HOGESCHOOL ROTTERDAM / CMI

Functional programming fundamentals

INFSEN02-1
2016-2017

Number of study points:	4 ects
Course owners:	Tony Busker

Module description

Module name:	Functional programming fundamentals
Module code:	INFSEN02-1
Study points and hours of effort:	This module gives 4 ects, in correspondence with 112 hours: • 2 X 3 x 6 hours of combined lecture and practical • the rest is self-study
Examination:	Written examination and practicums (with oral check)
Course structure:	Lectures, self-study, and practicums
Prerequisite knowledge:	all INFDEV courses.
Learning materials:	• Book: Friendly F#; authors Giuseppe Maggiore and Giulia Costantini, available from the school library • Book: Learn you a Haskell for a great good; available online for free • Slides: found on N@tschool and on the GitHub repository github.com/hogeschool/INFSEN01-1 • exercises and assignments, to be done at home and during the practical part of the lectures (pdf): found on N@tschool and on the GitHub repository github.com/hogeschool/INFDEV-Homework
Connected to competences:	realiseren en ontwerpen
Learning objectives:	At the end of the course, the student: • **understands** the fundamental semantic difference between functional and imperative programming. `(FP VS IMP)` • **understands** reduction strategies such as $\rightarrow_\beta$. `(RED)` • `understands` the basics of a functional type system. `(TYP)` • `can program` with the typical constructs of a modern functional language. The languages of focus are F# and Haskell. `(FP EXT)`
Course owners:	Tony Busker
Date:	February 16, 2017

1 General description

Functional programming and functional programming languages are increasing in popularity for multiple reasons and in multiple ways, to the point that even mainstream languages such as Python, C++, C#, and Java are being extended with more and more functional programming features such as tuples, lambda's, higher order functions, and even monads such as LINQ and async/await. Whole architectures such as the popular map/reduce are strongly inspired by functional programming.
"Java^{TM} developers should learn functional paradigms now, even if they have no immediate plans to move to a functional language such as Scala or Clojure. Over time, all mainstream languages will become more functional" [IBM].
"LISP is worth learning for a different reason — the profound enlightenment experience you will have when you finally get it. That experience will make you a better programmer for the rest of your days, even if you never actually use LISP itself a lot." – Eric S. Raymond
"SQL, Lisp, and Haskell are the only programming languages that I've seen where one spends more time thinking than typing." – Philip Greenspun
"I do not know if learning Haskell will get you a job. I know it will make you a better software developer." – Larry O' Brien
The reason for this growth is to be found in the safe and deep expressive power of functional languages, which are capable of recombining simpler elements into powerful, complex other elements with less space for mistakes and more control in the hands of the programmer. This comes at a fundamental cost: functional languages are structurally different from imperative and object oriented languages, and thus a new mindset is required of the programmer that wishes to enter this new world. Moreover, functional languages often require more thought and planning, and are thus experienced, especially by beginners, as somewhat less flexible and supporting of experimentation.

1.1 Relationship with other didactic units

This module completes and perfects the understanding and knowledge of programming that was set up in the preceding INFDEV courses.

2 Course program

The course is structured into eight lectures. The eight lectures take place during the eight weeks of the course, but are not necessarily in a one-to-one correspondance with the course weeks.

2.1 Chapter 1 - foundations (weeks 1, and 2)

Topics

- Recap of imperative programming language semantics: the *shared memory* model;
- Functional programming concepts: the lambda calculus and beta reduction;
- Adding static typing: the simply typed lambda calculus;
- Making the language usable: delta rules.

2.2 Chapter 2 - practical applications (weeks 3, 4, 5)

Topics

- From the lambda calculus to F# (with practical lecture);
- Lazy evaluation;
- From the lambda calculus to Haskell (with practical lecture);
- Advanced constructs: lists (and list comprehensions), records, tuples, discriminated unions;
- Interop with other languages.

3 Assessment

The course is tested with two exams: A series of assignments which have to be handed in, but will not be graded offline. There will be an oral/practicum check, which is based on the assignments, and a written exam. The final grade is determined as follows:

`if exam grade` $>= 5.0$ `then practicum-grade else 0`

Motivation for grade A professional software developer is required to be able to program code which is, at the very least, *correct*.
In order to produce correct code, we expect students to show: *i*) a foundation of knowledge about how the semantics of the programming language actually work; *ii*) fluency when actually writing the code.
The quality of the programmer is ultimately determined by his actual code-writing skills, therefore the written exam will contain require you to write code. This ensures that each student is able to show that his work is his own and that he has adequate understanding of its mechanisms.

