What is Type Theory?

Thorsten Altenkirch

My Own Motivation

Looking for the perfect programming language

Since 1975: from BASIC to **HoTT**

Foundations of Mathematics

Most mathematicians would answer:

- **Set Theory**, usually **ZFC**

ZFC designed around **1920**

Compare: first programming language (FORTRAN) in the **1950s**

People still use FORTRAN, but:
better alternatives exist for abstraction and modularity

Claim

Type Theory is a modern alternative to Set Theory

- better suited to modern mathematics
- supports abstraction and modularity
- matches the needs of computer science
- success of proof assistants (Lean, Coq, Agda)

Origins of Set Theory

Cantor's idea:

A set is the extension of a property.

Example: the property "being green" gives the set of all green things.

Pure set theory:

- no urelements
- everything is a set

Constructing the Naturals

Von Neumann construction:

$$0 = \{\}, \quad 1 = \{0\}, \quad 2 = \{0, 1\}, \ldots$$

Cantor's idea resembles the domain equation

$$\mathsf{Set} = \mathsf{Set} \to \mathsf{Prop}$$

A negative recursive domain equation: problematic from a CS perspective.

Russell showed naïve set theory is inconsistent.

Zermelo–Fraenkel fix: **restrict comprehension**.

The Birth of Type Theory

Per Martin-Löf (1970s)

Influences:

- System T, System F
- intuitionism: "mathematics is *created*, not discovered"

Goal:

- a new foundation for mathematics
- constructive and computational

Propositions as Types

In Set Theory:

- logic comes first (first-order logic)
- sets defined via axioms about $\in$

In Type Theory:

- **types come first**
- propositions are identified with types
- logic is **emergent**, not primitive

Typing vs Membership

Type Theory:

- Every element has a **unique** type
- Typing is a **judgement**
- Analogy: **static typing**

Set Theory:

- Membership is just a relation
- An object can belong to many sets
- Analogy: **dynamic typing**

Representation Matters (or Does It?)

In Set Theory:

- representation-level questions are meaningful
- e.g. "Is $\{\} \in \mathbb{N}$?"

But abstract mathematics shouldn't depend on representation.

Type Theory enforces **representation independence**.

Univalence

Voevodsky's Univalence Principle:

Isomorphic structures are equal.

Meaning:

- mathematics should depend only on structure
- bijective equivalences count as equality
- formalised in Homotopy Type Theory

Summary

- Set theory: classical, extensional, representation-sensitive
- Type theory: constructive, computational, structural
- Modern proof assistants rely on type theory
- Univalence captures structural equality

Type Theory is a foundation built for modern mathematics.

Agda

Today we will use **Agda**, an implementation of Type Theory that is very faithful to Martin-Löf's original ideas (unlike Lean, which includes more classical and set-theoretic features).

Agda is genuinely a **programming language** with a powerful dependent type system, suitable both for doing mathematics and for constructing programs in computer science.

Designed by **Ulf Norell** in 2007 and developed since by a large community.

Cubical Agda:

- implements a constructive version of Homotopy Type Theory
- supports **univalence**
- supports **higher inductive types**