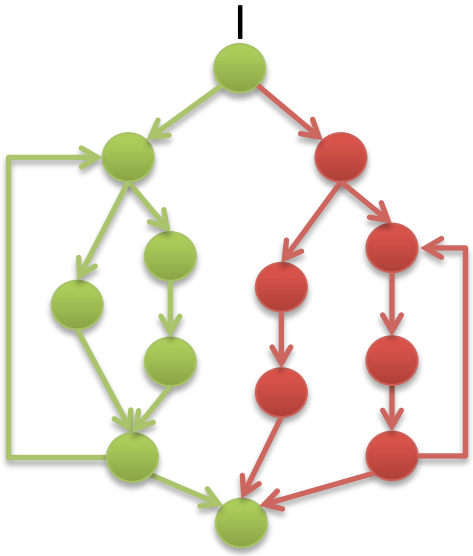


mutate



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

```
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)
```

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

```
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

```
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

```
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

```
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

```
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

```
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;
}
```

LIM

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

gcc bug autopsy

```
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

```
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

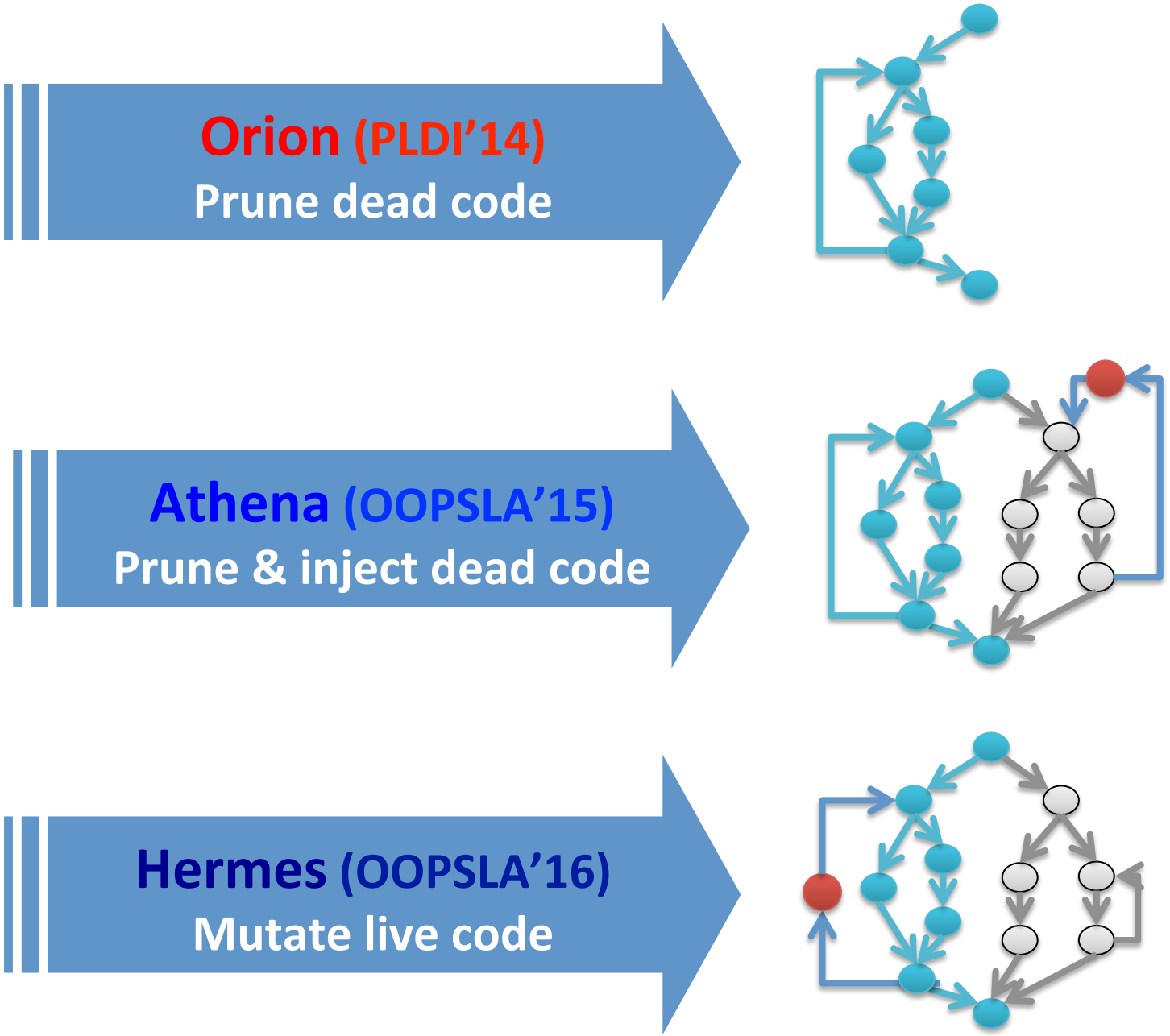
- First and simplest EMI realization
- Targeting C compilers
  - Randomly **prune** unexecuted code
  - Very **effective**

bug counts (till 03/2014)

	GCC	LLVM	TOTAL
Reported	111	84	**195**
Marked Duplicate	28	7	**35**
Confirmed	79	68	**147**
Fixed	56	54	**110**

bug types

	GCC	LLVM	TOTAL
Wrong code	46	49	**95**
Crash	23	10	**33**
Performance	10	9	**19**


Orion (PLDI'14)
Prune dead code
Athena (OOPSLA'15)
Prune & inject dead code
Hermes (OOPSLA'16)
Mutate live code

Athena: gcc bug 61383

Seed Program P

```
int a, c, d, e = 1, f;
int fn1 () {
 int h;
 for (; d < 1; d = e) {
   h = (f == 0) ? 0 : 1 % f;
   if (f < 1) c = 0;
   else c = 1;
 }
}
int main () {
 fn1 ();
 return 0;
}
```

Athena

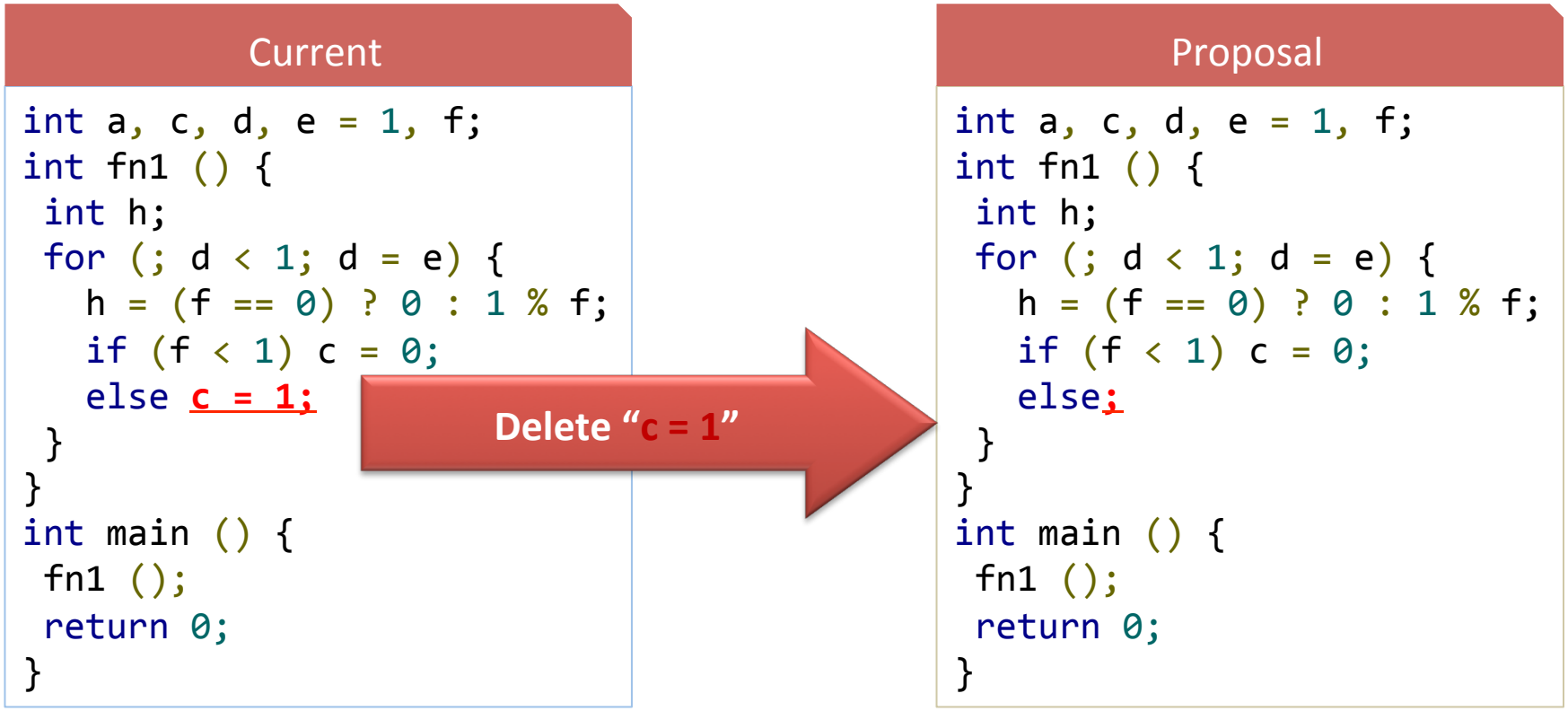
Bug-triggering Variant

