Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A. Lochbihler and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 1: First Steps in Haskell and Natural Deduction

Submission deadline: February 24th, 2014 (before 11:00 am)

Note: Assignments 1, 2(a)-(c) and 3 can be solved after the first lecture. For assignments 2(d), 4 and 5, material from the second lecture on Thursday is required.

Introduction

The exercise groups start *this week*, that is, February 18th (Tue) and February 19th (Wed). If you have missed the registration during the first lecture, you can send an e-mail to `omaric@inf.ethz.ch` and visit one of the exercise groups in the first week. You can look up the exercise group assignments on the course web page (`www.infsec.ethz.ch/education/ss2014/fmfp`).

In this course, you will use the Haskell interpreter GHCi. Install GHCi on your own computer or use the computers in the D-INFK student labs. The GHCi interpreter is included in the Haskell platform available at `http://www.haskell.org/platform/`. If you have problems with the Haskell platform, you can also install the GHC environment `www.haskell.org/ghc/` which includes GHCi.

There exists a wealth of resources on Haskell. You can find links to almost all of them on `www.haskell.org`. The most informative ones for you as a Haskell programmer in the making are:

- An online version of the introductory book "Learn You a Haskell for Great Good" by Miran Lipovača is available at `learnyouahaskell.com/`. Other introductory books are "Programming in Haskell" by Hutton and "Haskell: The craft of functional programming" by Thompson. However, there are no online versions of these. Note that many examples given in the lecture are from the book by Thompson, which you could use as a reference.

- The book "Real World Haskell" by Bryan O'Sullivan, Don Stewart, and John Goerzen contains advanced Haskell material. It is also available online: `book.realworldhaskell.org/read`.

- Search the Haskell libraries by name or type with `www.haskell.org/hoogle`.

- See also the *links on the course homepage* for a printable version of the GHCi prelude file and other material about Haskell.

Please send your solutions by e-mail to your tutor as a *Haskell file*. The subject of your e-mail should start with [FMFP]. You can find the e-mail address on the course web page. Hand in your solution no later than the given submission deadline.

You can use -- for single line comments and {- and -} to enclose multiline comments. You should use multiline comments for your answers to non-programming exercises. You can find an example of a Haskell file at the course web page. The name of your Haskell file must be sheet<nr>_<nethz-username>.hs and the maximal line length is 100 characters. You also must not use TAB characters in your Haskell files. This helps your tutor to sort and print all submissions easily.[1] In general, provide detailed comments in your solutions in order to help your tutor to understand your solutions.

Assignment 1:

The purpose of this assignment is to get used to GHCi and writing Haskell programs. You do not have to hand in your solution for this assignment; you can find a solution in the file sheet1_johndo.hs, which is available from the course web page.

Important prompt commands in GHCi are

:?	help
:load <filename> or :l <filename>	load the file filename
:reload or :r	repeat the last load command
:quit or :q	quit

We recommend using a decent text editor that supports syntax highlighting for editing your Haskell files. See the Haskell links on the course homepage for suggestions for all major operating systems.

(a) Download the file gcd.hs from the course web page and load it into GHCi. Use GHCi to calculate the greatest common divisor of 139629 and 83496. What happens if one of the arguments of the function gcdiv is negative? What happens if one of the arguments is 0?

Generalize the gcdiv function to a function gcdInt :: Int -> Int -> Int such that gcdInt x y = gcdiv x' y', where x' is the absolute value of x and y' is the absolute value of y. Does your function terminate for all inputs?

(b) GHCi has already many predefined functions. These functions are defined in the Prelude, which is automatically loaded when you start GHCi. You can find a link to the standard Prelude files on the course homepage.

Look at the Prelude and check whether you can simplify your solution in (a) by using some of the predefined functions.

(c) You can query GHCi for the type of a function with the prompt command :t. For instance, GHCi will output Int -> Int -> Int if you type in :t gcdiv.

What happens if you apply gcdiv to the floats 3.6 and 7.2? Change the type of gcdiv to Double -> Double -> Double. What is now the output of gcdiv 3.6 7.2? What is the output of gcdiv 3.6 7.1999999999999?

[1] TAB characters are also prone to result in strange GHCi error messages, as Haskell is layout sensitive.

Assignment 2:

Complex numbers can be represented as pairs of reals: the first coordinate of a pair represents the real part of the complex number and the second coordinate represents the imaginary part. In Haskell, we can use pairs of type `Double` for complex numbers.

