Numerical Program Analysis via Mathematical Execution

Zhendong Su

ETH Zurich

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, …

```c
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

$(\mathbf{p}, \phi)$

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 π

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

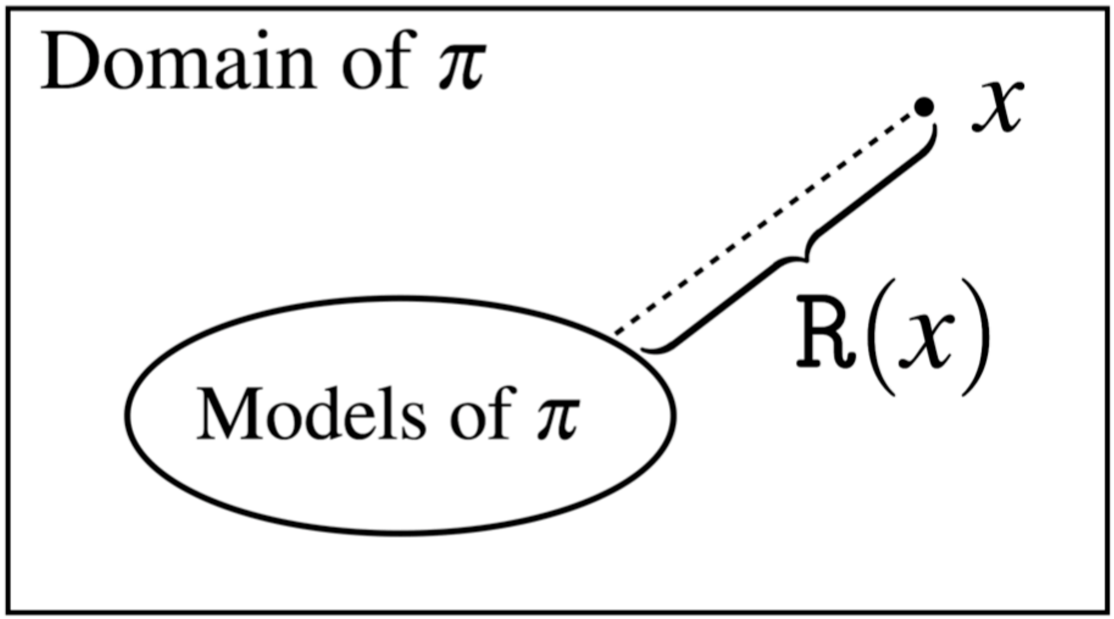
- Satisfiable if x is floating-point

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

- Unsatisfiable if x is real

