Practical Program Analysis

Series of lectures by Yegor Bugayenko

ABSTRACT:
The course is a high-level introduction to program analysis with a strong emphasis on its practical implementation in programming language design and static analyzers. Students may listen to this course if they plan to develop their own programming languages, compilers, IDEs, static analyzers, code refactoring, generating and optimization tools. The course combines theoretical study with the development of instruments that analyze source code and automatically modifies it.

What is the goal?

The primary objective of the course is to demonstrate how theoretical knowledge of program analysis may be applied to the design of software tools.

Who is the teacher?

I'm developing software for more than 30 years, being a hands-on programmer (see my GitHub account: @yegor256) and a manager of other programmers. At the moment I'm a director of an R&D laboratory in Huawei. Our primary research focus is software quality problems. You may find some lectures I've presented at some software conferences on my YouTube channel. I also published a few books and wrote a blog about software engineering and object-oriented programming. I previously tought two courses in Innopolis University (Kazan, Russia) and HSE University (Moscow, Russia): Software Systems Design and Ensuring Quality in Software Projects (all videos are available).

Why this course?

The quality of software code that most of us programmers write is way below the expectations of our customers. Two main reasons for that is the lack of understanding of how programming languages are designed internally and the absences of connection between theoretical knowledge about language design and the actual software we use every day to write code: IDEs, compilers, code analyzers and modifiers. This course may help build the bridge between theory and practice.

What's the methodology?

The course is organized in pairs of lectures. The first lecture in a pair is an introduction of a theory, while the second lecture is a demonstration of how the theory may be applied to the development of a software tool. During the course students will participate in the development of an automated code optimization tool for one of popular programming languages, like Java or C++.

Course Structure

Prerequisites to the course (it is expected that a student knows this):

- How to write code
- How to design software

After the course a student *hopefully* will understand the basics of:

- Abstract Interpretation
- Symbolic Execution
- Model Checking
- Type theory
- λ-calculus
- Functional Programming
- Axiomatic Semantics
- Operational Semantics
- Temporal Logic

Also, a student will be able to develop:

- A Programming Language
- A Compiler
- A Static Analyzer
- A Code Refactoring Tool

Grading

Students may form groups of up to four people. Each group will present their own public GitHub repository with a software module inside. The group will make a presentation of the moduld that is present in the repository. They will have to explain during a 10-minutes oral presentation with live GitHub demonstration via screen sharing:

- How the software is designed?
- What theory is used in it?
- How far away it is from the canonical theory?
- How the result are validated (tested)?

Most probably, there will be no more than 20% of "A" marks, no more than 40% of "B," and the rest will go to "C" and "D." However, this distribution is not mandatory: if all students make excellent presentations, everybody will get "A."

Attendance will be tracked at the lectures. If you attend more than 75% of all lectures, you will not get less than "C".

A retake exam is possible, following exactly the same procedure. However, the highest mark most probably possible at the retake is "C."