Effective Floating-Point Analysis via Weak-Distance Minimization

Zhoulai Fu

Advanced Software Analysis, lecture 4

Sep 2, 2020

Previous lectures

- Random testing
- Boundary input generation
- Solving Numerical Constraints

A common trait: these are all search problems.

Today: Weak-Distance Minimization

- Do not analyze floating-point programs
- Focus on how to run them efficiently
- Theoretical guarantee

Boundary value analysis (recall)

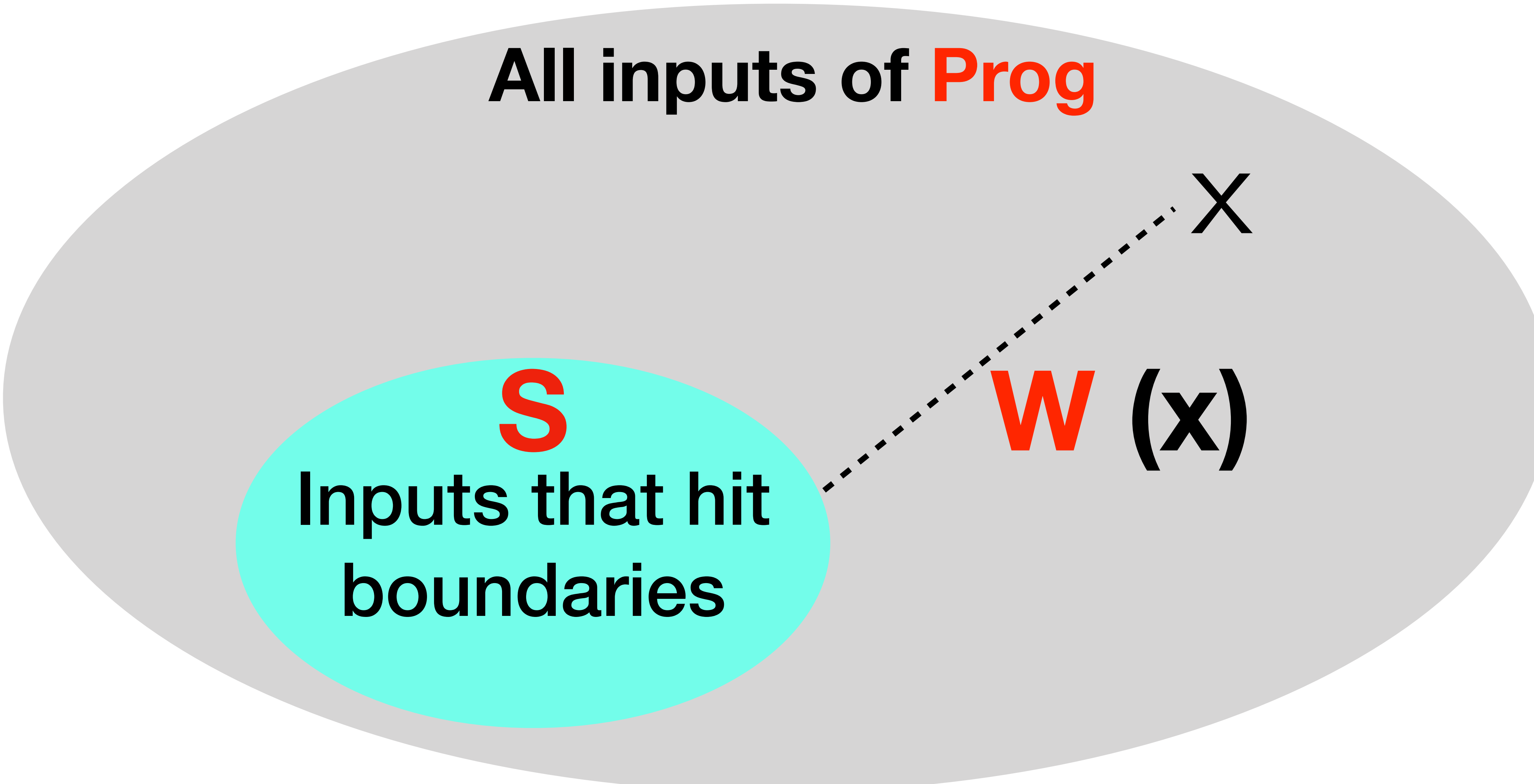
Boundary x == 1.0

Boundary y == 4.0

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

Goal: Generate the boundary inputs -3, 1, 2

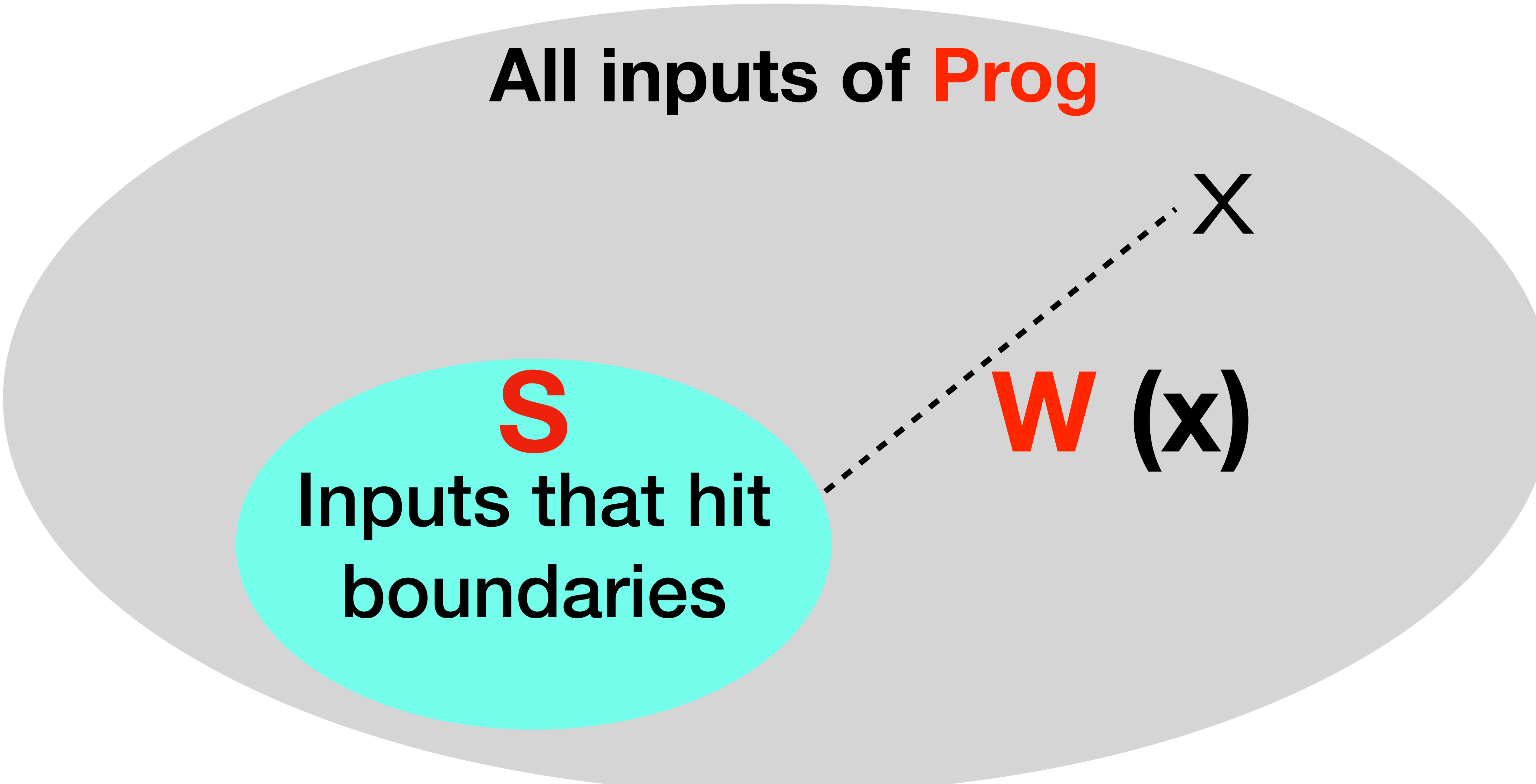
Boundary value analysis as a search problem



