软件分析与验证前沿

苏亭

软件科学与技术系
软件工程学院

Who am I?

- 苏亭(教授/博导)，软件科学与技术系，软件工程学院
- 个人主页：http://tingsu.github.io
- 研究方向
  - 软件分析、测试、验证、安全
  - 软件与系统的质量和安全保障
- 教育/工作背景
  - ECNU (B.S. & PhD) -» UCD (Visiting PhD) -» NTU (Postdoc) -» ETH (Postdoc) -» ECNU (Professor)
- 联系方式
  - 理科楼B1103，tsu@sei.ecnu.edu.cn

Course Information

- 课程目标
  1. 掌握软件分析、测试与验证的基本理论、技术原理和实践
  2. 了解相关的前沿研究进展
- 课程信息
  1. 课程时间：每周五晚上6:00-7:30
  2. 课程讲义：飞书群
  3. 上课地点：理科楼A228
  4. 考核形式：出勤*20%、平时课堂表现*30%、实验Labs（阅读研究论文、工具调研等）*50%

课程网站：https://tingsu.github.io/files/courses/pa2024.html
实验课仓库：https://github.com/ecnu-sa-labs/ecnu-sa-labs
实验课发布/提交平台：github classgroom
助教：明孟立（硕士研究生）、黄杉（博士研究生）、麻恩泽（博士研究生）

Course Information

- 课程目标
  1. 掌握软件分析、测试与验证的基本
  2. 了解相关的前沿研究进展
- 课程信息
  1. 课程时间：每周五晚上6:00-7:30
  2. 课程讲义：飞书群
  3. 上课地点：理科楼A228
  4. 考核形式：出勤*20%、平时课堂表现*
     工具调研等）*50%

课程网站：https://tingsu.github.io/files/courses/pa2024.html
实验课仓库：https://github.com/ecnu-sa-labs/ecnu-sa-labs
实验课发布/提交平台：github classgroom
助教：明孟立（硕士研究生）、黄杉（博士研究生）、麻恩泽（博士研究生）

Course History

- Pre 2022 - 软件分析与验证工具 (蒲戈光、郭建)
- 2022- 软件分析与验证前沿 (苏亭)

学生评价

第一年的课程是非常好的尝试，讲解清晰，选题也适宜，望老师能把这门课越办越好

系统性地学习到了很多软件分析的知识，老师上课细心讲解，让复杂的知识易于听懂，课堂 slides 制作精良，重点突出。助教认真负责，十分配合老师工作。整体课堂非常融洽，最后的汇报环境充分锻炼了我的能力。希望此课继续开设下去！

课程内容很扎实，感觉可以多提供学生自己可以课后探索一些学习的网站。其他都非常棒！

老师上课很认真负责，pre的奖励机制挺有效的（起码对我们组来说）！就是可能有几周作业比较多，可以平摊到其他周或者适当减少一点点

小广告

- **欢迎学有余力的同学来我们研究小组做科研实习**
- 良好的科研环境、有挑战的课题、有研究生指导、有科研津贴、推荐保研/企业实习、出国参加国际会议和短期访问…… https://tingsu.github.io/files/application.html

- 黄杉（19级本科、共同一作发表CCF-A类论文1篇、优秀本科毕业论文、本院直博、访问美国）
- 文贺（19级本科、参与发表CCF-A类论文1篇、ACM杰出论文奖、荷兰阿姆斯特丹大学硕士）
- 王可以（20级本科、参与软件安全方向课题研究、本院直研）
- 李丽坤（21级本科、本院直研）、史晓磊（21级本科） 、李恩兆（21级本科）……
- 我们实验室所有同学信息：https://tingsu.github.io/files//people.html

What is Program Analysis?

SEGMENT

What is Program Analysis?

What you probably know

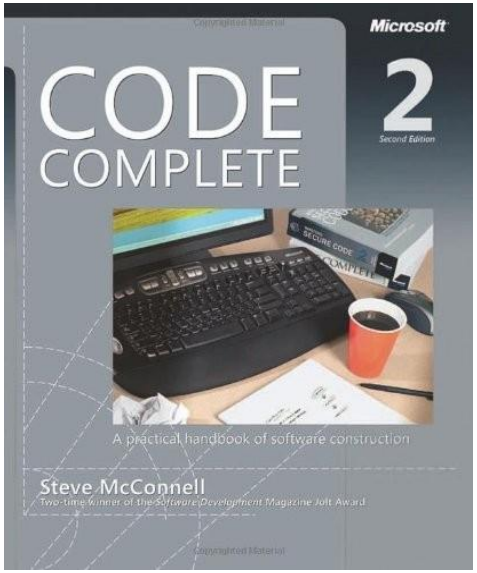
- Manual testing or semi-automated testing:
  - JUnit, Selenium, etc.
- Manual "analysis" of programs:
  - Code inspection, debugging, etc.

***Focus of this course*:**

Automated program analysis

Why Do We Need it?

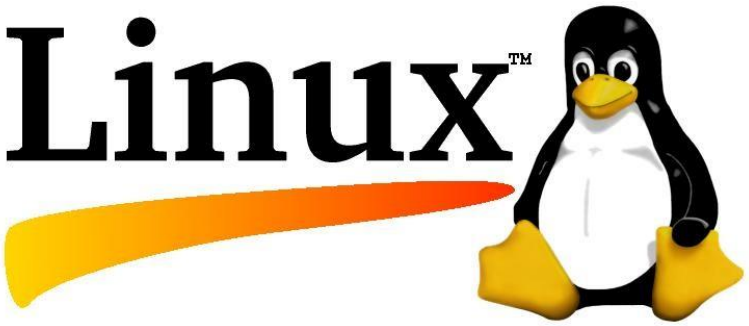
- All software has bugs
- Bugs are hard to find
- Bugs cause serious harm



0.5-25/KLoC in delivered software

Why Do We Need it?