```
int a, c, d, e = 1, f;
int fn1 () {
 int h;
 for (; d < 1; d = e) {
   h = (f == 0) ? 0 : 1 % f;
   if (f < 1) c = 0;
   else if (h) break;
 }
}
int main () {
 fn1 ();
 return 0;
}
```

```
$ gcc -O0 test.c ; ./a.out
$ gcc -O2 test.c ; ./a.out
Floating point exception (core dumped)
```

Current

```
int a, c, d, e = 1, f;
int fn1 () {
 int h;
 for (; d < 1; d = e) {
   h = (f == 0) ? 0 : 1 % f;
   if (f < 1) c = 0;
   else c = 1;
 }
}
int main () {
 fn1 ();
 return 0;
}
```


Current
int a, c, d, e = 1, f;
int fn1 () {
int h;
for (; d < 1; d = e) {
h = (f == 0) ? 0 : 1 % f;
if (f < 1) c = 0;
else c = 1;
}
}
int main () {
fn1 ();
return 0;
}
Delete “c = 1”
Proposal
int a, c, d, e = 1, f;
int fn1 () {
int h;
for (; d < 1; d = e) {
h = (f == 0) ? 0 : 1 % f;
if (f < 1) c = 0;
else;
}
}
int main () {
fn1 ();
return 0;
}

Current

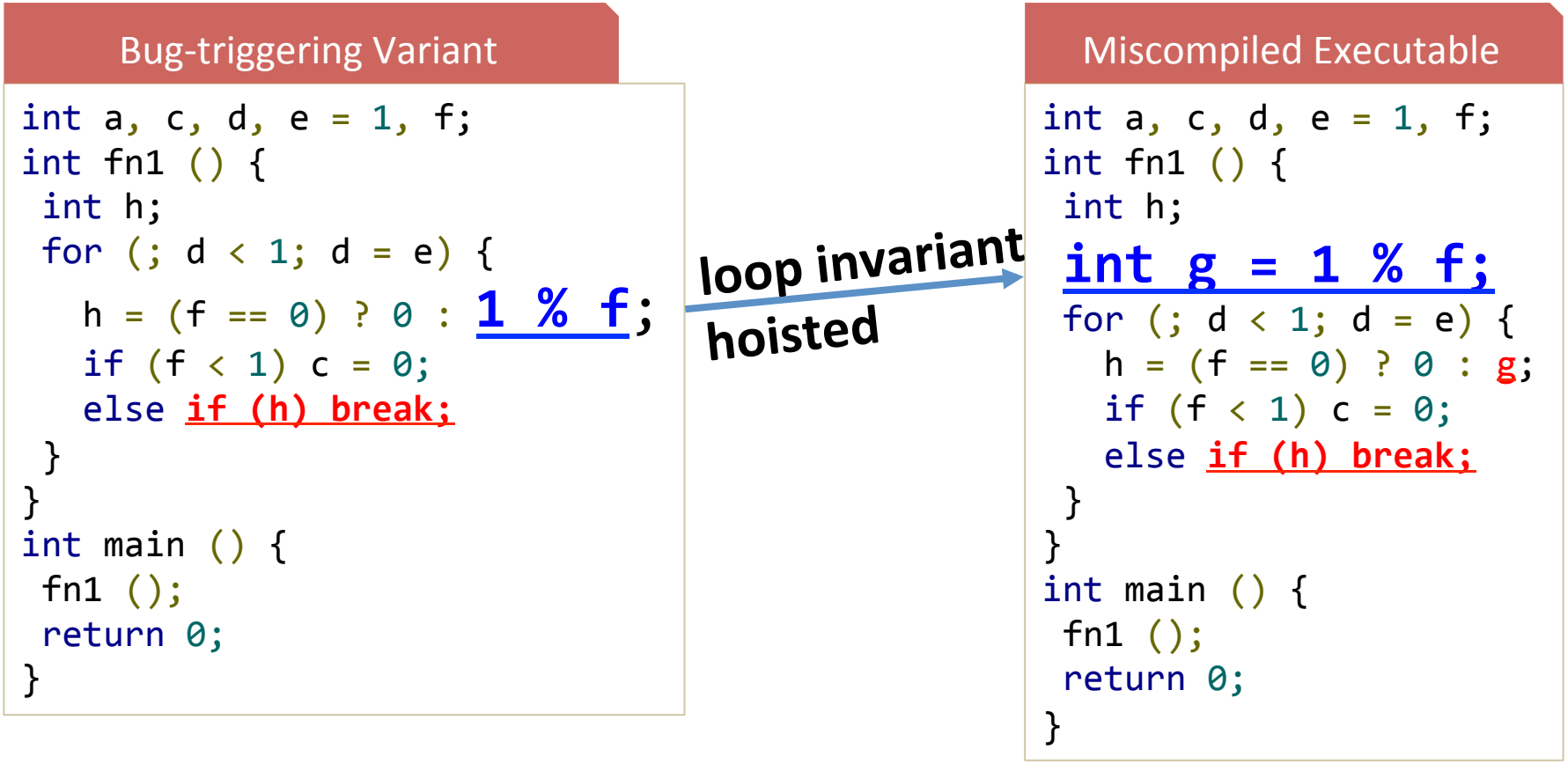
```
int a, c, d, e = 1, f;
int fn1 () {
 int h;
 for (; d < 1; d = e) {
   h = (f == 0) ? 0 : 1 % f;
   if (f < 1) c = 0;
   else;
 }
}
int main () {
 fn1 ();
 return 0;
}
```

Insert the statement below

Proposal

```
int a, c, d, e = 1, f;
int fn1 () {
 int h;
 for (; d < 1; d = e) {
   h = (f == 0) ? 0 : 1 % f;
   if (f < 1) c = 0;
   else if (h) break;
 }
}
int main () {
 fn1 ();
 return 0;
}
```

Context	Statement
1. Requires loop	`if (i)` `  break;`
2. `int i;`	


Bug-triggering Variant
int a, c, d, e = 1, f;
int fn1 () {
int h;
for (; d < 1; d = e) {
h = (f == 0) ? 0 : 1 % f;
if (f < 1) c = 0;
else if (h) break;
}
}
int main () {
fn1 ();
return 0;
}
loop invariant hoisted
Miscompiled Executable
int a, c, d, e = 1, f;
int fn1 () {
int h;
int g = 1 % f;
for (; d < 1; d = e) {
h = (f == 0) ? 0 : g;
if (f < 1) c = 0;
else if (h) break;
}
}
int main () {
fn1 ();
return 0;
}

Hermes

- **Profile** and **record variable values**
- Synthesize code snippets
  - Ensure **no undefined behavior**
  - Ensure **EMI property locally**
    - Maintain same program state at entry & exit

Hermes: llvm 26266

Seed Program **P**

```
char a;
int b, c = 9, d, e;

void fn1() {
 unsigned f = 1;
 int g = 8, h = 5;
 for (; a != 6; a--) {
   int *i = &h, *j;
   for (;;) {
// b=0,c=9,e=0,f=1,g=8,h=5
     if (d <= 8) break;
     *i = 0;
     for (; *j <= 0;);
   }
 }
}
int main() {fn1(); return 0;}
```

EMI Variant

```
   int *i = &h, *j;
   for (;;) {
// b=0,c=9,e=0,f=1,g=8,h=5
int backup_g = e, backup_f = ~1;
if (g && h) {
  backup_g = g;
  backup_f = f;
  f = -(~(c && b)|-~(e*~backup_f));
  if (c < f) abort();
}
g = backup_g;
f = backup_f;
     if (d <= 8) break;
     *i = 0;
     for (; *j <= 0;);
   }
```

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

EMI Variant

