Program Analysis – Lecture 1

Introduction and Basics

Join the course on Ilias! See link on http://software-lab.org/teaching/winter2019/pa/

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2019/2020
About Me: Michael Pradel

- Since 9/2019: Full Professor at University of Stuttgart

- Before
  - 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**
  - Course project
  - Mid-term and 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

Input ➔ Program ➔ Output
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**
- Analyse 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

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

Dynamic

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

E.g., compilers, lint-like tools
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"; // infeasible path
console.log(out);

Quiz: What are the possible outputs?

https://ilias3.uni-stuttgart.de/vote/KN2I
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
// 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);
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 Math.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 $P(i)$

- $P(i_n)$
- $P(i_2)$
- $P(i_3)$

All possible behaviors (what we want, ideally)
Underapproximation (e.g., testing, dynamic analysis)
Overapproximation (most static analyses)

False positive (e.g., benign warning)
False negative (e.g., missed bug)
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?

Improve the **quality** of your code

- Fewer bugs
- Better performance
- More secure software

Save **time** during manual testing

Become a **better developer**

- Get better understanding of program’s behavior
- Avoid common pitfalls
- Learn to use and write tools
Plan for Today

- **Introduction**
  - What the course is about
  - Why it is interesting
  - How it can help you

- **Organization**
  - Course project
  - Mid-term and final exam

- **Foundations**
  - Grammars, ASTs, CFGs, etc.
Organization

- Two lecture slots per week
  - But not all used: See course page
- Weekly reading material
- Throughout the semester:
  - Course project
- December 17: Mid-term exam
- End of semester: Final exam
Grading

- Two lecture slots per week
  - But not all used: See course page
- Weekly reading material
- Throughout the semester:
  - Course project: 40%
- December 17: Mid-term exam: 10%
- End of semester: Final exam: 50%
A Friendly Warning

This is not going to be an easy course!

- Read regularly (otherwise, you won’t be able to catch up)
- Work regularly on the course project
A Friendly Warning

This is not going to be an easy course!

- Read regularly (otherwise, you won’t be able to catch up)
- Work regularly on the course project

... but the effort is worth it!
Programming Language

Most concepts taught in this course: Language-independent

Most examples: JavaScript

- Very popular: client-side web applications, but also for server, mobile, and desktop applications
- Various interesting research challenges

Course project: Java and JavaScript

- Analysis written in Java
- Analysis of JavaScript code
Ilias

Platform for discussions, in-class quizzes, and sharing additional material

- Please register for the course
- Use it for all questions related to the course
- Messages sent to all students go via Ilias

Link to Ilias course on 
software-lab.org/teaching/winter2019/pa/
There is no script or single book that covers everything

- Slides and hand-written nodes: Available after lecture
- Pointers to papers, book chapters, and web resources
Course Project

- Design, implement, and evaluate a program analysis based on an existing framework
  - Data flow analysis of JavaScript code
  - Based on Google Closure compiler
- Individual, independent project
  - Mentor available for questions
Course Project: Timeline

- Project published on November 4
- Due on February 7
  - Implementation and results
  - Report
- Week of February 10 to 14
  - Presentation of projects
Academic Integrity

- Work you submit must be your own
- Unauthorized group efforts and any form of plagiarism are considered academic dishonesty and will be punished
- Allowed to discuss the problem with your peers, but not to reuse any part of an existing solution
Exams

Mid-term exam (written)
- Recommended but not mandatory
- 10% of final grade, but mostly to help you learn
- On Dec 17 in the lecture slot

Final exam (written)
- End of semester
Exams

Mid-term exam (written)
- Recommended but not mandatory
- 10% of final grade, but mostly to help you learn
- On Dec 17 in the lecture slot

Final exam (written)
- End of semester

For both: Open book exam
- Bring books, papers, etc.
- No electronic devices
- Will test your understanding, not your memory!
Plan for Today

■ Introduction
  □ What the course is about
  □ Why it is interesting
  □ How it can help you

■ Organization
  □ Course projects
  □ Term paper
  □ Mid-term and final exam

■ Foundations
  □ Grammars, ASTs, CFGs, CGs, PDGs, etc.
Programming languages

- Syntax ( = form)
- Semantics ( = meaning)
- Implementation ( = execution)

Implementation

a) Compilation

Source code → Lexical analyzer → Tokens → Syntax analyzer (parser) → Syntax tree → Semantic analysis

b) Interpretation
c) Hybrid
   e.g., Java, JavaScript

Machine code generator → Machine Language
Syntax

a) Grammar → which programs are syntactically correct

4 parts: terminals Σ, non-terminals N, productions P, initial symbol S ∈ N

Example: Arithmetic expressions

Σ = {0, 1, 2, ..., 9, +, -}
N = {Exp, Num, Op, Digit}
S = Exp
P : Exp → Num | Exp Op Exp
   Op → + | -
   Num → Digit | Digit Num
   Digit → 01234567890

What is part of this language?
A) 12-2  
B) 2+(12-4)  
C) 11*4  
D) 1234567890
5) Abstract Syntax Trees

- abstract grammar
  - e.g., \( E \rightarrow n \mid Op(E,E) \)
  - \( Op \rightarrow + \mid - \)

- terminals = tokens
  - e.g., 3 + 45

```
    +
   /  \
3    45
```
3) **Control flow graphs**

4) Model flow of control through program

\[ G = (N, E) \] where \( N \) - basic block (sequence of operations executed together) and \( E \) - possible transfers of control

**Ex. 1**

```javascript
if (c) {
  x = 5
} else {
  x = 7
}
console.log(x)
```

![Diagram of control flow graph with nodes and edges](image-url)
Ex. 2: 

```javascript
while (c) {
    x++; 
    y = x;
}
console.log(x);
```

How many edges and nodes in the CFG?
4) **Data Dependence Graph**

Model flow of data from "definitions" to "use".

\[ G = (N, E) \]

- \( N \): operations
- \( E \): possible def-use relations

\( e = (n_1, n_2) \) means: \( n_2 \) may use data defined at \( n_1 \)

**Example 1**

\[ x = 5 \]
\[ y = x + 1 \]
Ex. 2

\[ x = 3 \]
\[ y = 5 \]
\[ \text{if } (x \geq 1) \]
\[ y = x \]
\[ z = x + y \]

# Edges & # nodes in data dep. graph? 

```
\[ x = 3 \]
\[ y = x \]
\[ z = x + y \]
```

```
\[ y = 5 \]
```

```
\[ x \geq 1 \]
```

Plan for Today

- **Introduction**
  - What the course is about
  - Why it is interesting
  - How it can help you

- **Organization**
  - Course project
  - Mid-term and final exam

- **Foundations**
  - Grammars, ASTs, CFGs, etc.