An Introduction to Type Theory

Felix Bradley

Department of Computer Science
Royal Holloway, University of London

28 February 2024

Presentation Overview

ROYAL HOLLOWAY UNIVERSITY OF LONDON

A Brief History of Types

- Let's rewind back to 300 BC...
- Ancient Greek mathematician Euclid studied geometry, and published his work in *Elements* [Euclid, c. 300 BC].
- His work on geometry relied on five axioms; Euclid's *postulates*.
- Life was good...
- ...for approximately 2000 years.
- In 1868, Italian mathematician Eugenio Beltrami proved that Euclid's *parallel postulate* is independent of the other axioms.
- Is Euclidean geometry consistent?
- Is *anything* we've done so far correct?

A Brief History of Types

- German mathematician Gottlob Frege published his *Begriffschrift* in 1879
- Frege identified issues with prior work on logic and made a proposal:
- There's some basic rules we should expect of logic: consistency, completeness, and decidability.
  1) Consistency: you cannot prove contradictory propositions
  2) Completeness: every proposition is either provable or refutable
  3) Decidability: every proposition can be tested and every proof can be derived

A Brief History of Types

- The year is 1901.
- Along comes British mathematician Bertrand Russell.
- Russell asks a simple question:
  Does the set of all sets that do not contain themselves contain itself?

$$x := \{s | s \notin s\} \qquad x \in x?$$

- Modern mathematics says, "Trick question! That set isn't definable!"

"Unless it is, in which case: trick question! It's not a set, it's a *proper class*!"

- Is the question well-defined? Is it complete? Are we missing something?
- There's a rush to solve this problem; to find a complete, consistent and decidable basis for all mathematics.

A Brief History of Types

- It gets worse.
- Theorem: There exists mathematical objects which cannot be explicitly calculated or described.
- Theorem: There exists theorems in Peano arithmetic that cannot be proven with Peano arithmetic.
- It gets even worse.
- Along comes Austrian-Hungarian mathematician Kurt Gödel.
- Gödel publishes his *incompleteness theorems.*
  1) If a system of logic is consistent, then it is incomplete.
  2) A system of logic cannot prove it's own consistency.

A Brief History of Types

- So why bother?
- Russell demonstrated that naïve set theory is inconsistent; unrestricted set specification leads to contradiction.
- These questions led to the development of Zermelo-Frankael Set Theory (ZF) and the study of powerful tools such as the Axiom of Choice.
- Many other systems and bases of logic were studied and developed during this time.
- Russell's own proposal for solving this problem was a theory of *types*. (1908)
- Every mathematical object has a 'type', and these types form a hierarchy. Each type and its objects can be constructed only using subtypes.

Introducing Lambda Calculus

- In the 1930s, American mathematician Alonzo Church introduced lambda calculus.
- What is $f(x)$? What does $f(1)$ mean, if we only have $f(x)$ defined?
- $f(1)$ means to take all the xs in $f(x)$ and replace them with 1s.
- Church formalised by introducing λ and the notion of reduction

$$f(x) = x^2 + 2x + 3 \Rightarrow f := \lambda x.(x^2 + 2x + 3)$$

$$1 + 1 = 2$$

Introducing Lambda Calculus

- In the 1930s, American mathematician Alonzo Church introduced lambda calculus.
- What is $f(x)$? What does $f(1)$ mean, if we only have $f(x)$ defined?
- $f(1)$ means to take all the xs in $f(x)$ and replace them with 1s.
- Church formalised by introducing λ and the notion of reduction

$$f(x) = x^2 + 2x + 3 \Rightarrow f := \lambda x.(x^2 + 2x + 3)$$

$$1 + 1 \neq 2 \Rightarrow 1 + 1 \twoheadrightarrow^* 2$$

Introducing Lambda Calculus

ROYAL HOLLOWAY UNIVERSITY OF LONDON

- As an abstract calculus, lambda calculus is a universal model for computation.
- It's a powerful enough system to represent arithmetic and logical operations.
- As a logic, lambda calculus was shown by Kleene and Rosser to be inconsistent.

Examples of Lambda Calculus

- Consider the function

$$\omega := \lambda x.(xx)$$

- Let's consider some examples:
- $\omega(2) \rightharpoonup (\lambda x.(xx))(2) \rightharpoonup (xx)[2/x] \rightharpoonup 2(2)$
- $\omega(\omega) \rightharpoonup (\lambda x.(xx))(\omega) \rightharpoonup (xx)[2/x] \rightharpoonup \omega(\omega)$

Simply Typed Lambda Calculus

- How did Church solve this problem? Type theory!
- Church's Simply Typed Lambda Calculus restricted lambda calculus to only those terms which could be *typed.*
- $2 : \mathbb{N} \qquad (+) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$
- $d := \lambda(x : \mathbb{N}).2x$
- $d(2) \rightharpoonup^* 4$
- $d(\text{"}apple\text{"})$?
- $\omega := \lambda x.(xx)$ can't be typed.
- This restricted lambda calculus *is* logically consistent.

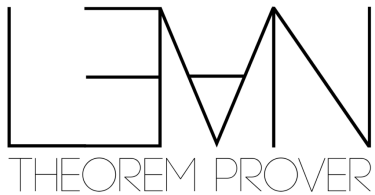
The Curry-Howard Correspondence

ROYAL HOLLOWAY UNIVERSITY OF LONDON

- By 1970, Mathematician Haskell Curry and logician William Alvin Howard realise something remarkable.
- Certain proof systems, such as natural deduction, behave kind of like an intuitionistic version of lambda calculus.
- This leads to the Curry-Howard Correspondence: that the act of writing a mathematical proof *has a one-to-one correspondence* with the act of writing a computer program.
- If A and B are propositions, then I may write a proof p that $A \to B$.
- If A and B are types, then I may describe a function $f : A \to B$ that takes in an object of type A and returns an object of type B.
- If we interpret *propositions as types*, then writing a proof is the same as describing a function.

Proof Assistants

- The Curry-Howard Correspondence has lead to the development of *proof assistants*.
- Proof assistants are advanced programming languages.
- You can formalise mathematics and write up your proofs.
- The proof assistant can verify your proof and ensure that you are correct.
- Mathematicians like Kevin Buzzard and Terence Tao are pushing proof assistants with cutting-edge research.
  [Buzzard, 2024, Tao, 2024].



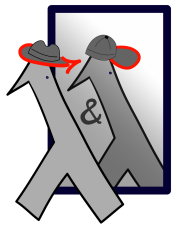
Programming Languages

ROYAL HOLLOWAY UNIVERSITY OF LONDON

- In fact, type theory has worked its way into every popular programming language.
- It stops you, the programmer, from making mistakes.
- Type-driven programming paradigms help to prevent bugs.

In Conclusion

- Type theory was developed as an alternative to set theory.
- Type theory has bloomed into something new and exciting.
- Type theory is an active area of research.
- We're trying to get students more involved with type theory!



Proof and Programs Club @ RHUL

Thank you for your time!

References

Buzzard, K. (2024).

How to prove fermat's last theorem.

Cambridge, UK. Faculty of Mathematics, University of Cambridge.

Euclid (300 B.C.).

Elements.

Various, Alexandria, Egypt.

Tao, T. (2024).

Machine assisted proof.

San Francisco, California. 2024 Joint Mathematics Meetings.
https://youtu.be/AayZuuDDKP0.