```
    int *i = &h, *j;
  for (;;) {
// b=0,c=9,e=0,f=1,g=8,h=5
int backup_g = e, backup_f = ~1;
if (g && h) {
  backup_g = g;
  backup_f = f;
  f = -(~(c && b)|-~(e*~backup_f));
  if (c < f) abort();
}
g = backup_g;
f = backup_f;
    if (d <= 8) break;
    *i = 0;
    for (; *j <= 0;);
  }
```

Clang (mistakenly) deems this predicate always **true**

bug counts

	GCC	LLVM	TOTAL
Reported	841	781	**1622**
Fixed	612	419	**1031**

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

LLVM 3.9 & 4.0 Release Notes

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

GCC's list of contributors

https://gcc.gnu.org/onlinedocs//gcc/Contributors.html

"Zhendong Su . . . for reporting numerous bugs"
"Chengnian Sun . . . for reporting numerous bugs"
"Qirun Zhang . . . for reporting numerous bugs"

program enumeration

- **Vision**: Bounded compiler verification
- **Goal**: Program enumeration
  - **Exhaustive** (all small test programs)
  - **Practical**
- **Idea**:
  - Context-free grammar to token sequences
  - Token sequences to programs (**SPE**)

SPE (PLDI'17)

```
a := 10;
b := 1;
while(a) do
  a := a - b;
```

(a) Program P

```
□ := 10;
□ := 1;
while(□) do
  □ := □ - □;
```

(b) Skeleton $\mathbb{P}$

```
b := 10;
a := 1;
while(b) do
  b := b - a;
```

(c) Program P_1

```
a := 10;
b := 1;
while(b) do
  b := a - b;
```

(d) Program P_2

example 1: wrong code

```
1 int a = 0;
```

Marek Polacek 2016-02-25 09:13:03 UTC Comment 1

Hm, this really seems like a wrong code. Started a looooong time ago, before r104500.

Richard Biener 2016-02-25 10:22:43 UTC Comment 2

We have a duplicate for this - basically (most) of alias analysis doesn't handle aliases (hah).

```
11   return a; // Bug: the program exits with 1
12 }
```

example 2: gcc crash

```
1 struct s { char c[1]; };
2 struct s a, b, c;
3 int d; int e;
4
5 void bar (void)
6 {
7 //e ? (d=       c).c : (e==0 ? b : c).c;
8   e ? (d=     b : c).c : (d==0 ? b : c).c;
9 }
```

Release blocking

example 3: clang crash

```
1  int a;
2  double b, *c;
3
4  void fn1(int p1) {
5    for (;; p1--) {
6      a = p1;
7      for (; p1 >= a; a--)
8        b = c[p1];
9    }
10 }
```

general, effective

- Found bugs in **many compilers**
  - Scala, Rust, Go, …
- Very **simple to apply**
- **No need to reduce**

how about CompCert?

```
// test.c
#include <stdio.h>
int main(){
  int i = '\214';
  printf("%d\n", i);
  return 0;
}
```

```
$ ccomp-2.4 test.c; ./a.out
$ -116
$ ccomp-2.0 test.c; ./a.out
$ 140
```

how about CompCert?

```
// test.c
volatile long long a;
unsigned b;
int main () {
  a = b;
  return 0;
}
```

```
$ ccomp-2.6 test.c
Fatal error: exception File "ia32/Asmexpand.ml", line 191, …
…
$
```

how about CompCert?

```
// test.c
#include <stdio.h>
int main () {
  int t = printf ("0\n");
  printf ("%d\n", t);
  return 0;
}
```

```
$ ccomp-2.6 -interp -quiet test.c
0
0
$ ccomp-2.7 -interp -quiet test.c
0
2
$
```

how about CompCert?

```
// test.c
int a, b, c, d, e, f, g;
void fn1 () {
 int h, i, j;
 if (g) {
    g = 1;
  L1: if (1) ;
  }
 short k = ~j;
 int l = 1 / (h & e & ~d + ((k & ~h) - ((1 | i) & (a | c)))), m = ~~j / (~h | d + a);
 j = l & m | h;
 if (j) ;
 j = k;
 int n = ~i | f, o = ~b - 1 / -n * ~i, p = f;
 goto L1;
}
int main () {
if (a) fn1 ();
 return 0;
}
```

```
$ ccomp-2.7 test.c
...
Fatal error: exception File "backend/Regalloc.ml", line 741, ...
...
$
```

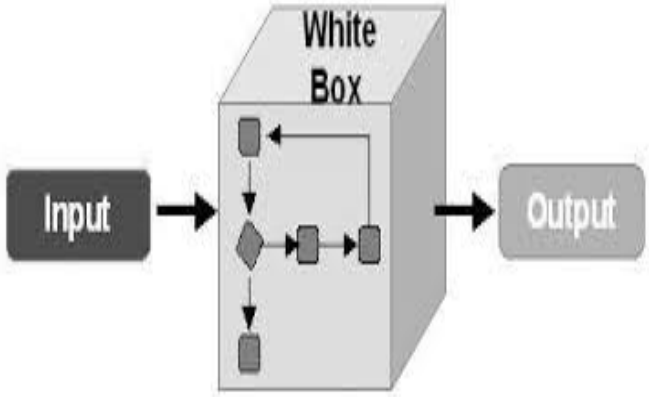
Relationship to MT

- Instance of metamorphic testing (MT)
  - Given input **i**
  - Constructed input **j**
  - Property **P** holds on **i**, **j**
- EMI is the most successful MT instance

a few key problems

- How to select **good** seed inputs?
- How to decide **where** & **how** to mutate?
- How to generate inputs that **satisfy** a property **P**?
- How to find **missed optimization** opportunities?
- How to **synthesize optimizations**?

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, …

