COMP2209 - Lambda Calculus Challenges Report

Daniel Best (Student ID: 29777127)

January 7, 2020

In order to complete the challenges that were assigned to me for this project, I made use of various tools and techniques.

I chose to use Visual Studio Code as the IDE in which I coded the challenges in. I installed a plug-in for this called "Haskell Syntax Highlighting" that adds appropriate mark-up to make the code easier to read and distinguish between. However, I did not use the in-built console to run GHCI and test and run my code - instead, I had a separate Windows Terminal window that was running an instance of command prompt. This allowed me to split the screen in half, being able to code and then perform some kind of test without having to switch between tabs.

To assist with the testing process, I enhanced the existing Tests.hs file with a more complex test suite that distinguishes between tests and identifiers the specific tests that passed and failed. I also added functionality that informed me what exactly the output was if the test failed, so I did not have to waste any time copy and pasting the test into the console to find this information out. I added additional tests to each challenge, varying based on the knowledge I had in regards to the correct output of each challenge. A significant portion of these covered aspects of the challenge that were not currently being tested by the existing test cases.

Whilst testing my code during its development, I utilised two main methods. The first was simply to use let expressions in the command-line to quickly set more complex parameters, and then to use these new values to input into the function that I wanted to test. This saved significant time in that I could reuse these values in multiple functions with relative easy. The second was to use the in-built debugger tool to check the output of complex functions at each 'helper' function. This allowed me to quickly identify any such 'helper' functions that were returning unexpected values, and therefore allow me to identify where a problem may be and fix it.

As a side note, it is worth mentioning that I used the debugger significantly less whilst coding in Haskell than if I were coding in an imperative language. This is because of the nature of a functional programming language, where each function is intended to perform only one task, meaning that the problematic line of code causing a logic error can be more easily identified and corrected.