universität innsbruck

WS 2021/2022

Functional Programming

Week 1 – Organisation and Introduction

René Thiemann Philipp Anrain Marc Bußjäger Benedikt Dornauer Manuel Eberl
Christina Kohl Sandra Reitinger Christian Sternagel

Department of Computer Science

Organization

Lecture (VO 2)

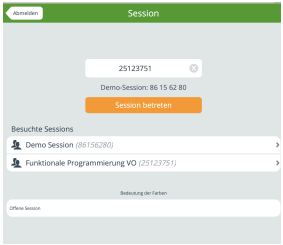
- LV-Number: 703024
- lecturer: René Thiemann
  consultation hours: Tuesday 10:00 – 11:00 in 3M09 (ICT building)
  or at `https://webconference.uibk.ac.at/b/ren-mxw-peh-o9v`
- time and place: Monday, 12:15 – 14:00 in HS B or online
- course website: `http://cl-informatik.uibk.ac.at/teaching/ws21/fp/`
- lecture will be in German with English slides
- slides are available online and contain links
- online registration required until February 5, 2022
- lecture will be recorded and streamed;
  videos and live-streams accessible in OLAT-VO;
  live-questions will be taken via ARSnova, session 25123751

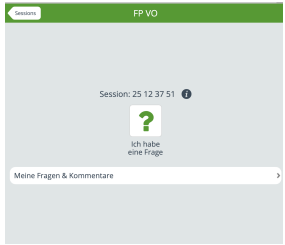
ARSnova

- login via browser

  https://arsnova.uibk.ac.at/mobile/#id/25123751

- main feature: ask questions





Schedule

lecture 1	October	5	lecture 8	November	29
lecture 2	October	11	lecture 9	December	6
lecture 3	October	18	lecture 10	December	13
lecture 4	October	25	lecture 11	January	10
lecture 5	November	8	lecture 12	January	17
lecture 6	November	15	lecture 13	January	24
lecture 7	November	22	Q & A	January	31

- no lecture on November 1 (all saints day)
- on January 31 no new content is presented

Proseminar (PS 1)

- LV-Number: 703025
- new exercise sheets available online on Tuesday or Wednesday
- solved exercises must be entered in OLAT-PS
  - mark which exercises have been solved (Kreuzliste)
  - upload solution: program source (everyone), everything (groups 10–12)
  - deadline: 6 am before PS on Wednesday
- solutions will be presented in proseminar groups
- first exercise sheet: today
- proseminar starts on October 6 or October 13
- proseminar on October 6
  - voluntary, discussion of basic topics (command line, . . .)
  - might be in different room (not in HSB 5), check LFU online
- attendance is obligatory starting from October 13
- registration deadline was in September
- exercise sheets will be English, seminar groups in German

Proseminar Groups

- in total 12 groups, cf. LFU online
- two kinds of proseminars
  - physical meetings: Bachelorstudium, Lehramtsstudium
  - virtual meetings: Erweiterungsstudium
- all groups are completely full
- change of groups only possible if you find exchange partner
  - go to OLAT-PS and use "Tauschbörse"
  - several swaps have already been conducted
  - exchanges are only possible until Friday this week

Tutorium

- opportunity to ask questions about topics of lecture and exercises
- presentation of more examples
- no new topics, no influence on grades, no solutions to exercises
- attendance voluntary
- tutor: Benedikt Dornauer
- Monday 18:15 – 19:00
  - starts next week
  - HS B
  - stream + recordings in OLAT-VO
  - online questions via ARSnova (session 52444256)

Weekly Schedule

- Monday 12:15 – 14:00: lecture on topic n
- Monday 18:15 – 19:00: tutorium on topic $n-1$ or n
- Tuesday or Wednesday: exercise sheet n on topic n available
- Wednesday 6 am: deadline for upload of solution of ex. sheet $n-1$
- Wednesday afternoon: proseminars on exercise sheet $n-1$
- ...