```
1  double foo(double x){
2    if (x<=1.0)
3      x++;
4
5    y = x*x;
6    if (y<=4.0)
7      x--;
8    return x;
9  }
```

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

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 π

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

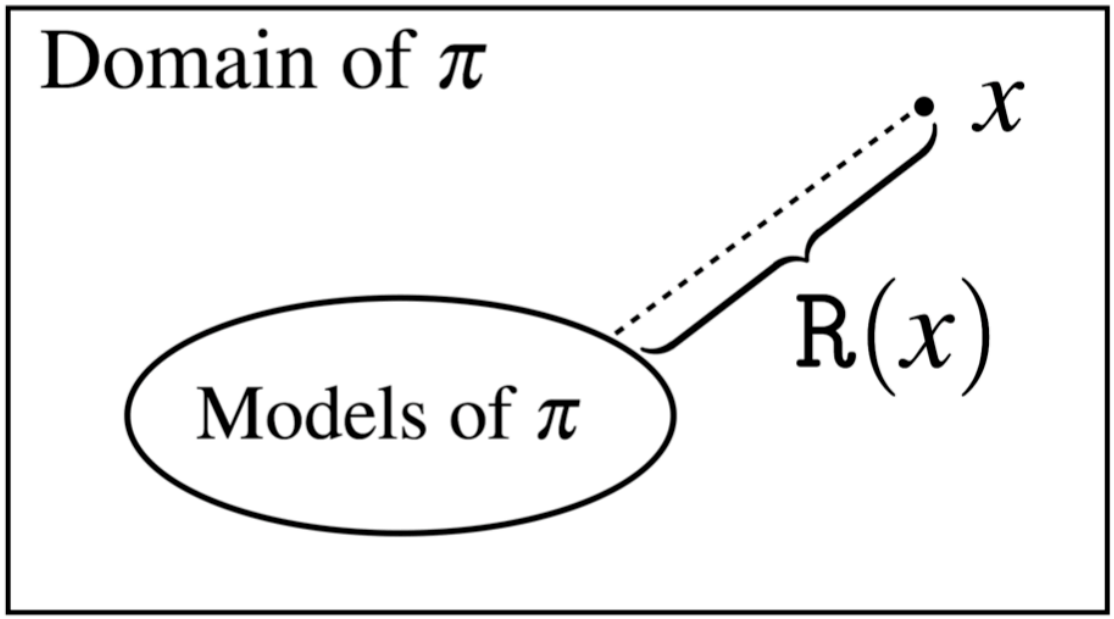
- Satisfiable if x is floating-point

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

- Unsatisfiable if x is real

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

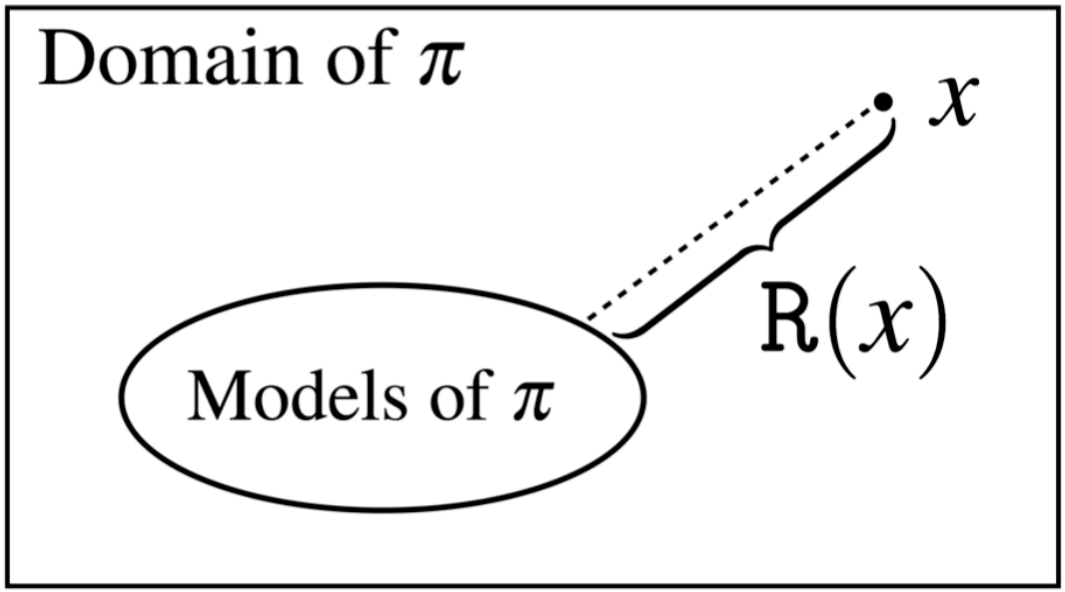
Step 1



Simulate π with a floating-point program R

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

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?

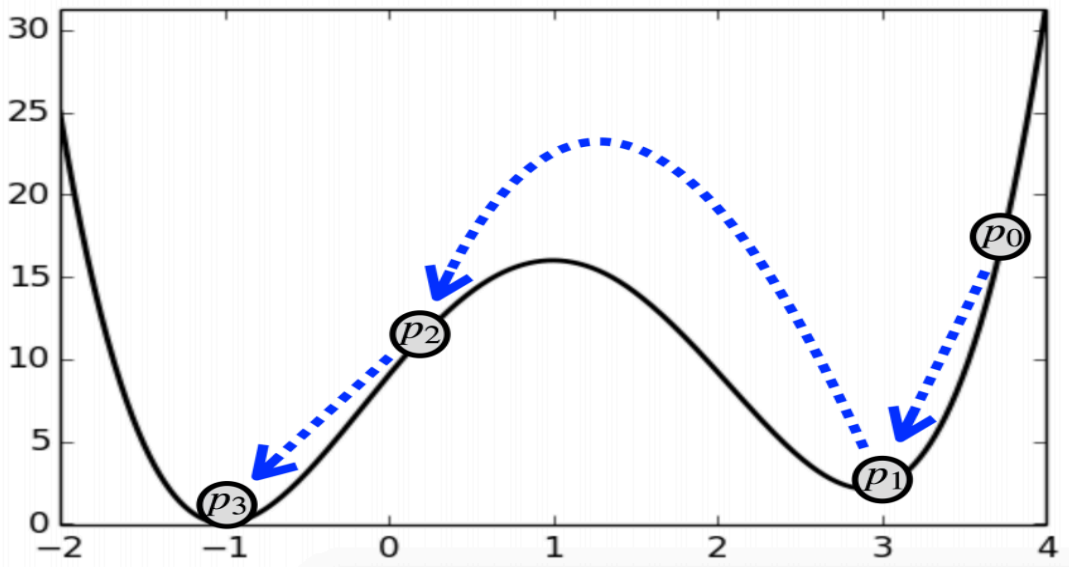
Constraint π	Program R
$x == y$	$(x-y)^2$
$x \leq y$	$x \leq y \ ? \ 0 : (x-y)^2$
$\pi_1 \wedge \pi_2$	$\mathsf{R}_1 + \mathsf{R}_2$
$\pi_1 \vee \pi_2$	$\mathsf{R}_1 * \mathsf{R}_2$

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

Generalizations

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

Coverage-based testing

Goal

To generate test inputs to cover all branches of a program like this:

- pointer operations: &, *
- type casting: (int*), (unsigned)
- bit operations ^, &, >>
- floating-point comparison

```
double __ieee754_fmod(double x, double y){
  ...
  Zero[] = {0.0, -0.0,};
  hx = *(1+(int*)&x);
  lx = *(int*)&x;
  hy = *(1+(int*)&y);
  ly = *(int*)&y;
  sx = hx&0x80000000;
  hx ^=sx;
  hy &= 0x7fffffff;

  if((hy|ly)==0||(hx>=0x7ff00000)||
     ((hy|((ly|-ly)>>31))>0x7ff00000))
    return (x*y)/(x*y);
  if(hx<=hy) {
    if((hx<hy)||(lx<ly)) return x;
    if(lx==ly)
      return Zero[(unsigned)sx>>31];
 }


