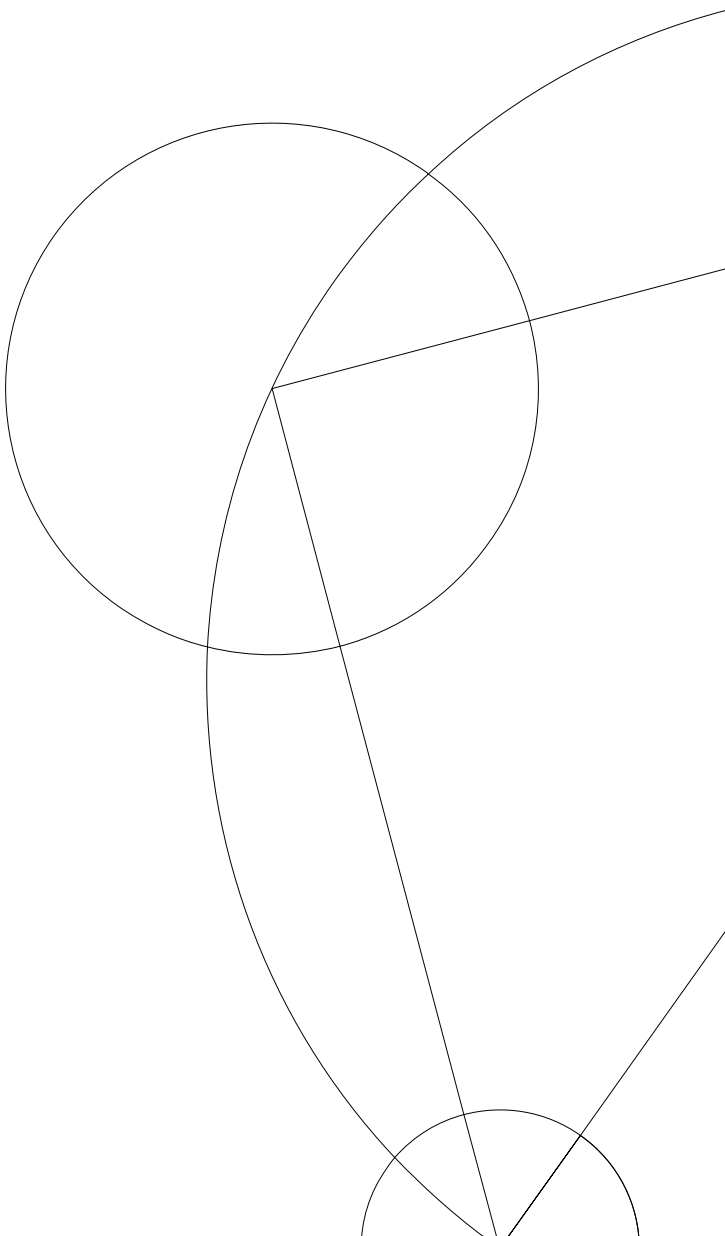
KØBENHAVNS UNIVERSITET

Home Assignment 1

AP-Haskell

qbp758 / zxf339

AP22 Assignment 1 Group 3



Date: 16. september 2022

Part 1: Description about Design and Implementation Choices

1. Design Description

In our code design, we try to minimize a function's function to make each function independent. We will implement a complex function through these independent functions.
For example, when we implement *evalErr* :

```
evalErr :: Exp -> Env -> Either ArithError Integer
...
```

In general, to check each errors of *Exp*, we need to nest a lot of *case...of...* in this functions:

```
case evalErr x initEnv of
        Right n -> Right n
        Left _ ->
            case evalErr d e of
                Left ae -> Left ae
                Right x ->
                    ...
```

In our design,we abstract a auxiliary function to process the *case...of...* expressions:

```
eHelper :: Either ArithError Integer -> Either ArithError Integer
             -> (Integer -> Integer ->  Either ArithError Integer) -> Either ArithError Integer
eHelper x y f = case x of
                  Left ae -> Left ae
                  Right x' ->
                    case y of
                      Left ae -> Left ae
                      Right y'-> f x' y'
```

As we can see below, in the *evalErr* function, we can use the *eHelper* function to simplify our codes.

```
evalErr (Add x y) e = eHelper (evalErr x e) (evalErr y e) (\ x y -> Right (x + y))
evalErr (Sub x y) e = eHelper (evalErr x e) (evalErr y e) (\ x y -> Right (x - y))
evalErr (Mul x y) e = eHelper (evalErr x e) (evalErr y e) (\ x y -> Right (x * y))
```

In the unit-testing, we import *Testy* framework to do unit-testing

```
import Test.Tasty
import Test.Tasty.HUnit

tests :: TestTree
tests = testGroup "All tests :"
  [testShowExp, testEvalSimple, testEvalFull, testEvalErr]
```

2. Implementation Choices and Why

Regarding how we chose to deal with errors in unneeded parts of Let-expressions, and why:
In our code, we prefer to figure out the body expression of the 'Let' first. Then, if we can figure it out directly - namely, it does depend on the variable of 'Let' scope - we can return the value of it. Specifically, it means that we do not check the errors of the previous expression in 'Let' if we figure it out directly.
The reasons why we select this method: Firstly, this method can save a lot of computing resources in the run-time; Secondly, it aims to get the result or a value from the 'Let' expression, so if we get the result, we can go to next step of the program. In other words, in this scenario, it is not our aim to care about the no-association variables.

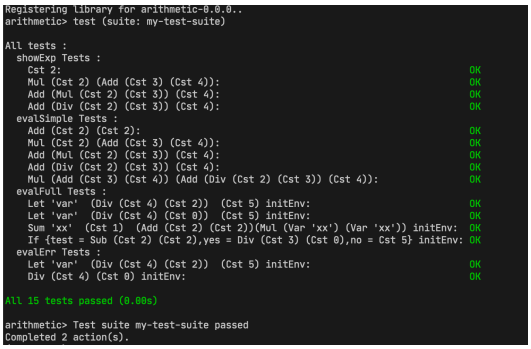
Part 2: Assessment of the Quality of Our code

A. Completeness

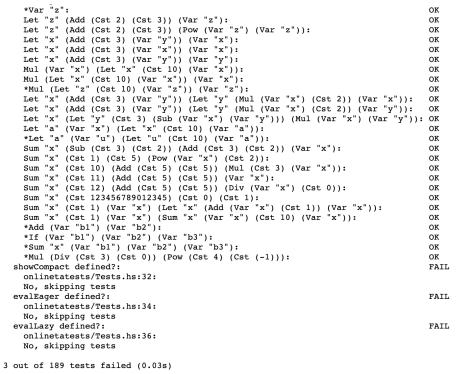
We have completed all the mandatory questions, but did not do the optional questions. Since we did not have enough time, we plan to complete them soon.

B. Correctness

The part of completion has passed both the unit-testing and the onlineTA-testing.



(a) Result of Unit Testing



(b) Result of OnlineTA Testing

C. Efficiency

As for the run time and space usage, We tried our best to improve the efficiency of code operation, but we didn't design too much for space usage.

D. Maintainability

The maintainability of our code is OK. First, we added enough common code to improve the readability of the code; Secondly, we abstracted higher-order-functions and auxiliary-functions to clarify our code logic.