Propositions as Types

Generated by LectureMate

February 22, 2025

1 Introduction

Phil Wadler, a professor at the University of Edinburgh, introduces the topic of the lecture: propositions of types. He encourages questions throughout the lecture and sets the stage for a discussion on computability theory and its historical context.

2 Historical Context of Computability

2.1 Early Algorithms

Algorithms have a long history, dating back to Euclid's algorithm around 300 BCE and al-Khwarizmi's work around 800 CE. However, a formal definition of an algorithm emerged only in the early 1900s.

2.2 Key Figures

In 1935, three pivotal papers were published:

- Alonzo Church's Lambda Calculus
- Kurt Gödel's Recursive Functions
- Alan Turing's Turing Machines

These developments were influenced by David Hilbert's desire to formalize mathematics and create a decision algorithm for logical statements.

2.3 Gödel's Incompleteness Theorem

Gödel's incompleteness theorem demonstrated that within any consistent formal system, there are true statements that cannot be proven. This revelation led to the understanding that no algorithm could determine the truth of all mathematical statements.

3 Definitions of Computability

3.1 Church's Lambda Calculus

Church introduced the lambda calculus, a formal system for expressing computation. He defined numbers using Church numerals, where:

$$1 \equiv \lambda f.\lambda x.fx$$

$$2 \equiv \lambda f.\lambda x.f(fx)$$

This encoding allowed for the definition of arithmetic operations.

3.2 Turing Machines

Turing machines provided a mechanical model of computation. Turing argued that what a computer could do is equivalent to what a Turing machine could do, establishing a foundational concept in computer science.

3.3 Equivalence of Models

Church, Gödel, and Turing's models of computation were shown to be equivalent, leading to the conclusion that they all defined the same concept of computability.

4 Propositions as Types

4.1 Jensen's Contributions

Gerhard Jensen developed natural deduction and sequent calculus, introducing rules for logical inference. His work emphasized the importance of introduction and elimination rules in formal logic.

4.2 Curry-Howard Correspondence

The Curry-Howard correspondence establishes a deep connection between logic and computation:

- Propositions correspond to types.
- Proofs correspond to programs.
- Simplification of proofs corresponds to evaluation of programs.

4.3 Simply Typed Lambda Calculus

Church later developed simply typed lambda calculus to avoid paradoxes and ensure termination of programs. In this system:

$$\text{If } L : A \to B \text{ and } m : A, \text{ then } L\,m : B$$

5 Practical Applications

5.1 Programming with Types

The concepts from lambda calculus are foundational in modern programming languages, particularly functional languages like Haskell. The type system ensures that programs are well-structured and free from certain classes of errors.

5.2 Proof Assistants

Proof assistants leverage the Curry-Howard correspondence to treat proofs as programs, allowing for automated verification of logical statements.

6 Conclusion

Wadler concludes by emphasizing the significance of lambda calculus as a model of computation and its implications for programming languages. He encourages students to think of lambda calculus as a powerful tool for solving complex problems.