II.3523 Formal Approaches, Programming Languages and Compilers

General Information

Course title: Formal Approaches, Programming Languages and Compilers
Course identifier: II.3523
Course supervisor: Patrick WANG
ECTS: 5
Expected course workload: 150h of individual work, with 42h in classrooms
Teamwork: Yes
Keywords:
- Formal approaches: Propositional logic, software design and verification, lambda-calculus
- Programming languages and compilers: Programming language paradigms, syntactical and lexical analysis, interpreters and compilers

Presentation

This course is divided into two topics: "Formal Approaches" on the one hand, "Programming Languages and Compilers" on the other.

Formal Approaches sub-module

The conventional programming languages do not allow to claim that a program will be bug-free when it runs. Even the more sophisticated test techniques can let some defects pass through a huge combinatory explosion happens when we mix the potential value of variables and the execution paths, making it impossible to exhaustively consider them. Formal methods lead to a solution to this problem for a certain class of application. In particular, critical applications such as bank transactional systems, automatic piloting programs for rockets, airplanes of metro need for the use of methods able to mathematically prove that a program will run as expected.

In this course, students will be introduced to formal methods and to software verification tools. Students, at the end of the course, are expected to understand how to apply formal methods to their own specific problems and to use said tools to verify the design of their programs with regards to their specifications.

Programming Languages and Compilers sub-module

Programming languages are the foundations that allow programmers to control the machine and make it solve more or less complex problems. Depending on the intended field of application, the level of abstraction and expressiveness of the language varies, which explains the great diversity of languages available on the market.

Compilers are programs that, from a program written in a given language, generate the code directly executable by the machine. The construction of a language and its associated compiler is a complex process that will be detailed in a practical way in this module.

In this course, students will be introduced to the various programming language paradigms. In particular, students will learn about the syntactical and lexical analysis of a program and will learn to design and implement a compiler for a toy language.

Learning Outcomes

Formal Approaches sub-module

At the end of this course, students will:

- Learn about proofs of program properties
- Learn about model-checking
- Learn about lambda-calculus
- Learn about software certification and testing
- Learn to specify and prove a program in a formal language
- Apply these concepts with several tools such as Coq, Prolog, or Frama-C

Programming Languages and Compilers sub-module

At the end of this course, students will:

- Learn how to choose a programming language adapted to their own problems and technical environment
- Learn how to define a Backus-Naur Form grammar
- Learn how to use parser generators and compiler generators
- Learn to implement a programming language interpreter
- Learn to compile a program and generate JVM bytecode

Prerequisites

A good knowledge of a programming language is essential (ideally C, C++, or Java). Knowledge of object-oriented programming is desirable.

The knowledge of basic math's concepts is also desirable.

Course syllabus

The following concepts will be covered in this course:

- Formal Approaches:
    - Propositional logic
    - Lambda-calculus
    - Model of computation
    - Proof systems
    - Model-checking
    - Software specification and verifications
- Programming Languages and Compilers:
    - Languages typing
    - Lexers
    - Grammars and syntax of programming languages
    - Abstract Syntax Trees (AST)
    - Type inference
    - Compilers construction

Tools used by the lecturer

Formal Approaches: The lecturer will use Coq, Frama-C, and Spin.

Programming Languages and Compilers: The lecturer will use Clojure, Clojurescript, Instaparse, and ASM.

Tools used by the learner

At the end of the course, students will have learnt how to use the following tools and methods:

- Formal Approaches:
  - Natural deduction,
  - Coq,
  - Frama-C,
  - Model-checking
- Programming Languages and Compilers:
  - Clojure and Clojurescript,
  - Instaparse,
  - ASM.

Course organization

General organization

Because this course is composed of two sub-modules, the semester is cut in half: the first 7 weeks will be dedicated to the Formal Approaches sub-module and the last 7 weeks will be dedicated to the Programming Languages and Compilers sub-module (or vice-versa).

Course materials

For both sub-modules, courses will consist in lectures and practical sessions.

Evaluation

- Formal Approaches: 60% final exam, 30% project (individual or dyad), 10% homework assignments.
- Programming Languages and Compilers: 40% written exam, 60% project (dyad or triad).

Finally, the final grade for this course will simply be the average of the grades obtained for these two sub-modules.

Language

All courses and course materials will be in English. Students can choose to write their report in either French or English.

Bibliography – Webography – Other resources

Formal Approaches: In addition to the slides, a notebook providing more details is also available on Moodle.

Programming Languages and Compilers:

- Stuart, T. (2013). Understanding Computation: From Simple Machines to Impossible Programs. O'Reilly Media, Inc.
- Queinnec, C. (2003). Lisp in small pieces. Cambridge University Press.
- Fogus, M., & Houser, C. (2014). The joy of Clojure. Shelter Island, NY: Manning Publications
- Steele, G. L. (1999). Growing a language. Higher-Order and Symbolic Computation, 12(3), 221-236.