Lab Sheet 10

Equational Reasoning

Functional Programming - Winter 2018/2019 - January 10, 2019 - Schupp/Lehmann

How to succeed with the labs and exercises?

Labs and exercise sheets are published every week on the course homepage at StudIP. As described in the first lecture, each successfully completed lab and exercise earns you bonus points towards your final score in this semester's exam. Keep in mind that you only get bonus points if you would pass the exam without them. **Cheating does not help you - but we will!**

How to complete a lab successfully?

In the lab, you will solve the lab sheet with fellow student(s). To find out who your teammates are, please look at the group pdf in StudIP's download area. **During the session, one participant of every group will be selected, who must then explain one of their task solutions to a lab assistant**.

How to complete an exercise successfully?

In order to complete an exercise sheet successfully, you must upload your answers using DOMjudge **before the deadline** printed on the exercise sheet. We will not consider any solutions handed in after the deadline! Furthermore, you must solve and hand in the exercises **individually** and your Haskell code **must compile** and **pass certain amounts of tests** as specified. During the exercise session, we develop possible solutions together. Please participate! We encourage you to ask and answer questions from fellow students.

Technically, Haskell files you submit using DOMjudge must have the format as specified in the task sheets(usually ".hs", ".lhs", or ".txt"). Furthermore, DOMjudge will only consider your last submission. Therefore, if you first submit successfully (your code compiles and tests are passed) and afterwards unsuccessfully (your code does not compile or certain tests fail again), your last submission counts, and - if it does not compile - will therefore be ignored. Make sure your last submission was successful!

How to get additional information?

We encourage you to discuss past and present exercise sheets with us. Either approach us during the exercise session, or visit us during the weekly office hours. We are also available via e-mail or on the StudIP forum. We try to reply as quick as possible and in general, you should get a reply the next weekday, but we cannot guarantee this.

Exercise 1 Rewrite the following function[1] so that no overlapping patterns are used anymore.

```
ackermann :: Integer -> Integer -> Integer
ackermann m n | m < 0 || n < 0 = error "only␣positive␣arguments␣allowed"
ackermann 0 n = n + 1
ackermann m 0 = ackermann (m - 1) 1
ackermann m n = ackermann (m - 1) (ackermann m (n - 1))
```

Exercise 2 Recall the definition of natural numbers, the `add` function, and the `mult` function:

```
data Nat = Zero | Succ Nat deriving (Show, Eq)

add :: Nat -> Nat -> Nat
add Zero m = m
add (Succ n) m = Succ (add n m)

mult :: Nat -> Nat -> Nat
mult Zero _ = Zero
mult (Succ n) m = add (mult n m) m
```

Your task is to use equational reasoning to prove that `Succ Zero` is a neutral element to `mult`.

a) Prove that for any natural number `x` of type `Nat`, `mult (Succ Zero) x` is equal to `x`. In other words, you should prove the following: $\forall x$: `mult (Succ Zero) x` $\equiv$ `x`.

b) Prove: $\forall x$: `mult x (Succ Zero)` $\equiv$ `x`.

Exercise 3 * Recap Implement the function `find :: Eq a =>a -> [a] -> [Int]` that takes a value `x` and a list `xs`. The function returns the list of the positions of `x` in `xs`. You are not allowed to use recursion or list comprehension.

Exercise 4 * Recap Consider the following user-defined type:

```
data NumChar = Num Int | Letter Char
```

Define a type synonym that can hold the same information, using only build in types or types exported by the Prelude.

[1] http://en.wikipedia.org/wiki/Ackermann_function