Introduction to homotopy type theory / univalent foundations

Tom de Jong

University of Nottingham, UK

10th autumn school *Proof and Computation*

15 September 2026, Fischbachau, Germany — Lecture 2/3

Overview

1. A few words on constructivism
2. Introduction to some fundamental notions from Voevodsky's univalent foundations
3. Very brief introduction to the syntax of MLTT

Constructivism and computation

Martin-Löf on mathematics and computation

From Per Martin-Löf's 1982 Constructive Mathematics and Computer Programming:
https://archive-pml.github.io/martin-lof/pdfs/Constructive-mathematics-and-computer-programming-1982.pdf

[...] the close connection between constructive mathematics and programming explains why the intuitionistic type theory (Martin-Löf 1975), which I began to develop solely with the philosophical motive of clarifying the syntax and semantics of intuitionistic mathematics, may equally well be viewed as a programming language.

↳ Proof assistants!

Martin-Löf on mathematics and computation

From Per Martin-Löf's 1982 Constructive Mathematics and Computer Programming:
https://archive-pml.github.io/martin-lof/pdfs/Constructive-mathematics-and-computer-programming-1982.pdf

[...] the close connection between constructive mathematics and programming explains why the intuitionistic type theory (Martin-Löf 1975), which I began to develop solely with the philosophical motive of clarifying the syntax and semantics of intuitionistic mathematics, may equally well be viewed as a programming language.

↳ Proof assistants!

Curry–Howard–Lambek correspondence:

- the proof of $P \to (Q \to P)$ corresponds to the program that takes two inputs and returns only the first; in λ-calculus notation $\lambda p.\lambda q.p$.
- the inference rule $\dfrac{P \to R \qquad Q \to R}{P \vee Q \to R}$ corresponds to a case analysis program.

In the concluding remarks of that paper, Martin-Löf writes:

[the] uninhibited use [of excluded middle] would lead to programs that you did not know how to execute.

Indeed, we can add excluded middle, $P \vee \neg P$, as an axiom to MLTT, but how would we compute with it?

Why cubical type theory?

- By Voevodsky's simplicial sets model, we know that we can consistently add his univalence axiom to MLTT.

  But just like with excluded middle, that does not give it a computational interpretation.

  In particular, proofs that use univalence may get "stuck" on it when we try to compute with them.

Thierry Coquand

Why cubical type theory?

- By Voevodsky's simplicial sets model, we know that we can consistently add his univalence axiom to MLTT.

  But just like with excluded middle, that does not give it a computational interpretation.

  In particular, proofs that use univalence may get "stuck" on it when we try to compute with them.

- Coquand et al. created **cubical type theory** to give a computational meaning to the univalence axiom: in cubical type theory there is a program that implements univalence.

- Cubical type theory was inspired by a **constructive** model in cubical (as opposed to simplicial) sets, so the (constructive) semantics came first.

Thierry Coquand

HoTT and constructive/classical mathematics

By default, HoTT is constructive in the sense that excluded middle or the axiom of the choice are not baked in. However, as detailed on the next slide,

HoTT is compatible with classical axioms (provided they are phrased carefully).

Question: So why is there often a focus on constructive mathematics in HoTT?

HoTT and constructive/classical mathematics

By default, HoTT is constructive in the sense that excluded middle or the axiom of the choice are not baked in. However, as detailed on the next slide,

HoTT is compatible with classical axioms (provided they are phrased carefully).

Question: So why is there often a focus on constructive mathematics in HoTT?

Some answers:

1. For historical/social reasons: type theory originated as a system for constructive mathematics.
2. Because of the connection to programming, an experience strenghtened / (best?) felt by formalizing mathematics (in Agda).
3. Because of the synthetic mathematics programme: even if you "believe in" excluded middle and the axiom of choice, these principles fail in some toposes.
   This is not a matter of philosophy/taste but of mathematics!

Voevodsky's univalent foundations

Voevodsky's univalent foundations

The simplicial sets model of HoTT is very relevant and important, but I focus on the novelties of Voevodsky's *Foundations* Coq library here.

The significance lied in his **new and elegant definitions**, among them:

- n-(truncated) types (called "hlevels" by Voevodsky), with **contractible** types as a fundamental instance;
- the **fiber** of a function;
- a well behaved notion of **equivalence**;
- the **univalence axiom** which connects equivalences with Martin-Löf's identity types.

n-types

Voevodsky realized that MLTT's identity types can be used to **define** (higher) groupoids *inside type theory*.

