Module Description, available in: EN

Advanced Programming Paradigms

General Information

Number of ECTS Credits

3

Module code

TSM_AdvPrPa

Valid for academic year

2026-27

Last modification

2024-10-08

Coordinator of the module

Daniel Kröni (FHNW, daniel.kroeni@fhnw.ch)

Explanations regarding the language definitions for each location:

- Instruction is given in the language specified for each location and module execution.
- Documentation is available in the language(s) listed for each location and module execution. If the documentation is in multiple languages, the percentage distributed is indicated (100% = all documentation provided).
- The examination, including both questions and answers, is provided entirely (100%) in the language(s) specified for each location and module execution. The exams are on-site.

	Lausanne			Lugano	Zurich		
Instruction					**X** E 100%		
Documentation					**X** E 100%		
Examination					**X** E 100%		

Module Category

TSM Technical scientific module

Profile Recommendations

This module is recommended for the following profiles:

	Zurich
Profile	CS

Lessons

2 lecture periods and 1 tutorial period per week

Entry level competences

Prerequisites, previous knowledge

All participants should be able to program in the functional and object-oriented styles to the degree that can be expected after attending an introductory-level undergraduate course on these topics.

Participants without prior experience in functional programming are required to work through the chapters 1 through 8 in the book "Programming in Haskell" (second edition) by Graham Hutton before starting the course.

Brief course description of module objectives and content

Although widespread, the currently mainstream imperative, object-oriented programming paradigm, with testing as its main method of quality assurance, has its limitations. Even though it allows novices to write programs relatively quickly and without much formal training, such programs tend to become complicated as soon as they need to do something non-trivial. This makes them increasingly hard to write and reason about, making assurance methods that give better guarantees than software testing intractable. Similarly, even though it is possible to write software tests relatively quickly and without much formal training, such tests are only able to show the presence of faults, but never their absence.

Few are aware that there exist a number of alternatives to imperative, object-oriented programming and testing. This course will start with a broad overview of these alternatives, and then focus on one of them, functional programming, and the use of formal methods to specify and prove the correctness of functional programs.

Functional programming is an approach to programming where simplicity, clarity, and elegance are the key goals. It is the application of formal mathematical and programming-language-based techniques for the construction and verification of computer programs. Although it has a long history, it has recently received attention due to its potential for writing elegant, correct and efficient programs that are easier to write, compose, and maintain, once one is familiar with its underlying concepts. Its simplicity also allows reasoning about the correctness of functional programs using techniques taught in high-school mathematics. These techniques can not only be used to state and prove functional correctness of smaller programs on paper, but also large systems using automated proof assistants. Such automated proof assistants (e.g. Agda, Isabelle/HOL, Idris, Lean and Coq) are themselves applications and further developments of functional programming.

In the final segment of the course, we broaden our focus to explore sophisticated type systems in imperative, object-oriented and functional programming languages. These advanced type systems serve as effective tools for program verification, and have proven effective for identifying and eliminating critical bugs.

Aims, content, methods

Learning objectives and competencies to be acquired

All participants are able to explain and apply formal programming-language-based techniques for the construction and verification of computer programs. Special emphasis will be made on techniques that surpass some of the limitations of mainstream programming in the imperative object-oriented programming style, and on alternatives to mainstream testing-based software assurance methods.

All participants should be able to:

- enumerate and explain a number of different programming paradigms.
- construct programs in the functional style.
- verify the correctness of functional programs using formal proof.
- apply advanced type systems to make imperative, object-oriented and functional programs more secure.

Module content with weighting of different components

The following is a rough overview of the content of the course and is subject to change:

- Overview (1 week)
  - A broad overview of different programming paradigms.
- Functional Programming (4 weeks)
  - Fast paced revision of functional programming prerequisites and Haskell.
  - Types for correctness, Input/Output in a pure language.
  - Functional design patterns: Functor, Applicative and Monad with parsing as an example application.
  - Application: Interpreters for a small imperative and functional programming language.
- Proving Programs Correct (5 weeks)
  - Equational reasoning
  - Structural induction
  - Automated theorem proving
  - Dependent types
- Advanced Type Systems (4 weeks)
  - Subtyping with generic types (co- and contravariance)
  - Subtyping lattices
  - Ownership typing
  - Effect systems

Teaching and learning methods

Interactive lectures explaining main concepts, interspersed with programming exercises and reading assignments.

Literature

You are required to have a copy of the following book during the course:

- Graham Hutton, Programming in Haskell, Second Edition, Cambridge, 2016.

Further material will be provided during the course as required.

Assessment

Additional performance assessment during the semester

The module does not contain an additional performance assessment during the semester

Basic principle for exams

As a rule, all standard final exams are conducted in written form. For resit exams, lecturers will communicate the exam format (written/oral) together with the exam schedule.

Standard final exam for a module and written resit exam

Kind of exam

Written exam

Duration of exam

120 minutes

Permissible aids

Aids permitted as specified below:

Permissible electronic aids

None

Other permissible aids

A summary on at most 2 pages DIN A4 (= 1 sheet DIN A4, written by hand or electronically and printed out). This summary will be collected together with the exam.

Exception: In case of an electronic Moodle exam, adjustments to the permissible aids may occur. Lecturers will announce the final permissible aids prior to the exam session.

Special case: Resit exam as oral exam

Kind of exam

Oral exam

Duration of exam

30 minutes

Permissible aids

No aids permitted