High-Assurance Scientific Computing: A Unified Approach

Zhoulai Fu

IT University of Copenhagen

October 14, 2021

Intended Learning Outcomes for today

- Mathematical Optimisation as a general software technique
- Satisfiability Solving
- Path Reachability
- Overflow Detection

Boundary value analysis (recall)

```
void Prog(double x) {
  if (x <= 1.0) x++;
  double y = x * x;
  if (y <= 4.0) x--;
}
```

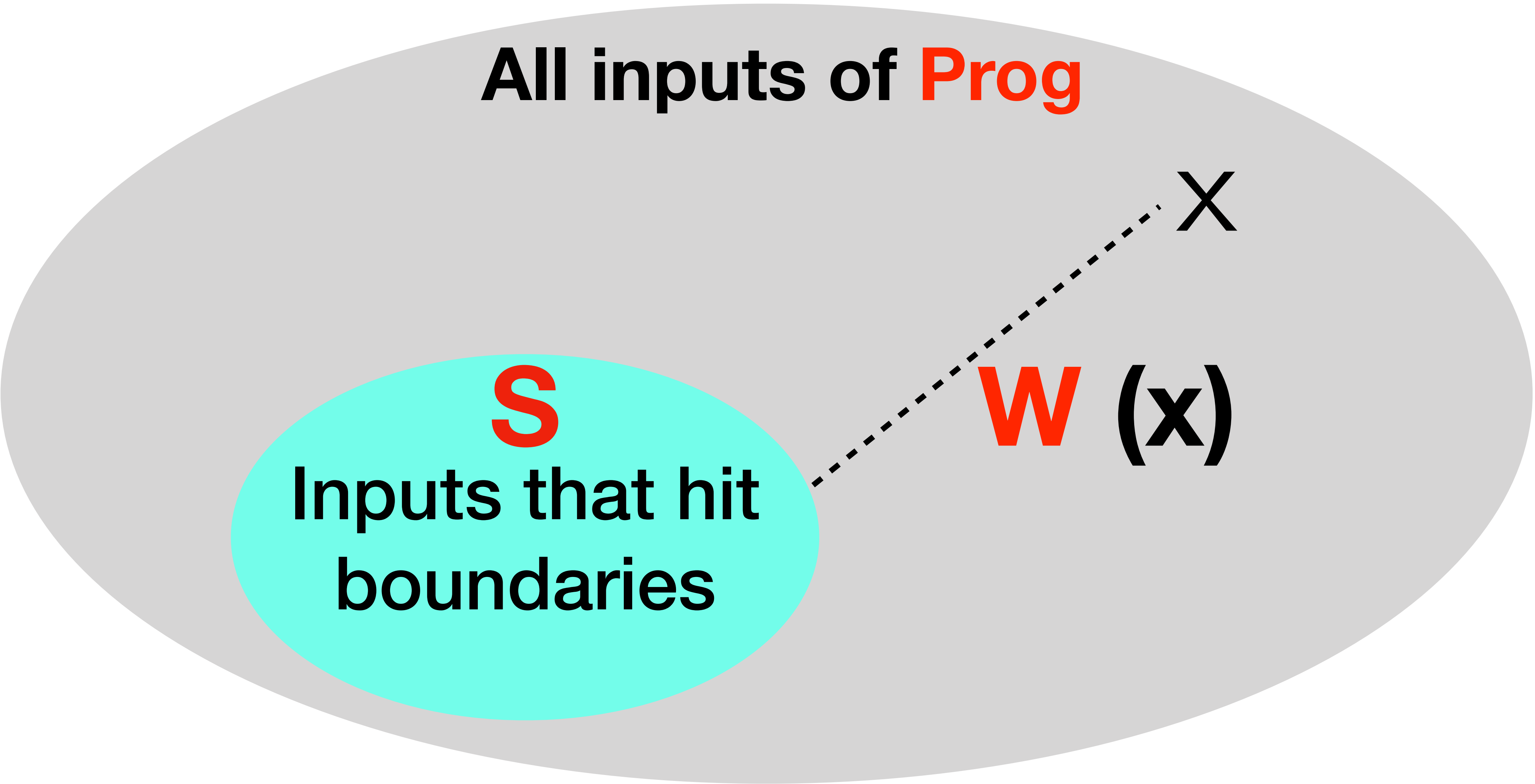
Insert
w = w * abs (a -b)

```
double w;
void Prog_W (double x) {
  w = w * abs(x - 1.0);
  if (x <= 1.0) x++;
  double y = x * x;
  w = w * abs(y - 4.0);
  if (y <= 4.0) x--;
}
double W (double x) {
  w = 1; Prog_W(x); return w;
}
```

