Recursion and Functional Programming in Haskell

Generated by LectureMate

February 22, 2025

1 Introduction

This lecture focuses on recursion, particularly in the context of functional programming using Haskell. The importance of understanding recursion is emphasized, as it is a fundamental concept in this course.

2 Careers Event

Before diving into recursion, an announcement was made regarding a careers event taking place in MacEwan Hall at the beginning of October. This event will feature employers looking for graduates and interns, making it a valuable opportunity for students.

3 Recursion Overview

Recursion is a method of defining functions in terms of themselves. The lecture builds on previous discussions about recursion on lists and extends it to integers, specifically natural numbers.

3.1 Recursion on Integers

An example of recursion on integers is provided through the Haskell function `enumFrom`. This function generates a list of integers from a lower bound m to an upper bound n.

3.1.1 Function Definition

The function is defined as follows:

```
enumFrom m n = if m > n then []
               else m : enumFrom (m + 1) n
```

The two cases are:

- Base case: If $m > n$, return an empty list.
- Recursive case: If $m \leq n$, return m cons the result of `enumFrom (m + 1) n`.

3.1.2 Computation Sequence

For example, calling `enumFrom 1 3` results in:

`enumFrom 1 3` $= 1 :$ `enumFrom 2 3` $= 1 : (2 :$ `enumFrom 3 3`$) = 1 : (2 : (3 :$ `enumFrom 4 3`$))$

The final result is $[1, 2, 3]$.

3.2 Factorial Function

The factorial function is another example of recursion:

```
factorial n = if n == 0 then 1
              else n * factorial (n - 1)
```

This function demonstrates a similar recursive pattern, where the factorial of n is defined in terms of the factorial of $n - 1$.

4 Lazy Evaluation

Haskell's ability to handle infinite data structures is introduced through lazy evaluation. This allows computations to be performed on infinite lists without requiring all values to be computed at once.

4.1 Infinite Lists

An example of generating an infinite list of natural numbers is:

```
enumFrom m = m : enumFrom (m + 1)
```

This function will generate numbers starting from m indefinitely.

4.2 Primes Example

A program to compute an infinite list of prime numbers is presented:

```
isPrime n = null [x | x <- [2..n-1], n `mod` x == 0]
primes = filter isPrime [2..]
```

This uses list comprehensions and demonstrates how to work with infinite lists.

5 The ZIP Function

The `zip` function in Haskell combines two lists into a list of pairs:

```
zip xs ys = case (xs, ys) of
    ([], _) -> []
    (_, []) -> []
    (x:xs', y:ys') -> (x, y) : zip xs' ys'
```

This function uses simultaneous recursion on both lists and has two base cases to handle empty lists.

5.1 Examples of ZIP

Using `zip` with two lists of different lengths will truncate the longer list:

```
zip [0, 1, 2] ['a', 'b', 'c', 'd'] = [(0, 'a'), (1, 'b'), (2, 'c')]
```

6 Conclusion

The lecture concludes with a discussion on the importance of recursion and functional programming techniques in Haskell. Students are encouraged to practice these concepts through coding exercises and examples.