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

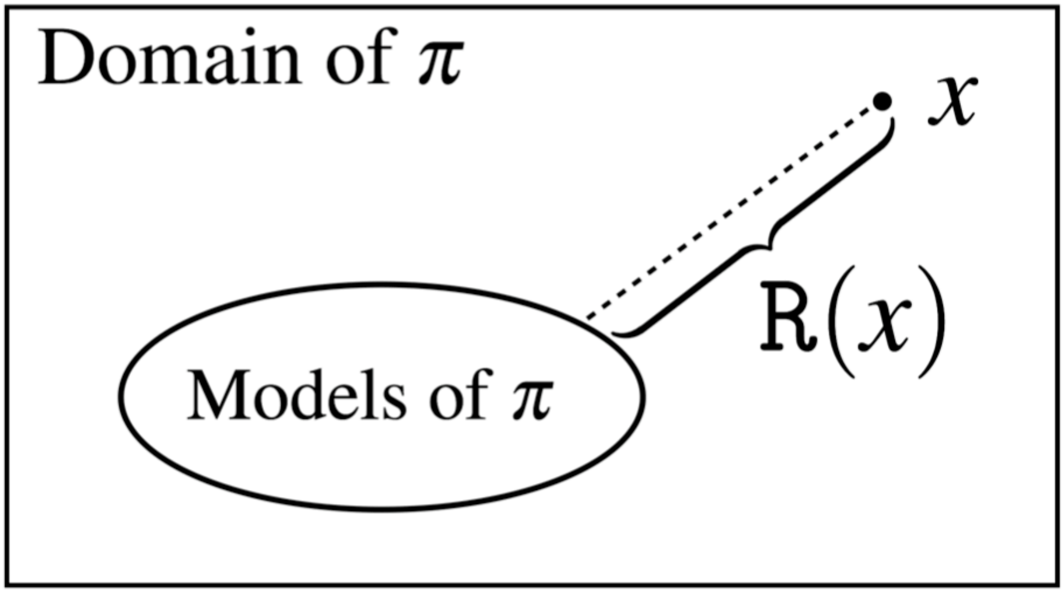
Step 1



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?

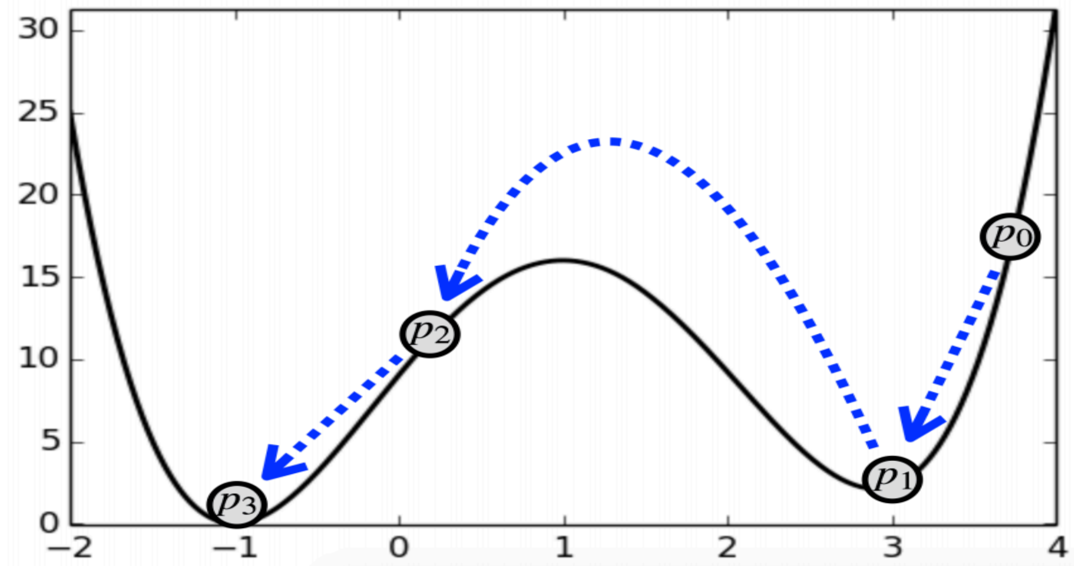
Constraint π	Program R
$x == y$	$(x-y)^2$
$x \leq y$	$x \leq y \; ? \; 0 : (x-y)^2$
$\pi_1 \wedge \pi_2$	$\mathsf{R}_1 + \mathsf{R}_2$
$\pi_1 \vee \pi_2$	$\mathsf{R}_1 * \mathsf{R}_2$

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

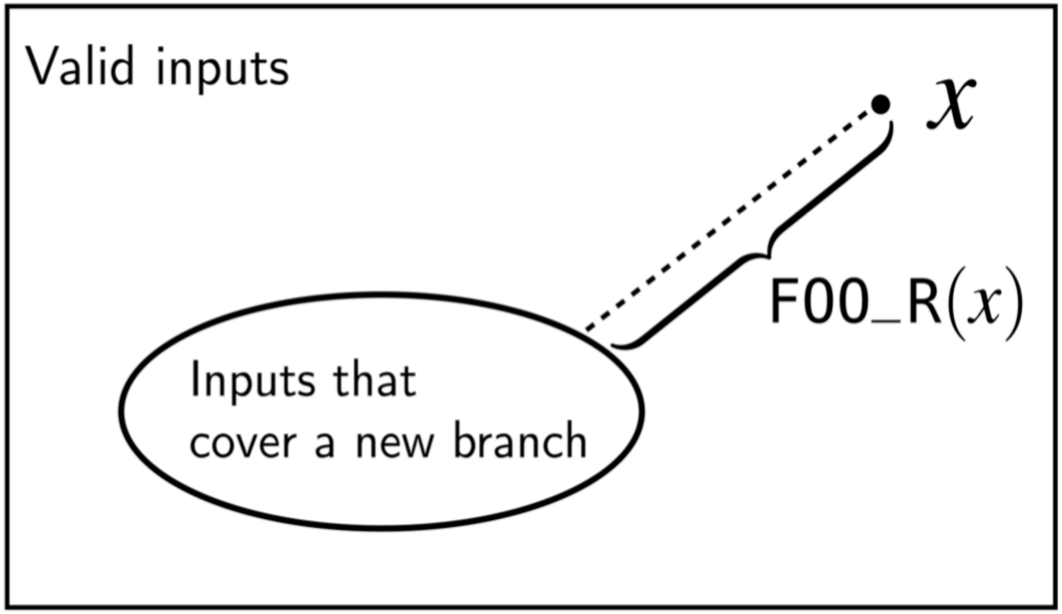