Property 1. W (x) >= 0

Property 2. W (x) = 0 if and only if x is a boundary input

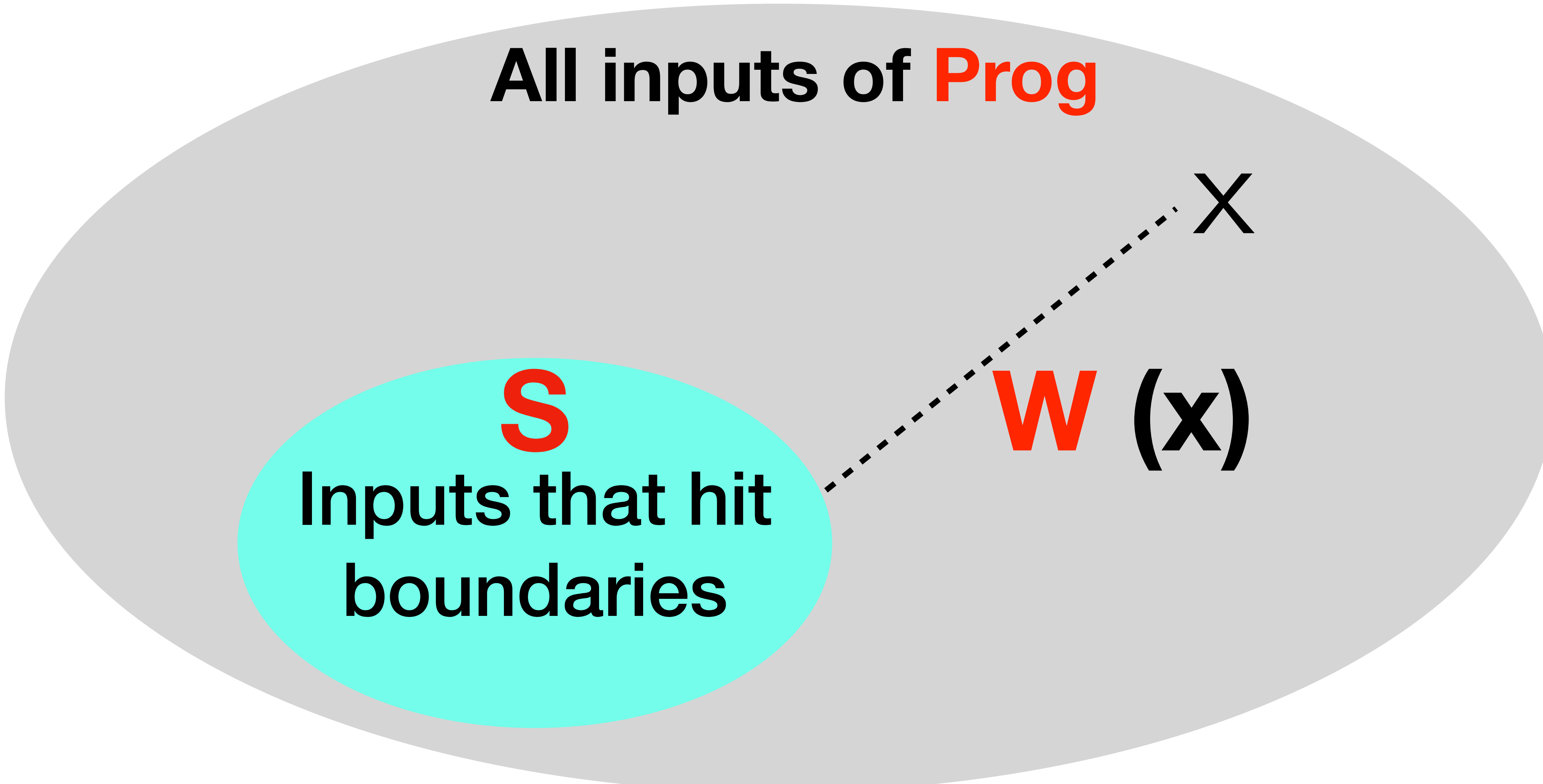
Boundary value analysis as a search problem



