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), Automata and process theory (2IT70).
Contact person: Bas Luttik (s.p.luttik@tue.nl)

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.

<table>
<thead>
<tr>
<th>Course code</th>
<th>Course name</th>
<th>Quarter/Slot</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>2IPH0</td>
<td>Declarative programming</td>
<td>Q1 / A</td>
<td>2</td>
</tr>
<tr>
<td>2ITA0</td>
<td>Process theory</td>
<td>Q4 / B</td>
<td>3</td>
</tr>
<tr>
<td>??????</td>
<td>Proviable programming (first offered in 2019-2020)</td>
<td>Q3 / E</td>
<td>3</td>
</tr>
</tbody>
</table>

This elective package will be gradually phased in: the course on Declarative programming (2IPH0) is available as of the academic year 2017-2018, the course on Process theory (2ITA0) will start in the academic year 2018-2019, and the course on Proviable programming is scheduled to be offered first in the academic year 2019-2020.

Course descriptions

Proviable 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. Firstly, 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. Secondly, 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. Thirdly, current tooling makes all this feasible, both in the sense of educationally realistic as well as for use 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 and a form of inheritance.

The principles and theory taught in the course are general; the emphasis is on practical verification rather than theory; and Dafny is used throughout to support the learning process.

**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 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, but will also see how the theory facilitates computer-assisted analysis in system design.

**Declarative programming**
Declarative programming refers to a 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. Perhaps the most prominent examples of a declarative programming style are functional programming and logic programming. Especially the importance of functional programming can hardly be overestimated. It is relevant for specification, prototyping and educating (provably) correct implementation, but it is also at the heart of many current internet developments. The functional concepts are structurally attractive and programs turn out to be way shorter, clearer and thus better maintainable.

The course acquaints students with the declarative 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 and logic expressions for algorithmic situations with appropriate recursion and type properties, to write interactive Haskell programs using monads to cope with purity and IO, and to stay in touch with recent developments in declarative programming.