- All software has bugs
- Bugs are hard to find
- Bugs cause serious harm



1.5 years to find a bug [Palix2011]

Why Do We Need it?

- All software has bugs
- Bugs are hard to find
- Bugs cause serious harm

Ariane 5

Northeast blackout

Therac-25

The Ariane Rocket Disaster (1996)

https://www.youtube.com/watch?v=PK_yguLapgA&t=50s

Post Mortem

- Caused due to numeric overflow error
  - Attempt to fit 64-bit format data in 16-bit space
- Cost
  - $370M's for loss of mission
  - Multi-year setback to the Ariane program
- Read more at https://www.bugsnag.com/blog/bug-day-ariane-5-disaster

Security Vulnerabilities

- Exploits of errors in programs
- Widespread problem
  - Moonlight Maze (1998)
  - Code Red (2001)
  - Titan Rain (2003)
  - Stuxnet Worm（蠕虫病毒；震网）
- Getting worse …

2011 Mobile Threat Report (Lookout™ Mobile Security)

- 0.5-1 million Android users affected by malware in first half of 2011
- 3 out of 10 Android owners likely to face web-based threat each year
- Attackers using increasingly sophisticated ways to steal data and money

What is Program Analysis?

- Discover useful *facts* about programs

- Broadly classified into three kinds:
  - Static (compile-time)
  - Dynamic (execution-time)
  - Hybrid (combining dynamic and static)

Static vs. Dynamic Analysis

- **Static**
  - Infer facts by inspecting *source or binary code*
  - Typically:
    - Consider *all* inputs
    - *Overapproximate* possible behavior

E.g., compilers, lint-like tools

- **Dynamic**
  - Infer facts by monitoring *program executions*
  - Typically:
    - Consider *current* input
    - *Underapproximate* possible behavior

E.g., automated testing tools, profilers

Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out = "maybe";
console.log(out);
```

What are the possible outputs?

Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out ="maybe";//infeasible path
console.log(out);
```

Overapproximation: ”yes”, ”no”, ”maybe”

- Consider all paths (that are feasible based on limited knowledge)

Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out ="maybe";//infeasible path
console.log(out);
```

Underapproximation: ”yes”

- Execute the program once

Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out ="maybe";//infeasible path
console.log(out);
```

Sound and complete: ”yes”, ”no”

- For this example: Can explore both feasible paths

Another Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out = r *2;
console.log(out);
```

What are the possible outputs?

Another Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out = r *2;
console.log(out);
```

Overapproximation: Any value

- Consider all paths (that are feasible based on limited knowledge about `random()`)

Another Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out = r *2;
console.log(out);
```

Underapproximation:

Some number in [0,2), e.g., 1.234

- Execute the program once

Another Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out = r *2;
console.log(out);
```

Sound and complete?

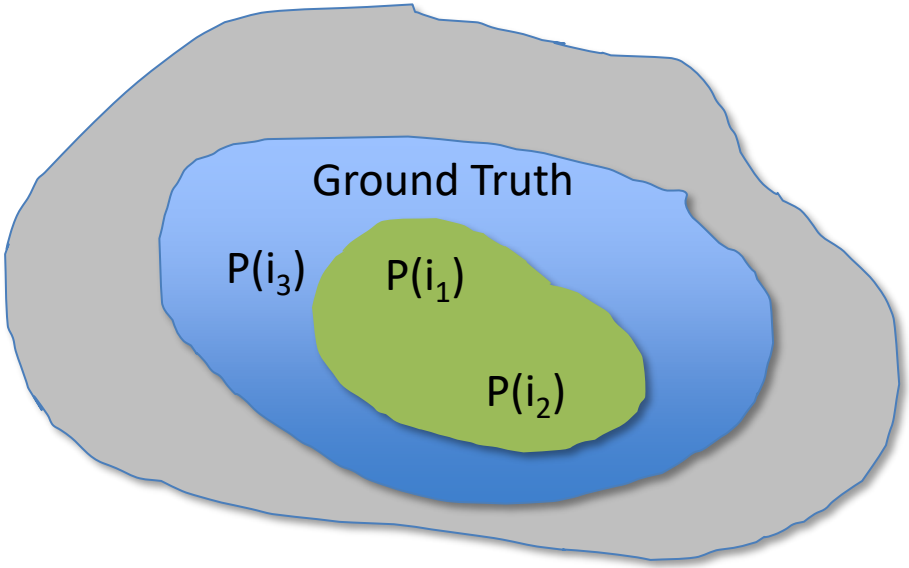
- Exploring all possible outputs:
  Practically impossible
- This is the case for most real-world programs

Terminology

- Over-approximation *v.s.* Under-approximation
- Soundness *v.s.* Completeness
- False positives *v.s.* False negatives
- Precision *v.s.* Recall

Under- & Over-approximation

Program P, Input i, Behavior P(i)