- Search an element of S among all inputs of Prog
- Minimize W as mathematical optimization,

looking for 0

Introducing the weak distance



- **W(x) is non-negative**
- **W(x) = 0 if and only if x reaches S**

Get the weak distance W from the syntax of Prog

Generalization

S

Analyzing floating-point code

- FP constraint solving
- Coverage-based testing
- Path reachability
- Boundary value analysis
- Overflow detection

$\Downarrow$

W

Mathematical Optimization

Input x satisfies S $\Leftrightarrow$ x minimizes W

(under the condition that S is non-empty)

The satisfiability problem

DEMO

$$
\begin{aligned}
3x + 2y - z &= 1 \\
2x - 2y + 4z &= -2 \\
-x + \tfrac{1}{2}y - z &= 0
\end{aligned}
$$

https://en.wikipedia.org/wiki/System_of_linear_equations

Satisfiability solving

- Planning
- Invariant Generation
- Type Checking
- Model Based Testing
- Termination
- ...

Example 2

$$a > b + 2, \qquad a = 2c + 10, \qquad c + b \leq 1000$$

SAT

Model

$$a = 0, \qquad b = -3, \qquad c = -5$$

$$0 > -3 + 2, \qquad 0 = 2(-5) + 10, \qquad (-5) + (-3) \leq 1000$$

The intro part taken from http://fm.csl.sri.com/SSFT12/introduction.pdf

Example 3

$$b + 2 = c, \quad f(read(write(a,b,3), c\text{-}2)) \neq f(c\text{-}b+1)$$

Example 3 (cont)

$$\boxed{b + 2 = c,}\ \ f(read(write(a,b,\boxed{3}),\ \boxed{c-2})) \neq f(\boxed{c-b+1})$$

Arithmetic

Example 3 (cont)

$$b + 2 = c, \quad f(\boxed{read}(\boxed{write}(a,b,3), c-2)) \neq f(c-b+1)$$

Array Theory

Example 3 (cont)

$$b + 2 = c, \quad \boxed{f}(read(write(a,b,3), c-2)) \neq \boxed{f}(c-b+1)$$

Uninterpreted Functions

Example 3 (cont)

$$b + 2 = c, \quad f(read(write(a,b,3), c-2)) \neq f(c-b+1)$$

Example 3 (cont)

$$b + 2 = c, \quad f(read(write(a,b,3), b+2-2)) \neq f(b+2-b+1)$$

Example 3 (cont)

$$b + 2 = c, \quad f(read(write(a,b,3), b)) \neq f(3)$$

Array Theory Axiom

$$\forall a,i,v : read(write(a, i, v), i) = v$$

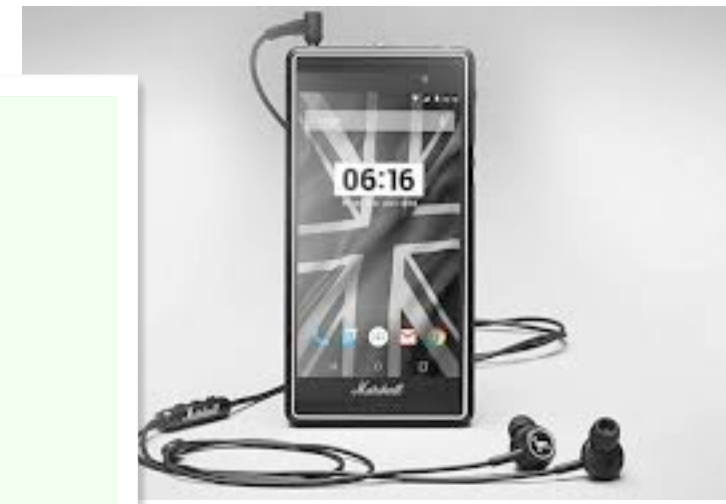
Example 3 (cont)

$$b + 2 = c, \quad f(3) \neq f(3)$$

UNSAT

Floating-point Satisfiability

Floating-point satisfiability solving in the digital age




```
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) {
    for (ix = -1043, i=lx; i>0; i<<=1) ix -=1;
```

