COMP2209 Coursework Report

Patrick Jfremov-Kustov
pjk1g22@soton.ac.uk

January 11, 2024

1 Introduction

The coding style follows a modular approach, where often my solutions are divided into different helper functions that each solve a particular sub-task. This, in turn, makes it more readable and more accessible to debug when you can isolate the problem. This is because you can pinpoint an error in the logic of a particular function, whereas having all the functionality in one block is much more challenging to fix.

I maintained consistent indentation for readability. The variable names are sometimes blatantly descriptive, as in more complex tasks, knowing precisely what a function can avoid mistaking it for other functions, changing it, and deviating further from a correct implementation. To create more robust code, I used "Maybe" types as part of my error handling to avoid any runtime errors.

To test my challenges, I used HUnit to handle various inputs from normal, error, edge and complex cases by specifying test inputs and expected outcomes using assertions. Then, I grouped the tests by having a separate main function for each challenge, allowing me to test specific parts of the code instead of testing the entire coursework. However, I also have a main function that combines all the challenges to test them by sequencing multiple IO actions.

For challenge 5 in particular, I created an alpha equivalence helper function so I could test if two lambda expressions are the same since two outputs could have different bound variables (leading to the test case saying it's incorrect) when, in fact, it comes from the same input.

In situations where I got stuck, I initially used debugging tools such as stack traces to understand how the output changes. However, due to Haskell's programming paradigm, using these tools gave less helpful information than expected compared to imperative languages. Otherwise, I would either revisit the Haskell documentation to further investigate how the functions would interact with the input given or write down how the input changes manually on paper line by line.

Reflecting on the challenges, my general programming strategy shifted from a previous imperative thinking style to a functional thinking style, which is seen through my extensive use of pattern matching, recursion, and higher-order functions to solve problems. Moreover, I've embraced the concepts of immutability and type safety to reduce the likelihood of runtime errors and simplify debugging overall.

2 Bibliography

Apart from adapting the template code from Challenge 6, all of my code is original.