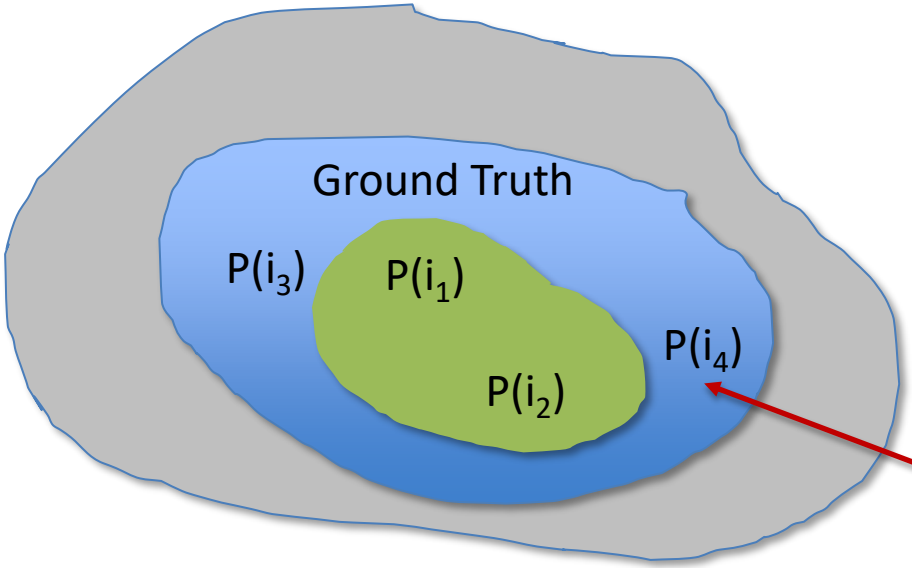
All possible behaviors (what we want, ideally)

Under-approximation (e.g., testing, dynamic analysis)

Over-approximation (most static analysis)

Under- & Over-approximation

Program P, Input i, Behavior P(i)



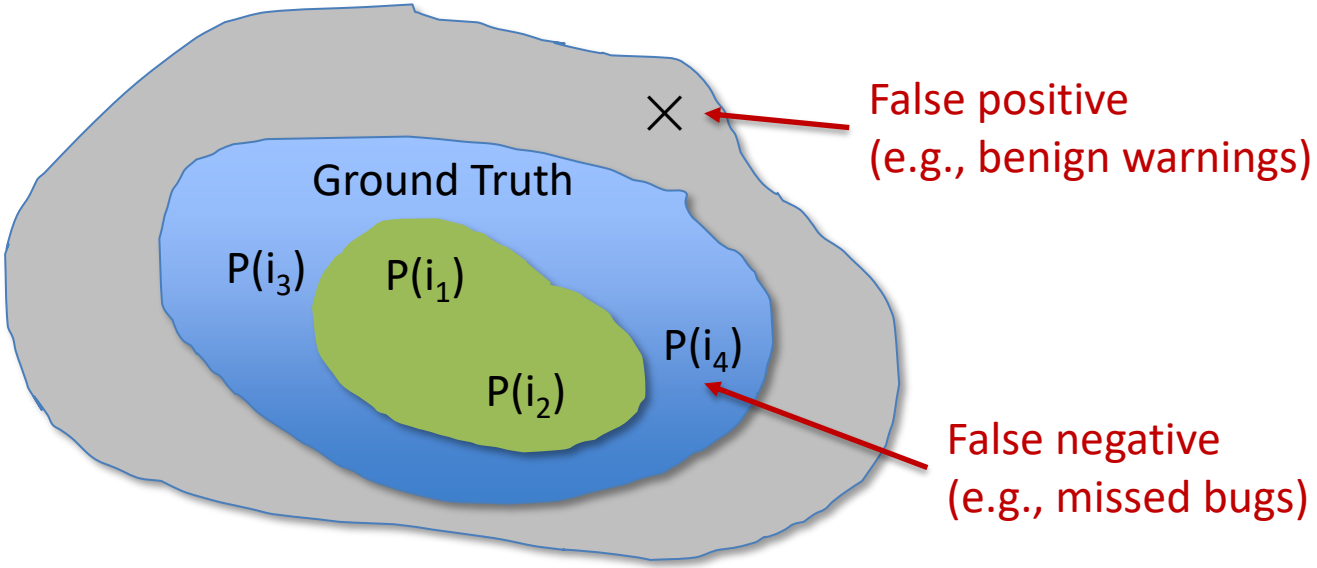
All possible behaviors (what we want, ideally)

Under-approximation (e.g., testing, dynamic analysis)

Over-approximation (most static analysis)

Under- & Over-approximation

Program P, Input i, Behavior P(i)



