Lecture Summary: Expression Trees and Propositional Logic in Haskell

Generated by LectureMate

February 22, 2025

1 Introduction

This lecture focused on expression trees as an algebraic data type in Haskell, particularly in the context of representing syntax for arithmetic and logical expressions. The discussion included evaluation of these expressions and practical applications in programming.

2 Expression Trees

2.1 Arithmetic Expressions

The first example discussed was arithmetic expressions involving integers, addition, and multiplication. The representation was done using three cases in Haskell's algebraic data types. The evaluation function utilized pattern matching to handle these cases.

2.2 Logical Propositions

The lecture then transitioned to logical propositions, which included propositional variables (strings), constants (true, false), and logical operations (negation, disjunction, conjunction). This representation was more complex, requiring six cases due to the inclusion of variables.

2.3 Evaluation of Propositions

To evaluate a proposition, an additional parameter, the valuation, was introduced. This valuation is a function that maps variable names to their boolean values. The evaluation function was defined with cases for each constructor, allowing for the evaluation of logical expressions based on the provided valuation.

3 Truth Tables

The concept of truth tables was introduced to illustrate the evaluation of logical expressions. For example, the expression $a \wedge \neg a$ was analyzed, showing that it always evaluates to false regardless of the value of a.

3.1 Example Evaluation

An example was provided where the valuation of variables was defined, and the evaluation function was applied to determine the result of a logical expression. The process involved breaking down the expression into subexpressions and recursively evaluating them.

4 Satisfiability and Tautology

The lecture discussed how to determine if a proposition is satisfiable (at least one true value) or a tautology (always true). The evaluation function could be extended to compute these properties by analyzing the results of all possible valuations.

5 Optional Values in Haskell

The lecture introduced the `Maybe` type in Haskell, which is used to represent optional values. This type can be useful for functions that may not return a value, such as division by zero.

5.1 Examples of Maybe Type

Two examples were provided:

- An exponentiation function that defaults to 2 if no value is provided.
- A safe division function that returns `Nothing` if the divisor is zero.

6 Disjoint Union Types

The lecture concluded with the introduction of the `Either` type, which allows for a list of values of different types. This type can be used to create lists that contain either integers or strings, with constructors to distinguish between the two.

6.1 Example Functions

Functions were demonstrated to process lists of `Either` types, such as summing integers or concatenating strings.

7 Conclusion

The lecture provided a comprehensive overview of expression trees, logical propositions, and their evaluation in Haskell. The introduction of optional values and disjoint union types showcased the flexibility of Haskell's type system in handling complex data structures.