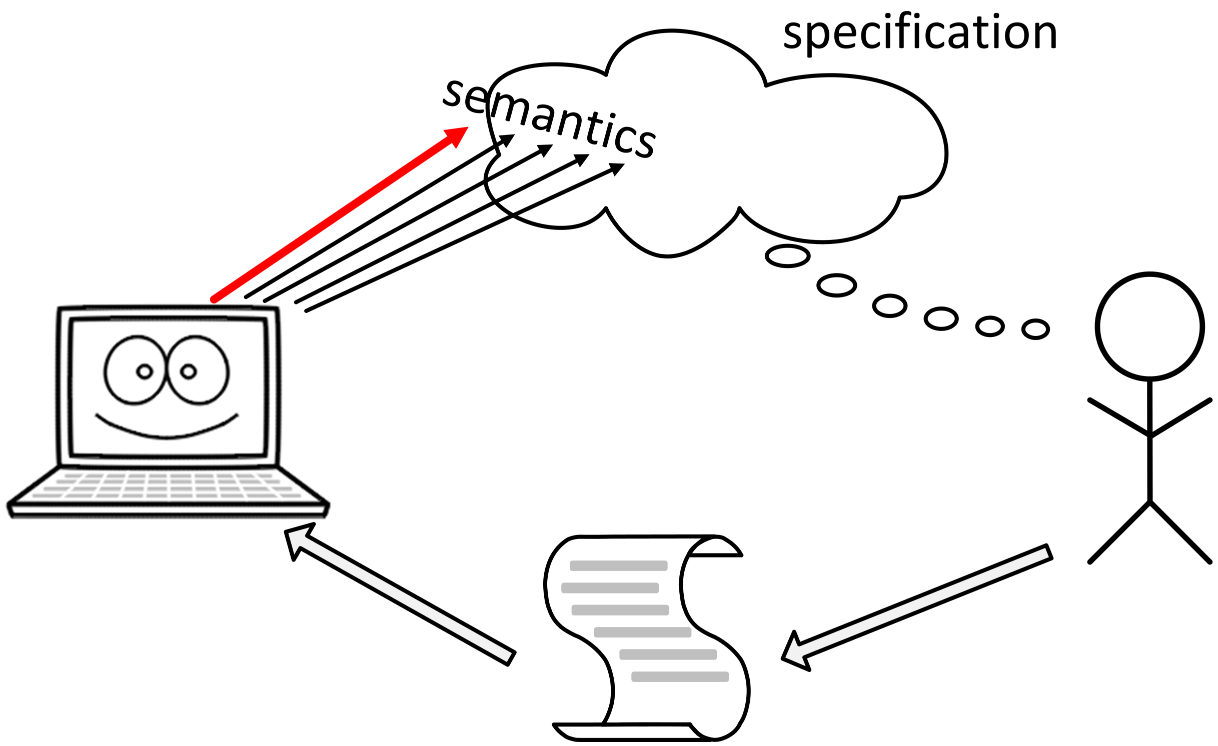
Previous lectures



- **Random testing: Finding defects in code by running it**

- **Boundary value analysis: transforming P to P', minimize P'**

Symbolic execution

Idea:

every path reachability corresponds to a constraint on the inputs

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

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

Satisfiability solving

x == 1 OR (x > 1 AND x * x = 4) OR
(x<=1.0 and (x + 1) ** 2 == 4)

Applications with satisfiability solving

- Planning
- Scheduling
- Constraint Solving
- Systems Biology
- Invariant Generation
- Type Checking
- Model Based Testing
- Termination
- ...

A brief Introduction to satisfiability and floating-point satisfiability solving

Zhoulai Fu

Advanced Software Analysis, lecture 3

Aug 31, 2020

Today's lecture

- **Program transformation combined with mathematical optimization, turns out to be a general method in software analysis, which we will illustrate with floating-point constraint solving today.**

- **We start by introducing the satisfiability problem**

Reference

- De Moura, Leonardo & Dutertre, Bruno & Shankar, Natarajan. (2007). A Tutorial on Satisfiability Modulo Theories. 20-36. 10.1007/978-3-540-73368-3_5.