Grading

- separate grades for lecture and proseminar
- lecture
  - grading solely via exam
  - 1st exam on February 7, 2022
  - online registration required from January 1 – 31 via LFU online (deregistration still possible in February)
  - 2nd and 3rd exam in March and September (tentative)
  - it suffices to pass one of the three exams
- proseminar
  - 80 %: scores from weekly exercises
  - 20 %: presentation of solutions

Literature

slides

- no other topics will appear in exam ...
- ... but topics need to be understood thoroughly
  - read and write functional programs
  - apply presented techniques on new examples
  - not only knowledge reproduction

Richard Bird. Introduction to Functional Programming using Haskell, 2nd Edition, Prentice Hall.

Introduction

(Functional) Programming

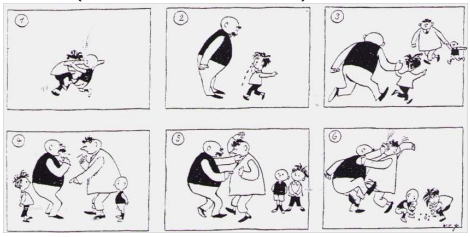
- task: solve problems
  - sort a list
  - generate a website
  - navigate from Innsbruck to Cologne
- distinguish between data ...
  - input `[1,5,2]` and output `[1,2,5]`
  - query "search for 'functional programming'" and resulting website
  - map of Europe, two locations and route
- ...and programs
  - control over how data should be processed
  - mostly written by humans
- usually computers are used for executing a program on some input, but computation can also be done on paper or in mind

How to Learn Programming

+ read, study and write programs (many)
+ actively attend lecture and proseminar
+ try to solve exercises (alone or discuss in small teams)
− copy solutions from other students or from the internet

Algorithms and Programs

story (language agnostic)



text (language dependent)

- Tom and Paul were struggling until ...
- Thomas und Paul rauften solange bis ...
- 토마스와 파울은 싸우고 있었는데...

algorithm (prog. language agn.)

- task: determine the maximum of m and a list of numbers
- if list is empty, result is m
- otherwise, change m to max. of head of list and m
- continue with tail of list

program (language dependent)

```
maxlist m [] = m
maxlist m (x : xs) =
  maxlist (max m x) xs
```

```
while (list != null) {
  m = max(m, list.head);
  list = list.next; }
return m;
```

Different Programming Styles

- Imperative Programming (VO Introduction to Programming)
  - state is mapping of variables to data
  - assignments instruct computer to update state
  - example
    - consider assignment `x := x + 5;`
    - if in a state `x` stores value 7
    - then after executing assignment `x` stores value 12
- Functional Programming (this lecture)
  - define functions (mathematical: same input implies same output)
  - new results (of function invocations) are computed,
    but there is no notion of state that can be updated
  - example
    - consider function definition `add5 x = x + 5` where `x` is parameter;
    - function invocation `add5 7`
    - is evaluated, e.g., `add5 7 = 7 + 5 = 12`
    - 7 is not changed into 12, there is no state with variable `x`
- Logic Programming, Object Oriented Programming, ...

Different Programming Styles

- fact: most programming languages are of equal power
- demand for different styles still reasonable
  - each style has its own distinguishing features and limitations
    (like in real languages: translate "Ohrwurm" or "Internetbrowser")
  - good programmer should know about alternatives:
    choose suitable style and language depending on problem and context
- advantages of functional programming
  - intuitive evaluation mechanism
  - suitable for verification
  - expressive language features
  - suitable for parallelization
- disadvantages of functional programming
  - more difficult to model state, side-effects, and I/O
  - not main-stream in industry, but getting more popular

Different Functional Programming Languages

- combinatory logic (Moses Schönfinkel 1924, Haskell Curry 1930): foundation of FP
- λ-calculus (Alonzo Church 1936): foundation of FP
- LISP (John McCarthy, 1958): List Processing
- ML (Robin Milner, 1973): Meta Language, several dialects
- Erlang (Ericsson, 1987): distributed computing
- Haskell (Paul Hudak and Philip Wadler, 1990): language in this course
- F# (Microsoft, 2002) and Scala (Martin Odersky, 2003): combine different programming styles, including FP