Search-based testing

- Fitness function
- Search strategies

Our approach

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

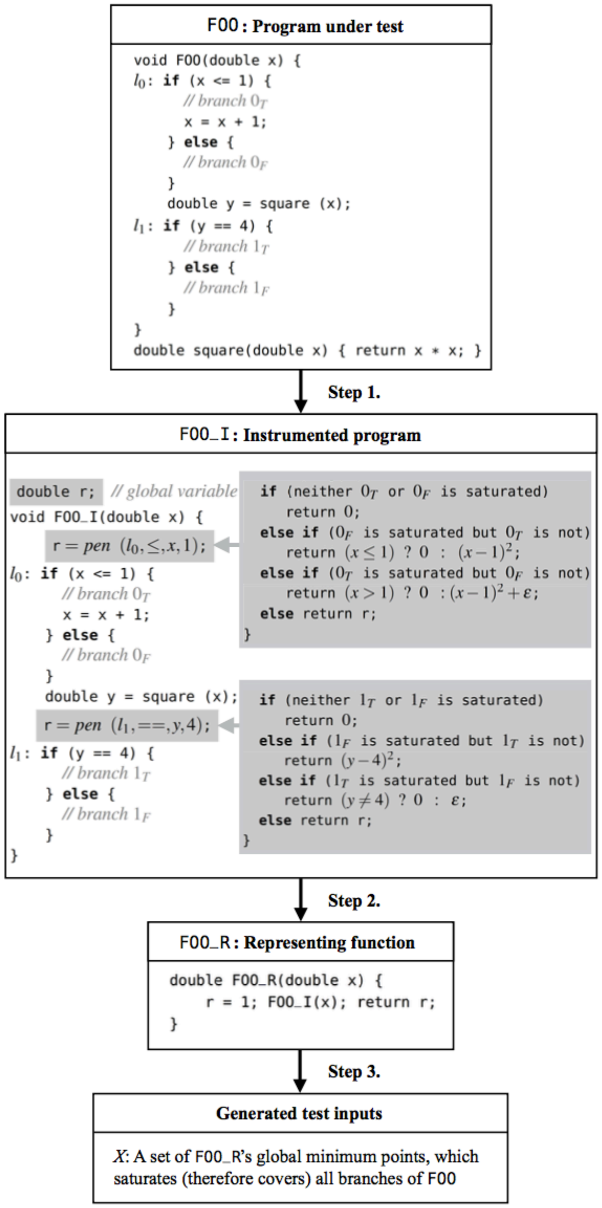
Our approach



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 (l_0, ≤, x, 1);
l_0: if (x <= 1) {
      // branch 0_T
      x = x + 1;
    } else {
      // branch 0_F
    }
    double y = square (x);
    r = pen (l_1, ==, y, 4);
l_1: if (y == 4) {
      // branch 1_T
    } else {
      // branch 1_F
    }
}
```