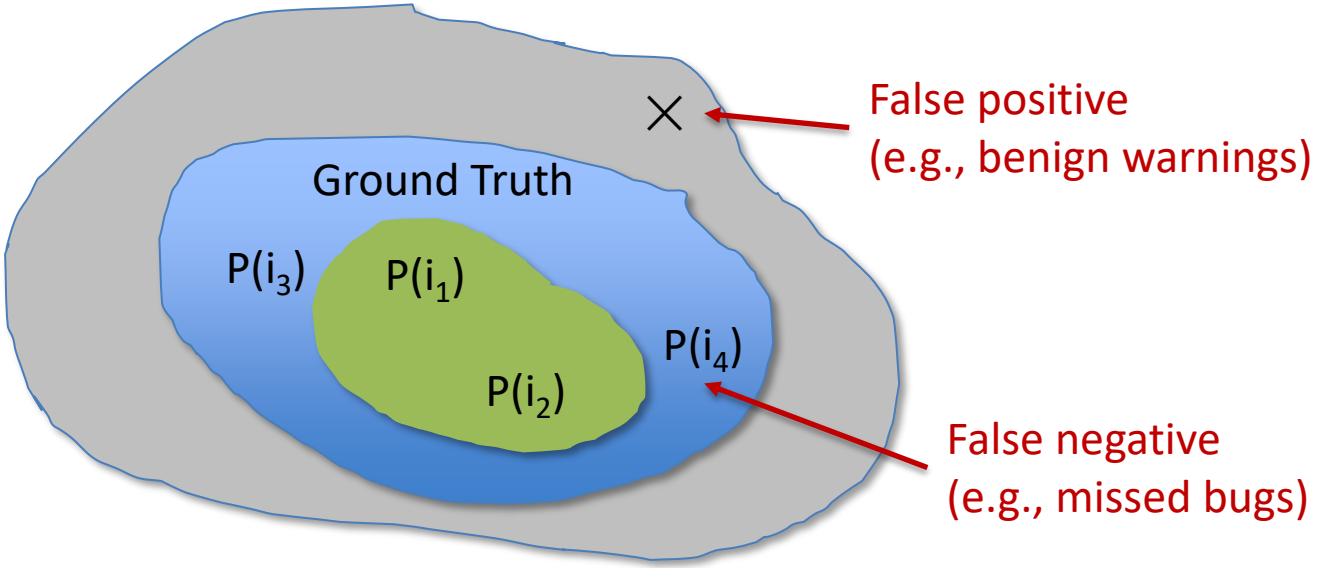
All possible behaviors (what we want, ideally)

Under-approximation (e.g., testing, dynamic analysis)

Over-approximation (most static analysis)

Soundness & Completeness

Program P, Input i, Behavior P(i)



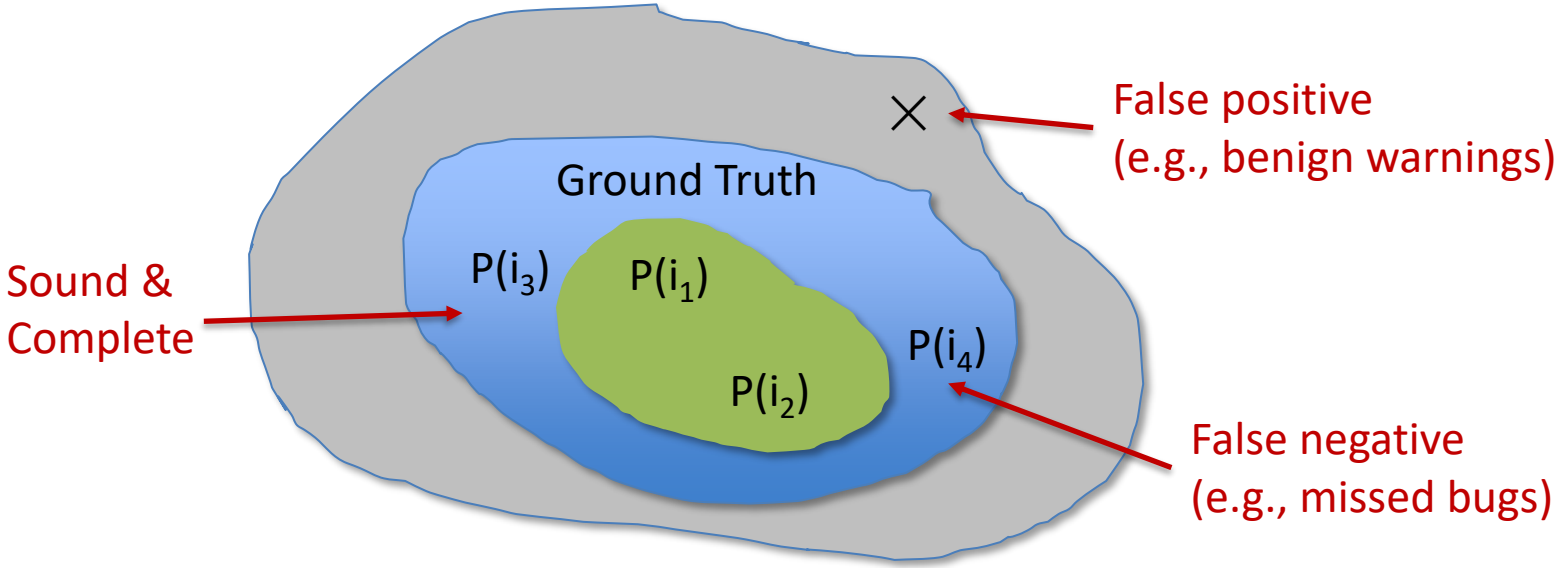

All possible behaviors (what we want, ideally)


Under-approximation (e.g., testing, dynamic analysis)


Over-approximation (most static analysis)

Soundness & Completeness

Program P, Input i, Behavior P(i)



