中国科学技术大学
University of Science and Technology of China

# Why Program Analysis

## (Static and Dynamic Analysis)

张昱

yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

# Code Review

# Code Review



## Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

A requirement for many safety-critical systems.

# Code Review

## Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

```
double foo(double[] d) {
    int n = d.length;
    double s = 0;
    int i = 0;
    while (i<n){
    s = s + d[i];
    i = i + 1;}
    double a = s / n;
    return a;
}
```

What can we improve in this (Java) code?

# Code Review



## Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

```
double foo(double[] d) {
    int n = d.length;
    double s = 0;
    int i = 0;
    while (i<n){
    s = s + d[i];
    i = i + 1;}
    double a = s / n;
    return a;
}
```

- Naming
- Indent
- New line

```
double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;

    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum / n;

    return avg;
}
```

What can we improve in this (Java) code?

# Apple's SSL/TLS bug (22 Feb 2014, CVE-2014-1266)

- In function SSLVerifySignedServerKeyExchange (sslKeyExchange.c)

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail; …
```

Apple’s “goto fail” bug:
A security vulnerability for 2 years!

# Terminology about PA

# Terminology and Important Concepts



- Precision vs. Recall (and FP/FN/TP/TN)
- Soundness vs. Completeness
- Concrete domain vs. Abstract domain
- Accuracy vs. Precision (and conservative analysis)

# Precision vs. Recall



## FP/FN/TP/TN

| Ground Truth \ Analysis result | Positive | Negative |
|---|---|---|
| Positive | | |
| Negative | | |

# Precision vs. Recall

## FP/FN/TP/TN

**Analysis result**

| **Ground Truth** | Positive | Negative |
|---|---|---|
| Positive | TP | FN漏报 |
| Negative | FP误报 | TN |

# Precision vs. Recall

- FP/FN/TP/TN

| Ground Truth \ Analysis result | Positive | Negative |
| --- | --- | --- |
| Positive | TP | FN |
| Negative | FP | TN |

Recall:

$$\frac{|TP|}{|TP| + |FN|}$$

Precison:

$$\frac{|TP|}{|TP| + |FP|}$$

# Soundness vs. Completeness

- FP/FN/TP/TN
- Soundness vs. Completeness

**Soundness:
no FNs无漏报**

**Analysis result**

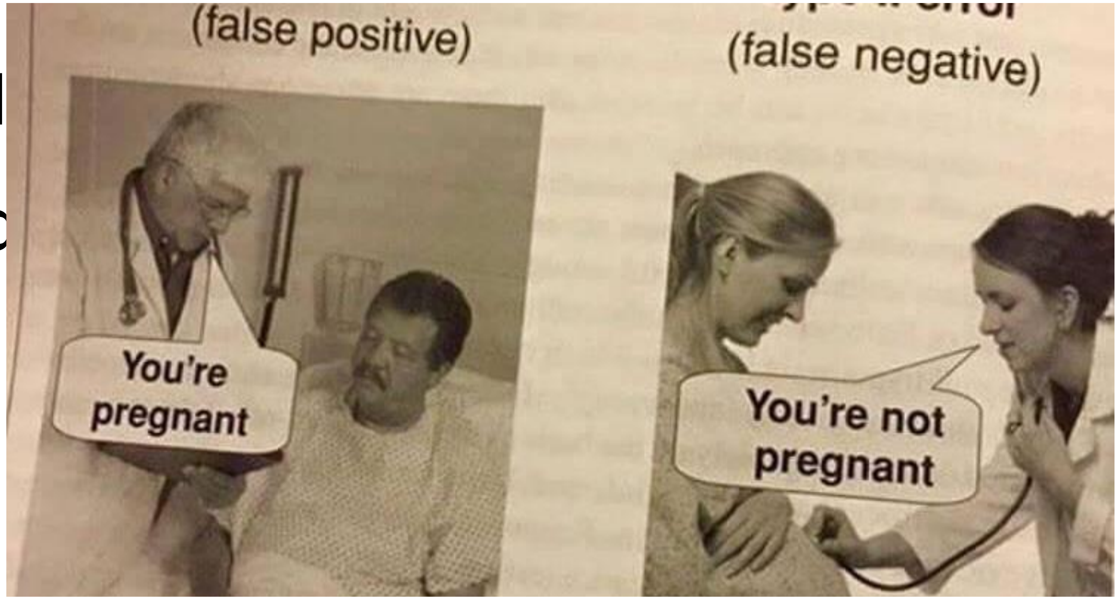
| Ground Truth | Positive | Negative |
|---|---|---|
| Positive | TP | FN |
| Negative | FP | TN |

**Completeness:
no FPs无误报**

# Soundness vs. Completeness



- FP/FN
- Sound




**Soundness: no FNs无漏报**

**Analysis result**

| Ground Truth | Positive | Negative |
|---|---|---|
| Positive | TP | FN |
| Negative | FP | TN |

**Completeness: no FPs无误报**

# Concrete domain vs. Abstract domain

- Soundness (no FNs无漏报) vs. Completeness (no FPs无误报)

| Concrete domain | Abstract domain |
| --- | --- |
| 0, 1, 2, 3, 4, … | even, odd, … |

# Accuracy vs. Precision

- Soundness (no FNs无漏报) vs. Completeness (no FPs无误报)
- Accuracy = correct estimate vs.
  Precision = small estimate

# Static vs. Dynamic Analysis

# Static vs. Dynamic Analysis

## ☐ Static Analysis

- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

* Some static analyses are unsound; dynamic analyses can be sound.

# Selecting an Abstract Domain

University of Science and Technology of China

$\langle\, x = 2;\ y = 5\, \rangle$