Where challenges lie

Solving constraints with:

- **floating-point** arithmetic
- **non-linear** properties
- **external** functions, such as SIN, LOG, EXP

An example of solving floating-point constraints

Find a floating-point x to satisfy

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

Existing solutions would not solve it

Solvers of **Reals** cannot solve this constraint

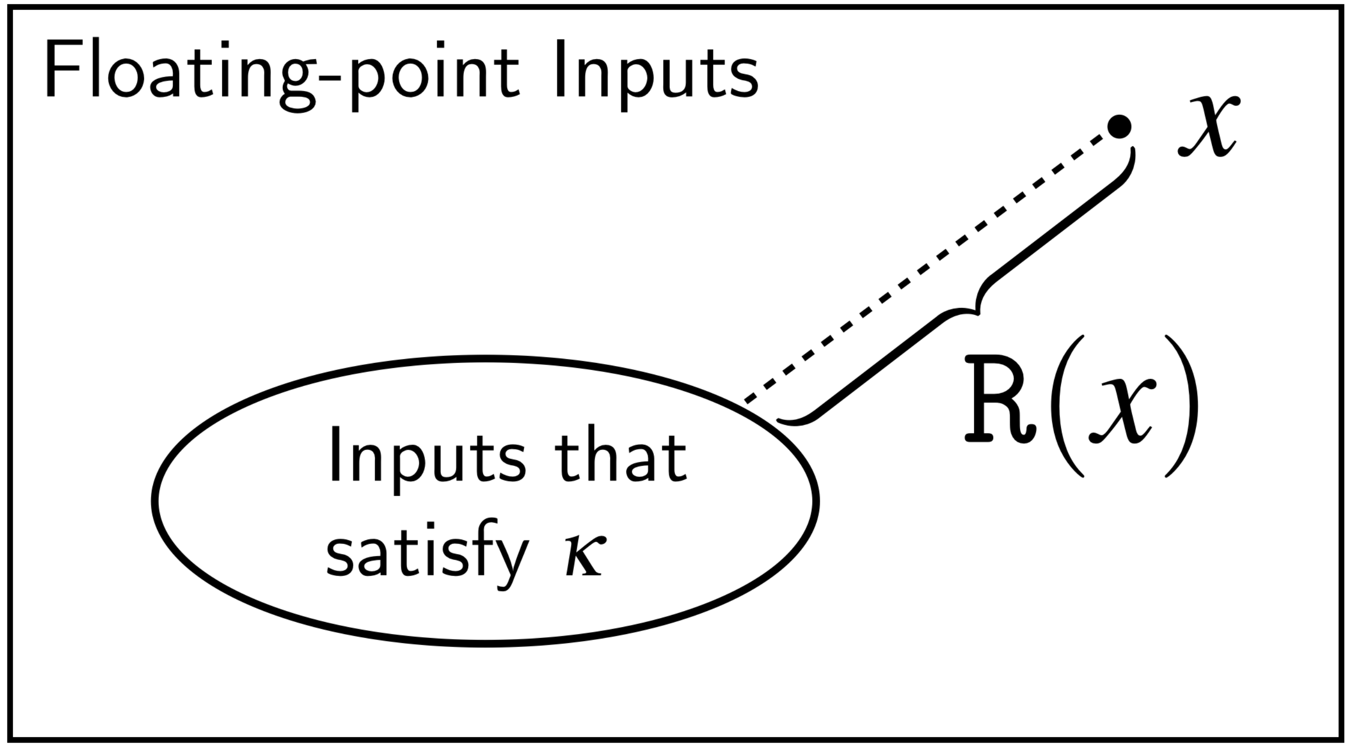
$$\text{If } x \in \mathbb{R}, SIN(x) = x \Leftrightarrow x = 0$$

Reducing to boolean satisfiability (bit-blasting) would require semantics approximation.