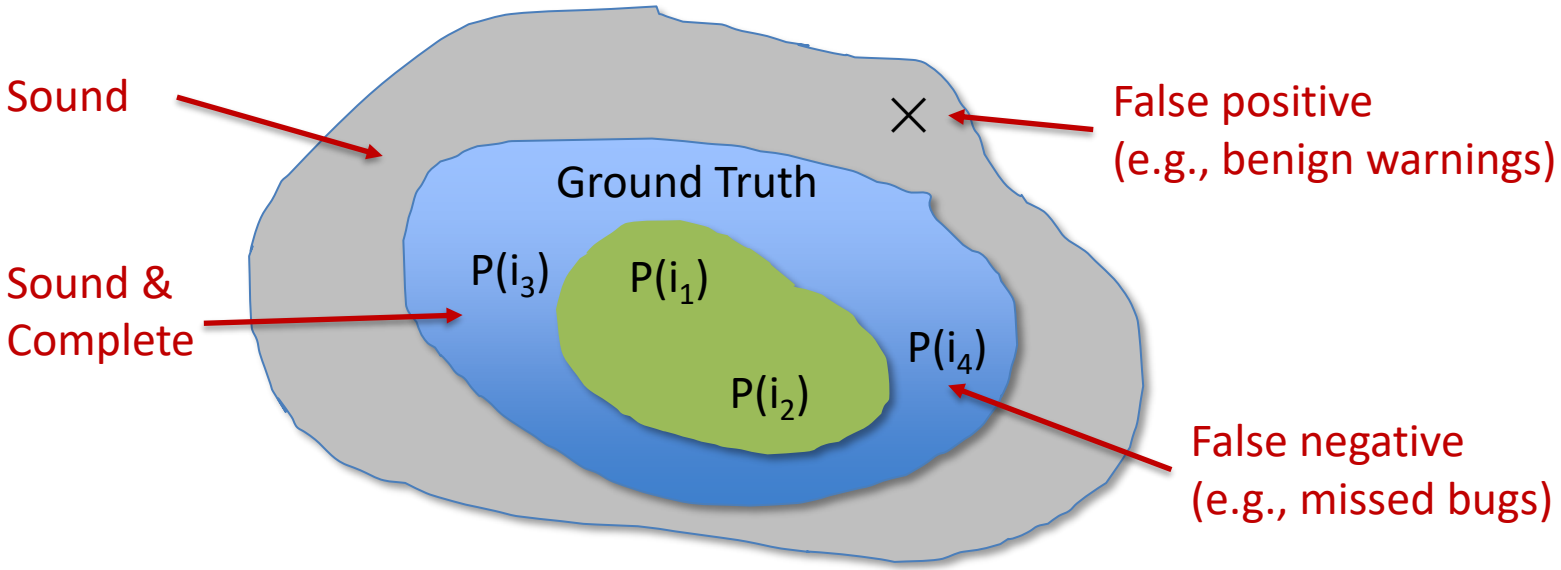
All possible behaviors (what we want, ideally)

Under-approximation (e.g., testing, dynamic analysis)

Over-approximation (most static analysis)

Soundness & Completeness

Program P, Input i, Behavior P(i)



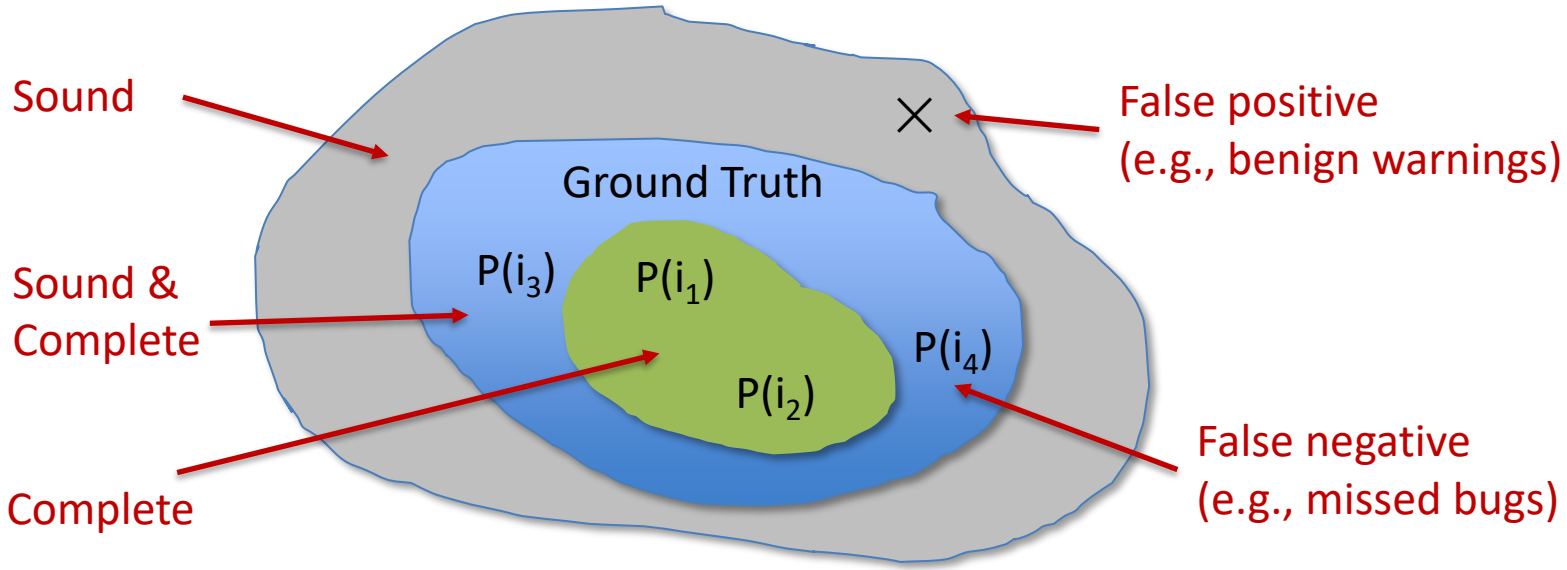
All possible behaviors (what we want, ideally)

