

Formalizing mathematics with cubical type theory

Michael Zhang

October 19, 2024

Intro

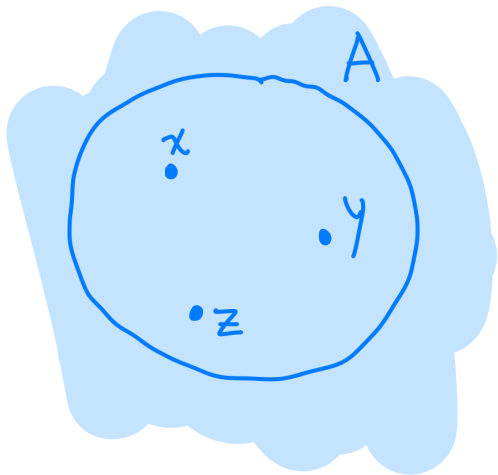
- ▶ Why formalize proofs?
- ▶ What can we formalize?
- ▶ What are some existing theorem provers?
 - ▶ Rocq (Coq), Lean, Agda

Demo

Set theory

- ▶ Math is built upon sets
- ▶ We often operate at a much higher level
- ▶ Skip the details
- ▶ We can't do this for mechanized systems!

Introducing the type



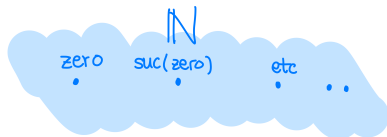
Proofs are programs

Logic side	Programming side
formula	type
proof	term
formula is true	type has an element
formula is false	type does not have an element
logical constant \top (truth)	unit type
logical constant \perp (falsehood)	empty type
implication	function type
conjunction	product type
disjunction	sum type
universal quantification	dependent product type
existential quantification	dependent sum type

Important features of Martin-Löf type theory

- ▶ Stratified universes: $\text{Type}_0, \text{Type}_1, \text{Type}_2, \dots$
- ▶ Inductive types, defined by type former, constructors, eliminators, and computation rules

Important features of Martin-Löf type theory



- ▶ For example, \mathbb{N} is defined with 2 possible constructors: zero and suc.

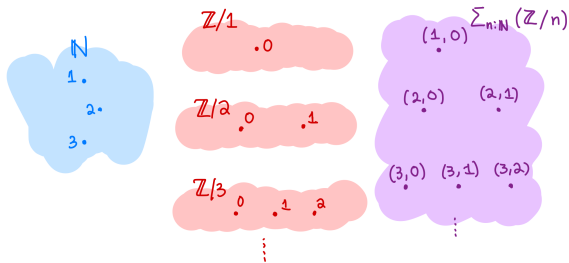
$$\frac{}{\Gamma \vdash \text{zero} : \mathbb{N}} \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \text{suc}(n) : \mathbb{N}}$$

- ▶ Elimination rule for \mathbb{N} :

$$\frac{\Gamma \vdash c_z : C \quad \Gamma, (x : \mathbb{N}), (y : C) \vdash (c_s : C) \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \text{ind}_{\mathbb{N}}(c_z, x.y.c_s, n) : C}$$

Important features of Martin-Löf type theory

- ▶ *Dependent sums*, or Σ -types



$\text{isEven} : \mathbb{N} \rightarrow \text{Type}$

$\text{Even} : \sum_{n:\mathbb{N}} \text{isEven}(n)$

Important features of Martin-Löf type theory

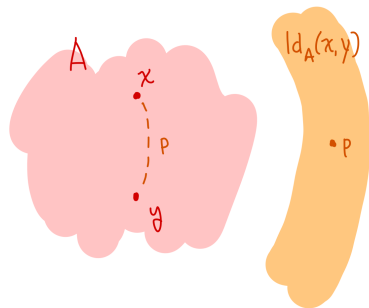
- ▶ *Dependent functions*, or Π -types

List : Type \rightarrow Type

Vec : Type $\rightarrow \mathbb{N} \rightarrow$ Type

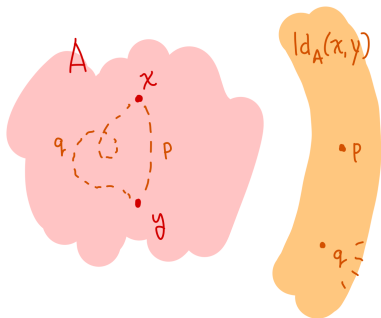
append : $\prod_{A:\text{Type}} \left(\prod_{m,n:\mathbb{N}} (\text{Vec}(A, m) \rightarrow \text{Vec}(A, n) \rightarrow \text{Vec}(A, m + n)) \right)$