"IEEE-754 contains recommendations for trigonometric functions and exponentials but neither are mandated. The accuracy of implementations of these functions vary significantly, making it very hard to come up with logical models that are widely applicable. . ."

[Ref] Brain et al., "An automatable formal semantics for IEEE-754 ng-point arithmetic." In Computer Arithmetic 2015

Approach with Mathematical Optimisation



Step 1: Represent constraint κ by a function R

- $R(x) \geq 0$ for all x
- $R(x) = 0 \Leftrightarrow x \models \kappa$

Step 2: Minimize R

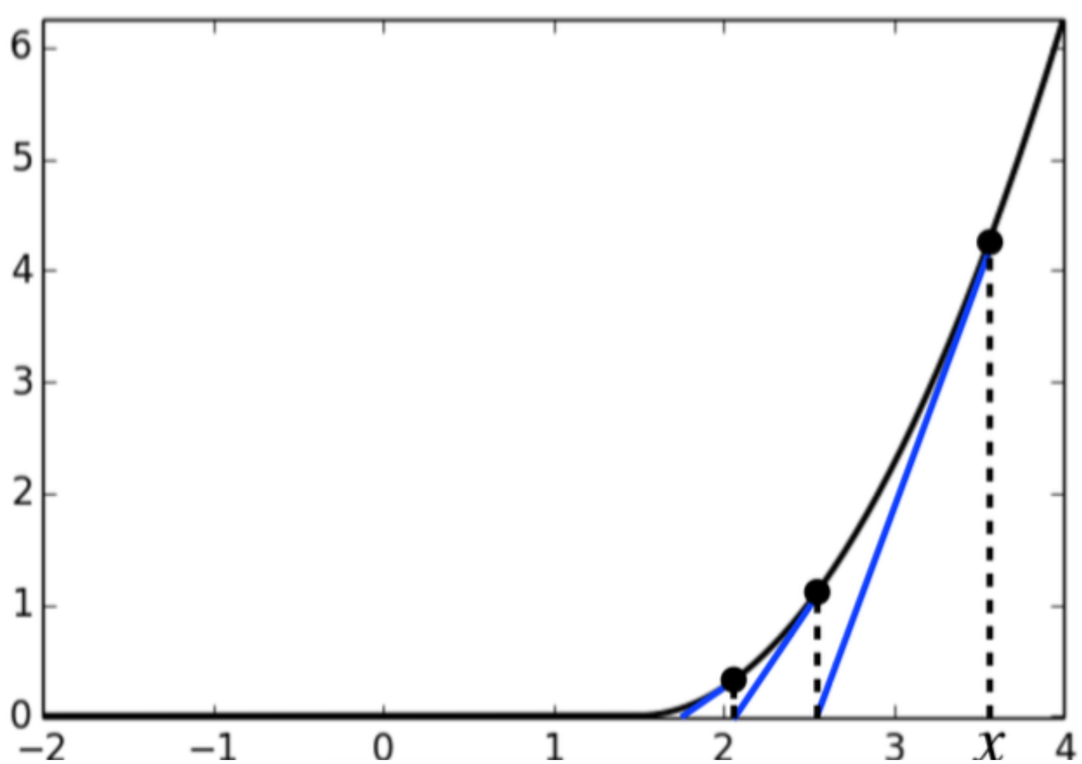
Theoretical guarantee: κ satisfiable $\Leftrightarrow R(x^*) = 0$
where x^* is a minimum point

Example κ_1 $\boxed{x \leq 1.5}$

Step 1. Transform κ_1 to

$$R_1(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x \leq 1.5 \\ (x-1.5)^2 & \text{otherwise} \end{cases}$$

Step 2. Minimize R_1 (local optimization)