3.1 Theoretical examination INFSEN02-1

The general shape of an exam for `INFSEN02-1` is made up of a short series of highly structured open questions. In each exam the content of the questions will change, but the structure of the questions will remain the same. For the structure (and an example) of the theoretical exam, see the appendix.

3.2 Practical examination INFSEN02-1

There is only one assignment, which is mandatory, and formatively assessed for feedback.

- The assignment must be uploaded to N@tschool or Classroom in the required space (Inlevermap or assignment);
- Each assignment is designed to assess the students knowledge related to one or more learning goal. If the teacher is unable te assess the students' ability related to the appropriate learning goal based on his work, then no points will be awarded for that part.
- *The teachers still reserves the right to check the practicums handed in by each student, and to use it for further evaluation.*
- The university rules on fraude and plagiarism (Hogeschoolgids art. 11.10 – 11.12) also apply to code;

  The practical exam requires to complete an interpreter for a virtual machine. The detailed specification will be provided during the course.

  The solution should include the following features from functional programming languages: (*i*) Tuples, (*ii*) Discriminated Unions, (*iii*) Records, (*iv*) Functions, and (*v*) Recursive functions. At the practical assessment the student will be asked to solve exercises based on these features. The final score is a weighted average of the assignments score and the exam score computed as follows:

  ```
  grade = 0.3 * assignmentScore + 0.7 * practicalExamination
  ```

Structure of exam INFSEN02-1

The general shape of a theoretical exam for `DEV 3` is made up of only two, highly structured open questions.

3.2.0.1 Question 1:

General shape of the question: *Given the following lambda program, and a series of delta rules, show the beta reductions for this program.*
Concrete example of question:
Program:

```
(((TRUE ∧ TRUE) T) F)
```

Delta rules:

```
TRUE ≡ λt→f→t
FALSE ≡ λt→f→f
∧ ≡ λp→q→((p q) p)
```

Concrete example of answer:

```
(((TRUE ∧ TRUE) T) F)

(((( ∧ TRUE) TRUE) T) F)

(((((λp→q→((p q) p)) TRUE) TRUE) T) F)

(((((λp→q→((p q) p)) TRUE) TRUE) T) F)

(((((λp→q→((p q) p)) (λt→f→t)) TRUE) T) F)

(((((λp→q→((p q) p)) (λt→f→t)) TRUE) T) F)

(((((λp→q→((p q) p)) (λt→f→t)) (λt→f→t)) T) F)

(((((λp→q→((p q) p)) (λt→f→t)) (λt→f→t)) T) F)

((((λq→(((λt→f→t) q) (λt→f→t))) (λt→f→t)) T) F)

((((λq→(((λt→f→t) q) (λt→f→t))) (λt→f→t)) T) F)

(((((λt→f→t) (λt→f→t)) (λt→f→t)) T) F)

(((((λt→f→t) (λt→f→t)) (λt→f→t)) T) F)

((((λf→t→f→t) (λt→f→t)) T) F)

((((λf→t→f→t) (λt→f→t)) T) F)

(((λt→f→t) T) F)

(((λt→f→t) T) F)

(((λt→f→t) T) F)

((λf→T) F)

((λf→T) F)

T
```

Points: *4 (50% of total).*

Grading: *Full points for all correct steps and result. -1 for each wrong step. Zero points with 4 or more errors.*
Associated learning objective: understands reduction strategies such as $\rightarrow_\beta$. `(RED)`

3.2.0.2 Question 2:

General shape of question: *Given a lambda calculus program, and a series of typing rules, give the full typing derivation for the program.*
Points: *4 (50% of total).*
Grading: *Full points for all correct steps and result. -1 for each wrong step. Zero points with 4 or more errors.*
Associated learning objective: `understands` the basics of a functional type system. `(TYP)`

Appendix 1: Assessment matrix

Learning objective	Dublin descriptors
`FP VS IMP`	1, 4, 5
`RED`	1, 2, 4, 5
`TYP`	1, 4, 5
`FP EXT`	1, 2

Dublin-descriptors:

1. Knowledge and understanding
2. Applying knowledge and understanding
3. Making judgments
4. Communication
5. Learning skills