Important features of Martin-Löf type theory



- ▶ The identity type, $\text{Id}_A(x, y)$, for some $x, y : A$
- ▶ Also written as $x \equiv_A y$, or just $x \equiv y$ if A is obvious
- ▶ Single constructor: $\text{refl}_x : \text{Id}_A(x, x)$

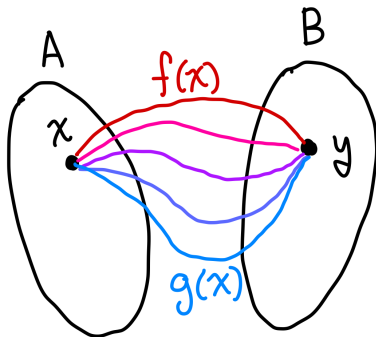
Important features of Martin-Löf type theory



- ▶ There can be multiple paths between points!

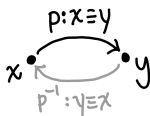
Homotopy type theory

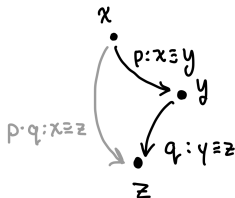
- ▶ A homotopy, from algebraic topology, is a way to continuously deform one path into another ($A \times [0, 1] \rightarrow B$)
- ▶ In type theory, we consider two functions $f, g : A \rightarrow B$ as being homotopic if we can inhabit $h : (x : A) \rightarrow f(x) \equiv g(x)$



Homotopy type theory

$$p: x \equiv x$$

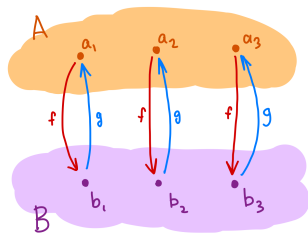

$$p: x \equiv y$$

$$p^{-1}: y \equiv x$$

$$p: x \equiv y$$

$$q: y \equiv z$$
$$p \cdot q: x \equiv z$$

Isomorphism

- ▶ A function $f : A \rightarrow B$
- ▶ A function $g : B \rightarrow A$

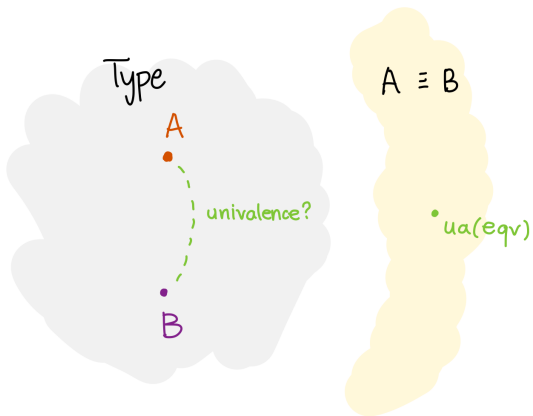
- ▶ $\prod_{(a:A)} g(f(a)) \equiv a$
- ▶ $\prod_{(b:B)} f(g(b)) \equiv b$



$$\bullet \xrightarrow{f} \bullet \xrightarrow{g} \bullet = \text{id}_A$$

$$\bullet \xrightarrow{g} \bullet \xrightarrow{f} \bullet = \text{id}_B$$

Univalence



- ▶ Equivalences *are* identities

Univalence

- ▶ Equivalences *are* identities

$$(A \simeq B) \simeq (A \equiv B)$$

- ▶ Intuitively, if you wanted to use a B where you wanted to use an A , just convert it and then convert it back later
- ▶ Importantly, there can be multiple inhabitants of $A \equiv B$
 - ▶ Booleans!
ua(id) and ua(not) are different elements of $\text{Bool} \equiv \text{Bool}$
- ▶ Transport allows you to use A and B interchangeably

$\text{transport} : (P : (X : \text{Type}) \rightarrow \text{Type}) \rightarrow (p : x \equiv_X y) \rightarrow P(x) \rightarrow P(y)$

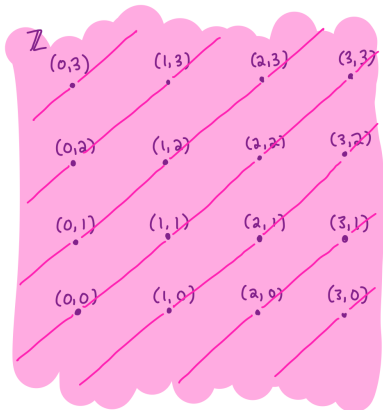
Univalence axiom?