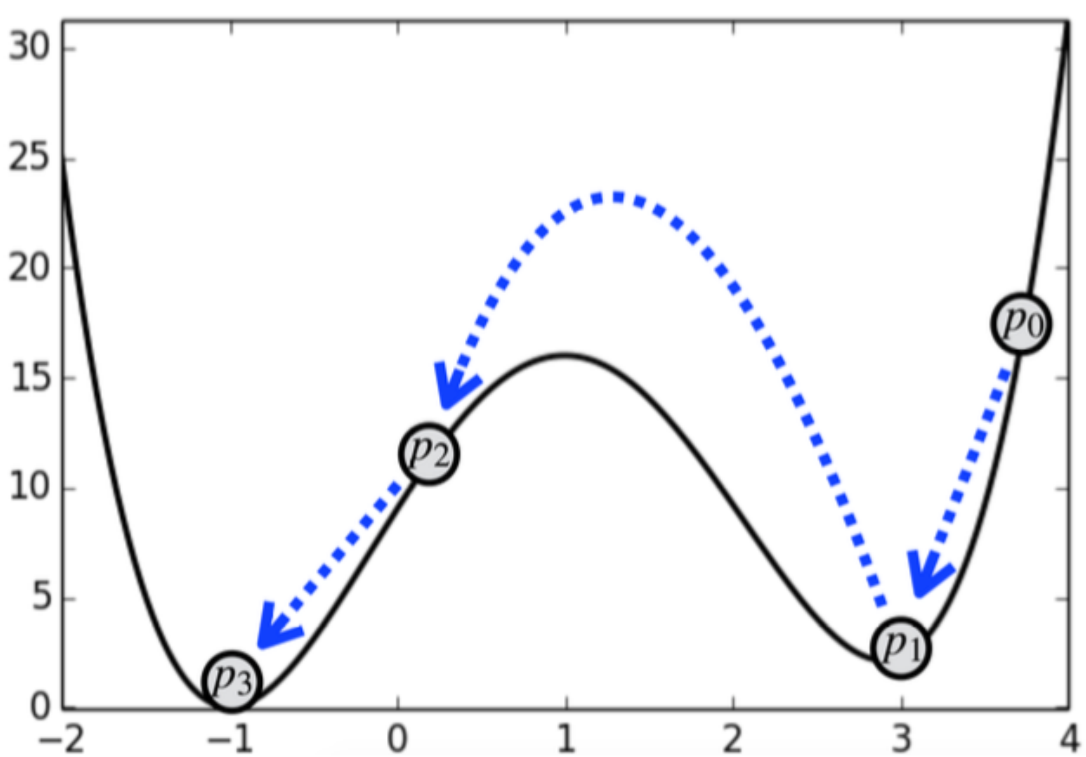
- x^* can be anything ≤ 1.5
- $R_1(x^*) = 0 \implies x^* \models R_1$

Example κ_2 $\boxed{(x-1)^2 == 4 \ \wedge \ x \leq 1.5}$

Step 1. Transform κ_2 to R_2

$$((x-1)^2-4)^2+\begin{cases}0 & \text{if } x \leq 1.5 \\ (x-1.5)^2 & \text{otherwise}\end{cases}$$

Step 2. Minimize R_2 (Basinhopping)



- $x^* = -1$
- $R_2(x^*) = 0 \implies x^* \models R_2$

Example κ_3 $\boxed{SIN(x) == x \ \wedge \ x \geq 10^{-10}}$

Step 1. Transform κ_3 to R_3:

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

Step 2. Minimize R_3 (Basinhopping)

- $x^* = 9.0 * 10^{-9}$ (can be others)
- $R_3(x^*) = 0 \implies x^* \models R_3$

Construct R systematically

Constraint κ	Program R
$x == y$	$(x-y)^2$
$x \leq y$	$x \leq y\ ?\ 0 : (x-y)^2$
$\kappa_1 \wedge \kappa_2$	$R_1 + R_2$
$\kappa_1 \vee \kappa_2$	$R_1 * R_2$

Constraint κ	Program R
$x == y$	$(x-y)^2$
$x \leq y$	$x \leq y \; ? \; 0 : (x-y)^2$
$\kappa_1 \wedge \kappa_2$	$R_1 + R_2$
$\kappa_1 \vee \kappa_2$	$R_1 * R_2$