```
y = x++;
```

$\langle\, x = 3;\ y = 2\, \rangle$

$\langle\, x = \{3, 5, 7\};\ y = \{9, 11, 13\}\, \rangle$

```
y = x++;
```

$\langle\, x = \{4, 6, 8\};\ y = \{3, 5, 7\}\, \rangle$

⟨ x is odd; y is odd ⟩

```
y = x++;
```

⟨ x is even; y is odd ⟩

$\langle x{=}3, y{=}11\rangle, \langle x{=}5, y{=}9\rangle, \langle x{=}7, y{=}13\rangle$

```
y = x++;
```

$\langle x{=}4, y{=}3\rangle, \langle x{=}6, y{=}5\rangle, \langle x{=}8, y{=}7\rangle$

⟨ x is prime; y is prime ⟩

```
y = x++;
```

⟨ x is anything; y is prime ⟩

$\langle\, x_n = f(a_{n-1},\ldots,z_{n-1});\ y_n = f(a_{n-1},\ldots,z_{n-1})\, \rangle$

```
y = x++;
```

$\langle\, x_{n+1} = x_n + 1;\ y_{n+1} = x_n\, \rangle$

# Static vs. Dynamic Analysis



## Static Analysis

- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

## Dynamic Analysis

- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

* Some static analyses are unsound; dynamic analyses can be sound.

# Static Analysis: Examples

University of Science and Technology of China

## Type checking (also compiler optimizations)

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0.0;
  while (i<n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum / n;

  return avg;
}
```

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum / n;

  return avg;
}
```

# Static Analysis: Examples

University of Science and Technology of China

- Rule/pattern-based analysis (PMD, Findbugs, etc.)

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum / n;

  return avg;
}
```

https://github.com/pmd/pmd An extensible multilanguage static code analyzer.

https://github.com/findbugsproject/findbugs Find Bugs in Java Programs

# Dynamic Analysis: Examples

中国科学技术大学
University of Science and Technology of China

- Software testing (also monitoring and profiling)