- 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

Boundary value analysis: Construct W

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

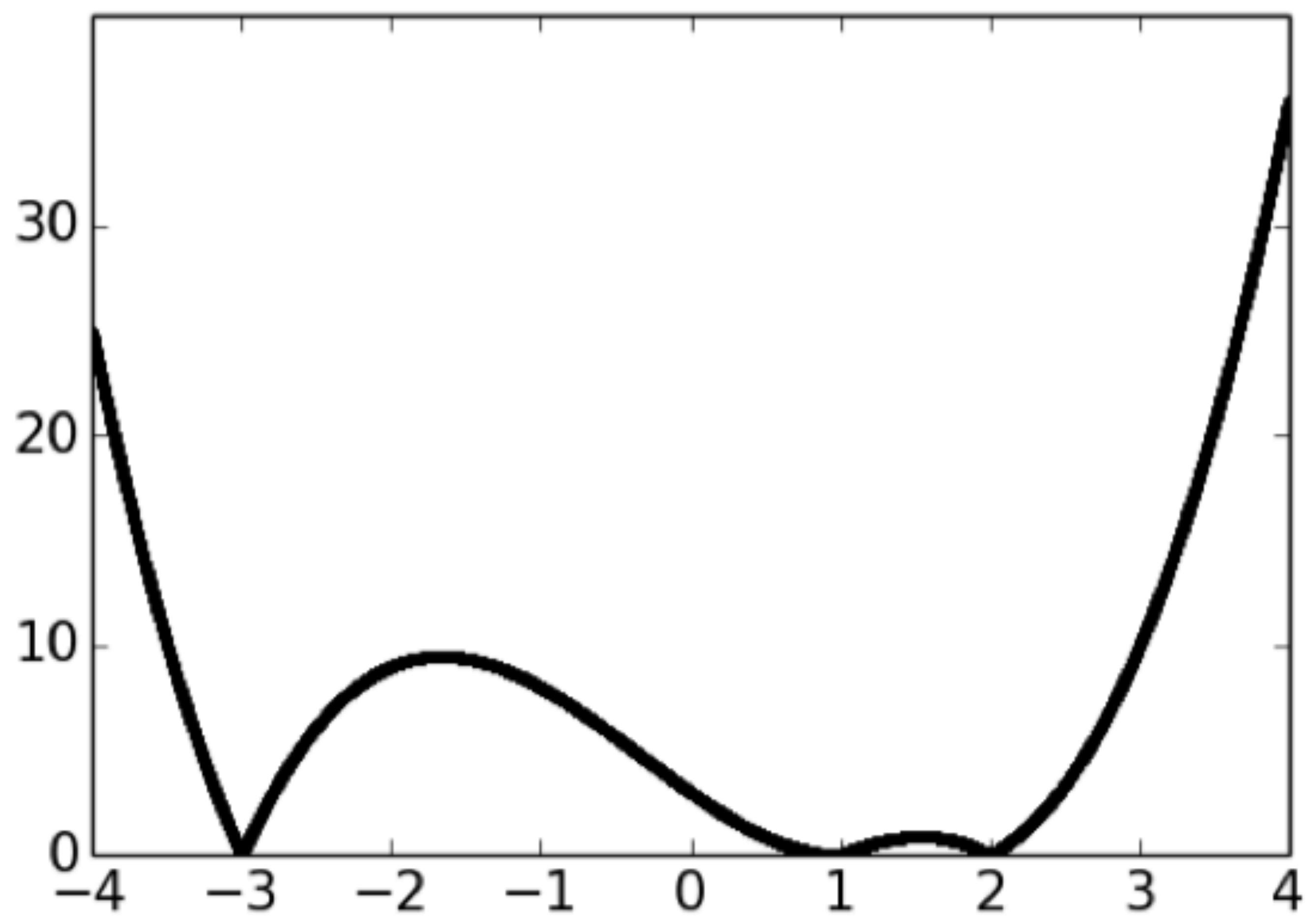
Insert
w = w * abs (a -b)

```
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

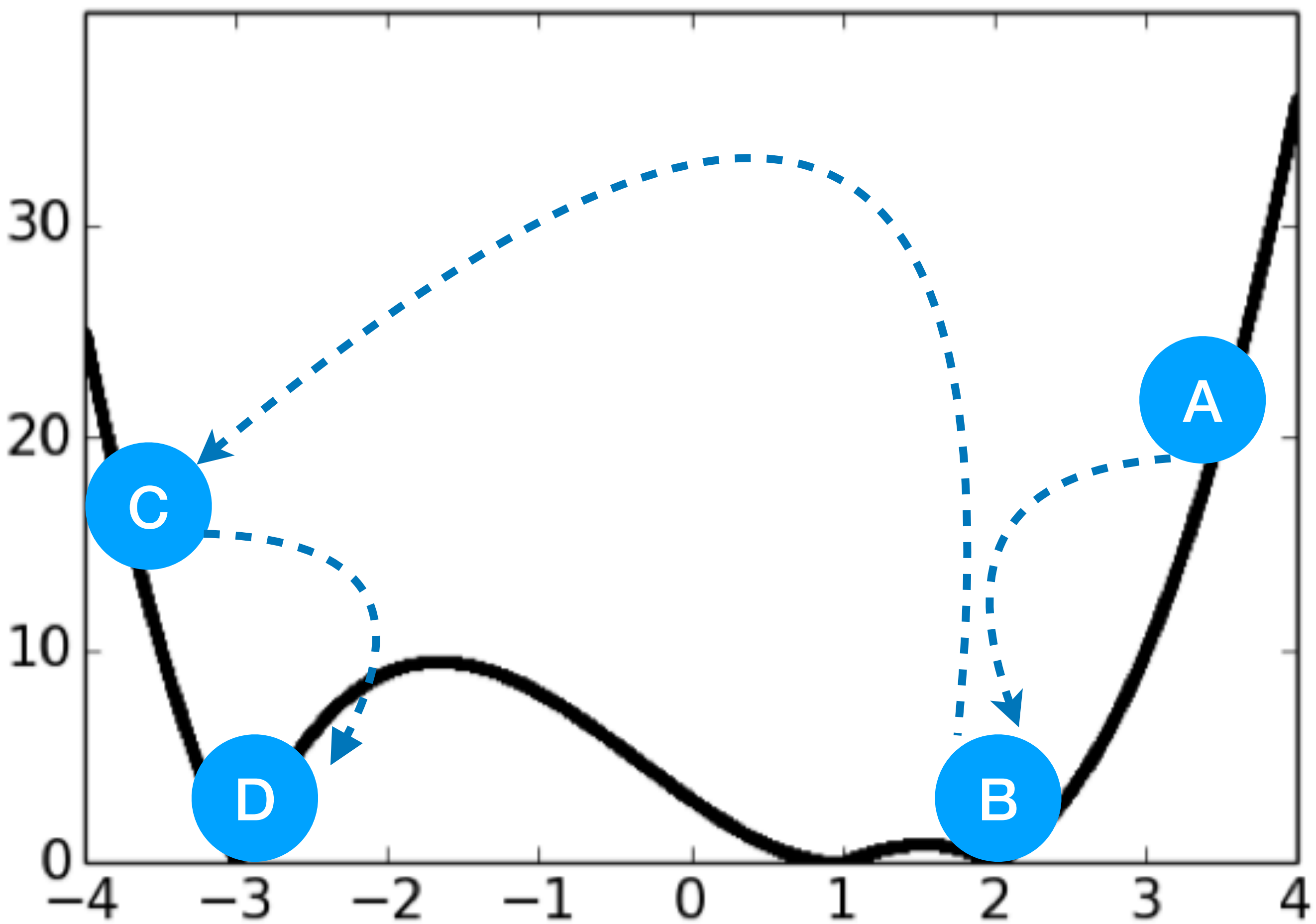
Property 2. W (x) = 0 if and only if x is a boundary input

Property 3. W is continuous



Graph of the weak distance W

Boundary value analysis: Minimize W



Mathematical optimization

- **Black-box**
- **Local & global**
- **Can be very efficient**

Demo (if time permits) => Locating -3, 1, 2