For $n \in \{-2, -1, 0, \ldots\}$, we can define a hierarchy of n-types (a.k.a. n-truncated types)

$$-2\text{-types} \subset -1\text{-types} \subset 0\text{-types} \subset \ldots \qquad \text{with}$$

- 0-types being ordinary sets ("types à la Axiom K");
- 1-types being ordinary groupoids ("types à la Hofmann–Streicher");

---

n-types

Voevodsky realized that MLTT's identity types can be used to **define** (higher) groupoids *inside type theory*.

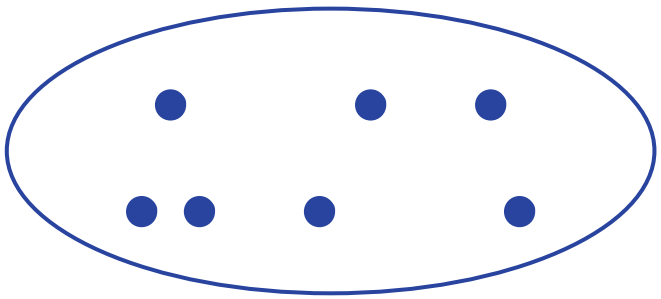
For $n \in \{-2, -1, 0, \ldots\}$, we can define a hierarchy of n-types (a.k.a. n-truncated types)

$$(-2)\text{-types} \subset (-1)\text{-types} \subset 0\text{-types} \subset \ldots \qquad \text{with}$$

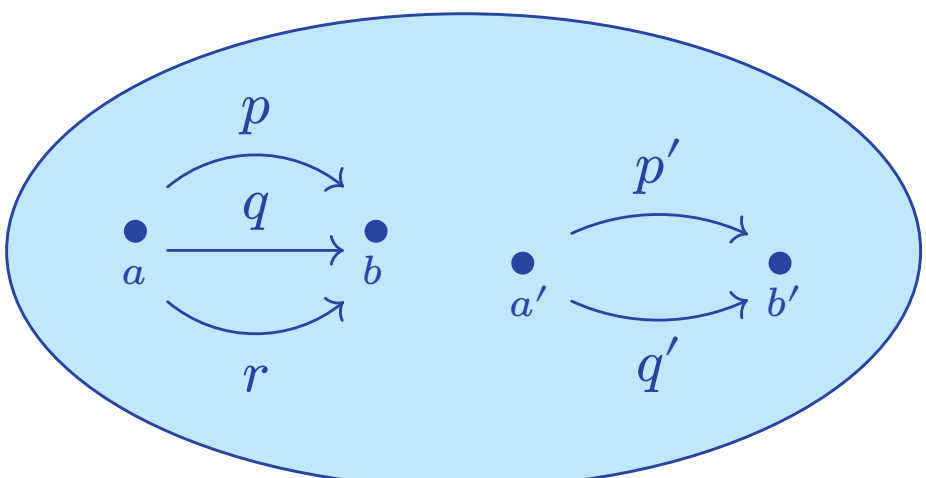
- 0-types being ordinary sets ("types à la Axiom K");
- 1-types being ordinary groupoids ("types à la Hofmann–Streicher");

- (-1)-types are the **propositions** (or **subsingletons**) — this is where we will **recover logic**;
- (-2)-types have just a single point.
  These are known as **contractible** — this notion is surprisingly fundamental!