Theoretical guarantee and limitation

- In theory, κ is satisfiable $\Leftrightarrow R(x^*) = 0$
- In practice, $R(x^*)$ may be inaccurate

Path reachability

FOO: Program under test

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

Goal: Search inputs that reach the path "L0-true, L1-true"

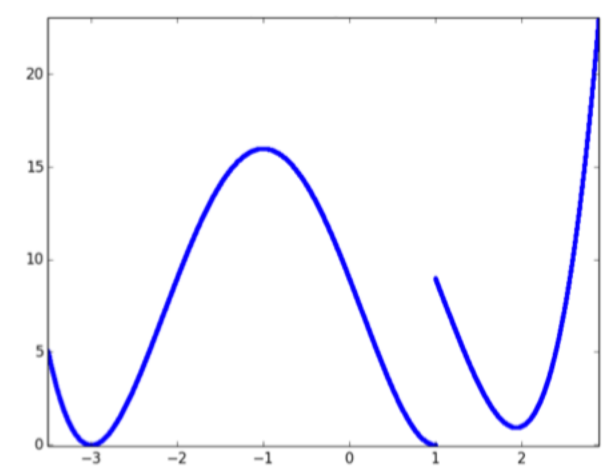
Construct a weak distance

FOO: Program under test

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

```
double r;
void FOO_I(double x){
    r = r + (x ≤ 1) ? 0 : (x − 1)^2;
l0: if (x <= 1) x++;
    double y = square(x);
    r = r + (y − 4)^2;
l1: if (y == 4) ... ;}
```

```
double FOO_R(double x){
    r = 0; FOO_I(x); return r;}
```



Quiz: FOO_R is a weak distance iff. ___

DEMO: /Users/zhfu/Google Drive/active/19_teaching_asa/python/demo4.py

Overflow Detection

GNU Scientific Library's bessel function

```
int gsl_sf_bessel_Knu_scaled_asympx_e(const double nu,
const double x, gsl_sf_result* result) {
  double mu = 4.0 * nu * nu;
  double mum1 = mu - 1.0;
  double mum9 = mu - 9.0;
  double pre = sqrt(M_PI / (2.0 * x));
  double r = nu / x;
  result->val = pre * (1.0 + mum1 / (8.0 * x) +
                       mum1 * mum9 / (128.0 * x * x));
  result->err = 2.0 * GSL_DBL_EPSILON *
    fabs(result->val) + pre * fabs(0.1 * r * r * r);
  return GSL_SUCCESS;
}
```