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

```
  if (neither 1_T or 1_F is saturated)
     return 0;
  else if (1_F is saturated but 1_T is not)
     return (y − 4)^2;
  else if (1_T is saturated but 1_F 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
    }
}
```

$pen(l_0, \leq, x, 1)$:

```
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;
}
```

$pen(l_1, ==, y, 4)$:

```
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;
}
```

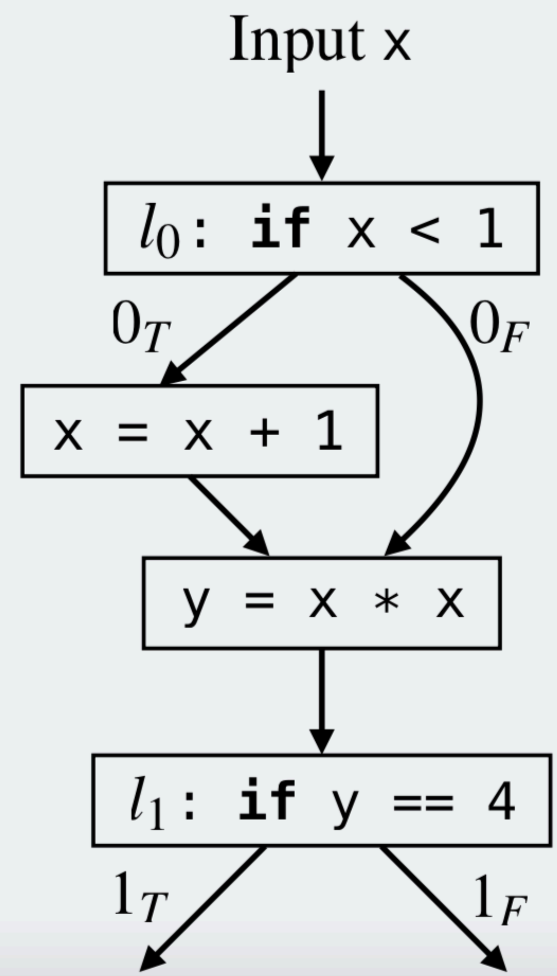
Generated test inputs

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 $\{0_T, 0_F, 1_T, 1_F\}$

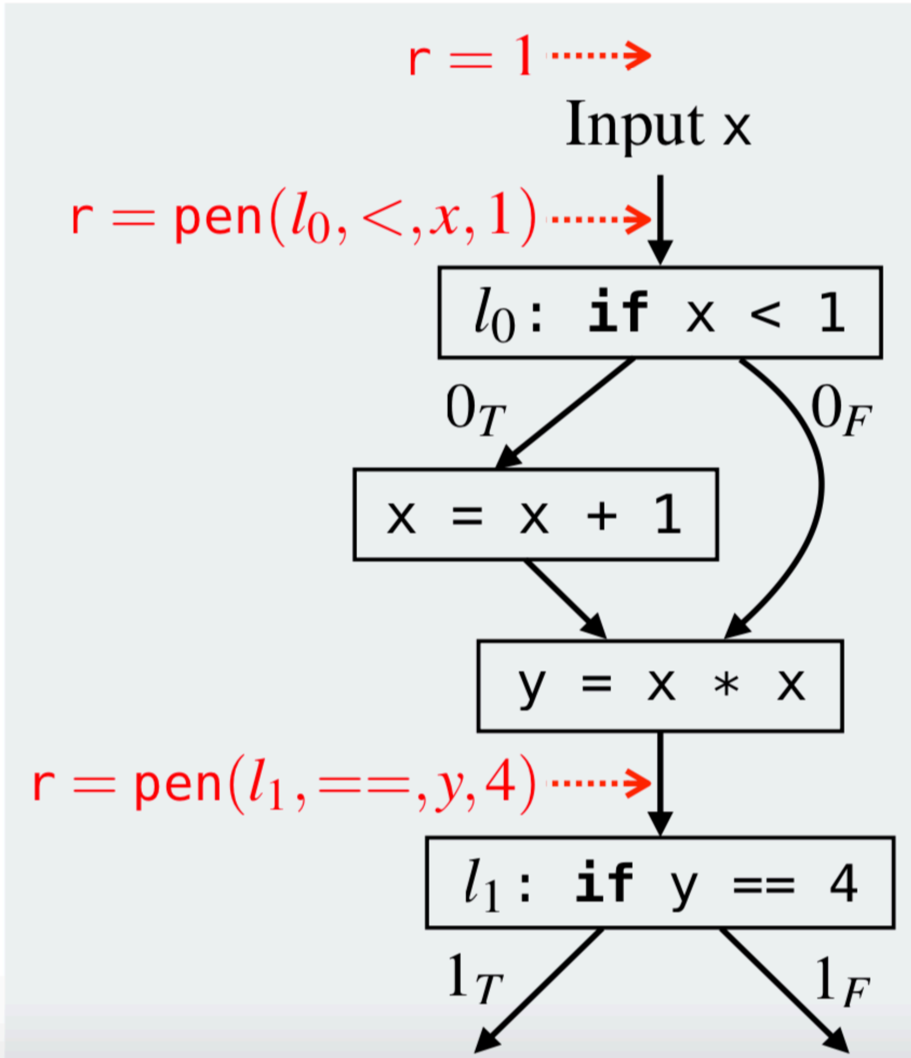
```
void FOO (double x){
l0: if (x < 1)
        x++;
    double y = x * x;
l1: if (y == 4)
        ...
}
```