- Fu, Z. and Su, Z., 2016, July. XSat: a fast floating-point satisfiability solver. In *International Conference on Computer Aided Verification* (pp. 187-209). Springer, Cham.

The satisfiability problem

DEMO

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

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

The satisfiability problem

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

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

The satisfiability problem

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

The satisfiability problem

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

Arithmetic

The satisfiability problem

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

Array Theory

The satisfiability problem

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

Uninterpreted
Functions

The satisfiability problem

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

The satisfiability problem

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

The satisfiability problem

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

Array Theory Axiom

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

The satisfiability problem

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

UNSAT

Floating-point Satisfiability

Floating-point satisfiability solving in the digital age


```c
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;
```

Goal

Solving constraints with:

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

An intermezzo on conjunctive normal form (CNF)

$$p_1 \lor \neg p_2, \qquad \neg p_1 \lor p_2 \lor p_3, \qquad p_3$$

$$p_1 = true, \qquad p_2 = true, \qquad p_3 = true$$

CNF is a set (conjunction) set of clauses
Clause is a disjunction of literals
Literal is an atom or the negation of an atom

[Slides taken from http://fm.csl.sri.com/SSFT12/introduction.pdf](http://fm.csl.sri.com/SSFT12/introduction.pdf)

Quiz: Are these CNF forms, and why?

- $\neg(B \vee C)$
- $\neg B \wedge \neg C$
- $(A \wedge B) \vee C$
- $A \wedge (B \vee (D \wedge E))$,

… a formula is in conjunctive normal form (CNF) … if it is a conjunction of one or more clauses, where a clause is a disjunction of literals; otherwise put, it is **an AND of ORs.**

From Wikipedia

CNF examples

- $(A \lor \neg B \lor \neg C) \land (\neg D \lor E \lor F)$
- $(A \lor B) \land C$
- $A \lor B$
- A

Not CNF

- $\neg(B \lor C)$
- $(A \land B) \lor C$
- $A \land (B \lor (D \land E))$

CNF is a fundamental form in logic — all formula can be transformed to them

- $\neg(B \lor C)$
- $(A \land B) \lor C$
- $A \land (B \lor (D \land E)),$

- $\neg B \land \neg C$
- $(A \lor C) \land (B \lor C)$
- $A \land (B \lor D) \land (B \lor E).$