- ▶ Unfortunately, univalence does not compute in Book HoTT
- ▶ Assume the functions we want as axioms

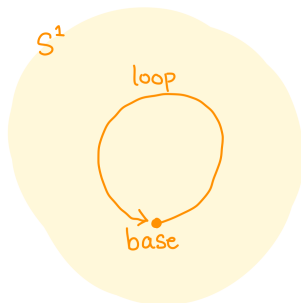
$$\text{ua} : (A \simeq B) \rightarrow (A \equiv B)$$

Higher inductive types

- ▶ Normally inductive types give us ways to construct *points*
- ▶ Higher inductive types give us ways to construct *paths*



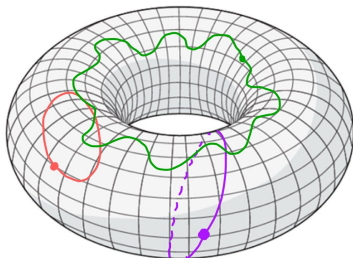
Circle (S^1)



- ▶ base : S^1 , some arbitrary base point
- ▶ loop : base \equiv base, a path representing the rest of the circle

The fundamental group π_1

- ▶ One central idea of algebraic topology: identify which spaces are different from each other (i.e donuts and coffee mugs are different from a solid sphere)
- ▶ The fundamental group is one metric of identifying spaces
- ▶ The fundamental group measures the "loop space"



The fundamental group of the circle

- ▶ The circle (S^1) is an example of a simple but non-trivial space because of the loop
- ▶ Determining fundamental groups of n -spheres is a difficult problem in algebraic topology
- ▶ Fortunately, $\pi_1(S^1)$ has a known solution

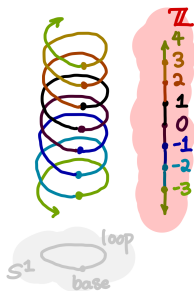
The fundamental group of the circle

- ▶ Fundamental group asks us about the loops in the circle space
- ▶ $\text{base} \equiv \text{base}$ because we don't care about choice of base point
- ▶ Some example elements:
 - ▶ ...
 - ▶ $\text{loop}^{-1} \cdot \text{loop}^{-1}$
 - ▶ loop^{-1}
 - ▶ refl
 - ▶ loop
 - ▶ $\text{loop} \cdot \text{loop}$
 - ▶ ...
- ▶ The fundamental group of the circle space is the integers

$$\pi_1(S^1) \simeq \mathbb{Z}$$

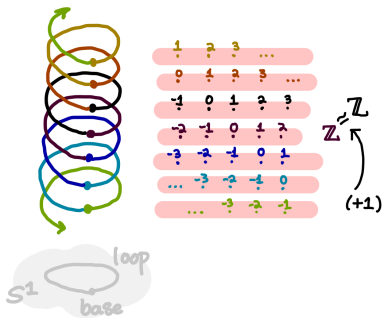
The fundamental group of the circle, core idea

- ▶ Use a winding helix to represent both!



The fundamental group of the circle, core idea

- ▶ Problem: we want multiple loopings to map us to different integers.
- ▶ Idea: define a custom type family that takes different number of loops to different "rotations" of the integers!
- ▶ Define the encoding:
 - ▶ $\text{code} : S^1 \rightarrow \text{Type}$
 - ▶ $\text{code}(\text{base}) = \mathbb{Z}$
 - ▶ $\text{code}(\text{loop}) = \text{ua}(\text{suc})$



The fundamental group of the circle, core idea

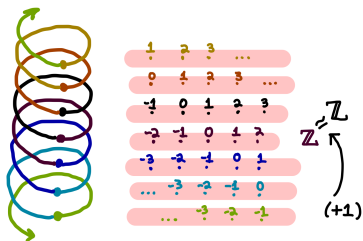
- ▶ Need to define the following data to prove the equivalence:

$$(\text{base} \equiv_{S^1} c) \simeq \text{code}(c)$$

- ▶ For some $(c : S^1)$ and $(n : \mathbb{N})$ that encodes how many loopings a path to c could take
 - ▶ $f : (\text{base} \equiv_{S^1} c) \rightarrow \text{code}(c)$
 - ▶ $g : \text{code}(c) \rightarrow (\text{base} \equiv_{S^1} c)$
 - ▶ $(g \circ f)(p) \equiv \text{id}_{\text{base} \equiv_{S^1} c}$
 - ▶ $(f \circ g)(n) \equiv \text{id}_{\mathbb{Z}}$

Homotopy type theory proof idea

- ▶ For $f : (\text{base} \equiv_{S^1} c) \rightarrow \text{code}(c)$, just use the winding map!
- ▶ This allows us to turn any loop into an integer
- ▶ For some $p : \text{base} \equiv \text{base}$, $\text{transport}^{\text{code}}(p, 0)$ turns 0 into any integer by mapping over the equivalence between integers



