ITI0212: Functional Programming

TalTech

Spring 2022

Course Web Page

`https://compose.ioc.ee/courses/2022/functional_programming`

Lectures

Wednesdays 10:00–11:30 in ICT-315

Labs

Fridays 10:00–11:30 in ICT-122

Instructors

Matt Earnshaw	(`https://www.ioc.ee/~matt`),
Philipp Joram	(`https://www.ioc.ee/~philipp`),
Ed Morehouse	(`https://www.ioc.ee/~ed`)

Course Description

An important aspect of the expressiveness of a programming language comes from what sorts of things it can manipulate as data. In this course we study programming in a *dependently typed purely functional* language. In such a language both types and functions are treated as data.

In a *dependently typed* programming language types are ordinary terms. This implies that terms that are types and those that are not can interact in interesting ways, including the following.

- Types may appear in the definitions of types, allowing for type constructors, which can be regarded as functions from types to types such as the parameterized type of lists, `List : Type -> Type`.
- Types may appear in the definitions of terms that are not types, allowing for parametrically polymorphic or "generic" functions such as parametric identity function, `id : (a : Type)-> (x : a)-> a ; id a x = x`.
- Terms that are not types may appear in the definitions of types, allowing for indexed types such as the indexed type of lists of strings of a given length, `Vect 3 String : Type`.

Because types are terms, they compute just like other terms and we can write functions where the type of the result depends on the value of an argument.

In a *functional* programming language functions are ordinary terms. This lets us write functions that manipulate other functions. For example, we can write a function that applies a given function to each element of a list, `map : {a , b : Type} -> (a -> b)-> List a -> List b`. In a *purely functional* programming language the only sort of terms available are *expressions*, which are terms that get evaluated to values, but which do not produce any effects. In contrast, in imperative programming languages we also have *statements*, which are terms that are executed in order to cause a change of state. Purely functional languages tend to have clear mathematical interpretations and are relatively easy to parallelize.

One of the benefits of working in a dependently typed purely functional language is that we can express properties of programs as types. Under this *propositions-as-types interpretation* we can prove facts about our programs, giving us much stronger guarantees of program correctness than is possible with unit testing. Indeed, such languages have been proposed as a foundation for constructive mathematics, providing a bridge between computation and logic.

In this course we will study dependently typed purely functional programming using the language Idris (`https://www.idris-lang.org`).

Course Structure

The main components of the course are as follows.

lecture: Each week there is a lecture, which introduces new theoretical concepts together with examples of their application.

lab: Each week there is also a laboratory practicum, in which students complete tasks designed to build experience and competence with the concepts presented in lecture.

homework: Approximately every three weeks there is a homework assignment consisting of a number of short exercises intended to build further competence and provide a basis for assessment of student progress.

project: There is a term project, consisting of a larger and less precisely specified programming task, allowing students to demonstrate creative problem-solving in addition to programming skills.

exam: At the end of the course there is an individual interactive final examination, in which instructors ask students to solve short problems and explain concepts based on what they have learned in the course.

Student Evaluation

Course grades are determined from several components, weighted as follows:

Homework Problem Sets	25%
Programming Project	25%
Final Exam	50%

When assessing each assignment, marks will be weighted as follows:

Correctness	70%
Style (clarity, concision, efficiency)	30%

Assignment Submission

All assignment submission will be through the TalTech GitLab server (`https://gitlab.cs.ttu.ee`). Students should create a project named "`iti0212-2022`",

to which course staff are automatically granted read access. Shortly after an assignment deadline, student submissions will be pulled for marking and grades will be communicated to students through the GitLab issue tracker.

Academic Integrity

Collaboration and learning from one another are encouraged, while copying answers and cheating are strictly forbidden. You are expected to be able to distinguish the two. If you are contemplating an action, and you're not sure into which category it falls, you should consider whether what you intend to submit for evaluation is the product of your own efforts and represents your own understanding of the concepts involved. If it is/does not, then you should not submit the work as your own.

Academic Accommodations

Your instructors are committed to supporting an accessible and inclusive learning environment where all students feel welcome, comfortable, and treated fairly. If you have any concerns or would like to request an individual accommodation please contact your instructors.