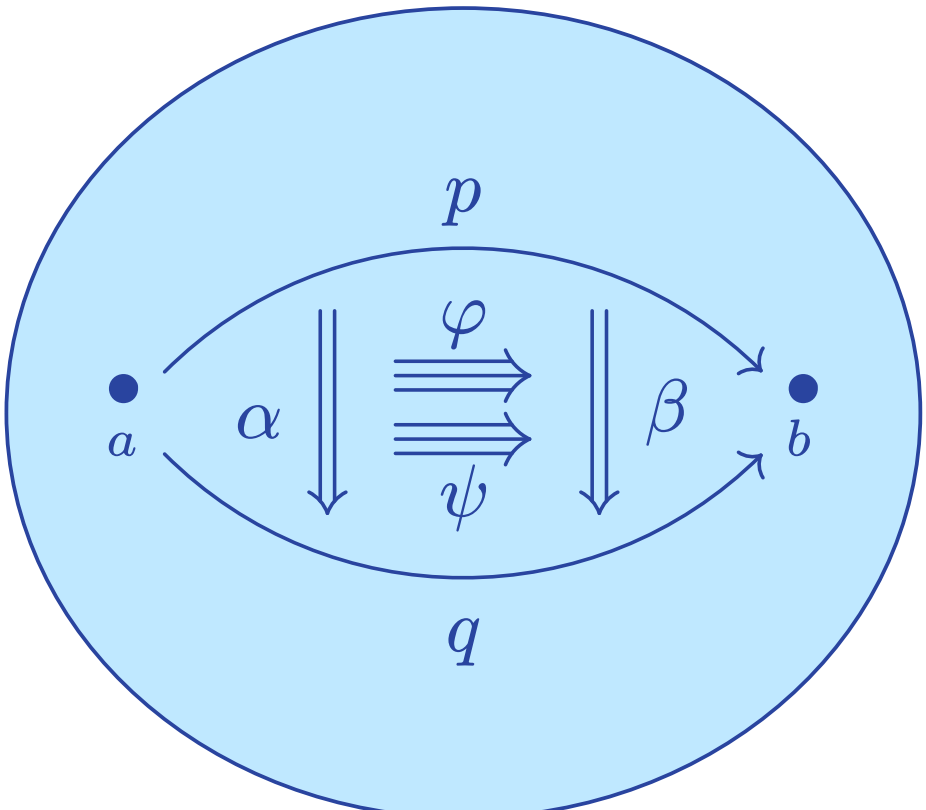
For $n \geq 0$, a good way to think of n-types is as those types whose (iterated) identity types trivialize above level n.



An illustration of a 0-type.



An illustration of a 1-type.



An illustration of a higher type.

Examples of (higher) types

- A **proposition**, **subsingleton**, or **-1-type** is a type with at most one element.
- A **0-type** or **set** is a type whose identity types all have at most one element, i.e. all its identity types are propositions.
- An **$n+1$-type** is a type whose identity types are all n-types.

Examples of (higher) types

- A **proposition**, **subsingleton**, or **-1-type** is a type with at most one element.
- A **0-type** or **set** is a type whose identity types all have at most one element, i.e. all its identity types are propositions.
- An **$n+1$-type** is a type whose identity types are all n-types.

- The empty type $\mathbb{0}$ and the type $\mathbb{1}$ with exactly one element are examples of propositions, i.e. -1-types.
- We assume to have an inductive type $\mathbb{N}$ of natural numbers. It can be shown to be a 0-type.

  We can construct $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ as usual, and the resulting types are 0-types.
- With **univalence**, we can construct examples of n-types for any n, and even of untruncated types (i.e. types with nontrivial identity types at any level).

Function extensionality (warm up to the univalence axiom)

Question: When should two functions be equal?

Answer: Two functions $f, g : A \to B$ should be equal when they are pointwise equal, i.e. for every point a in A, the values $f(a)$ and $g(a)$ are equal.

Function extensionality (warm up to the univalence axiom)

Question: When should two functions be equal?

Answer: Two functions $f, g : A \to B$ should be equal when they are pointwise equal, i.e. for every point a in A, the values $f(a)$ and $g(a)$ are equal.

- Suppose we have functions $f, g : A \to B$ in MLTT. We can construct a type $f \sim g$ expressing that f and g are pointwise equal.
- We can also consider the identity type $\mathrm{Id}_{A \to B}(f, g)$. How does it compare to $f \sim g$?

Function extensionality (warm up to the univalence axiom)

Question: When should two functions be equal?

Answer: Two functions $f, g : A \to B$ should be equal when they are pointwise equal, i.e. for every point a in A, the values $f(a)$ and $g(a)$ are equal.

- Suppose we have functions $f, g : A \to B$ in MLTT. We can construct a type $f \sim g$ expressing that f and g are pointwise equal.
- We can also consider the identity type $\mathrm{Id}_{A\to B}(f, g)$. How does it compare to $f \sim g$?
- We always have $\mathrm{Id}_{A\to B}(f, g) \longrightarrow f \sim g$, but the converse is not derivable in plain MLTT.
- Therefore, people considered the **function extensionality** axiom:

$$f \sim g \longrightarrow \mathrm{Id}_{A\to B}(f, g) \quad \text{“pointwise equal functions are equal”.}$$

The univalence axiom: universe extensionality

Question: When should two types, living in a universe $\mathcal{U}$, be equal?

Voevodsky's answer: Two such types A and B should be equal when they are **equivalent**.

The univalence axiom: universe extensionality

Question: When should two types, living in a universe $\mathcal{U}$, be equal?

Voevodsky's answer: Two such types A and B should be equal when they are **equivalent**.

