Foundations of Program Construction and System Behavior

Foundations of Program Construction and System Behavior	
Offered by	Department of Mathematics and Computer Science
Language	English
Primarily interesting for	Computer Science and Engineering and Applied Mathematics
Prerequisites	Imperative and object-oriented programming (2IP90/2IS60/2WH20), Logic and set theory (2IT60/2WF40), Introduction to discrete structures (2IT80), Design patterns (2IPC0). Furthermore, students are expected to have had some prior exposure to formal topics in computer science, e.g., because they have done Automata, language theory and complexity (2IT90) or Software Specification (2IX20).
Contact person	dr. Bas Luttik (s.p.luttik@tue.nl)

Content and composition

What is this package about?

For reasoning about the correctness of computer systems, it is important to have a thorough understanding of the constructs and mechanisms used to realize them. This elective package takes a formal (i.e., mathematical) approach to the consideration of programming constructs and mechanisms, which will open the way towards analyzing computer programs and systems using tool support.
After completing the courses in this package, the student will understand that it is possible to give a mathematical semantics to programming constructs, that the behavior of systems can be specified and analyzed by mathematical means, and that computer tools may then support the activity of showing that a computer system does what it should do.

Course code	Course name	Level classification
2IPH0	Functional Programming	3
2ITA0	Process Theory	3
2ITB0	Provable Programming	3

Course description

2IPH0 Functional Programming
Functional programming refers to a declarative style of programming in which programs describe the desired result, rather than explicitly specifying the steps that need to be taken to obtain the result. The importance of functional programming can hardly be overestimated. It is relevant for specification, prototyping, and the teaching of (provably) correct implementation. But it is also at the heart of many current internet developments and the evolution of programming languages. The functional concepts are structurally attractive, and programs turn out to be way shorter, clearer, and thus better maintainable than their non-declarative counterparts.

The course acquaints students with the functional programming paradigm, emphasizing the pure functional and lazy approach. It addresses the leading roles of types, generalization, calculation, and proof. Students will get an algebraic mind-set with respect to programming, will learn to define correct functional expressions for algorithmic situations with appropriate recursion and type properties, to write interactive Haskell programs using

Foundations of Program Construction and System Behavior

monads to cope with purity and IO, and to stay in touch with recent developments in functional programming.

Prerequisite knowledge and skills
This course presupposes that students are familiar with the elementary notions of (predicate) logic and set theory and proving skills as taught in, e.g., Logic and Set Theory (2IT60) and Introduction to Discrete Structures (2IT80) or Set Theory and Algebra (2WF40). Students are also expected to have had some prior exposure to (i) formal topics in computer science, e.g., because they have done Automata, language theory and complexity (2IT90), and (ii) to design patterns, e.g. because they have done Programming Methods (2IPC0), and (iii) to algorithms and data structures, e.g. because they have done Data Structures (2IL50) or Graphs and Algorithms (2MMD30).

2ITA0 Process theory
Modern computing systems typically consist of many components running in parallel and interacting with each other and their environment. They are designed not so much to compute a result, but rather to execute a process and thus to exhibit behavior. For a correct design of a modern computing system, it is therefore essential to be able to specify its intended behavior, at different levels of abstraction, and to analyze and reason about this behavior. This course offers a formal approach to specifying, analyzing and reasoning about system behavior. It starts from the notion of labelled transition system as a mathematical representation of system behavior, considering various notions of behavioral equivalence. Furthermore, it will discuss a language for specifying concurrent and interactive behavior, and logics to specify properties of behavior. Students will get a theoretical perspective on system behavior and will also see how the theory facilitates computer-assisted analysis in system design.

Prerequisite knowledge and skills
This course presupposes that students are familiar with the elementary notions of (predicate) logic and set theory and proving skills as taught in, e.g., Logic and Set Theory (2IT60) and Introduction to Discrete Structures (2IT80), or Set Theory and Algebra (2WF40). Students are also expected to have had some prior exposure to formal topics in computer science, e.g., because they have done Automata, language theory and complexity (2IT90) or Software Specification (2IX20).

2ITB0 Provable programming

After an introductory course in object-oriented programming, the need arises to get more grip on whether a program does or does not correctly implement its intended task. This course aims to satisfy this need. First, the notion of correctness of a program is made explicit by providing a formalism to specify what a program should do, and when a program satisfies such a specification — and prove it. Second, an important claim to fame of object-oriented programming is that it enables to design programs in a compositional manner from parts. The specification language supports this approach, as it also enables to describe these parts in an abstract yet precise manner. Third, current tooling is used that makes all this feasible, both in the sense of educationally realistic as well as usable in practice. In this course, Dafny (developed at Microsoft) is used. Dafny is a programming language with a built-in specification language (first-order logic, intermediate assertion style) and a tool for static verification (i.e., a verifying compiler), using a theorem prover. The language is imperative and sequential and supports generic classes, dynamic allocation, inductive datatypes. The principles and theory taught in the course are general; the emphasis is on practical verification rather than theory; Dafny is used throughout to support the learning process.

Prerequisite knowledge and skills
This course presupposes that students are familiar with the elementary notions of (predicate) logic and set theory and proving skills as taught in, e.g., Logic and Set Theory (2IT60) and Introduction to Discrete Structures (2IT80), or Set Theory and Algebra (2WF40). Students are also expected to have had some prior exposure to (i) formal topics in computer science, e.g., because they have done Automata, language theory and complexity (2IT90), and (ii) to design patterns, e.g., because they have done Programming Methods (2IPC0), and (iii) to algorithms and data structures, e.g., because they have done Data Structures (2IL50) or Graphs and Algorithms (2MMD30).