  if(hx<0x00100000) {
   if(hx==0) {
```

State-of-the-art & Challenges

Symbolic execution

- Path explosion
- Constraint solving

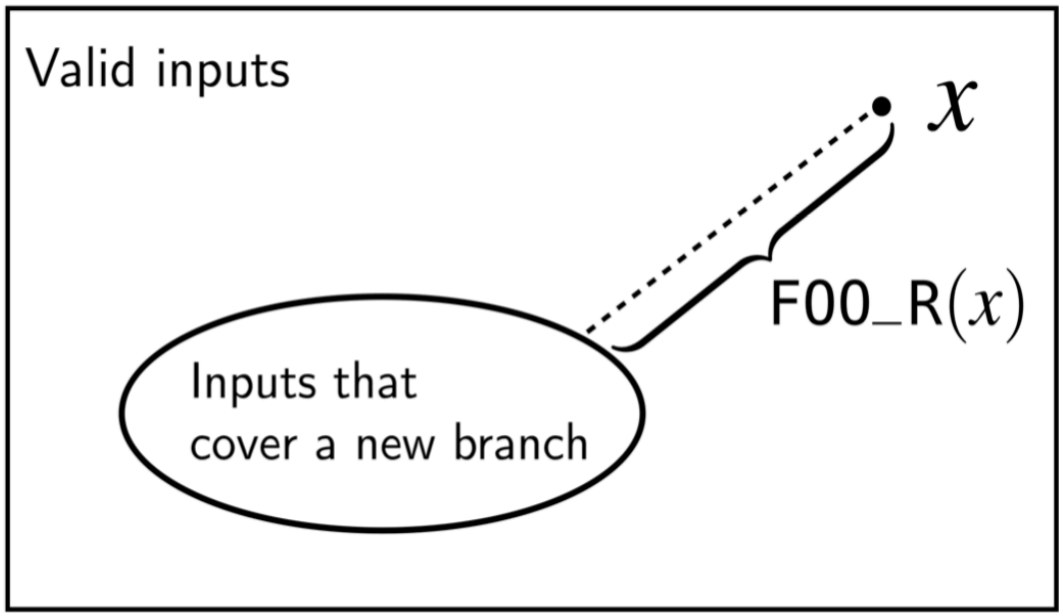
Search-based testing

- Fitness function
- Search strategies

Our approach

- No path issues
- No need to solve constraints
- Effective for FP programs

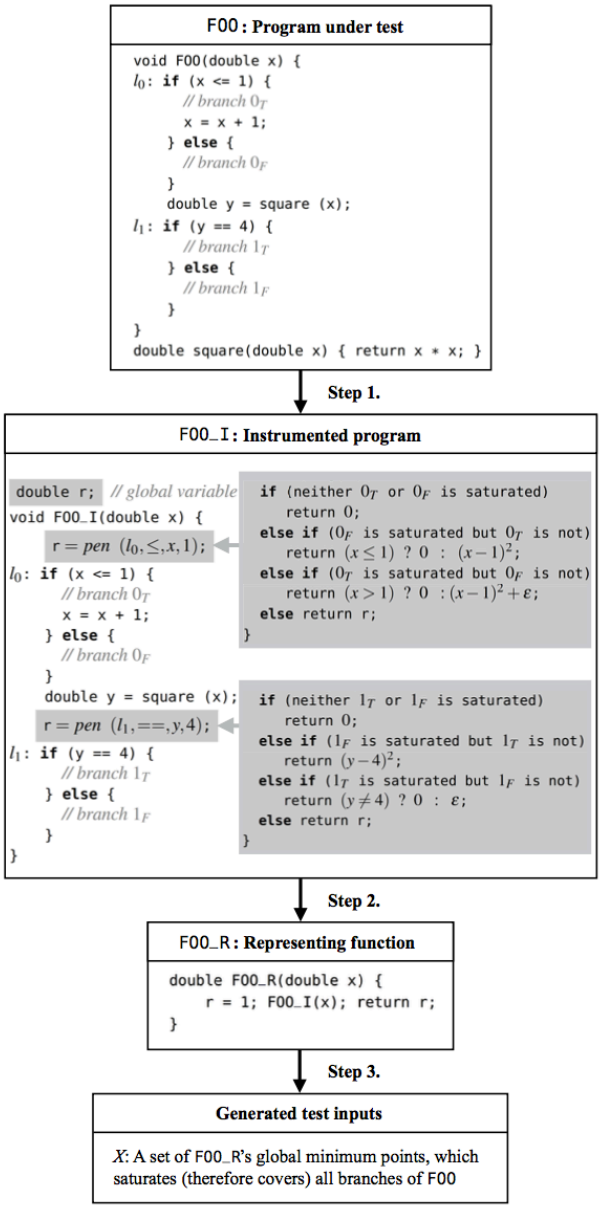
Our approach



Step 1: Derive a program FOO_R from FOO *s.t.*

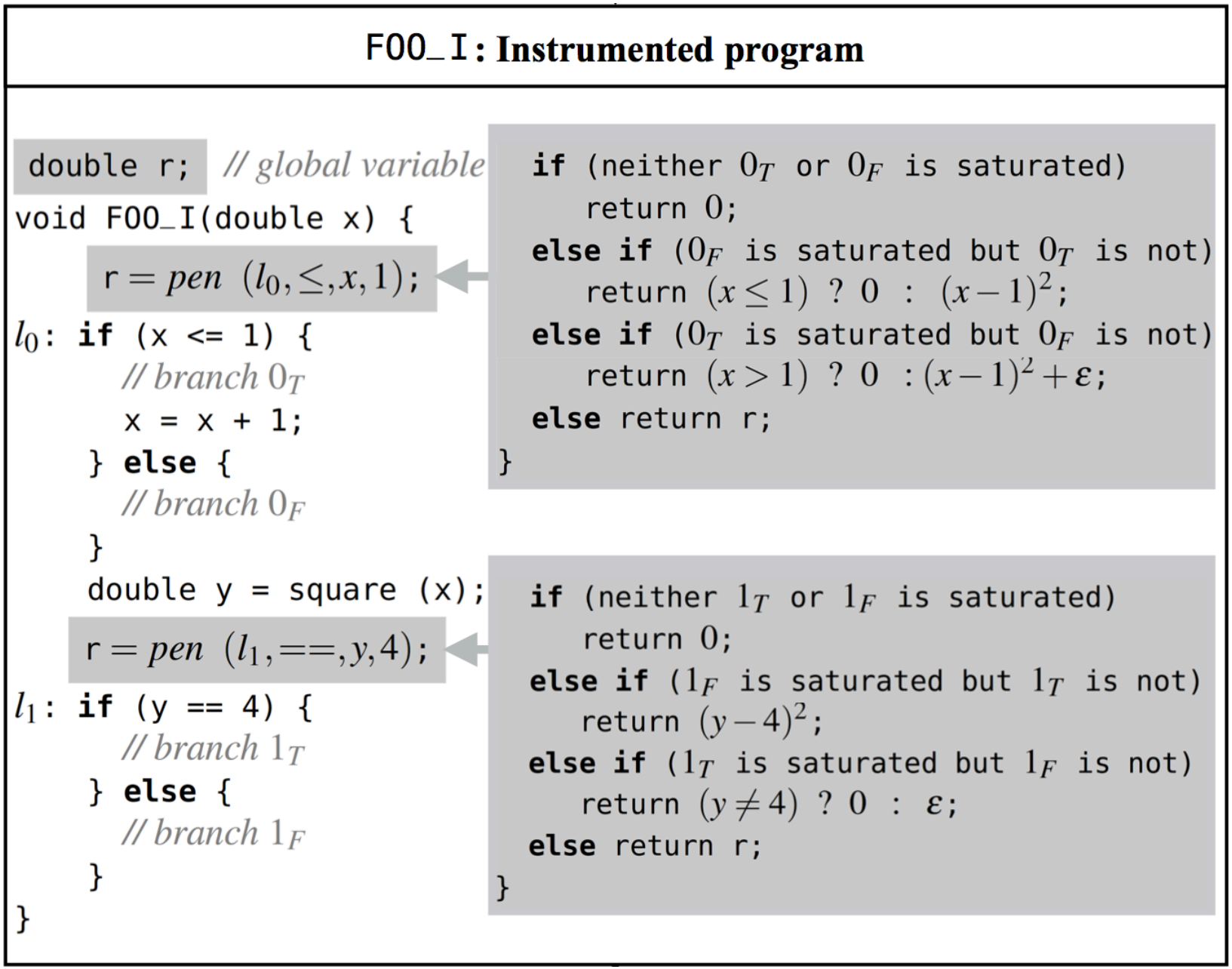
- FOO_R$(x) \geq 0$ for all x, and
- FOO_R$(x) = 0 \Leftrightarrow x$ covers a new branch

Step 2: Repeatedly minimize FOO_R until > 0



FOO : Program under test

```
void FOO(double x) {
l0: if (x <= 1) {
      // branch 0T
      x = x + 1;
    } else {
      // branch 0F
    }
    double y = square (x);
l1: if (y == 4) {
      // branch 1T
    } else {
      // branch 1F
    }
}
double square(double x) { return x * x; }
```


FOO_I : Instrumented program
double r; // global variable
void FOO_I(double x) {
r = pen (l0, ≤, x, 1);
l0: if (x <= 1) {
// branch 0T
x = x + 1;
} else {
// branch 0F
}
double y = square (x);
r = pen (l1, ==, y, 4);
l1: if (y == 4) {
// branch 1T
} else {
// branch 1F
}
}
if (neither 0T or 0F is saturated)
return 0;
else if (0F is saturated but 0T is not)
return (x ≤ 1) ? 0 : (x − 1)^2;
else if (0T is saturated but 0F is not)
return (x > 1) ? 0 : (x − 1)^2 + ε;
else return r;
}
if (neither 1T or 1F is saturated)
return 0;
else if (1F is saturated but 1T is not)
return (y − 4)^2;
else if (1T is saturated but 1F is not)
return (y ≠ 4) ? 0 : ε;
else return r;
}

FOO_I: Instrumented program

```
double r;  // global variable
void FOO_I(double x) {
    r = pen (l0, ≤, x, 1);
l0: if (x <= 1) {
      // branch 0T
      x = x + 1;
    } else {
      // branch 0F
    }
    double y = square (x);
    r = pen (l1, ==, y, 4);
l1: if (y == 4) {
      // branch 1T
    } else {
      // branch 1F
    }
}
```

```
if (neither 0T or 0F is saturated)
    return 0;
else if (0F is saturated but 0T is not)
    return (x ≤ 1) ? 0 : (x−1)^2;
else if (0T is saturated but 0F is not)
    return (x > 1) ? 0 : (x−1)^2 + ε;
else return r;
}
```

```
if (neither 1T or 1F is saturated)
    return 0;
else if (1F is saturated but 1T is not)
    return (y−4)^2;
else if (1T is saturated but 1F is not)
    return (y ≠ 4) ? 0 : ε;
else return r;
}
```

FOO_R: Representing function