Solving numerical constraints (recall)

Constraints of FP	$\pi := \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid e_1 \bowtie e_2$
Arithmetic expressions	$e := c \mid X \mid \mathtt{foo}(e_1, \cdots e_n) \mid e_1 \oplus e_2$

Goal: search a model (if any) from the domain (namely, possible values of the free variables) of the constraint

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)

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"

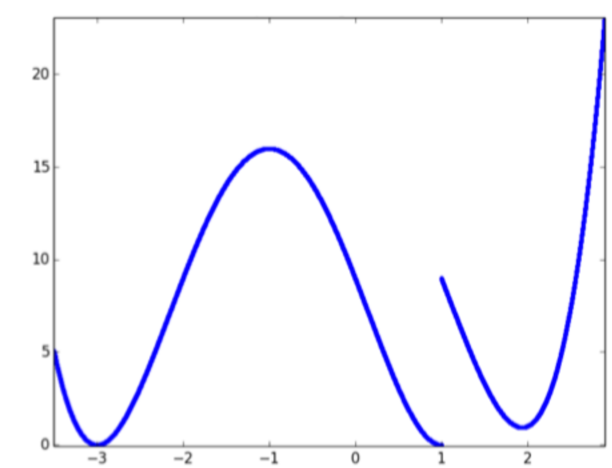
Construct a weak distance

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;
}
```

