CS 4402-01 Comparative Programming Languages

Instructor: Professor Rajesh Soni

Name: Ryohei Hayashi

Learning Journal 7

**Overview of the Week**

This week’s learning focused on the fundamentals of functional programming. Specifically, we covered lambda calculus, the differences between functional and imperative programming, functions and compound types, first-class functions (higher-order functions), currying, lazy and eager evaluation, and the basics of Haskell. The learning objectives included understanding key concepts of functional programming and creating simple programs in Haskell. Additionally, in the discussion forum, we debated the advantages of lazy evaluation and the significance of higher-order functions and currying.

**Personal Reflections**

Functional programming is vastly different from the imperative programming I have been familiar with, requiring me to adopt a new way of thinking. While imperative programming typically involves executing commands sequentially, functional programming emphasizes treating functions as first-class citizens, avoiding side effects, and maintaining a stateless approach. Initially, I found these concepts challenging, but I gradually began to appreciate their advantages.

Exploring Haskell, in particular, gave me deeper insights into functional programming. Unlike other programming languages, Haskell is highly abstract and allows intuitive manipulation of functions. Although its syntax and concepts took some time to get used to, working through the exercises helped me grasp how programs are abstracted, making functional programming increasingly fascinating.

The discussion forum offered valuable opportunities to exchange ideas with peers. For example, I gained new perspectives on lazy evaluation and currying. Lazy evaluation, especially in handling infinite lists, demonstrated its immense power. Additionally, seeing how others understood and utilized higher-order functions helped refine my understanding and broaden my approach.

**Topics Studied in Depth**

This week, I explored three key areas in detail:

**1. Advantages of Lazy Evaluation**

Haskell's lazy evaluation processes only the necessary parts of a computation, enabling efficient memory usage and large-scale data handling. For instance, using “take 10” on an infinite list to extract the first 10 elements showcased this feature effectively. Lazy evaluation eliminates unnecessary computations, which significantly boosts efficiency. Unlike imperative languages like Python, where handling infinite lists is impossible, Haskell’s lazy evaluation overcomes this limitation and proves to be a remarkable strength.

**2. Higher-Order Functions and Currying**

Higher-order functions, which take other functions as arguments or return functions as results, are incredibly versatile tools. They enhance code reusability and enable concise and modular programming. Currying, which involves applying arguments to functions incrementally, was particularly impactful. By partially applying functions, I realized how flexible and general-purpose the code becomes. For example, defining “max 4” as a function allows passing the remaining arguments later, showcasing remarkable adaptability in constructing programs.

**3. Fundamentals of Haskell**

Learning the basic structure of Haskell programs, from a simple “Hello World” to recursive Fibonacci sequence generation, offered a solid foundation in functional programming. I was impressed by Haskell’s abstraction capabilities and how it specializes in functional programming. Exercises involving recursion and infinite lists highlighted the language's strengths, such as its concise syntax and lazy evaluation. Haskell’s expressive power makes it a language I am eager to explore further.

**Future Challenges and Goals for Next Week**

Moving forward, my primary challenge is to deepen my ability to apply functional programming concepts. Understanding how to solve real-world problems using Haskell and comparing it with other functional languages are areas I aim to focus on. Next week, I plan to study indexing and hashing techniques in depth, which are crucial for algorithm design. Additionally, I intend to tackle more complex functional programming tasks in Haskell, aiming to develop skills applicable to practical use cases.

Word Count: 567

References

1. Ben-Ari, M. (2006). *Understanding programming languages.* Weizman Institute of Science.