```
double FOO_R(double x) {
    r = 1; FOO_I(x); return r;
}
```

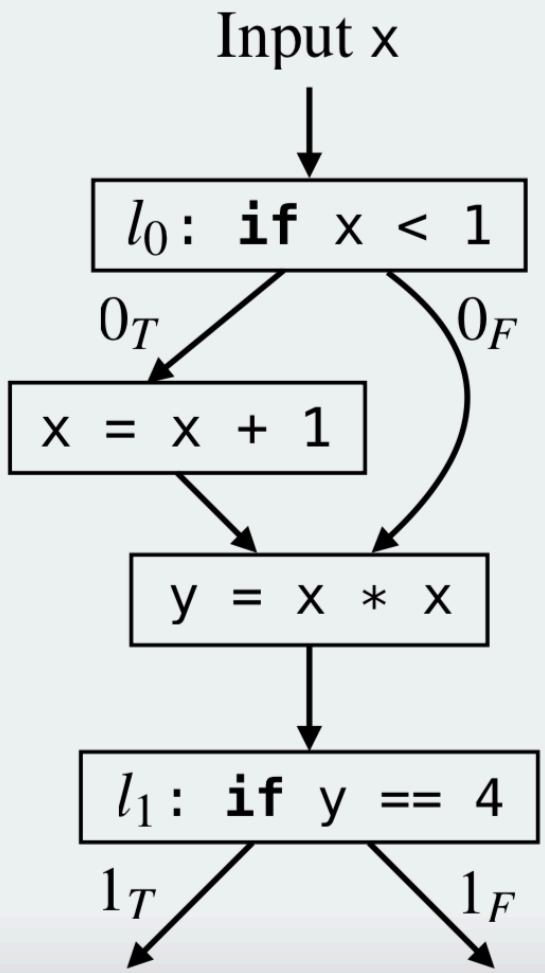
Generated test inputs

X: A set of FOO_R's global minimum points, which saturates (therefore covers) all branches of FOO

Example

Generate an input set to cover $\{0_T, 0_F, 1_T, 1_F\}$

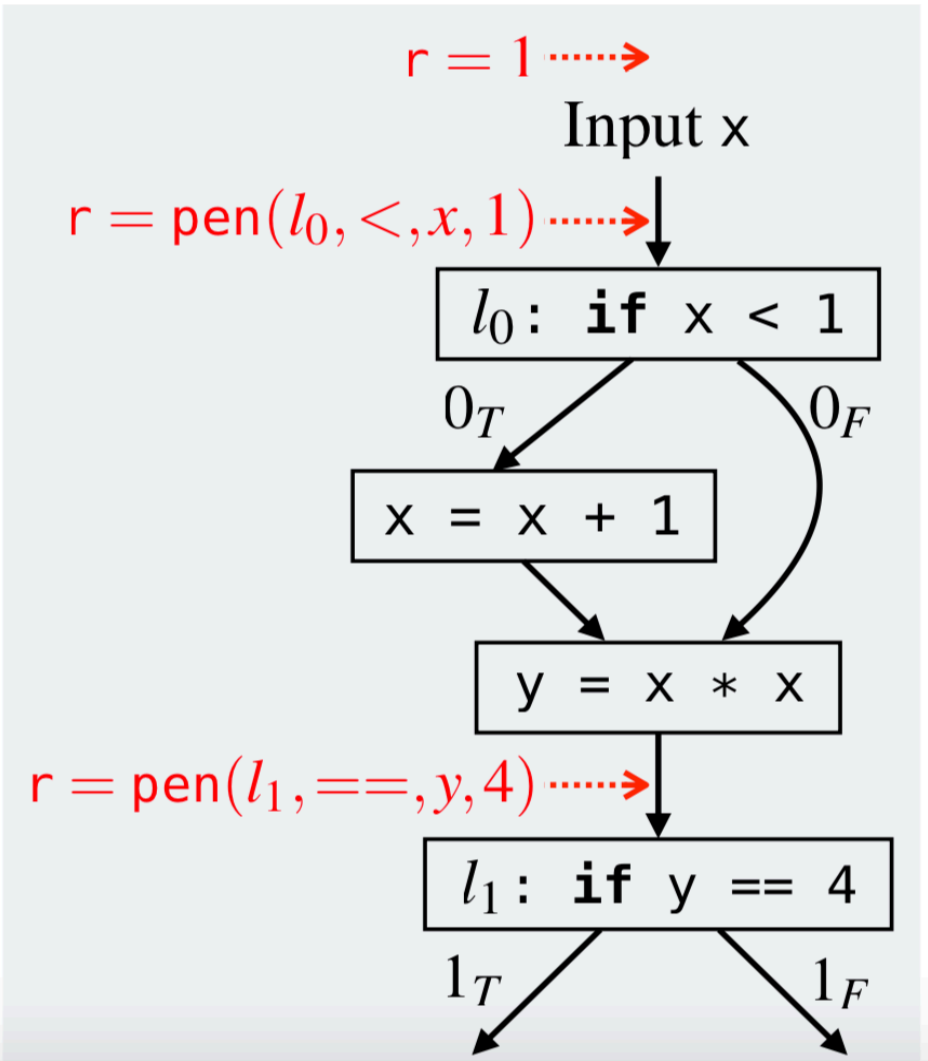
```
void FOO (double x){
l0: if (x < 1)
        x++;
    double y = x * x;
l1: if (y == 4)
        ...
}
```



Step 1: Construct FOO_R

covered at l_i	$pen(l_i, op, a, b)$
$\emptyset$	0
$\{i_F\}$	$R_{a\ op\ b}$
$\{i_T\}$	$R_{\neg(a\ op\ b)}$
$\{i_T, i_F\}$	r

- r: global variable
- FOO_R : $x \rightarrow$ r
- $R_{a\ op\ b}$:Branch distance



Branch distance $R_{a\ op\ b}$

A helper function to quantify how far a and b are from attaining branch $a\ op\ b$.

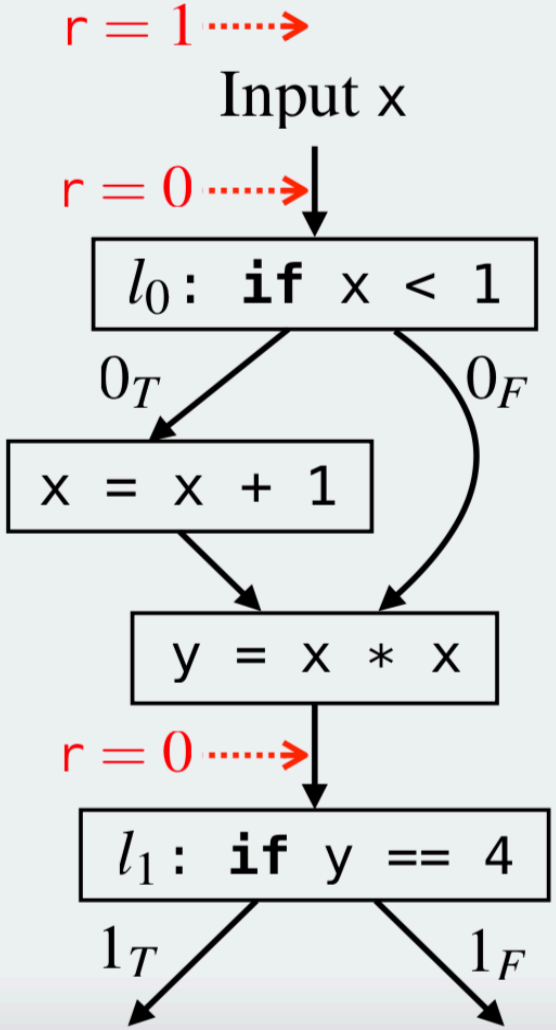
$R_{a==b}$ defined as $(a-b)^2$

$R_{a\geq b}$ defined as $(a \geq b)\ ?\ 0 : (a-b)^2$

Step 2: Minimize FOO_R