Some rules for the transformation

- De Morgan $\quad \neg(P \lor Q) \iff (\neg P) \land (\neg Q).$

- Distributive laws

$$(P \land (Q \lor R)) \Leftrightarrow ((P \land Q) \lor (P \land R))$$

An example of solving floating-point CNF

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

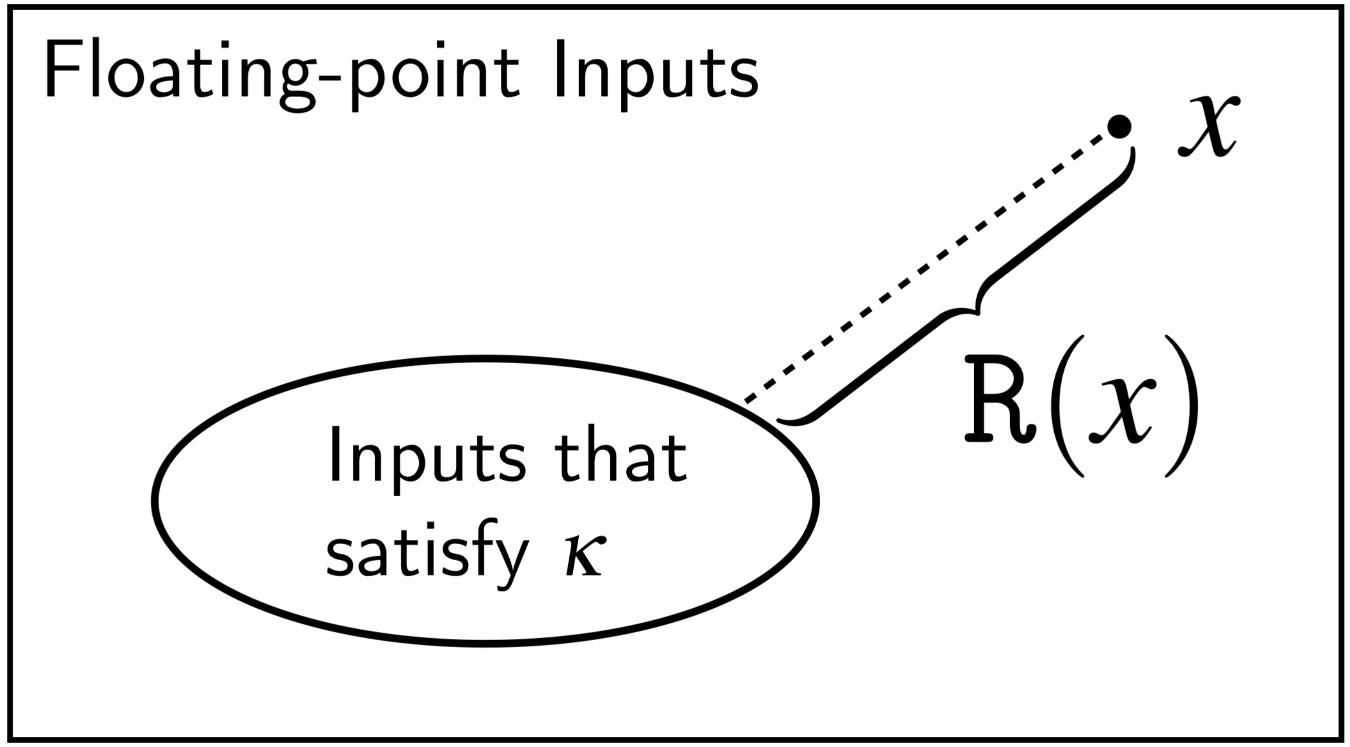
$$\text{If } x \in \mathbb{R}, SIN(x) = x \Leftrightarrow x = 0$$

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



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

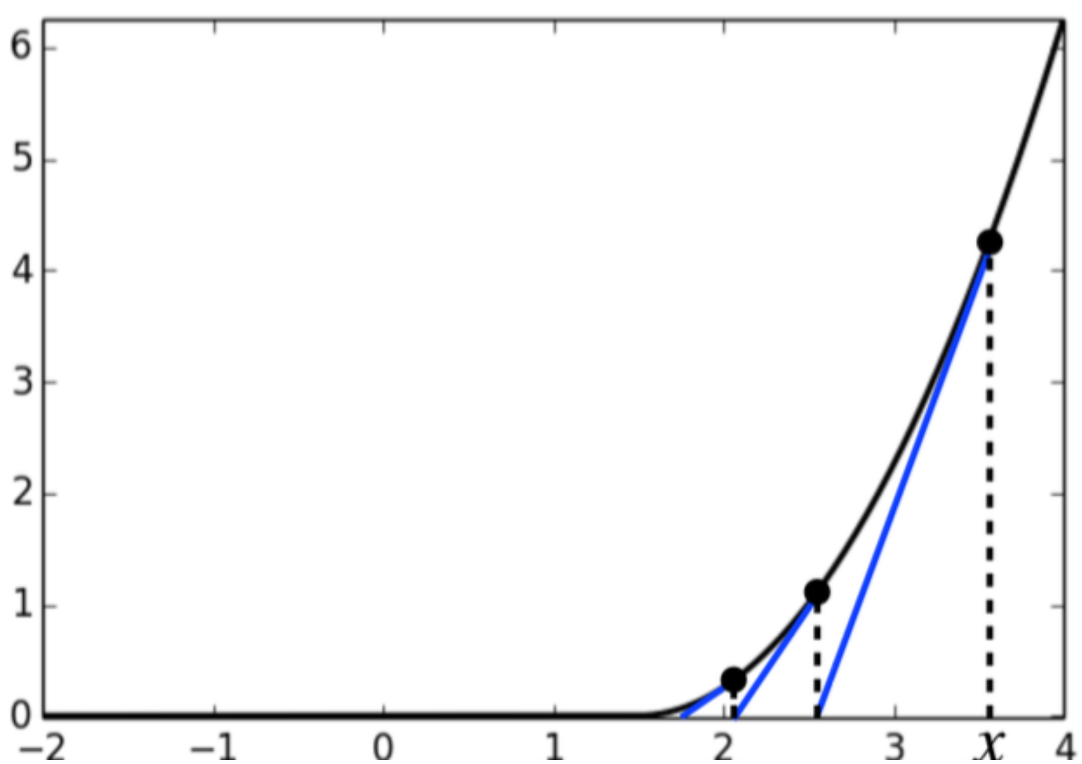
Theoretical guarantee: κ satisfiable $\Leftrightarrow R(x^*) = 0$
where x^* is a minimum point

