University of Southampton

Edward Wray ew7g23

COMP2209 Programming III (2024-25)

Haskell Programming Exercises Report

Submission date: December 31, 2024

Development Process

During development, I used VSCode as my integrated development environment (IDE). Its features, such as syntax highlighting, error detection, and suggestions for code improvement, greatly enhanced my workflow by identifying potential issues and offering ways to clean up code or remove redundancies. I discovered that using pattern matching, guards, and case of constructs provided far greater clarity and readability compared to traditional if statements. Additionally, where clauses and let..in expressions proved to be clean and effective tools for structuring code. For debugging, the Debug.Trace module was crucial, allowing me to use simple print statements (similar to those in other programming languages) to gain deeper insights into the program's behavior.

I did not find much use for lambda functions, as many simple lambda functions I considered writing already had some form of implementation in the standard prelude, reducing the need to define them manually. Monads were particularly valuable in question 4 and during testing, as they allowed me to write clear, concise, and readable code. That said, I chose to avoid using monads in other questions, as simpler solutions sufficed for those cases.

Testing

During testing, I did not use any frameworks or libraries, opting instead to write my own test functions. I defined a custom assertEquals function for each test, tailored towards the data types compared, and printed the results to the terminal. To run these tests, they needed to be manually copied into the terminal after loading the relevant question into GHCI. I designed the tests to build inductively, starting with basic functionality and progressively combining these into more complex cases. I also aimed to cover as many edge cases as possible. For instance, in question 6, I specifically tested edge cases related to the bound on the number of reductions.

I found that some questions could naturally complement each other in testing. For example, I used calcInteractions from question 1 to generate the observed rays needed to test question 2. Similarly, the tests for unparse and parseLamMacro from questions 3 and 4 are almost the same but applied in reverse. Additionally, since cpsTransform from question 5 is a component of compareInnerOuter in question 6, any test cases that worked for question 6 also provided further evidence of the validity of question 5.

For questions 5 and 6, I found it challenging to calculate the expected values of complex test cases manually. To address this, I focused on proving assumptions using smaller, simpler test cases. These allowed me to simplify the analysis of more intricate tests. For question 5, I began by testing whether variables clashed in straightforward scenarios. Once these tests were successful, I verified that my program produced the expected structure, assuming there were no variable name conflicts. For question 6, I created basic tests for the CPS-transformed reductions. After confirming these worked as intended, I assumed that if tests were correct for the non-CPS style, they would also hold for the CPS-transformed style. To ensure reliability, I still evaluated at least one complex test case in full for each question.

Code Fragments Included

1. **Graham Hutton's Parsing Module:** Functional parsing library from Chapter 13 of *Programming in Haskell*, Graham Hutton, Cambridge University Press, 2016. Included fully in Question 4.