Advanced programming - Assignment 1

Nicolas Dyhrman (LZG210)

September 2022

1 Completeness

I have to my knowledge implemented all the functions needed to hand in the assignment. Including *showExp*, *evalSimple*, *extendEnv*, *evalFull* and *evalErr*. I have not really looked into any of the optional parts of the assignment and have therefore not really solved anything about them or looked too much into them.

2 Correctness

I have created automatic test through the testing script given in the assignment that uses all paths possible for the program to go through, in each pattern of the functions. I have not tested fully that it necessary evaluates first the left argument and then the right as described it should under evalErr. I have also done some automated test if an error was expected as it would crash the program if i tried to create the automatised test with that.

Throughout the assignment some design choices have been made that i will discuss in this section. One of the design choices made was about how we would solve unwanted parts of Let expressions. This refers to if an error is done in the variable binding if an error should be thrown or not if it is not used in the body to evaluate. I have for evalFull chosen to not throw an error if that were to happen. This i have done because it is easier than making it fail. It is easier because Haskell uses lazy evaluation. So the variable result value is not even evaluated before it is used in the body, and will therefore not fail. To make it throw an error we would have to evaluate the variable first by using it in an if statement or some other way of force evaluate it. Which would create more messy code without giving a greater payoff as it is not used anyway.
For evalErr I have done it different, as here it will evaluate an error if the variable has an error in the value expression. This time it is because we evaluate each expression and then add the already calculated expression to the environment, and therefore it will throw an error. This has been done as we want it to be throwing an error as expected if it is evaluated, in the correct manner with an *Left errorType*, which would be harder to do if it was not pre evaluated for the evaluation of the expression in the Let binding. Other expressions such as If expressions still counts towards only evaluating the desired path in both evalErr and evalFull.

3 Efficiency

The code submitted can get much better as there are multiple places where the code would evaluate the same expression multiple times. This can be seen in evalFull and evalSimple when they evaluate a division statement. This could have been made better if using an let expression with the variable being equal to these expresions and the expresion being the same but with a variable instead,

so it would not have to be evaluated multiple times. This is a common feature that will happen in multiple areas.

4 Maintainability

4.1 Code

The maintainability of the code has a couple of problems a lot of the same code has been reused multiple times, for each of the 3 evaluation methods, where some auxiliary functions could have been used instead, so it would have been easier to maintain.

It has also not many comments about the functions so any non trivial thing that is not done to force a evaluation is not commented and can therefore be hard to find what it is. I have not used super hard to understand ways of doing any of my functions so should still be easy to understand what happens in each pattern.

4.2 Test suite

The test suit as written earlier is given good enough names to recognise what they test and can easily be changed if anything should change in how i were to interpret or in the language. But have not used any auxiliary functions if possible as I could not find any good ways. I have reused a lot of the testing between the different evaluate functions as I did not see the need to find new values, although I did not use the auxiliary functions and therefore should test everything on each evaluate function.