Program Testing and Analysis:
Introduction and Basics

Subscribe to the course via Piazza:
http://piazza.com/tu-darmstadt.de/fall2016/20000933iv/

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.
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

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Analyse source code, byte code, or binary</td>
<td></td>
</tr>
<tr>
<td>- Typically:</td>
<td></td>
</tr>
<tr>
<td>- Consider all inputs</td>
<td></td>
</tr>
<tr>
<td>- Overapproximate possible behavior</td>
<td></td>
</tr>
<tr>
<td>- E.g., compilers, lint-like tools</td>
<td>- Analyze program execution</td>
</tr>
<tr>
<td>- Typically:</td>
<td></td>
</tr>
<tr>
<td>- Consider current input</td>
<td></td>
</tr>
<tr>
<td>- Underapproximate possible behavior</td>
<td></td>
</tr>
<tr>
<td>- E.g., automated testing, profilers</td>
<td></td>
</tr>
</tbody>
</table>
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
Program $P$, Input $i$, Behavior $P(i)$

All possible behaviours (what we want ideally)

Underapproximation (e.g., testing, most dyn. analyses)

$\Rightarrow$ False negative

Overapproximation (e.g., most static a.)

$\Rightarrow$ False positive
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
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
- End of November: Mid-term exam
- End of semester: Final exam
Grading

■ Weekly lectures

■ Weekly reading material

■ Throughout the semester:
  □ Course project $\rightarrow$ 33%
  □ Term paper $\rightarrow$ 33%

■ End of November: Mid-term exam $\rightarrow +10$

■ End of semester: Final exam $\rightarrow 33%$
Grading

■ Weekly lectures
■ Weekly reading material
■ Throughout the semester:
  □ Course project 33%
  □ Term paper 33%
■ End of November: Mid-term exam +10%
■ End of semester: 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

Course projects and most 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:
http://piazza.com/tu-darmstadt.de/fall2016/20000933iv/
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
Course Project: Tools

Based on existing frameworks and tools

- Jalangi: Dynamic analysis framework
- Esprima & Escodegen: ASTs, parsing, code generation
Course Project: Organization

Timeline

- Oct 31: Register teams and preferred projects
- Week of Nov 7–11: Meeting with mentor
- Week of Dec 12–16: Meeting with mentor
- Week of Jan 16–20: Meeting with mentor
- Feb 6: Presentation of results
- Feb 20: 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 20, 2017
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: $\frac{3}{4} \cdot$ final paper + $\frac{1}{4} \cdot$ 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

- Oct 31: Register with preferred topics
- Jan 9: Submit paper for peer review
- Jan 30: Reviews due
- Feb 20: Final version of paper due
Exams

Mid-term exam (written)
- Advisable but not mandatory
- Can improve overall grade up to 10%
- On Dec 12 in the lecture slot

Final exam (written)
- After the last lecture (exact date TBD)
Exams

Mid-term exam (written)
■ Advisable but not mandatory
■ Can improve overall grade up to 10%
■ On Dec 12 in the lecture slot

Final exam (written)
■ After the last lecture (exact date TBD)

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.
Programming Language

→ syntax  (= form)
→ semantics  (= meaning)
→ implementation  (= execute)
Syntax

a) Grammar → which programs are syntactically correct

4 parts: terminals $\Sigma$, non-terminals $N$, productions $P$, initial symbol $S \in N$

Example: Arithmetic expressions

$\Sigma = \{0, 1, 2, \ldots, 9, +, -\}$
$N = \{\text{Exp, Num, Op, Digit}\}$
$S = \text{Exp}$
$P = \{\text{Exp} \rightarrow \text{Num} \mid \text{Exp Op Exp}, \text{Op} \rightarrow + \mid -, \text{Num} \rightarrow \text{Digit} \mid \text{Digit Num}, \text{Digit} \rightarrow 0123456789\}$

What is part of the language?

A) $12-2$ ✓
B) $2+ (12-4)$ ×
C) $11 * 4$ ×
D) $1234567890$ ✓
5) Abstract Syntax Trees

abstract grammar for example:

\[ E \rightarrow n \mid Op \ (E, E) \]

\[ Op \rightarrow + \mid - \]

- terminals = tokens

- e.g.: 3 + 45

\[ + \]

\[ 3 \]

\[ 45 \]
Control Flow Graph → models flow of control of a program

\[ G = (N, E) \]

- \( N \): basic blocks (sequence of operations executed together)
- \( E \): possible transfers of control

\[ e = (n_1, n_2) \in E \quad \text{... } \quad n_2 \text{ may execute after } n_1 \]

**Example:**

if \( (c) \)

- \( x = 5 \)
- else
- \( x = 7 \)
- console.log \((x)\)
Ex. 2

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

Quiz: CFG?

→ Nb. edges?
→ Nb. nodes?
Outlook

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