Under-approximation (e.g., testing, dynamic analysis)

Over-approximation (most static analysis)

Soundness & Completeness

Program P, Input i, Behavior P(i)



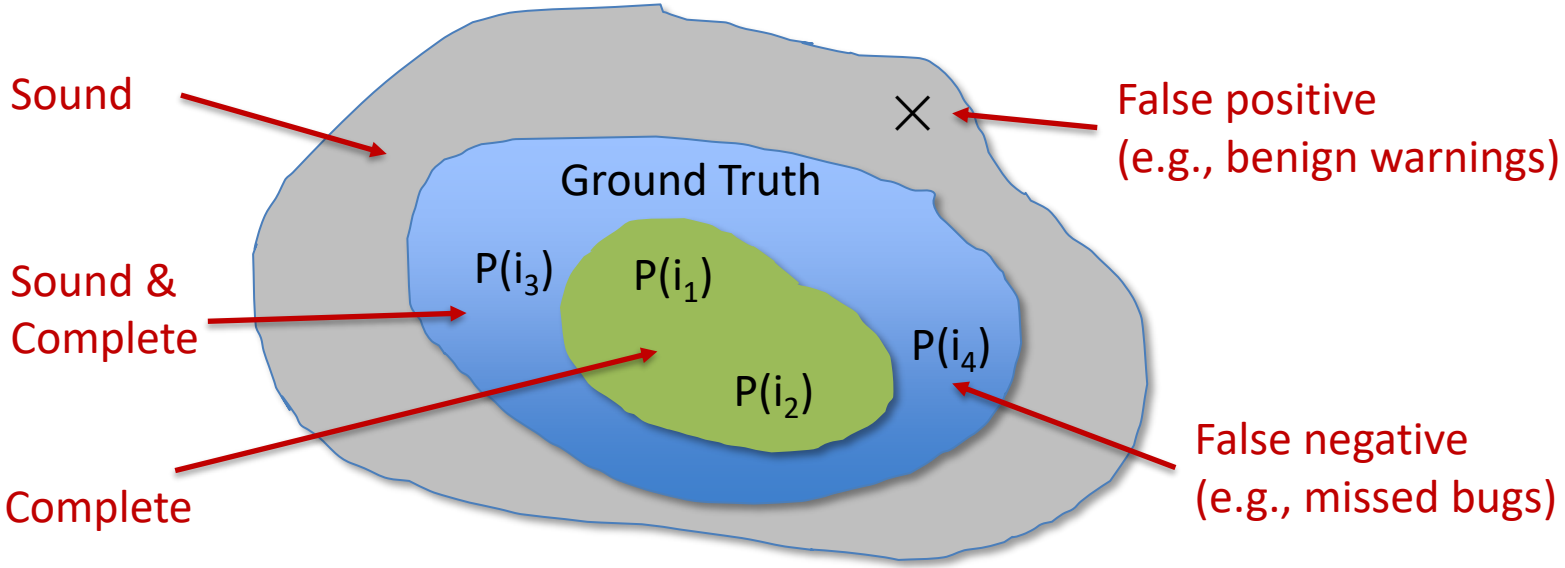
All possible behaviors (what we want, ideally)

Under-approximation (e.g., testing, dynamic analysis)

Over-approximation (most static analysis)

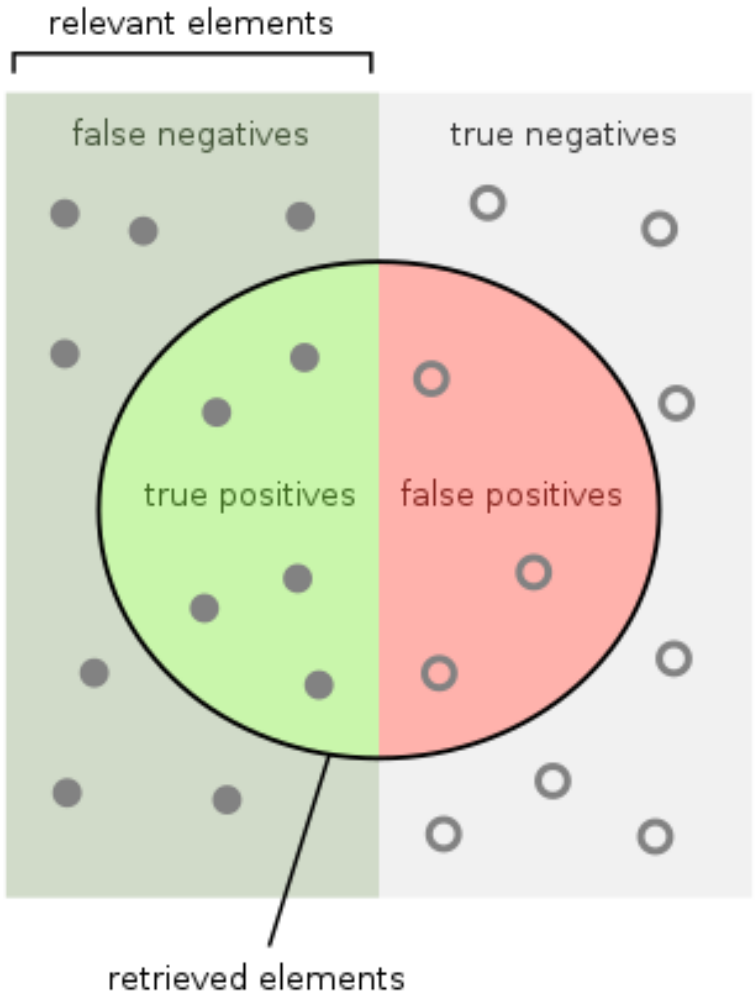
Soundness & Completeness

Program P, Input i, Behavior P(i)



- 妥协soundness => unsound, 会产生false negatives
- 妥协completeness => incomplete，会产生false positives

Precision & Recall



How many retrieved items are relevant?