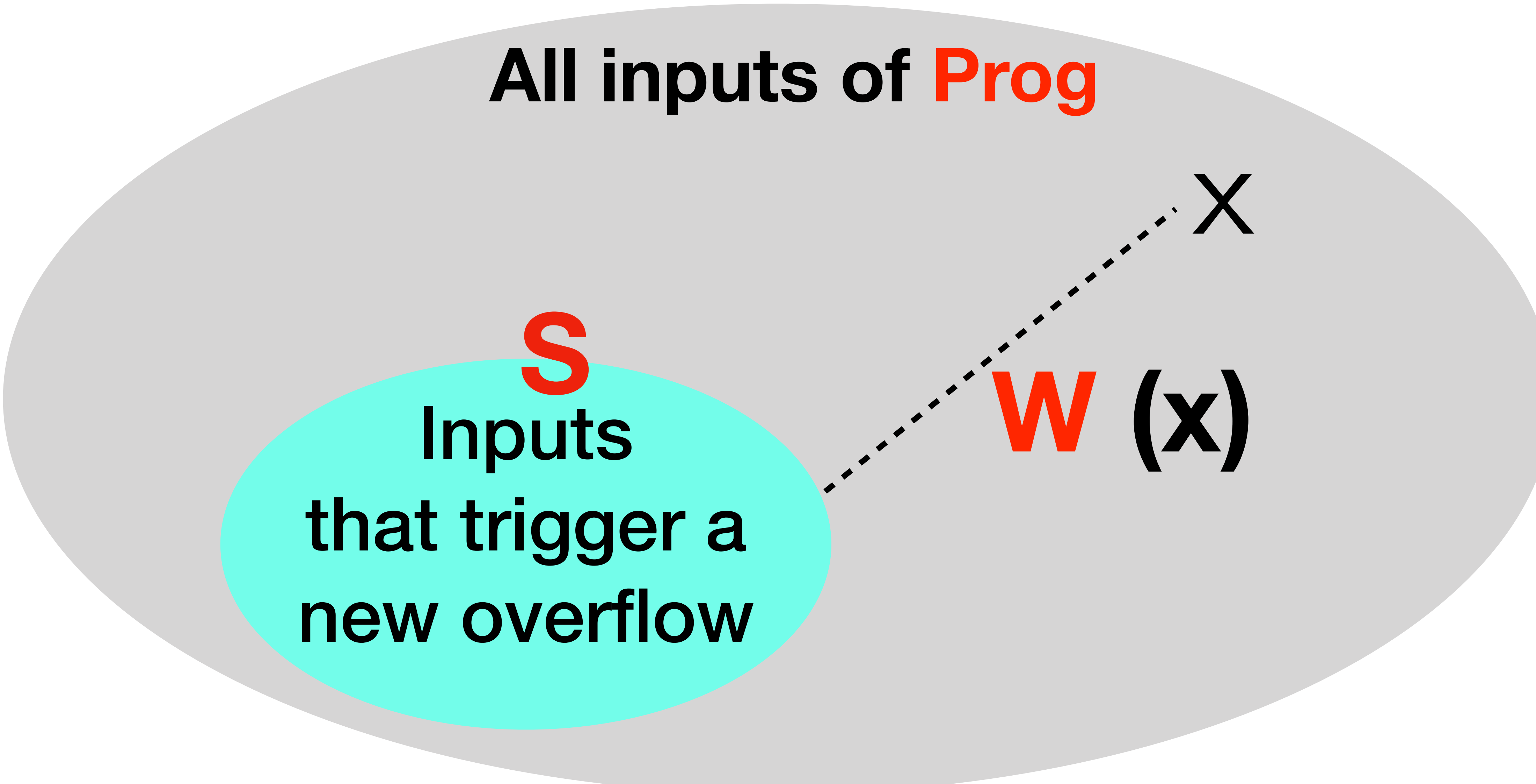
LLVM IR

l_1: `t = 4.0 * nu`

l_2: `mu = t * nu`

Goal: Trigger FP overflow for the first statement

Overflow detection via weak distance minimization



Step 1. Construct W

- Non-negative for all x
- W = 0 if and only if x reaches **S**

Step 2. Minimize W repeatedly until > 0

```
Program_after_insertion (const double nu,...)
{
  l1: t = 4.0 * nu
  if (l1 is not in L) w = |t|<MAX? MAX-|t| : 0

  l2: mu = t * nu
  if (l2 is not in L) w = |mu|<MAX? MAX-|mu| : 0
```

L: Overflowed instructions

Round 1

nu	w
0	$\mathtt{Max}$
$\vdots$	$\vdots$
$\frac{1}{8}\sqrt{\mathtt{Max}}$	$\frac{15}{16}\mathtt{Max}$
$\frac{1}{4}\sqrt{\mathtt{Max}}$	$\frac{3}{4}\mathtt{Max}$
$\frac{1}{2}\sqrt{\mathtt{Max}}$	0

Round 2

nu	w
0	$\mathtt{Max}$
$\vdots$	$\vdots$
$\frac{1}{16}\mathtt{Max}$	$\frac{3}{4}\mathtt{Max}$
$\frac{1}{8}\mathtt{Max}$	$\frac{1}{2}\mathtt{Max}$
$\frac{1}{4}\mathtt{Max}$	0

Summary: Weak-distance minimization

+ A general method

+ Do not analyze the FP code; minimize another one

+ Theoretical guarantee

- Minimizing is inherently incomplete (see exercises)

Conclusions

S

Analyzing floating-point code

- FP constraint solving
- Coverage-based testing
- Path reachability
- Boundary value analysis
- Overflow detection

$\Downarrow$

W

Mathematical Optimization

Input x satisfies S $\Leftrightarrow$ x minimizes W

(under the condition that S is non-empty)