Program Analysis: Introduction and Basics

Course page:
http://software-lab.org/teaching/winter2020/pa/

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

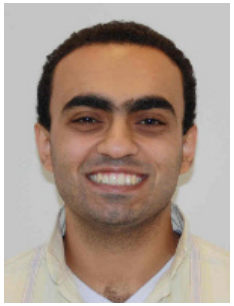
Winter 2020/2021

About Me: Michael Pradel

- **Since 9/2019: Full Professor at University of Stuttgart**
- **Before Stuttgart**
  - Studies at TU Dresden, ECP (Paris), and EPFL (Lausanne)
  - PhD at ETH Zurich, Switzerland
  - Postdoctoral researcher at UC Berkeley, USA
  - Assistant Professor at TU Darmstadt
  - Sabbatical at Facebook, Menlo Park, USA

About the Software Lab






- **My research group since 2014**
- **Focus: Tools and techniques for building reliable, efficient, and secure software**
  - Program testing and analysis
  - Machine learning, security
- **Thesis and job opportunities**

Plan for Today

- **Introduction**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization**
  - Lectures, exercises, course project
  - Final exam
- **Foundations**
  - Grammars, ASTs, CFGs, etc.

Program Testing & 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:

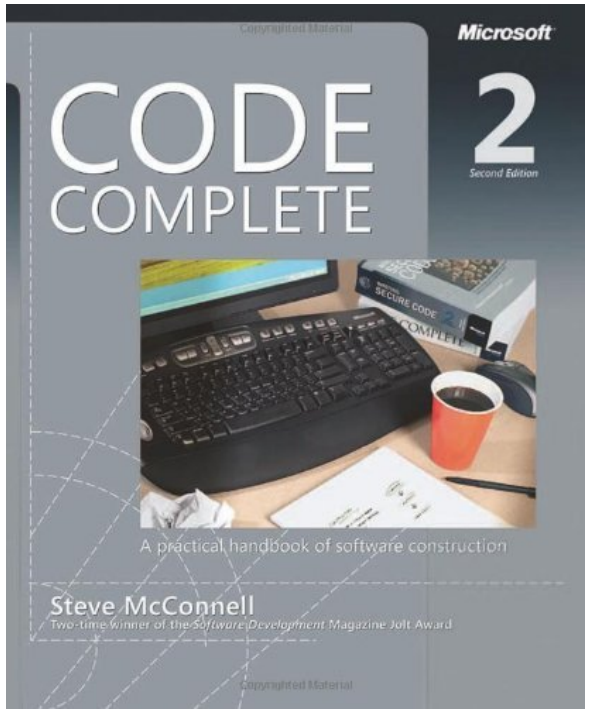
Automated testing and program analysis

Why Do We Need It?

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

Why Do We Need It?

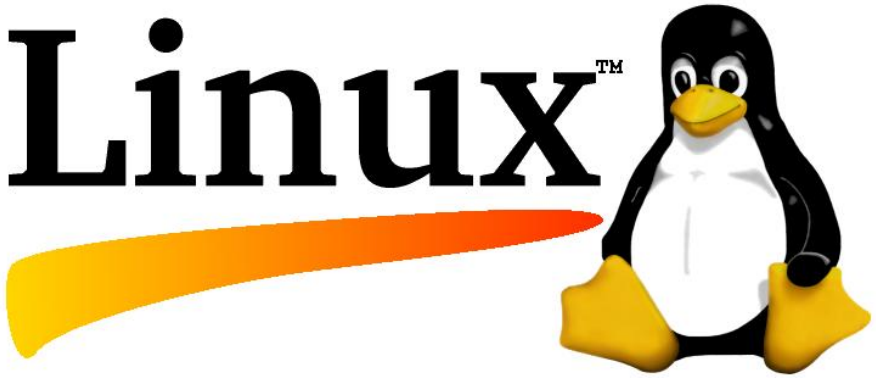
- All software has bugs
- Bugs are hard to find
- Bugs cause serious harm



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

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

Static vs. Dynamic Analysis

Static

- Analyze source code, byte code, or binary
- Typically:
  - Consider all inputs
  - Overapproximate possible behavior