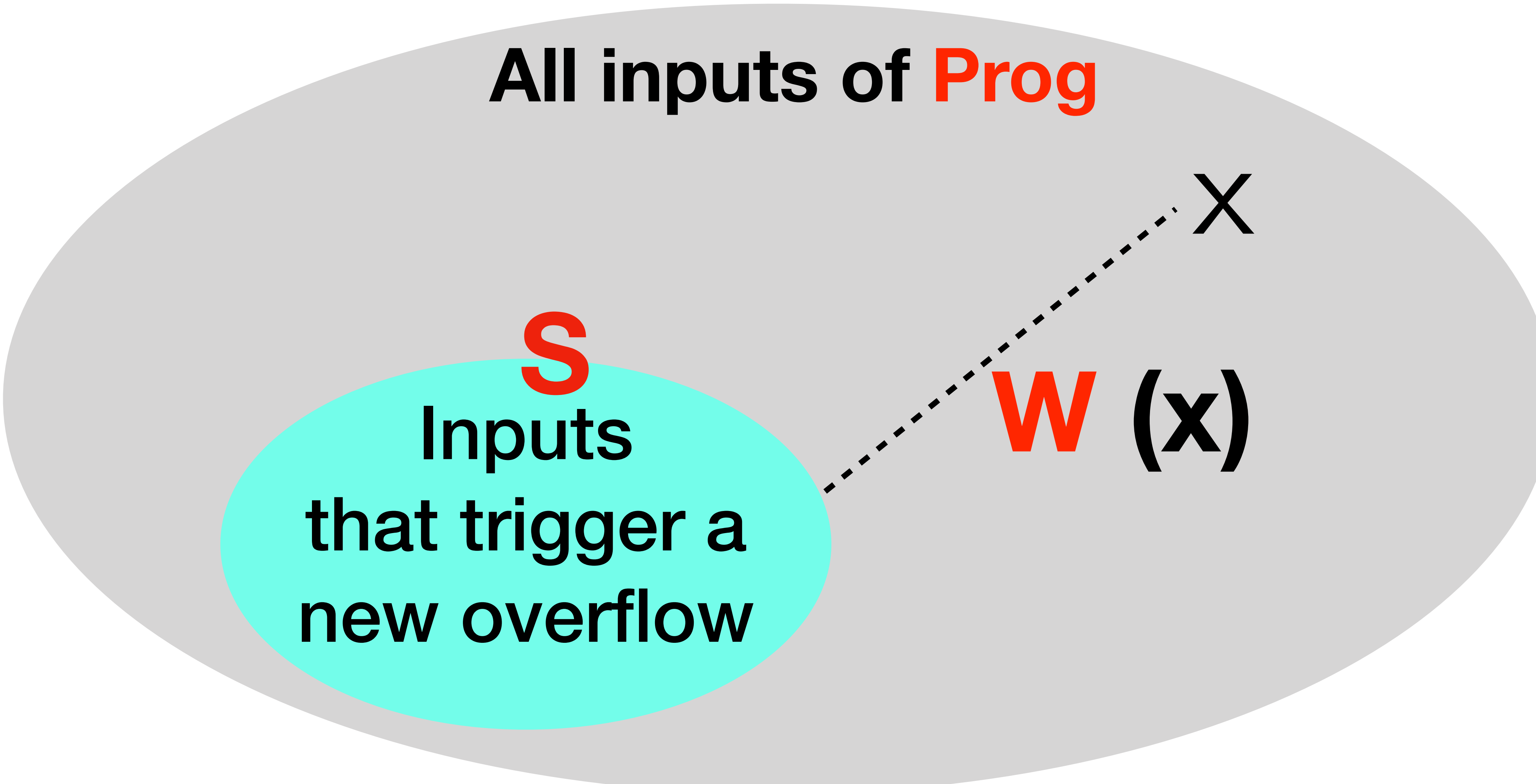
LLVM IR

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)