covered at l_i	$pen(l_i, op, a, b)$
$\emptyset$	0
$\{i_F\}$	$R_{a\ op\ b}$
$\{i_T\}$	$R_{\neg(a\ op\ b)}$
$\{i_T, i_F\}$	r

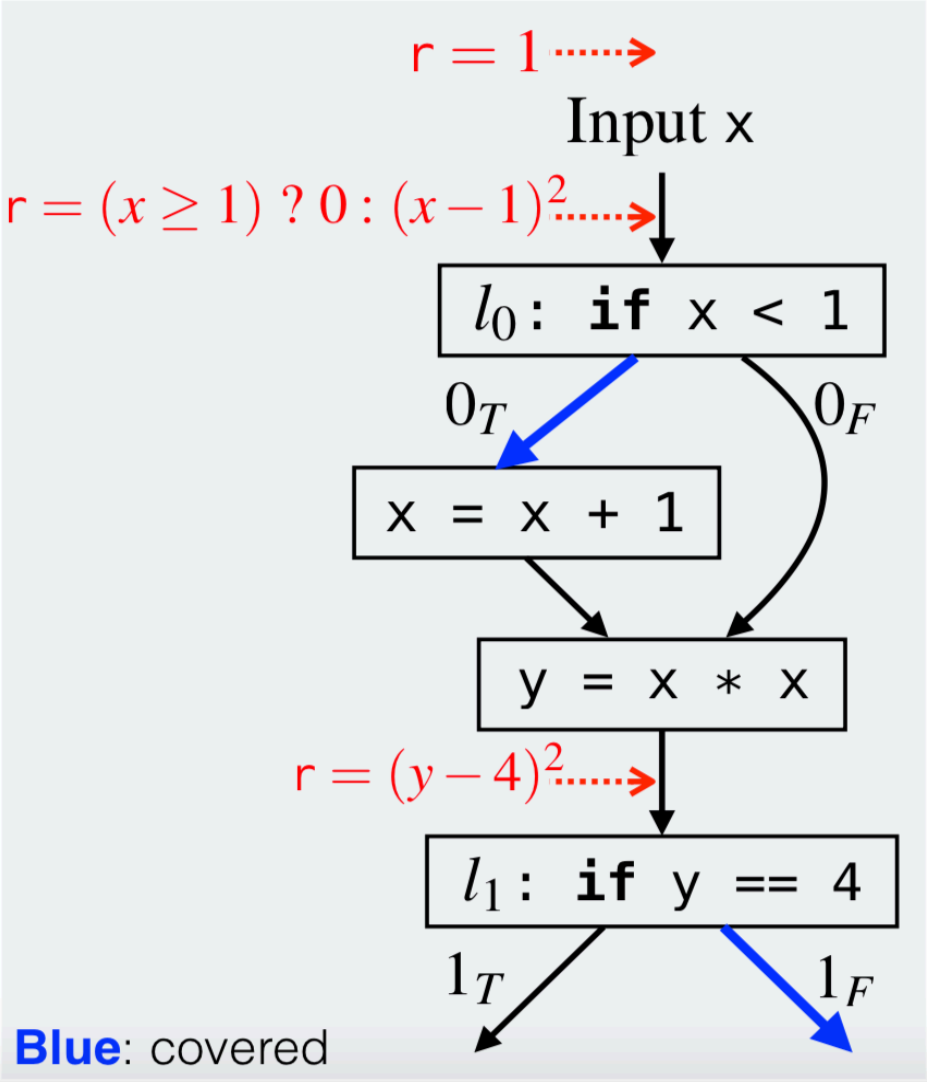
- No branch is covered
- Any input is a minimum point
- Assume $x^* = 0.7$



Step 2: Minimize FOO_R

covered at l_i	$pen(l_i, op, a, b)$
$\emptyset$	0
$\{i_F\}$	$R_{a\ op\ b}$
$\{i_T\}$	$R_{\neg(a\ op\ b)}$
$\{i_T, i_F\}$	r

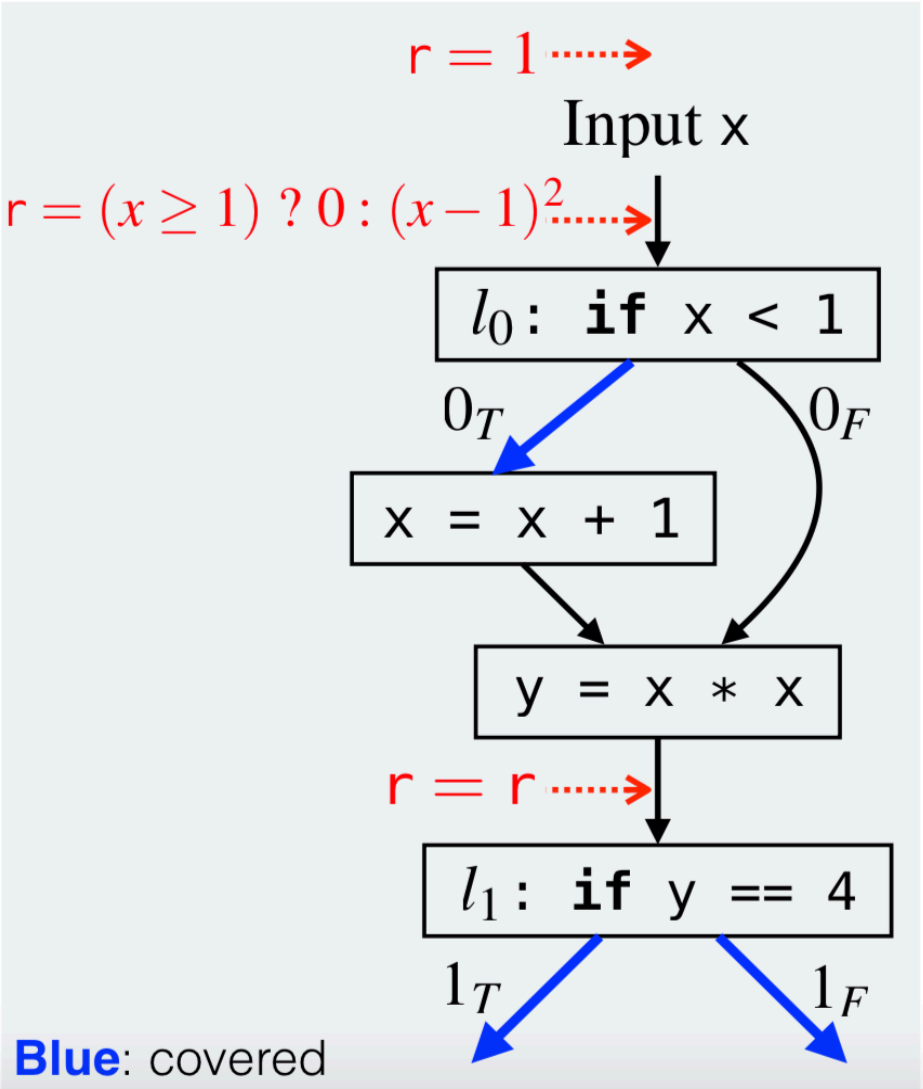
- $1_F, 0_T$ are covered
- FOO_R attains minimum at -3 or 2
- Assume $x^* = -3$



Step 2: Minimize FOO_R

covered at l_i	$pen(l_i, op, a, b)$
$\emptyset$	0
$\{i_F\}$	$R_{a\ op\ b}$
$\{i_T\}$	$R_{\neg(a\ op\ b)}$
$\{i_T, i_F\}$	r

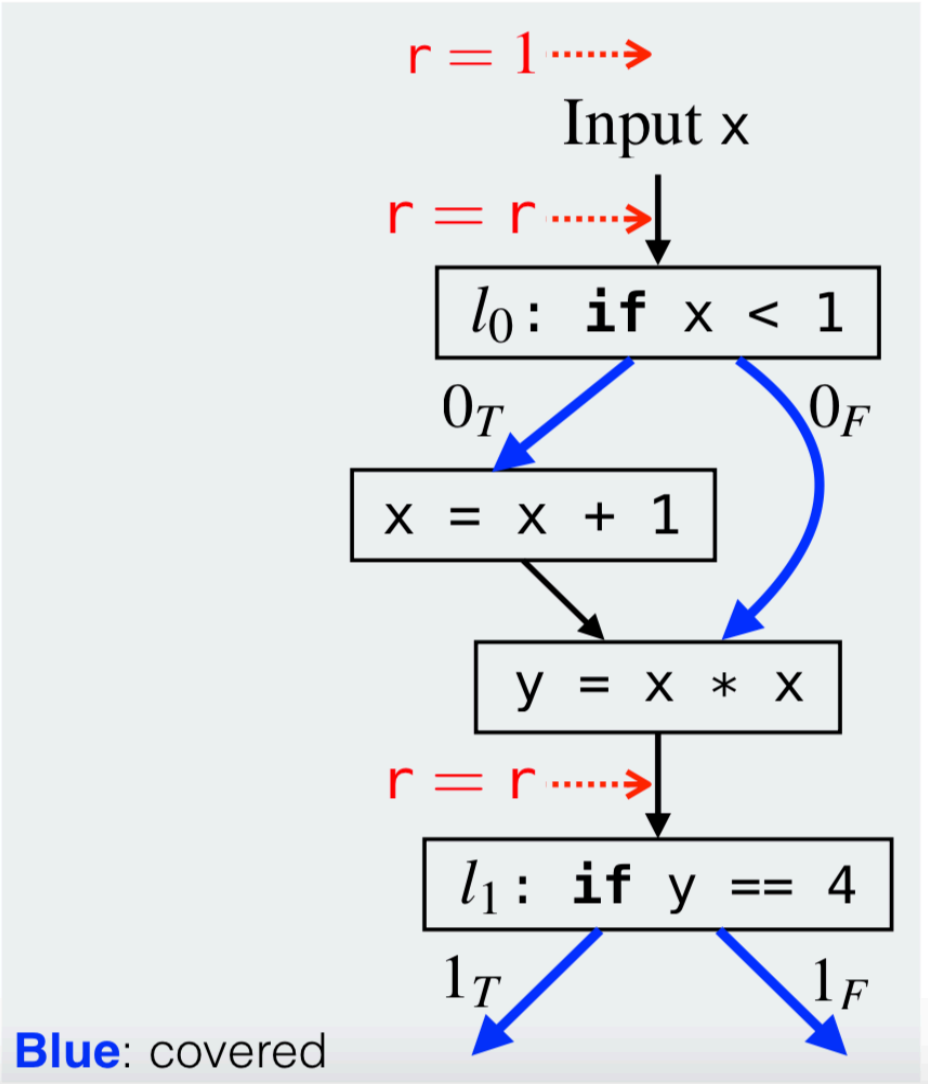
- $1_F, 1_T, 0_T$ are covered
- FOO_R attains minimum at ≥ 1
- Assume $x^* = 5.1$



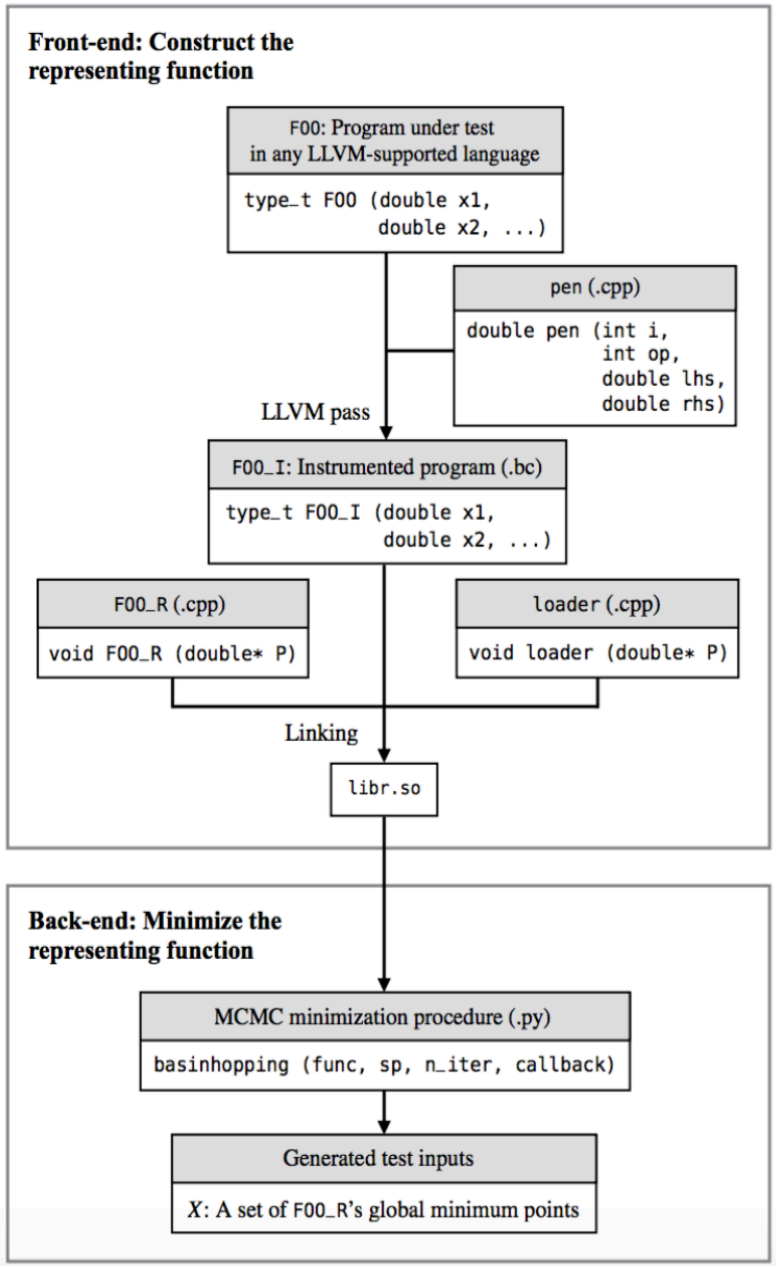
Step 2: Minimize FOO_R

covered at l_i	$pen(l_i, op, a, b)$
$\emptyset$	0
$\{i_F\}$	$R_{a\ op\ b}$