Example κ_1 $\boxed{x \le 1.5}$

Step 1. Transform κ_1 to

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

Step 2. Minimize R_1 (local optimization)



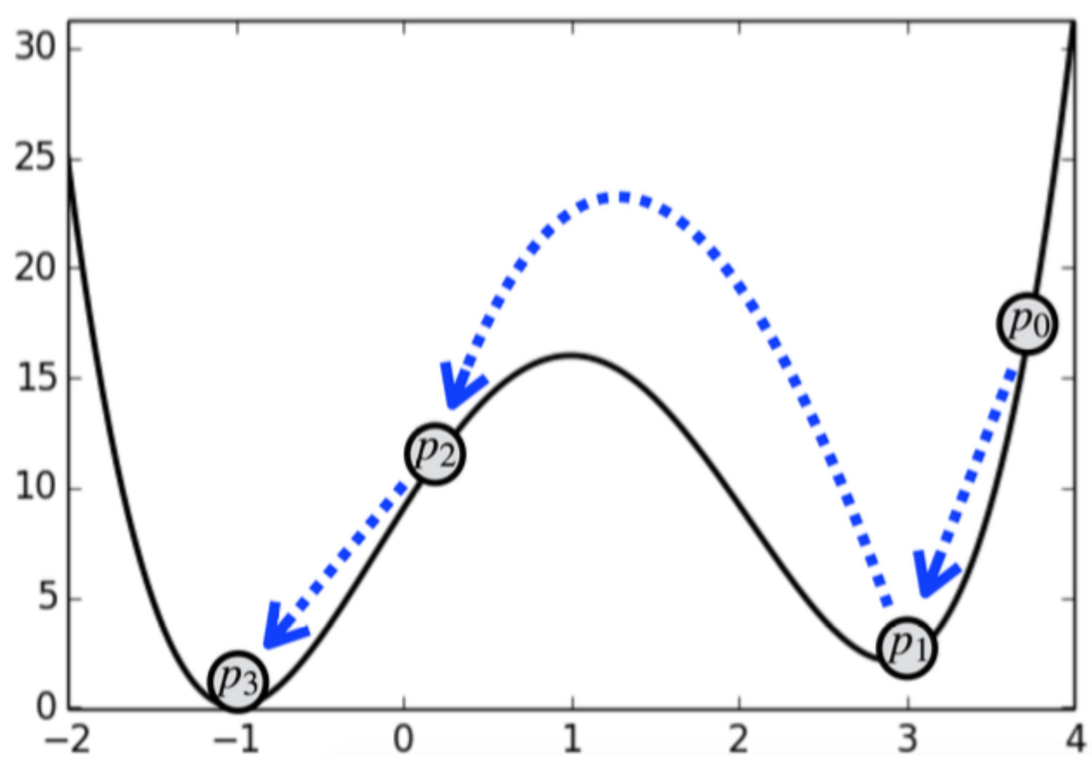
- x^* can be anything ≤ 1.5
- $R_1(x^*) = 0 \implies x^* \models R_1$

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

Quiz1: solving this floating-point constraint

$$a * a = 3 \text{ and } a >= 0$$

Quiz2: solving this floating-point constraint

$$2^x \leq 5 \wedge x^2 \geq 5 \wedge x \geq 0$$

Conclusions

- A brief introduction to satisfiability solving
- Solving floating-point satisfiability problems via function minimization
- The approach shares a similar pattern as how we tested boundary inputs in the previous lecture