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 $R_{a\ op\ b}$

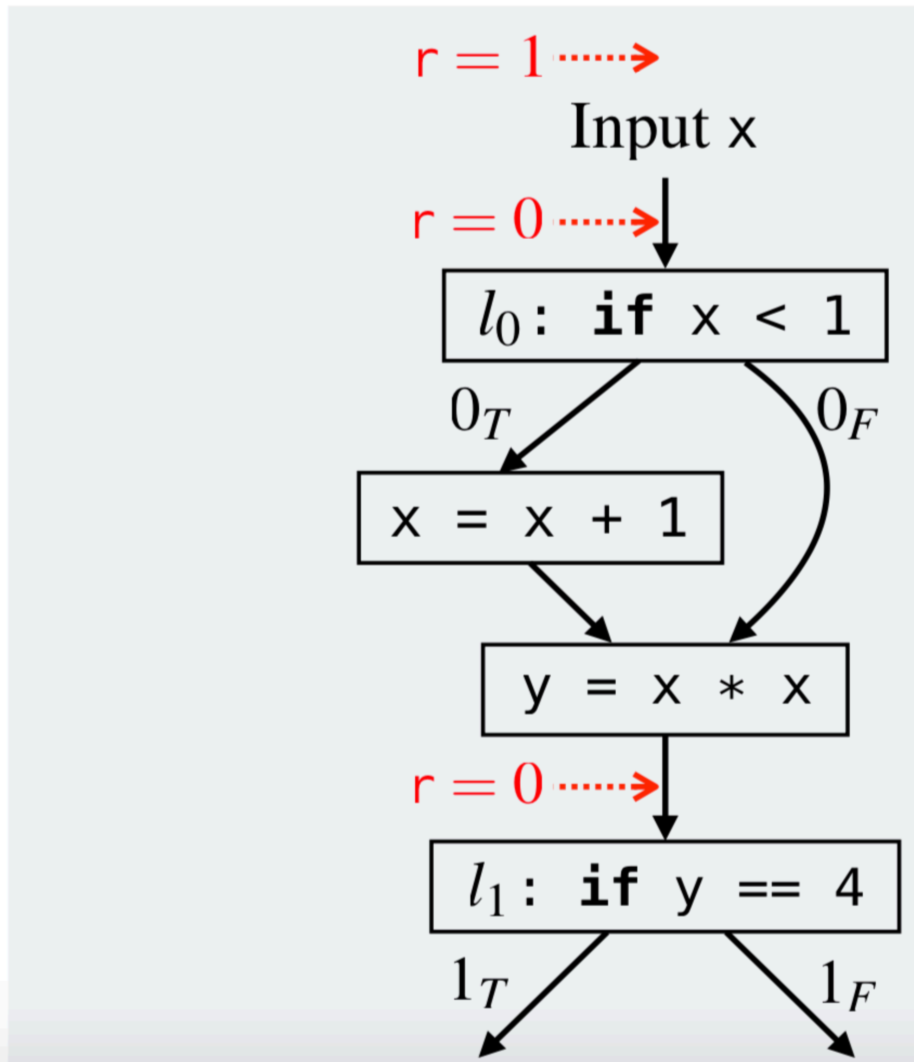
A helper function to quantify how far a and b are from attaining branch *a op b*.

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

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

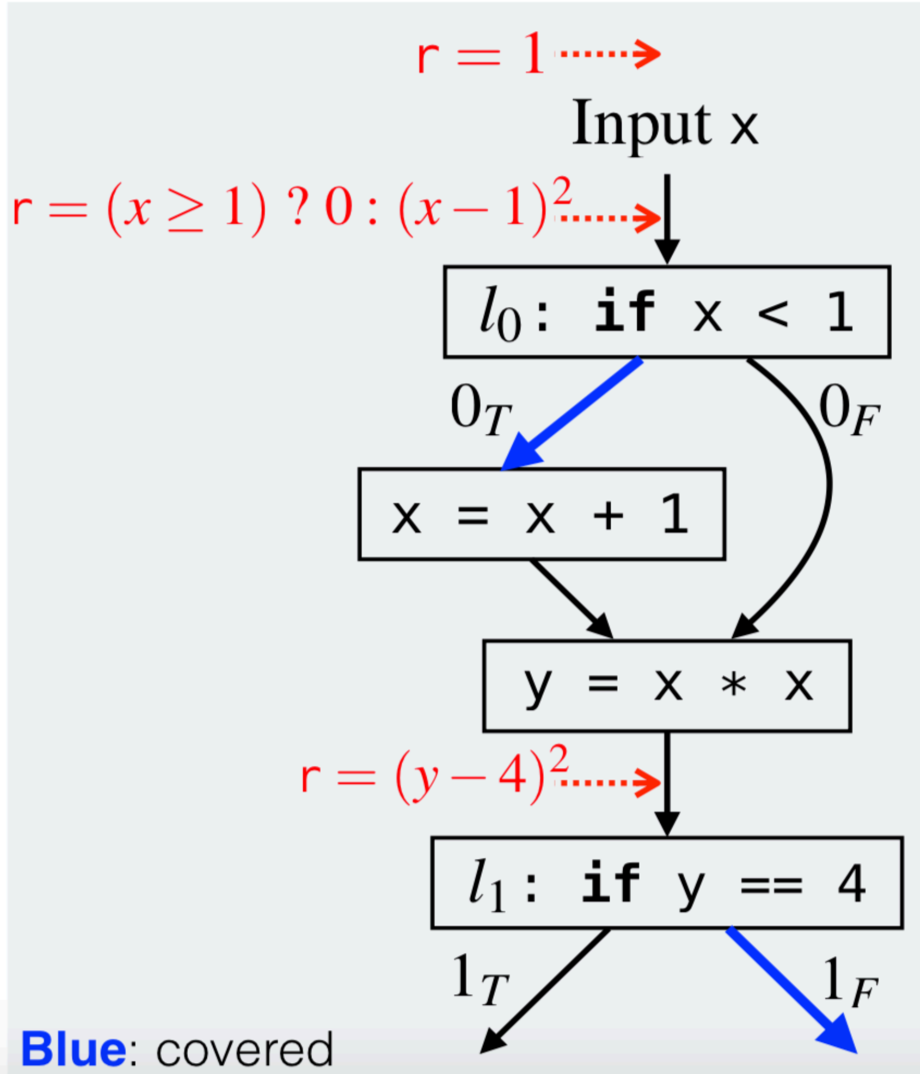
- No branch is covered
- Any input is a minimum point
- Assume $x^* = 0.7$