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

A test for the avg function:

```
@Test
public void testAvg() {
  double nums =
       new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

# Static vs. Dynamic Analysis

University of Science and Technology of China

**What are the key challenges?**

# Static vs. Dynamic Analysis

**What are the key challenges?**

- **Static analysis: choose good abstractions**
  - Chosen abstraction determines cost (time and space)
  - Chosen abstraction determines precision (what information is lost)
- **Dynamic analysis: choose good representatives (tests)**
  - Chosen tests determine cost (time and space)
  - Chosen tests determine accuracy (what executions are never seen)

# Static vs. Dynamic Analysis

## Summary

### Static analysis

- Abstract domain
- Conservative due to abstraction
- Sound due to conservatism
- Slow if precise

### Dynamic analysis

- Concrete domain
- Precise no approximation
- Unsound, does not generalize
- Slow if exhaustive

# Google: Why developers don’t use static analysis?

- **Not integrated** into the developer’s workflow.
- Reported **issues** are **not actionable**.
- Developers **do not trust the results** (FPs).
- Fixing an issue **is too expensive** or risky.
- Developers **do not understand** the reported **issues**.
- **Issues** theoretically possible but **don’t manifest in practice.**

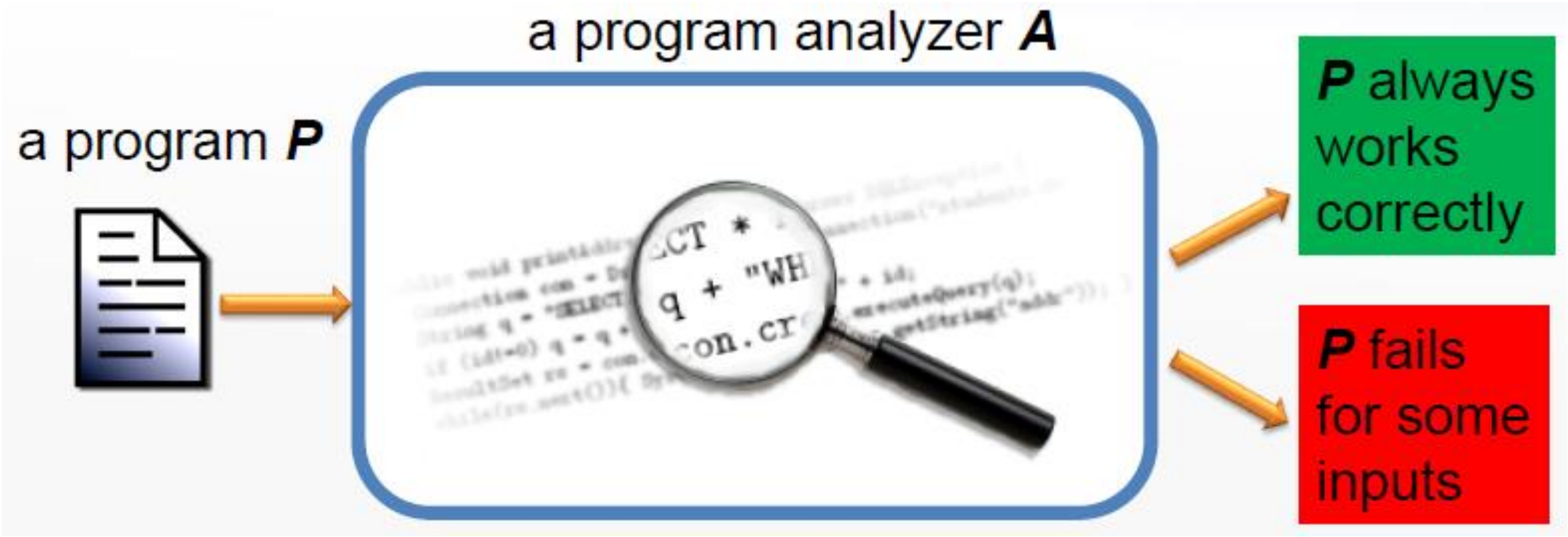
*“Produce **less than 10% effective false positives**. Developers should feel the check is pointing out an actual issue at least 90% of the time.”*

[ICSE 2013] Google: Why developers don’t use static analysis?