Precision = $\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$

How many relevant items are retrieved?

Recall = $\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$

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

Revist the Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out = "maybe";
console.log(out);
```

What are the possible outputs?

Revist the Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out ="maybe";//infeasible path
console.log(out);
```

Overapproximation: ”yes”, ”no”, ”maybe”

- Consider all paths (that are feasible based on limited knowledge)

Revist the Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out ="maybe";//infeasible path
console.log(out);
```

Underapproximation: ”yes”

- Execute the program once

Revist the Example

```
//JavaScript
var r =Math.random();//value in [0,1)
var out ="yes";
if(r <0.5)
  out ="no";
if(r ===1)
  out ="maybe";//infeasible path
console.log(out);
```

Sound and complete: ”yes”, ”no”

- For this example: Can explore both feasible paths

Example: Program Invariants

An invariant at the end of the program is (z == c) for some constant c. What is c?

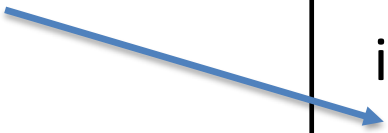
```
int p(int x) { return x * x; }

void main() {
    int z;
    if  (getc() == 'a')
        z = p(6) + 6;
    else
        z = p(-7) – 7;

}
```

z = ?

Example: Program Invariants

An invariant at the end of the program is (z == c) for some constant c. What is c?

Disaster averted!



```
int p(int x) { return x * x; }

void main() {
    int z;
    if (getc() == 'a')
        z = p(6) + 6;
    else
        z = p(-7) – 7;

  if (z != 42)
      disaster();
}
```

z = 42

Discovering Invariants By Dynamic Analysis

(z == 42) *might be* an invariant

(z == 30) is *definitely not* an invariant

```
int p(int x) { return x * x; }

void main() {
    int z;
    if  (getc() == 'a')
        z = p(6) + 6;
    else
        z = p(-7) – 7;

  if (z != 42)
      disaster();
}
```

z = 42

Discovering Invariants By Static Analysis

is definitely

(z == 42) ~~*might be*~~ an invariant

(z == 30) is *definitely not* an invariant

```
int p(int x) { return x * x; }

void main() {
    int z;
    if  (getc() == 'a')
        z = p(6) + 6;
    else
        z = p(-7) – 7;

  if (z != 42)
      disaster();
}
```

z = 42

QUIZ: Dynamic vs. Static Analysis

Match each box with its corresponding feature.

	Dynamic	Static
Cost		
Effectiveness		

A. Unsound (may miss errors)

B. Proportional to program's execution time

C. Proportional to program's size

D. Incomplete (may report spurious errors)

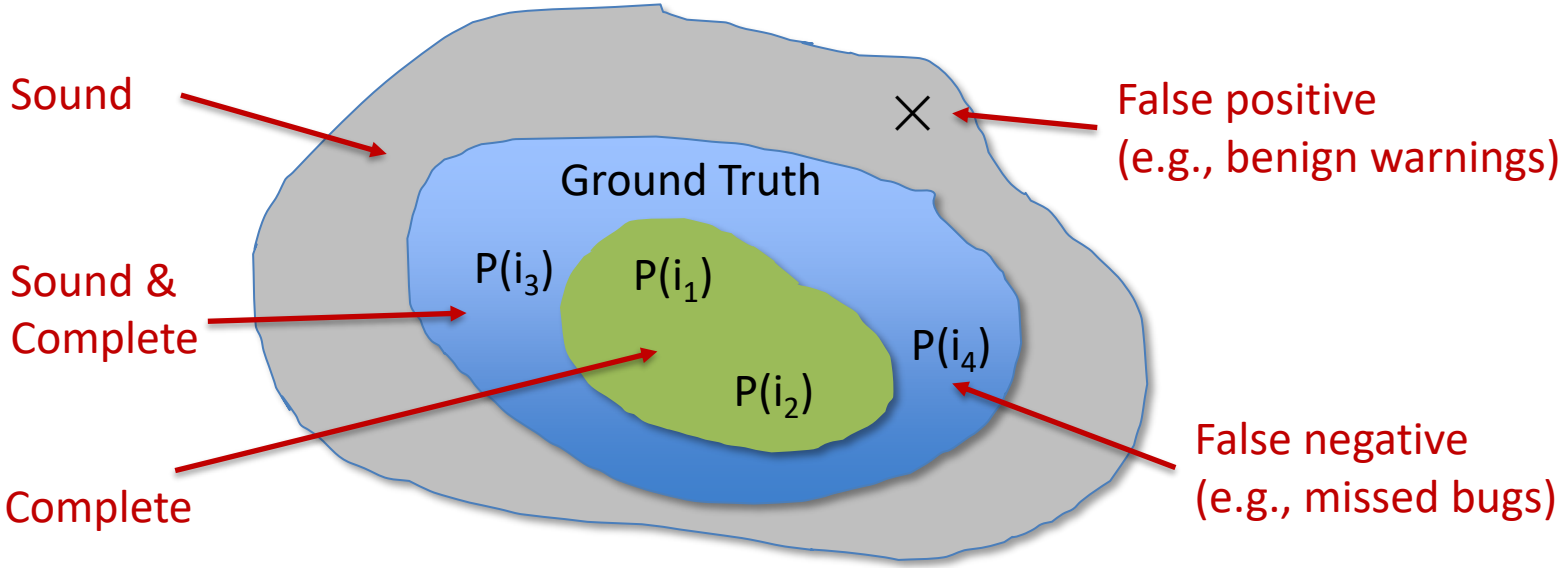
QUIZ: Dynamic vs. Static Analysis

Match each box with its corresponding feature.

	Dynamic	Static
Cost	B. Proportional to program's execution time	C. Proportional to program's size
Effectiveness	A. Unsound (may miss errors)	D. Incomplete (may report spurious errors)

Recall: Soundness & Completeness

Program P, Input i, Behavior P(i)



- 妥协soundness => unsound, 会产生false negatives
- 妥协completeness => incomplete， 会产生false positives

Undecidability of Program Properties

- Can program analysis be sound and complete?
  - Not if we want it to terminate!
- Questions like “is a program point reachable on some input?” are undecidable
- Designing a program analysis is an art
  - Tradeoffs dictated by consumer

Why Take This Course?

- Learn methods to improve software quality
  - reliability, security, performance, etc.
- Become a better software developer/tester
- Build specialized tools for software analysis, testing and verification
- Finding Jobs & Do research

Who Needs Program Analysis?

Three primary consumers of program analysis:

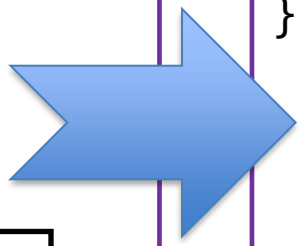
- Compilers
- Software Quality Tools
- Integrated Development Environments (IDEs)

Compilers

- Bridge between high-level languages and architectures
- Use program analysis to generate efficient code

