Program Testing and Analysis: Introduction and Basics

Subscribe to the course via Piazza: https://piazza.com/tu-darmstadt.de/winter2018/20000933iv/

Prof. Dr. Michael Pradel
Software Lab, TU Darmstadt
About Me

- Michael Pradel
- At TU Darmstadt since 2014

Before joining TUDA
- Master-level studies in Dresden and Paris
- Master thesis at EPFL, Switzerland
- PhD at ETH Zurich, Switzerland
- Postdoctoral researcher at UC Berkeley, USA
About the Software Lab

- My research group since 2014
- Focus: Tools and techniques for building reliable, efficient, and secure software
  - Program analysis
  - Test generation
- Thesis and job opportunities
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.
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

- 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";
console.log(out);

Quiz: 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
// 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 $P(i)$

All possible behaviors (what we want, ideally)
Underapproximation (e.g., testing, dyn. analysis)
Overapproximation (e.g., 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?

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 projects
  - Term paper
  - Mid-term and final exam

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

- Weekly lectures
- Weekly reading material
- Throughout the semester:
  - Course project
  - Term paper
- December 22: Mid-term exam
- March 20: Final exam
Grading

- Weekly lectures
- Weekly reading material
- Throughout the semester:
  - Course project: 33%
  - Term paper: 33%
- December 22: Mid-term exam: +10%
- March 20: Final exam: 33%
Grading

- Weekly lectures
- Weekly reading material
- Throughout the semester:
  - Course project 33%
  - Term paper 33%
- December 22: Mid-term exam +10%
- March 20: Final exam 33%
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
- Schedule enough time to work on the term paper
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
- Schedule enough time to work on the term paper

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

Most concepts taught in this course: Language-independent

Most course projects and examples: JavaScript (specifically: ECMAScript 6)

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

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

- Please register and enroll for the class
- Use it for all questions related to the course
- Starting from next week, messages sent to all students go via Piazza (not TUCaN!)

Subscribe to the course via Piazza:
https://piazza.com/tu-darmstadt.de/winter2018/20000933iv/
Learning Material

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

- Independent research project
- Design, implement, and evaluate a program analysis and/or test generator
- Teams of 2 students
  - In principle: Individual grades
  - Typically: One team, one grade
Based on existing frameworks and tools

- Jalangi: Dynamic analysis framework
- Esprima & Escodegen: ASTs, parsing, code generation
- Google Closure compiler: Static analysis of JavaScript
- Soot: Static analysis framework for Java
Course Project: Organization

Timeline

- Nov 10: Register teams and preferred projects
- Throughout the semester:
  - Up to three meetings with mentor
- Feb 12–16: Presentation of results
- Feb 25: Final submission

Project proposals will be available via Piazza
Course Project: Deliverables

1) Implementation and results
   ■ Source code and everything needed to reproduce the results

2) Report
   ■ 10 pages maximum, English
   ■ Written like a scientific paper

Due on Feb 25, 2018
Term Paper

Write a scientific article that summarizes and compares three existing papers

- Topic & papers: Based on lecture content
- Individual work
- 6 pages maximum, English
- Peer reviewing
Term Paper

Write a scientific article that summarizes and compares three existing papers

- Topic & papers: Based on lecture content
- Individual work
- 6 pages maximum, English
- Peer reviewing

Grading: 75% final paper + 25% reviews
Term Paper: Some Advice

- Don’t waste space on basics
- Examples are your secret weapon
- Most important part: Comparison of the three papers
- Bad English distracts from good content
- Revise, revise, revise
Term Paper: Rules

- No verbatim copying of text (exception: quotes)
- You may copy some figures (e.g., result graphs) and refer to the source
- You must use your own example(s)
Term Paper: Reviews

- Imitates peer reviewing process
- Each student reviews three term papers
- Revise your term paper after getting reviews
  - Grade will be for final term paper
- Plain text format
- About 1 page, English
Reviews: Some Advice

- Be constructive
- Be polite
- Your reviews contribute to your grade, not to the reviewee’s grade
Term Paper: Organization

Timeline

- Nov 10: Register with preferred topics
- Jan 12: Submit paper for peer review
- Jan 26: Reviews due
- Feb 25: Final version of paper due
Exams

Mid-term exam (written)
- Recommended but not mandatory
- Can improve overall grade up to 10%
- On Dec 22 in the lecture slot

Final exam (written)
- Mar 20, 2018
Exams

Mid-term exam (written)
- Recommended but not mandatory
- Can improve overall grade up to 10%
- On Dec 22 in the lecture slot

Final exam (written)
- Mar 20, 2018

For both:
- Open book: Bring books, papers, etc.
- Corollary: Will test your understanding, not your memory!
Academic Integrity

- Work you submit must be your own/your team’s work
- 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 copy or reuse any part of an existing solution
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.
1) **Programming Languages**

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

a) Implementation
   - Compilation

Source code → **Lexical analyzer** → Tokens → Syntax Analyzer (Parser) → Syntax tree → Semantic analysis

→ Machine code generator → Machine language
b) Interpretation

c) Hybrid

- e.g. Java, JavaScript
2) Syntax

a) Grammar → which programs are syntactically correct

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

Example: Arithmetic expression

Σ = \{0, 1, 2, ..., 9, +, -\}

N = \{Exp, Num, Op, Digit\}

S = Exp

P : Exp → Num | Exp Op Exp
   Op → + | -
   Num → Digit | Digit Num
   Digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

What is part of the language?

A) 12-2  ✓
B) 2+(12-4)
C) 11*4
D) 123456790  ✓
b) Abstract Syntax Trees

- abstract grammar
- \( E \rightarrow n \mid Op \ (E, E) \)
  - \( Op \rightarrow + \mid 1 \)
- terminals = tokens
- e.g., \( 3 + 45 \)
3) **Control flow graphs**

- Models flow of control of program

\[ G = (N, E) \] where \( N \) are basic blocks (sequence of operations executed together)

\( E \) are possible transfers of control

**Ex. 1**

\[
\text{if} (c) \\
x = 5
\]

*(Use)*

\[
x = 7
\]

*console.log\( (x) \)*

\[
\text{if} (c) \\
x = 5 \\
x > 7
\]

*console.log\( (x) \)*
Ex. 2

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

# Edges & # nodes of control flow graph?

```
while (c)
```

```
x++; 
```

```
y = x
```

```
console.log(x)
```

```
3 + 3
```
4) **Data Dependence Graph**

Model flow of data from "definitions" to "uses"

\[ G = (N, E) \]

- \( N \): 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

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

# Edges and # nodes in data dep. graph?

Diagram:

- Node: \( x = \ldots \)
- Node: \( y = 5 \)
- Node: \( x \geq 1 \)
- Node: \( y = x \)
- Node: \( z = x + y \)
Outlook

- Operational semantics
- Manual testing
- Random and fuzz testing
- Symbolic and concolic testing
- Testing concurrent programs
- Program slicing
- Information flow analysis
- Specification mining
- Performance profiling
- Path profiling