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

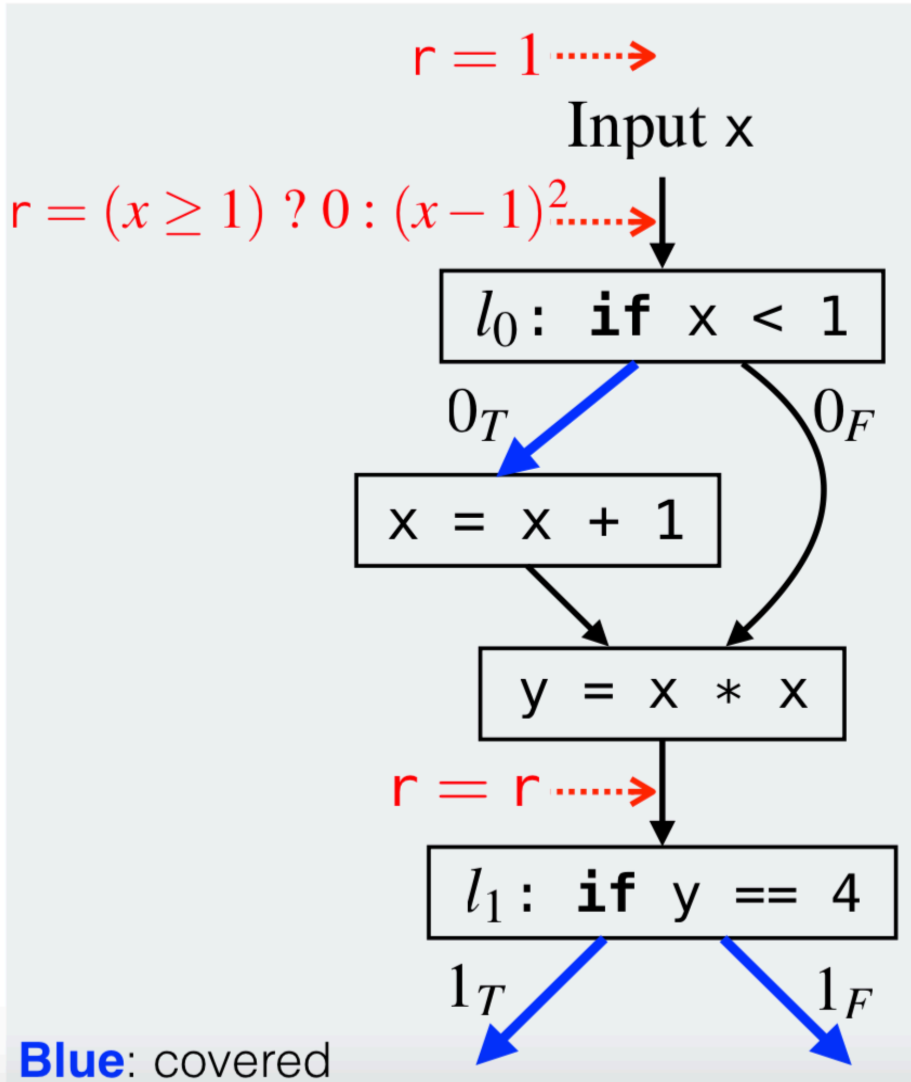
- $1_F, 0_T$ are covered
- FOO_R attains minimum at -3 or 2
- Assume $x^* = -3$