```
int p(int x) { return x * x; }
void main(int arg) {
   int z;
   if (arg != 0)
      z = p(6) + 6;
   else
      z = p(-7) - 7;

 print (z);
}
```

z = 42



```
int p(int x) { return x * x; }
void main() {
   print (42);
}
```

- Runs faster
- More energy-efficient
- Smaller in size

Software Quality Tools

- Primary focus of this course
- Tools for testing, debugging, and verification
- Use program analysis for:
  - Finding programming errors
  - Proving program invariants
  - Generating test cases
  - Localizing causes of errors
  - …

```
int p(int x) { return x * x; }

void main() {
    int z;
    if  (getc() == 'a')
        z = p(6) + 6;
    else
        z = p(-7) – 7;

  if (z != 42)
      disaster();
}
```

z = 42

Example: Software Quality Tools

- Static Program Analysis

Suspicious error patterns
Lint, SpotBugs, Coverity

Memory leak detection
Facebook Infer

Checking API usage rules
Microsoft SLAM

Verifying invariants
ESC/Java

The Coverity Platform - From a Developer's Perspective:
https://www.youtube.com/watch?v=_Vt4niZfNeA

Example: Software Quality Tools

- Dynamic Program Analysis

Array bound checking
Purify

Datarace detection
Eraser

Memory leak detection
Valgrind

Finding likely invariants
Daikon

Integrated Development Environments

- Examples: Eclipse and VS Code
- Use program analysis to help programmers:
  - Understand programs
  - Refactor programs
    - Restructuring a program without changing its behavior
- Useful in dealing with large, complex programs

实验课-Labs (Tentative)

Labs	Lab Title
lab1	Introduction to Software Analysis
lab2	The LLVM Framework
lab3	Random Testing / Fuzzing
lab4	Property-based Testing
lab5	Delta Debugging
lab6	Dataflow Analysis
lab7	Pointer Analysis
lab8	Dynamic Symbolic Execution

实验课仓库：https://github.com/ecnu-sa-labs/ecnu-sa-labs

Course Topics (Tentative)

- LLVM & Compilers
- Random Testing & Fuzzing
- Metamorphic Testing & Property-based Testing
- Delta debugging
- Data-flow Analysis
- Pointer Analysis
- Taint Analysis
- Symbolic Execution
- SMT Theory
- Formal verification (model checking)
- Security Analysis
- ……

Supplementary Materials

- Static Program Analysis, Anders Møller and Michael I. Schwartzbach https://cs.au.dk/~amoeller/spa/

- Mayur Naik (University of Pennsylvania)
- Michael Pradel (University of Stuttgart)
- 南京大学（李樾、 谭添老师）的程序分析课程
- 北京大学（熊英飞老师）的程序分析课程
- 国防科大（陈立前老师）的程序分析课程

What Have We Learned?

- What is program analysis?
- Dynamic vs. static analysis: pros and cons
- Terminologies in program analysis
- Undecidability => program analysis cannot ensure
  termination + soundness + completeness
- Why we need to learn program analysis?

Readings

- What is static program analysis?
  - https://matt.might.net/articles/intro-static-analysis/
- What is soundness (in static analysis)?

- http://www.pl-enthusiast.net/2017/10/23/what-is-soundness-in-static-analysis/

- *"Finding and Understanding Defects in Static Analyzers by Constructing Automated Oracles*". Proc. ACM Softw. Eng. 1(FSE): 1656-1678 (2024)