- This requires a good notion of equivalence! Fortunately, Voevodsky provided this; he defined what it means for a function $f : A \to B$ to be an equivalence.
  Voevodsky: f is an equivalence when its fibers are contractible types. Alternatively, when it has a left and right inverse.
- We can collect all equivalences from A to B into a type $A \simeq B$.
- A naive formulation of the above answer would be to assume that there is a map $A \simeq B \longrightarrow \mathrm{Id}_{\mathcal{U}}(A, B)$.

The univalence axiom: universe extensionality

Question: When should two types, living in a universe $\mathcal{U}$, be equal?

Voevodsky's answer: Two such types A and B should be equal when they are **equivalent**.

- This requires a good notion of equivalence! Fortunately, Voevodsky provided this; he defined what it means for a function $f : A \to B$ to be an equivalence.
  Voevodsky: f is an equivalence when its fibers are contractible types. Alternatively, when it has a left and right inverse.
- We can collect all equivalences from A to B into a type $A \simeq B$.
- A naive formulation of the above answer would be to assume that there is a map $A \simeq B \longrightarrow \mathrm{Id}_{\mathcal{U}}(A, B)$.
- A more refined formulation, **Voevodsky's univalence axiom**, says that the canonical map $\mathrm{Id}_{\mathcal{U}}(A, B) \longrightarrow A \simeq B$ which can always be defined is itself an equivalence.

Univalence at work: examples of higher types

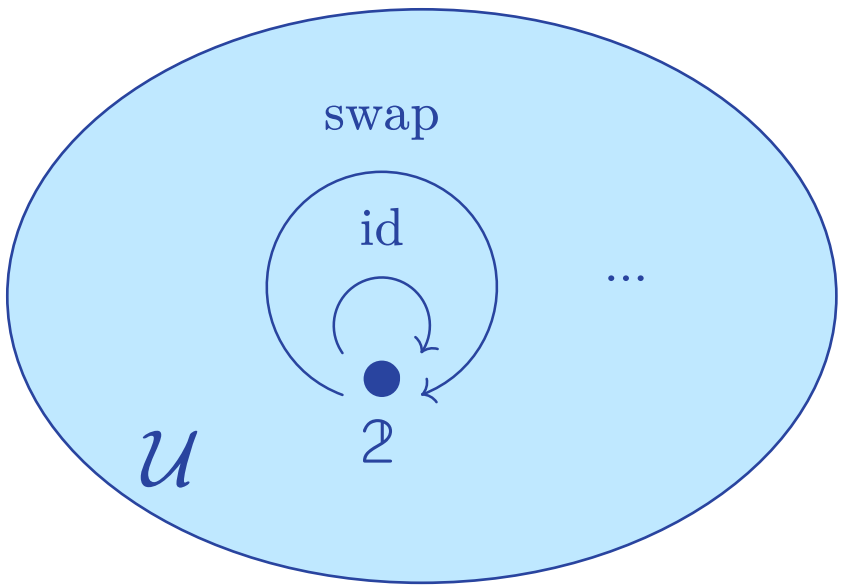
Universes

Let 2 be the type with exactly two elements (0 and 1, say) and let $\mathcal{U}$ be a universe that contains it.

If $\mathcal{U}$ is univalent, then $\mathrm{Id}_{\mathcal{U}}(2, 2)$ and $2 \simeq 2$ are equivalent.

But it's easy to see that there exactly two equivalences from 2 to itself, namely the identity and the map that swaps 0 and 1.

Thus, the identity type $\mathrm{Id}_{\mathcal{U}}(2, 2)$ has exactly two elements, and $\mathcal{U}$ is an example of a higher type.

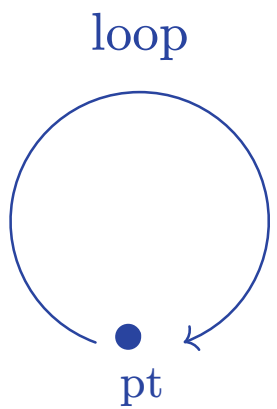


Circle

The circle is an example of a **higher inductive type**.

While the type $\mathbb{N}$ of natural numbers is inductively generated by two constructors $0 : \mathbb{N}$ and $\mathrm{succ}(n) : \mathbb{N}$ (for $n : \mathbb{N}$), the circle S^1 is inductively generated by

- a single point constructor $\mathrm{pt} : S^1$,
- and a *path* constructor $\mathrm{loop} : \mathrm{Id}_{S^1}(\mathrm{pt}, \mathrm{pt})$.



With univalence, we can show that $\mathrm{Id}_{S^1}(\mathrm{pt}, \mathrm{pt})$ is equivalent to $\mathbb{Z}$, the type of integers.

