Lambda Expressions and Currying in Haskell

Generated by LectureMate

February 22, 2025

1 Introduction

This lecture continues the discussion on higher-order functions, focusing on lambda expressions and currying in Haskell. We will explore the concepts of function application, partial application, and function composition.

2 Currying

Currying is a technique in functional programming where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument.

2.1 Function Types

In Haskell, a function with two arguments can be represented as:

$$\text{add} : \text{Int} \to \text{Int} \to \text{Int}$$

This indicates that the function takes an integer and returns another function that takes an integer and produces an integer.

2.2 Example of Currying

Consider the function `add` defined as:

```
add x y = x + y
```

When applied to two arguments, it can be viewed as:

$$\text{add}(3, 4) = 3 + 4 = 7$$

However, we can also apply it partially:

$$\text{add}(3) \to \text{a function that adds 3 to its argument}$$

This leads to the slogan: "A function of two numbers is the same as a function of the first number that returns a function of the second number."

2.3 Partial Application

Partial application allows us to apply a function to fewer arguments than it expects. For example:

```
addThree = add 3
```

Here, `addThree` is a function that adds 3 to its argument.

3 Lambda Expressions

Lambda expressions provide a way to define anonymous functions in Haskell.

3.1 Definition and Syntax

A lambda expression can be defined as:

$$\lambda x.(x \times x)$$

In Haskell, this is written as:

```
\x -> x * x
```

This defines a function that takes an argument x and returns $x \times x$.

3.2 Example of Lambda Expressions

Using lambda expressions, we can define a function to sum the squares of positive numbers in a list:

```
sumOfSquares = foldr (+) 0 (map (\x -> x * x) (filter (\x -> x > 0) list))
```

4 Sections

Sections are a convenient notation for partially applied functions in Haskell.

4.1 Definition of Sections

A section allows us to specify one argument of a binary function while leaving the other unspecified. For example:

```
(> 0)
```

This represents a function that checks if a number is greater than zero.

4.2 Examples of Sections

Sections can be used with various operators:

- $(\times 2)$ represents a function that doubles its input.
- $(2^{)}$ represents a function that raises 2 to the power of its input.

5 Function Composition

Function composition allows us to combine functions to create new functions.

5.1 Definition of Function Composition

In Haskell, function composition is denoted by the dot operator:

$$(f \circ g)(x) = f(g(x))$$

The type of function composition is:

$$(A \to B) \to (B \to C) \to (A \to C)$$

5.2 Associativity and Identity

Function composition is associative, meaning:

$$f \circ (g \circ h) = (f \circ g) \circ h$$

The identity function, which returns its input, acts as the identity element for composition:

$$\mathrm{id}(x) = x$$

6 Conclusion

This lecture covered the concepts of currying, lambda expressions, sections, and function composition in Haskell. These concepts are fundamental to understanding functional programming and enable powerful abstractions in code.