Dynamic

- Analyze program execution
- Typically:
  - Consider current input
  - Underapproximate possible behavior

Static vs. Dynamic Analysis

Static

- Analyze source code, byte code, or binary
- Typically:
  - Consider all inputs
  - Overapproximate possible behavior

E.g., compilers, lint-like tools

Dynamic

- Analyze program execution
- Typically:
  - Consider current input
  - Underapproximate possible behavior

E.g., automated testing, 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

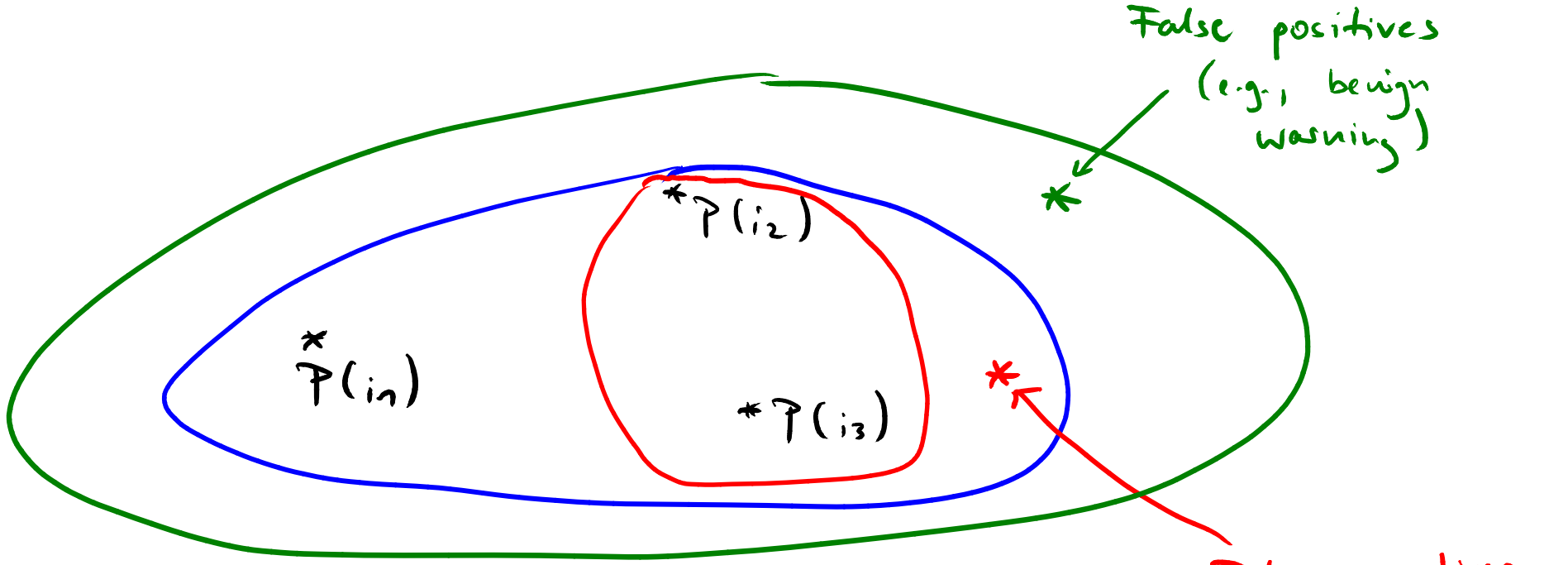
Another Example

```
// 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

Program P, Input i, Behavior $P(i)$



All possible behaviors (what we want, ideally)

Underapproximation (e.g., testing, dynamic analysis)

Overapproximation (most static analyses)

Test Generation

- **Dynamic analysis:**
  Requires input to run the program
- **Test generation:**
  Creates inputs automatically
- **Examples**
  - Generate JUnit tests:
    Input = sequence of method calls
  - UI-level test generation:
    Input = sequence UI events
  - Fuzz-test a compiler: Input = program

How Does All This Help Me?

- **Use program analysis tools**
  - Improve the quality of your code
- **Understand program analysis**
  - Better understanding of program behavior
- **Create your own tools**