Lecture Summary: Haskell Functions and Recursion

Generated by LectureMate

February 22, 2025

1 Introduction

This lecture covers important Haskell functions: `select`, `take`, and `drop`. These functions are essential for manipulating lists and understanding recursion in Haskell programming.

2 Careers Event Reminder

Before diving into the lecture content, a reminder was given about a careers event taking place at McEwan Hall, George Square. This event provides opportunities for internships and networking with potential employers.

3 Musical Introduction

The lecture began with a brief discussion of the Queen of the Night aria from Mozart's *Magic Flute*, highlighting its complexity and the performance of an eight-year-old singer.

4 Overview of Functions

The focus of the lecture is on three built-in Haskell functions:

- `select`: Selects an element from a list at a specified index.
- `take`: Returns the first n elements from a list.
- `drop`: Removes the first n elements from a list.

4.1 Function Definitions

4.1.1 Select

The `select` function is defined as follows:

```
select :: [a] -> Int -> a
select (x:_) 0 = x
select (_:xs) n = select xs (n - 1)
```

This function uses recursion to navigate through the list, decrementing the index until it reaches zero.

4.1.2 Take

The `take` function can be defined recursively:

```
take :: Int -> [a] -> [a]
take 0 _ = []
take _ [] = []
take n (x:xs) = x : take (n - 1) xs
```

This function returns an empty list if n is zero or the input list is empty. Otherwise, it includes the head of the list and recursively calls itself on the tail.

4.1.3 Drop

The `drop` function is defined similarly:

```
drop :: Int -> [a] -> [a]
drop 0 xs = xs
drop _ [] = []
drop n (_:xs) = drop (n - 1) xs
```

This function returns the original list if n is zero, and it recursively drops elements from the list until n reaches zero.

5 List Comprehensions

The lecture also covered how to define these functions using list comprehensions:

```
selectComp :: [a] -> Int -> [a]
selectComp xs i = [x | (x, j) <- zip xs [0..], j == i]

takeComp :: Int -> [a] -> [a]
takeComp n xs = [x | (x, j) <- zip xs [0..], j < n]

dropComp :: Int -> [a] -> [a]
dropComp n xs = [x | (x, j) <- zip xs [0..], j >= n]
```

These definitions utilize the `zip` function to pair elements with their indices.

6 Recursive Definitions

The lecture emphasized the importance of recursion in Haskell. Recursive definitions of lists and natural numbers were discussed:

- A list is either empty or consists of a head and a tail.
- Natural numbers can be defined recursively as either zero or the successor of a natural number.

7 Practical Examples

The practical applications of these functions were demonstrated through examples, showing how they can be used in real-world programming scenarios.

8 Infinite Lists

The lecture concluded with a discussion on infinite lists in Haskell. It was shown that while Haskell can handle infinite lists, care must be taken to avoid non-terminating computations.

9 Piano Arithmetic

Finally, the concept of Piano arithmetic was introduced, explaining how basic arithmetic operations can be defined recursively using natural numbers.

10 Conclusion

The lecture wrapped up with a Q&A session, inviting students to ask questions about the material covered. The next lecture will focus on higher-order functions in Haskell.