CO7008 Semantics of Programming Languages

Credits: 15  Convenor: Dr. F. J. de Vries  Semester: 2nd

Prerequisites: Essential: MSc in Software Engineering for the e-Economy students are assumed to have the required background. Previous experience with functional programming is recommended. Students should consult the convenor in case of any doubt.

Assessment: Coursework: 30%  Three hour exam in May/June: 70%

Lectures: 30 hours  Problem Classes: 5 hours
Surgeries: 10 hours  Private Study: 77.5 hours

Subject Knowledge

Aims  To give a broad account of operational and abstract machine semantics, and to explain type assignment and checking, for both imperative and functional programming.

Learning Outcomes  Students should be able to: specify a formal language syntax for various languages, sometimes from a novel and informal description; explain type inference and checking, and apply both simple, and complex (algorithm W) methods for type inference; explain evaluation and transition operational semantics, for various languages, and be able to develop simple formal semantics from descriptions of run time executions; they will be able to solve problems in operational semantics, and explain basic principles; explain abstract machines for two languages and the connections with operational semantics; do simple proofs based on inductive definitions and Rule Induction, at all stages of the module.

Methods  Class sessions together with course notes, worksheets, some printed solutions, and some additional hand-outs and web support.

Assessment  Marked coursework, traditional written examination.

Skills

Aims  To teach students scientific writing and problem solving skills. To teach programming principles which can be adapted to future languages.

Learning Outcomes  Students will be able to: write short, clear, note based, summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen); understand new languages by applying the programming principles taught.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework, traditional written examination.

Explanation of Prerequisites  Students taking this module should have a sound knowledge of simple discrete mathematics, such as that found in CO1011; and of basic imperative and functional programming, such as that found in CO1003, CO1004 and CO2008. In particular, the module will make use of sets, functions, (equivalence) relations, elementary (classical) logic, and mathematical induction. An understanding of basic programming constructs such as loops, conditionals, and assignments is required, but not necessarily large scale programming. Knowledge of basic functional programming is also required, but again, not on a large scale. Any student without such knowledge will need to undertake background reading to become familiar with the basic principles of functional and possibly object oriented programming; however, the material in CO3008 is taught from first principles.

Course Description  Syntax is the formal arrangement of symbols and words, often to create a language; and all programming languages have a particular syntax. Semantics is the study of meaning, and in this module we shall be concerned with the meaning of programming languages. An example will help. Consider if true then x:=3 else x:=4; and if (true) x=3; else x=4; The syntax of the two statements is clearly different, but Pascal
and C(++) programmers will know that their semantics should be the same. In this example, the semantics of each statement is the “effect on a computer at run time”. If you would like to learn more about the ideas behind modern programming languages, how they work, why they work, and gain a clear picture of how high and low level languages interact, then this is the course for you!

All programming languages should have a clear syntax and semantics, from the low level of microprogramming in a CPU, right up to high level programming languages. In this course you will learn methods for giving a run-time semantics to various programming languages. The languages range from high level programming languages (such as Java, C, and Haskell) to intermediate languages (which resemble assembler). We shall see that we can give both high and low level semantics to the same language syntax, and show that any one of the semantic descriptions is equivalent to any other. The high level semantics describes how a large program executes by the step-by-step execution of smaller sub-programs, whose execution is easy to specify precisely. The low level semantics works by translation of a high level language into a low level language; the simpler low level language can also be executed simply and precisely.

You will also learn more about the notion of a type, which you will have met in previous programming courses, and how types can be used to reduce errors in program code. For example, Java was claimed to be type safe, which means that if a program compiles, certain run time errors cannot occur. In 1997, Java was shown to not be type safe, using ideas similar to those met in this module.

By the end of this module, you will have a very sound grasp of the basic ideas on which modern programming languages are based, which will be of benefit to your future understanding of software systems.


**Reading List**


**Resources** Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers.

**Module Evaluation** Course questionnaires, course review.