Syntax and Semantics

- syntax of a (programming) language defines valid sentences (programs)
  - "This is a proper English sentence."
  - "this one not propper"
  - computers refuse programs that contain syntactical errors!
- semantics defines the meaning of valid sentences / programs
  - "Clean your room!" ✔
  - `let xs = 1 : 1 : zipWith (+) xs (tail xs) in take 9 xs` ✘
- we will learn both syntax and semantics of Haskell

Haskell Scripts

```
-- This script is stored in file example.hs

add5 x = x + 5

{- the following function takes a temperature in
   degree Fahrenheit and converts it into Celsius -}

fahrenheitToCelsius f = (f - 32) * 5 / 9
```

- a Haskell script (= program) has file extension `.hs`
- a script is a collection of (several) function definitions
- comments are just for humans, ignored by computer
- single-line and multi-line comments
  - single: `-- everything right of -- is a comment`
  - multi: `{- multi-line comments can deactivate areaRectangle width height = width * height parts of script easily -}`

Writing Haskell Scripts

```
-- This script is stored in file example.hs
add5 x = x + 5
fahrenheitToCelsius f = (f - 32) * 5 / 9
```

- coloring
  - when entering a Haskell script, one does not add colors in a text editor
  - syntax highlighting: often editors for computer programs automatically add colors to simplify reading; quickly distinguish
    - comments, keywords, names of functions, names of parameters, ...
- function- and parameter-names (`add5`, `x`, ...)
  - always start with a lowercase letter, may contain digits
  - convention: long names use camelCase (`fahrenheitToCelsius`, ...)
- white-space (spaces, tabs, newlines, ...)
  - in Haskell white-space matters
  - for the moment, start every new line without blanks
  - the following script is not accepted

```
add5 x = x + 5
 fahrenheitToCelsius f = (f - 32) * 5 / 9
```

Functional Programming – Sessions

- starting a session is like activating your calculator
- we use ghci, an interpreter for Haskell

```
rene$ ghci                        -- start the interpreter
Prelude> 42                       -- enter a value
42
Prelude> 5 * (3 + 4)              -- evaluate an expression
35
Prelude> :load example.hs         -- load script from file
[1 of 1] Compiling Main  ( example.hs, interpreted )
Ok, 1 module loaded.              -- script was accepted
*Main> fahrenheitToCelsius 95  -- invoke our function
35.0
*Main> :quit
```

Workflow for Functional Programming

- define functions in script
- load script (will compile script or deliver error message)
  - parse error: `5 +` (argument missing)
  - type error: `5 + "five"` (cannot add number and text)
  - error-messages are sometimes cryptic
- enter expression and start evaluation to get result (read-eval-print loop, REPL)
  - result: (canonical representation of some) value which cannot be further simplified,
    e.g., `42`, `"hello"`, `[7,1,3]`, ...,
    but not `5 + 7`, `fahrenheitToCelsius 8`, ...
  - evaluation uses
    - built-in functions (`+`, `*`, `:`, `++`, `head`, `tail`, ...), defined in Prelude
    - user-defined functions (`fahrenheitToCelsius`,...) from script-files

Compare FP to Calculator

- enter expression and let it compute result
- restricted to numbers and built-in functions

Comparison: FP vs Calculator

- task: convert many temperatures from Fahrenheit to Celsius: 8, 9, 300, ...
- calculator: enter the following expressions
  - $(8 - 32) * 5/9$
  - $(9 - 32) * 5/9$
  - $(300 - 32) * 5/9$
  - ... (quite tedious: enter same formula over and over again)
- FP
  - write one program: `fahrenheitToCelsius f = (f - 32) * 5 / 9`
  - just evaluate the function on the various inputs
    - `fahrenheitToCelsius 8`
    - `fahrenheitToCelsius 9`
    - `fahrenheitToCelsius 300`
    - ... (concise, readable, easy: just invoke function)
  - or just: `map fahrenheitToCelsius [8,9,300,...]`
- program(s): a recipe to turn inputs into desired outputs

Summary

- Haskell scripts are stored in .hs-files
- functional programming: specify functions (input-output-behaviour)
- ghci loads scripts and evaluates expressions
- next lecture: beyond numbers — structured data