Lab Sheet 8

Recursive User-defined Types

STS Software Technology Systems

Functional Programming - Winter 2018/2019 - December 12, 2018 - 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 Recall the definition of natural numbers from the lecture:

```
data Nat = Zero | Succ Nat deriving Show
```

a) Represent the natural numbers 1 and 4 as a value of type `Nat`.

b) Implement the functions `intToNat` and `natToInt` that convert between positive Haskell integrals and `Nat`s.

Exercise 2 Provide a complete design recipe for the function `mult` that takes two `Nat`s and multiplies them without converting to regular Haskell numbers. **Hint:** You can use the `add` function from the lecture.

Exercise 3 The following exercises are about Binary Decision Diagrams (BDD)[1]. Essentially, BDDs are graph representations of ordinary truth tables. Looking at Figure 0.1, you can see that BDDs consist of three node types:

a) True sinks

b) False sinks

c) Nodes that have a label and two sub-BDDs
One of these sub-BDDs corresponds to a partial valuation where the value of the boolean described by the label is true, and one to a partial valuation where the value of the boolean described by the label is false.

Design a datatype that encodes BDDs.

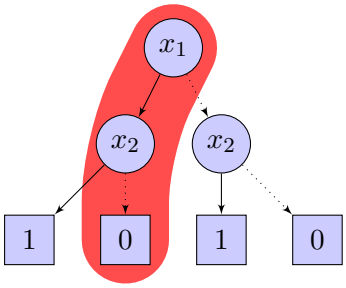

Figure 0.1: BDD representing f

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	0
1	1	1

Table 0.1: Truth table for f

Exercise 4 Represent the BDD from Figure 0.1 as a value of your type from Exercise 3.

Exercise 5 * To evaluate a BDD, you start at the top node and look-up the boolean value of the variable in the node (here: x_1). If this variable is True, then you follow the straight edge, otherwise the dotted edge. Repeat the procedure until you arrive at a sink (square node). The label inside this node is the result. Figure 0.1 shows the resulting path for $x_1 = \mathsf{True}$, $x_2 = \mathsf{False}$.

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

Implement an `eval` function that takes a complete list of variable assignments and a BDD, and evaluates the BDD using the assignments. **Hint:** The function lookup might help.

Exercise 6 ** MonadMagic Recall the definition of an error from a previous lab:

```
-- Error Code Description
data Error a = Error Int String | Result a deriving Show
```

And the bindE function:

```
bindE :: Error a -> (a -> Error b) -> Error b
(Error eId eName) `bindE` _ = Error eId eName
Result a `bindE` k = k a
```

a) Define a Monad instance for `Error`.

b) Read about the do notation e.g. at https://en.wikibooks.org/wiki/Haskell/do_Notation and use your Monad instance and the do notation in an example. Possible functions that you could define that may fail are a division function that returns an `Error` if given 0 as divisor, and a predecessor function that returns an `Error` if the result would be negative otherwise.