$\{i_T\}$	$R_{\neg(a\ op\ b)}$
$\{i_T, i_F\}$	r

- All branches are covered
- $\forall x, \mathsf{FOO\_R}(x) = 1$
- Termination



Our implementation CoverMe



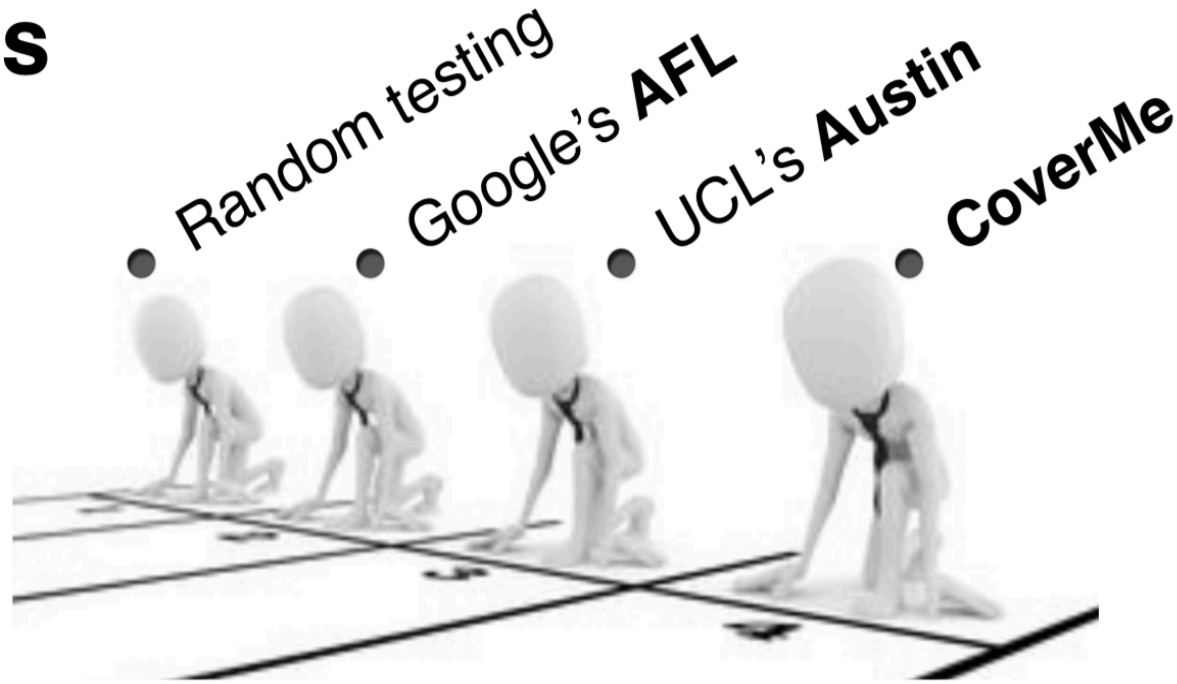
- **Front-end**: Program transformation in LLVM
- **Back-end**: Basinhopping

Experiments

Benchmarks: **Fdlibm**

- Sun's math library
- Reference for Java SE 8's math library
- Used in Matlab, JavaScript and Android
- Heavy on branches (max=114, avg=23)

Results



CoverMe covers

- ≈ 90% branches in 7 seconds
- ≈ 18% more branches than AFL with 1/10 time
- ≈ 40% more branches than Austin with

  speedups of several orders of magnitudes

ME in the long run

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

Thank you!

- Feedback always welcomed
- Bachelor's thesis topics available
  - Programming languages & compilers
  - Software engineering
  - Software security
  - Machine learning & AI
  - Education technologies (EdTech)