(a) Write functions `re :: (Double, Double) -> Double` and `im :: (Double, Double) -> Double` that return the real part and the imaginary part of a complex number, respectively.

(b) Write a function `conj :: (Double, Double) -> (Double, Double)` that conjugates a complex number.

(c) Write functions for addition and multiplication of complex numbers, and write a function that returns the absolute value of a complex number.

(d) Write a main function with I/O so the user can enter a complex number and receive its absolute value. Example interaction, with the user typing 3 and 4:

```
Enter your complex number's real component:
3
Enter your complex number's imaginary component:
4
Your complex number's absolute value is: 5
```

Assignment 3:

The Fibonacci numbers are defined as

$$fib(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ fib(n-1) + fib(n-2) & \text{otherwise.} \end{cases}$$

Louis Reasoner writes the following Haskell program for computing Fibonacci numbers:

```
fibLouis :: Int -> Int
fibLouis 0 = 0
fibLouis 1 = 1
fibLouis n = fibLouis (n-1) + fibLouis (n-2)
```

Eva La Tour writes another Haskell program for the Fibonacci numbers:

```
fibEva :: Int -> Int
fibEva n = fst (aux n)
    where aux 0 = (0,1)
          aux n = next (aux (n-1))
          next (a,b) = (b, a+b)
```

(a) Complete the evaluation steps in Haskell given below for `fibLouis 4`.

```
fibLouis 4 =
(fibLouis (4-1) + fibLouis (4-2)) =
(fibLouis 3 + fibLouis (4-2)) =
...
```

(b) Complete the evaluation steps for `fibEva 4`.

```
fibEva 4 =
fst (aux 4) =
...
```

Assignment 4:

Recall that $\rightarrow$ is right-associative, while $\wedge$ and $\vee$ are left-associative. Moreover, $\neg$ binds stronger than $\wedge$, which binds stronger than $\vee$, which in turn binds stronger than $\rightarrow$. Hence, the formula

$A{\wedge}B{\vee}C \rightarrow \neg E \rightarrow C{\vee}A{\wedge}B$ is parenthesized as $((A{\wedge}B){\vee}C) \rightarrow ((\neg E) \rightarrow (C{\vee}(A{\wedge}B)))$.

We recommend to always parenthesize formulas before proving them using natural deduction. This simplifies matching the inference rules and you avoid trivial parsing errors.

(a) Parenthesize the following formulas.

(i) $A \vee B \rightarrow C \rightarrow A \wedge C \vee B \wedge C$

(ii) $(A \rightarrow B \rightarrow C) \rightarrow A \wedge B \rightarrow C$

(b) Prove that the formulas in (a) are tautologies in intuitionistic logic using natural deduction. Give complete proof trees and label each rule application with the rule's name. For your convenience, the rules for natural deduction in intuitionistic logic are copied below.

$$\frac{}{\Gamma, A \vdash A}\,\textit{axiom} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow \textit{I} \qquad \frac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \rightarrow \textit{E}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\,\bot\textit{E} \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\,\neg\textit{I} \qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash B}\,\neg\textit{E}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge\textit{I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge\textit{EL} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge\textit{ER}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee\textit{IL} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee\textit{IR} \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\,\vee\textit{E}$$

(c) We define $A \leftrightarrow B$ as $(A \rightarrow B) \wedge (B \rightarrow A)$. Provide suitable introduction and elimination rules for $\leftrightarrow$ and use them to prove the validity of $(A \leftrightarrow B) \rightarrow (B \leftrightarrow A)$.

Assignment 5 (headache of the week[2]):

Recall that one way to make the above inference system complete for classical logic is to add an axiom formalizing the "law of excluded middle" (lat. "tertium non datur").

$$\frac{}{\Gamma \vdash A \vee \neg A} \textit{ TND}$$

Prove that the formula $((A \rightarrow B) \rightarrow A) \rightarrow A$ is valid in classical logic.
Hint: Consider first the simpler case where B is replaced by $\bot$. Recall that $\neg A$ is syntactic sugar for $A \rightarrow \bot$. Generalize your proof for $(\neg A \rightarrow A) \rightarrow A$ to $((A \rightarrow B) \rightarrow A) \rightarrow A$.

[2]Our weekly headaches are challenging problems. They are meant as supplements to test your FMFP skills.