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

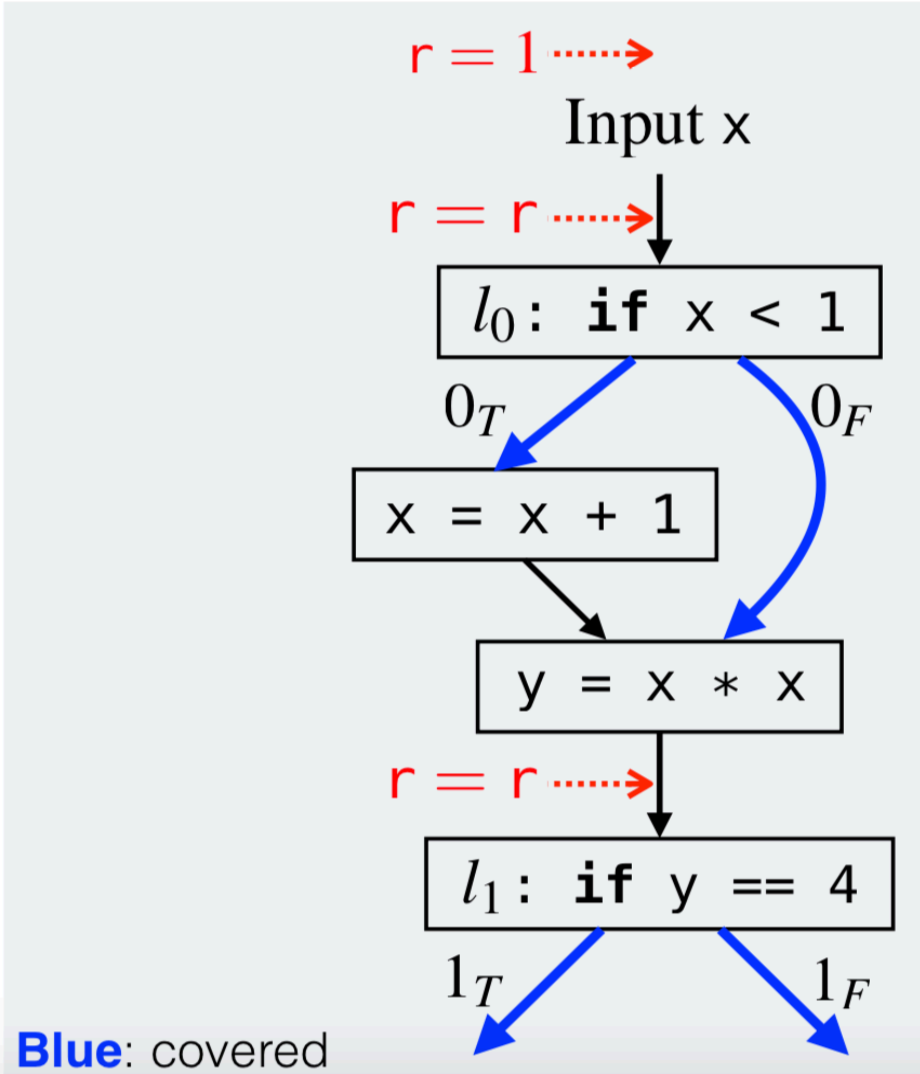
- $1_F, 1_T, 0_T$ are covered
- FOO_R attains minimum at ≥ 1
- Assume $x^* = 5.1$