# Program Analyzer

# Programs that reason about programs

- ☐ **Soundness(可靠性): don’t miss any errors**
- ☐ **Completeness(完备性): don’t raise false alarms**
- ☐ **Termination(终止性): always give an answer**



# Why are the Answers Interesting?



- Increase efficiency
  - Resource usage
  - Optimization

- Ensure correctness
  - Verify behavior
  - Catch bugs early
- Support program understanding
- Enable refactorings

# Rice's theorem, 1953

- H.G. Rice: *Classes of recursively enumerable sets and their decision problem*

- Rice定理：Any nontrivial property of the behavior of programs in a Turing-complete language is undecidable!

- 
  **递归可枚举语言的所有非平凡(nontrival)性质都是不可判定的**
  平凡性质：要么对全体程序都为真，要么对全体程序都为假
  非平凡性质：所有不平凡的性质

# Approximation



- ***Approximate* answers may be decidable!**
  - Output yes/no => output yes/no/unknown
- **The approximation must be *conservative***
- **More subtle approximations if not only yes/no**
  - E.g. memory usage, pointer targets

# The Engineering Challenge

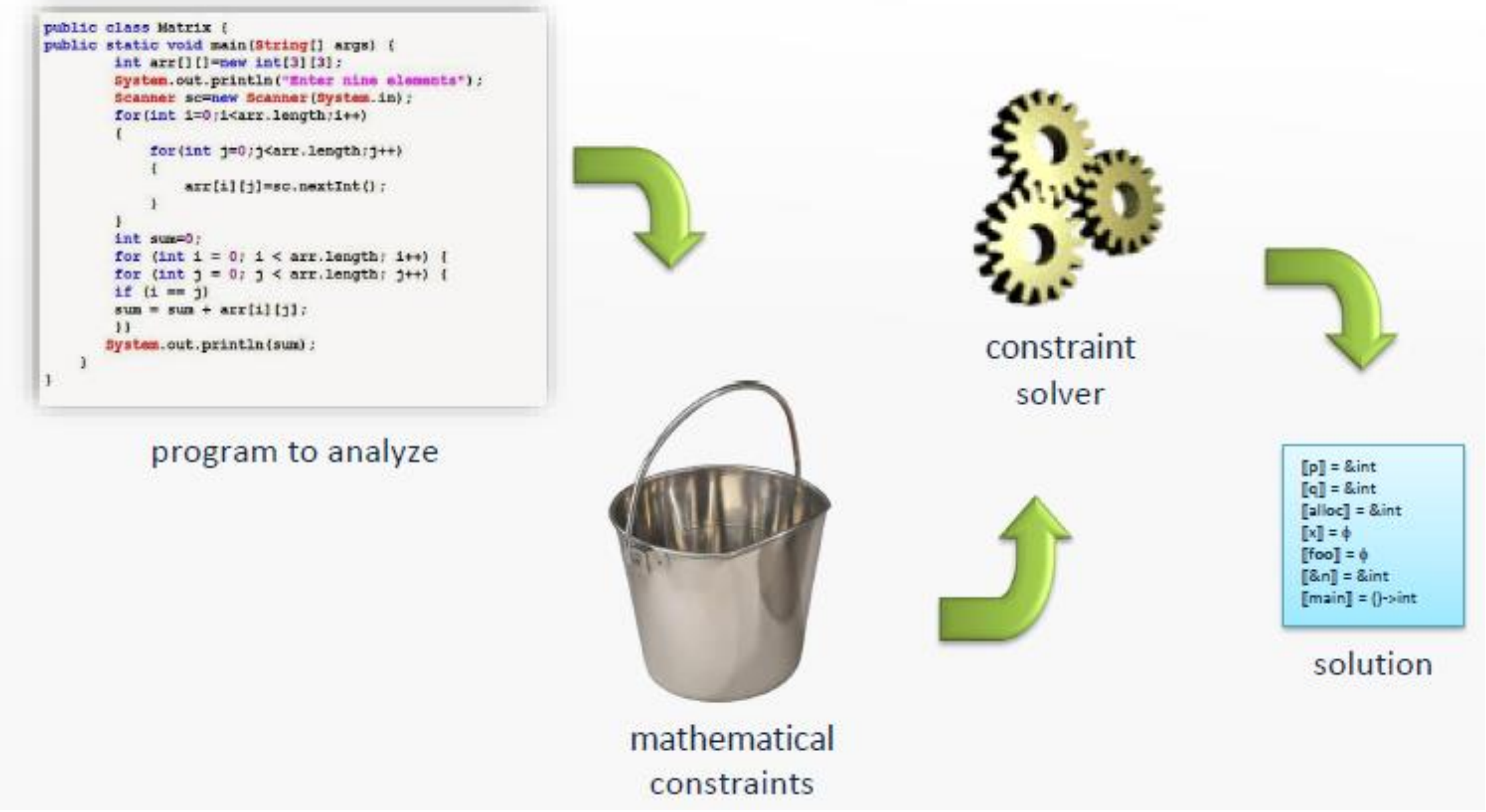
University of Science and Technology of China

- A correct but trivial approximation algorithm may just give the useless answer every time
- The engineering challenge is to give the useful answer often enough to fuel the client application
- … and to do so within reasonable time and space
- Hard (but fun) part of static analysis

# A Constraint-based Approach



- Conceptually separates the **analysis specification** from **algorithmic** aspects and **implementation** details



# Challengeing Features in Modern PLs



- Higher-order functions
- Mutable records or objects, arrays
- Integer or floating-point computations
- Dynamic dispatching
- Inheritance
- Exceptions
- Reflection
- …

# Thanks