We can similarly introduce higher spheres (via iterated suspensions, in fact).

Univalence at work: isomorphic structures are equal

Pick your favourite mathematical structure, e.g. boolean algebras, topological spaces, groups, etc. For concreteness, I will pick groups here.

- We can define a type Group of groups.
- For groups, i.e. elements of this type $G, H : \mathrm{Group}$, we can consider $\mathrm{Id}_{\mathrm{Group}}(G, H)$.

Univalence at work: isomorphic structures are equal

Pick your favourite mathematical structure, e.g. boolean algebras, topological spaces, groups, etc. For concreteness, I will pick groups here.

- We can define a type Group of groups.
- For groups, i.e. elements of this type $G, H : \mathrm{Group}$, we can consider $\mathrm{Id}_{\mathrm{Group}}(G, H)$.
- It is a consequence of the univalence axiom that two groups are equal* if and only if the groups G and H are isomorphic.

  [*] i.e. we have an element of the type $\mathrm{Id}_{\mathrm{Group}}(G, H)$

Univalence at work: isomorphic structures are equal

Pick your favourite mathematical structure, e.g. boolean algebras, topological spaces, groups, etc. For concreteness, I will pick groups here.

- We can define a type Group of groups.
- For groups, i.e. elements of this type $G, H : \mathrm{Group}$, we can consider $\mathrm{Id}_{\mathrm{Group}}(G, H)$.
- It is a consequence of the univalence axiom that two groups are equal* if and only if the groups G and H are isomorphic.

  [*] i.e. we have an element of the type $\mathrm{Id}_{\mathrm{Group}}(G, H)$

- If we have an element p of $\mathrm{Id}_{\mathrm{Group}}(G, H)$, then *any* true statement about G must also be true of H, since we can transport the proof of the statement for G along p.

So with univalence, isomorphic groups **really** do behave the same.

The syntax of MLTT

Quick overview

Type (former)	Elements/terms	Set theoretic anal. of the type
Empty type $\mathbb{0}$	none	$\varnothing$
Unit type $\mathbb{1}$	$*$	singleton set
Product type $A \times B$	(a, b) with $a : A$ and $b : B$	cartesian product
Coproduct type $A + B$	$\mathrm{inl}(a)$ with $a : A$ and $\mathrm{inr}(b)$ with $b : B$	disjoint union
Dependent pair type $\sum(x : A), B(x)$	(a, b) with $a : A$ and $b : B(a)$	$\coprod_{x \in A} B(x)$
Function type $A \to B$	f such that for $a : A$, we get $f(a) : B$	total, single-valued relations on A and B
Dependent function type $\prod(x : A), B(x)$	f such that for $a : A$, we get $f(a) : B(a)$	set of sections of the projection $\left(\coprod_{x \in A} B(x)\right) \to A$

Type (former)	Elements/terms	Set theoretic anal. of the type
Identity type $\mathrm{Id}_A(a,b)$ for a 0-type A	"proof that a is equal to b"	equality of FOL
Identity type $\mathrm{Id}_A(a,b)$ for a general type A	"path from a to b in A"	none (not directly anyway)

From now on, instead of $\mathrm{Id}_A(a,b)$, I write $a =_A b$ or even just $a = b$ when A can be inferred.

Using type theory to encode mathematics

Recall that a **monoid** is a set M with an associative binary operation $\cdot$ such that we have a *neutral* element $e \in M$, i.e. $m \cdot e = e \cdot m = m$ for all $m \in M$.

We thus (naively) define **the type of monoids** as

$$\begin{aligned}\sum(M : \mathcal{U}), \sum(- \cdot - : M \times M \to M), \sum(e : M),& \\ \left(\prod(m : M), (m \cdot e = m) \times (e \cdot m = m)\right)& \\ \times \left(\prod(x, y, z : M), (x \cdot y) \cdot z = x \cdot (y \cdot z)\right).&\end{aligned}$$

An element of this type is a tuple $(M, \cdot, e, \nu, \alpha)$ where M is the carrier, $\cdot$ is the monoid operation, e is the neutral element, and ν and α resp. witness neutrality and associativity.

Notice the use of dependent types!

A refinement of the monoid example

- The theory of monoids is *not* about higher dimensional structures.

  Therefore, in HoTT, we futher add the condition

  $$\text{is-0-type}(M)$$

  to the earlier (preliminary) type of monoids.

- This addition also ensures that ν and α are elements of propositions (-1-types) and can be thought of as **logical proofs**, rather than *additional pieces of data*.

- We will return to the distinction between data and property in tomorrow's lecture.