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 the design of programming languages and code analyzers. Students may listen to this course if they plan to develop their own programming languages, compilers, IDEs, static and dynamic 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?

Yegor is developing software for more than 30 years, being a hands-on programmer (see his GitHub account: @yegor256) and a manager of other programmers. At the moment Yegor is a director of an R&D laboratory in Huawei. His primary research focus is software quality problems. Some of the lectures he has recently presented at some software conferences could be found at his YouTube channel. Yegor also published a few books and wrote a blog about software engineering and object-oriented programming. Yegor 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 a) the lack of understanding of how programming languages are designed internally and b) 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 (45 minutes each). 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. Either existing GitHub projects will be used for the demonstration or new projects will be developed on-stage.

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:

- Formal Grammar
- Syntax Analysis
- Abstract Syntax Tree
- Formal Semantics
- Abstract Machines
- Program Analysis
- Data Flow Analysis
- Symbolic Execution
- Model Checking

Also, a student will be able to develop:

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

Lectures

The following topics are discussed:

1. Formal Grammar
    - Notation
    - Production Rules
    - Parse Tree
    - Ambiguity of Grammar
    - Chomsky's Four Types of Grammars
    - Regular Grammar
    - Context Free Grammar (CFG)
    - Linear Grammars
    - Precedence and Associativity
    - Recursive Rules
    - Leftmost and Rightmost Derivation
    - Non-deterministic CFG
    - Left Factoring
2. Syntax Analysis
    - Extended Backus-Naur Form
    - Lexical Analysis (Grammar + Lexer)
    - Tokenization
    - Syntactic Analysis (Parsing)
    - Top-down (LL) and Bottom-up (LR) Parsing
    - Flex and Bison
    - ANTLR
    - Off-side Rule
3. Abstract Syntax Tree
    - Contextual Analysis
    - Semantic Analysis
    - Intermediate Language
    - Control Flow Graph
    - AST in XML
4. Formal Semantics
    - Inference Rules, Axioms, Proof Trees
    - Natural Semantics (Denotational)
    - Structural Semantics (Operational)
    - Reduction Semantics
5. Abstract Machines
    - Turing Machine
    - Finite-State Machine
    - SECD Machine
    - Graph-based VM

6. Program Analysis
    - Rice’s Theorem
    - Static vs Dynamic Analysis
    - Soundness and Completeness
    - Precision and Recall
    - Abstract Interpretation
    - Approximation
    - Lattices
7. Data Flow Analysis
    - Basics Blocks, Transfer Function, and Join Operation
    - Iterative Round-Robin Algorithm
    - Work List Approach
    - Forward and Backward Analysis
    - Live Variable Analysis
    - Definite Assignment Analysis
    - Available Expressions Analysis
    - Taint Checking
8. Symbolic Execution
    - Constraint Solvers (SAT/SMT)
    - Itra- vs Inter-procedural Analysis
    - Concolic Execution
    - Path Explosion
    - KLEE
9. Model Checking
    - Verification vs. Validation
    - Program Graph
    - Transformation System
    - SPIN

Grading

Students may form groups of up to three people. Each group will present their own public GitHub repository with a software module inside, which may be one of the following:

- Compiler,
- Static analyzer,
- Transpiler,
- Code refactoring tool.

Higher grades will be given for (in this order):

- Higher formalism of documentation,
- Higher complexity, and
- Higher test coverage,

Attendance will be tracked at the lectures. If a student attends more than 75% of all lectures, they will not get less than “C”.

At the laboratory classes each group will have to complete three home works and defend them verbally on-site. A completion of less than two will give everybody in the group a negative point, a completion of three — will give a positive point; the point will be added to the grade given by the lecturer.

A retake exam is possible, following exactly the same procedure. However, the highest mark possible at the retake is “C.”

Learning Material

The following books are highly recommended to read (in no particular order):

Robert Harper, *Practical Foundations for Programming Languages*

Xavier Rival et al., *Introduction to Static Analysis: An Abstract Interpretation Perspective*

Peter Linz, *An Introduction to Formal Languages and Automata*

Terence Parr, *The Definitive ANTLR 4 Reference*

Benjamin C. Pierce, *Programming Language Foundations*

Glynn Winskel, *Formal Semantics of Programming Languages*

Flemming Nielson et al., *Principles of Program Analysis*

Anders Møller et al., *Static Program Analysis*

Patrick Cousot, *Principles of Abstract Interpretation*

Uday Khedker et al., *Data Flow Analysis: Theory and Practice*

Christel Baier, *Principles of Model Checking*

It is also recommended to watch YouTube lectures of Michael Pradel. @nesoacademy, Dmitry Soshnikov, Michael Ryan Clarkson, Kristopher Micinski, and Joost-Pieter Katoen.