Homotopy type theory proof idea

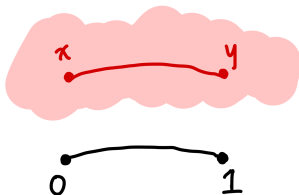
- ▶ For $g : \text{code}(c) \rightarrow \text{base} \equiv_{S^1} c$, we can iteratively compose loops to refl using a function loop^n
 - ▶ $\text{loop}^{-1+n} = \text{loop}^n \cdot \text{loop}^{-1}$
 - ▶ $\text{loop}^0 = \text{refl}$
 - ▶ $\text{loop}^{n+1} = \text{loop}^n \cdot \text{loop}$
- ▶ Then, use this to define g
 - ▶ $g(c : \text{code}(\text{base})) = \text{loop}^c$
 - ▶ $g(c : \text{code}(\text{loop})) : (\text{loop}^c \equiv_{\text{loop}}^{\lambda x \mapsto \text{code}(x) \rightarrow \text{base} \equiv x} \text{loop}^c)$
 - ▶ This can be defined in two ways

Homotopy type theory proof

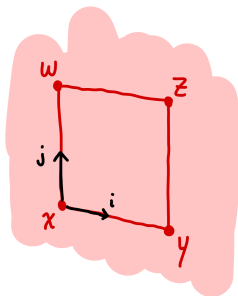
- ▶ The homotopies $(g \circ f \equiv \text{id}_{(\text{base} \equiv_{S^1} c)})$ and $(f \circ g \equiv \text{id}_{\text{code}(c)})$ can be proven just by applying path induction and these groupoid laws of paths:
 - ▶ Identity: $(p : x \equiv_A y) \rightarrow p \cdot \text{refl} \equiv p$
 - ▶ Identity: $(p : x \equiv_A y) \rightarrow \text{refl} \cdot p \equiv p$
 - ▶ Inverse: $(p : x \equiv_A y) \rightarrow p \cdot p^{-1} \equiv \text{refl}$
 - ▶ Inverse: $(p : x \equiv_A y) \rightarrow p^{-1} \cdot p \equiv \text{refl}$
 - ▶ Associativity: $(p : x \equiv_A y) \rightarrow (q : y \equiv_A z) \rightarrow (r : z \equiv_A w) \rightarrow (p \cdot q) \cdot r \equiv p \cdot (q \cdot r)$

Cubical type theory

- ▶ Make paths based on the interval type I which represents $[0, 1]$
- ▶ The interval is a primitive construct!



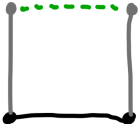
Cubical type theory



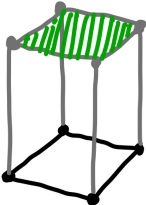
- ▶ Construct squares and cubes to create paths by composition and filling
- ▶ Consequences
 - ▶ Transport is a primitive notion
 - ▶ Makes the univalence "axiom" definable

Composition

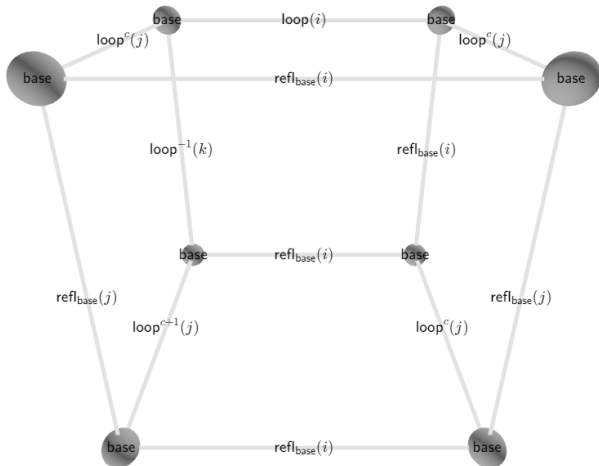
2 dimensions



3 dimensions



Defining g with cubes



Current work

- ▶ Re-formalizing results from HoTT book chapters 7 on n -types and 8 on homotopy theory
- ▶ Re-formalizing results from Floris van Doorn's dissertation on spectral sequences

Conclusion and references

► Code: <https://git.mzhang.io/michael/type-theory>

- [1] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: <https://homotopytypetheory.org/book>, 2013.
- [2] Cyril Cohen et al. “Cubical Type Theory: a constructive interpretation of the univalence axiom”. In: [arXiv:1611.02108](https://arxiv.org/abs/1611.02108) (Nov. 2016). [arXiv:1611.02108](https://arxiv.org/abs/1611.02108) [cs, math]. DOI: [10.48550/arXiv.1611.02108](https://doi.org/10.48550/arXiv.1611.02108). URL: <http://arxiv.org/abs/1611.02108>.