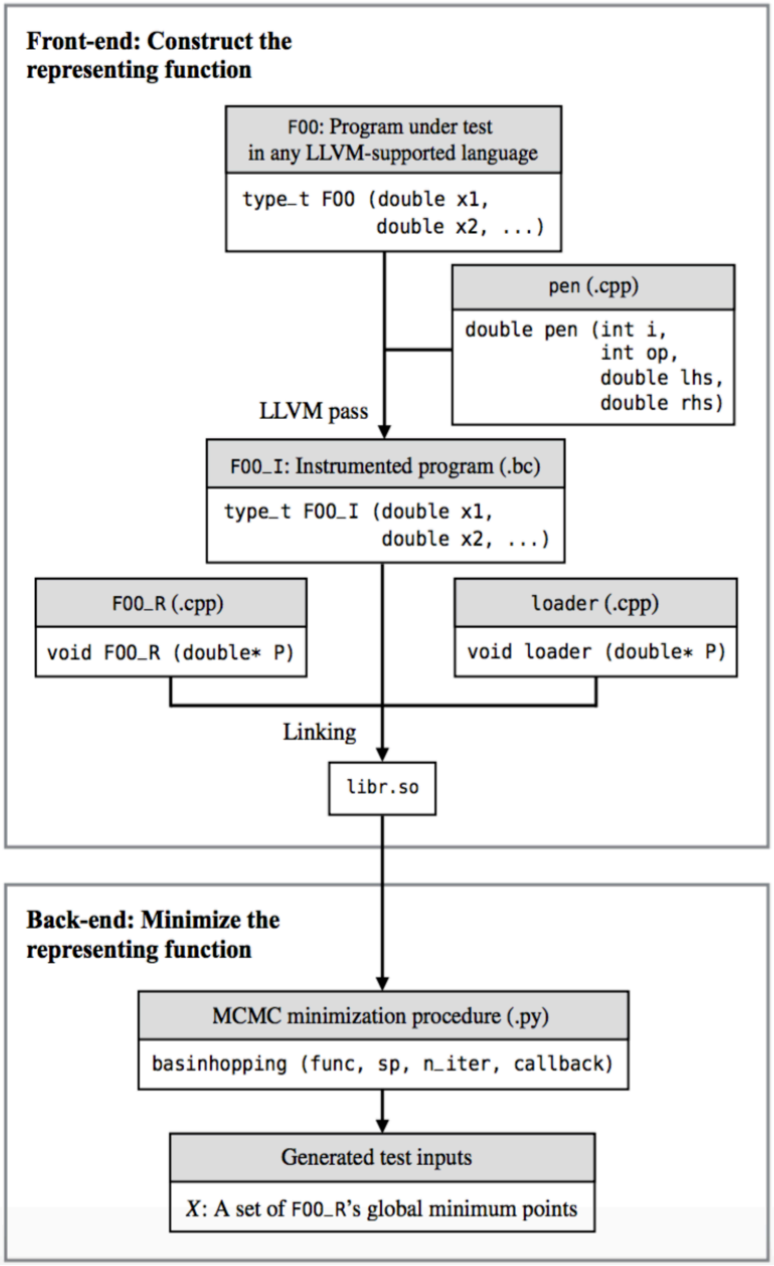
Step 2: Minimize FOO_R

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
- $\forall x, \text{FOO\_R}(x) = 1$
- Termination



Our implementation CoverMe



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

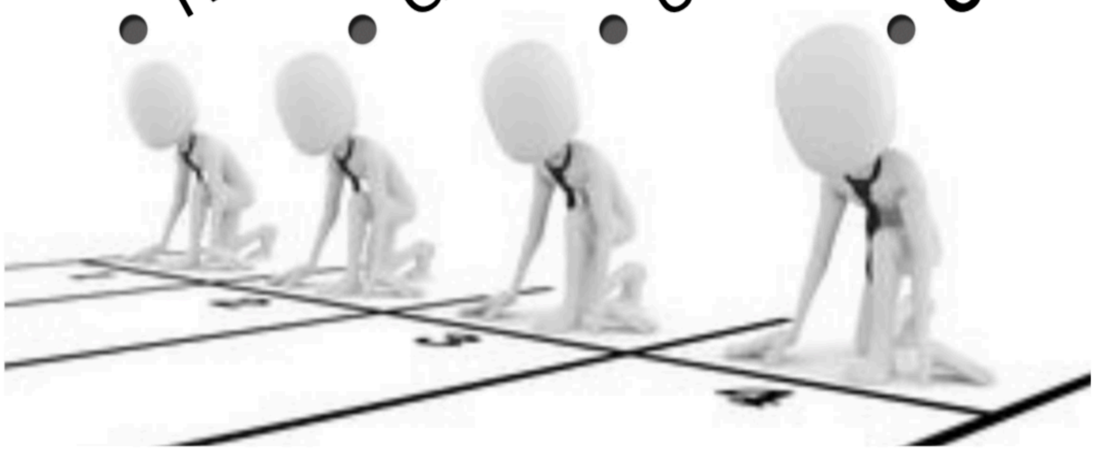
Experiments

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

- Random testing
- Google's **AFL**
- UCL's **Austin**
- **CoverMe**



CoverMe covers

- $\approx$ 90% branches in 7 seconds
- $\approx$ 18% more branches than AFL with 1/10 time
- $\approx$ 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 (AE)

New perspective: ME

$(\mathbf{p}, \phi)$

Analyzing numerical programs

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

Floating-point constraint solving

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

$\mathbf{r}$ **Mathematical optimization (MO)**

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