# Introduction to homotopy theory type / univalent foundations

**Tom de Jong**

University of Nottingham, UK

**10th autumn school *Proof and Computation***

16 September 2026, Fischbachau, Germany — Lecture 3/3

# Overview

1. Logical statements as propositions
2. Propositional truncation
3. Brief comparison with `Prop` in Calculus of (Inductive) Constructions
4. Truncations and higher types

# Logical statements as propositions

# Logical statements as propositions

- Logical statements are encoded as *types*.
- Proofs of such a statement are *elements* of such types.
- But which types should count as interpreting logical statements?
  - It seems reasonable that $\mathbb{1}$ and $\mathbb{0}$ encode $\top$ and $\bot$.
  - It seems unreasonable for $\mathbb{N}$ to encode a logical statement.

# Logical statements as propositions

- Logical statements are encoded as *types*.
- Proofs of such a statement are *elements* of such types.
- But which types should count as interpreting logical statements?
  - It seems reasonable that $\mathbb{1}$ and $\mathbb{0}$ encode $\top$ and $\bot$.
  - It seems unreasonable for $\mathbb{N}$ to encode a logical statement.
- What distinguishes these types is that $\mathbb{1}$ and $\mathbb{0}$ both have **at most one element**. We call such types **propositions** (or **subsingletons** or $-1$-types) and define for a type $A$, a new type

$$\text{is-prop}(A) :\equiv \prod(a, b : A), a = b$$

  expressing that any two elements of $A$ are identified, i.e. $A$ has at most one element if we have an element of $\text{is-prop}(A)$.
- We chose to interpret logical statements as *propostions*, i.e. types $P$ with $\text{is-prop}(P)$.

# Logical connectives: conjunction

It makes sense to encode logical conjunction using the product type, but does this restrict to propositions? That is, given two propositions $P$ and $Q$, is $P \times Q$ again a proposition?

**Yes**, because we can prove, for general types $P$ and $Q$ that

$$(p, q) =_{P \times Q} (p', q') \quad \simeq \quad (p =_P p') \times (q =_Q q').$$

Thus, if $P$ and $Q$ are propositions, then we always have $(p, q) = (p', q')$ because we always have both $p = p'$ and $q = q'$.

# Logical connectives: implication

For implication, we would like to use function types. Suppose $P$ and $Q$ are propositions. Is $P \to Q$ a proposition?

**Yes**, at least in the presence of function extensionality.

The **function extensionality axiom** asserts that for all functions $f, g : A \to B$, the canonical map $f = g \longrightarrow (\prod(a : A), f(a) = g(a))$ is an equivalence.

# Logical connectives: implication

For implication, we would like to use function types. Suppose $P$ and $Q$ are propositions. Is $P \to Q$ a proposition?

**Yes**, at least in the presence of function extensionality.

> The **function extensionality axiom** asserts that for all functions $f, g : A \to B$, the canonical map $f = g \longrightarrow (\prod(a : A), f(a) = g(a))$ is an equivalence.

Now, if $P$ and $Q$ are propositions, then so is $P \to Q$, for if we have $f, g : P \to Q$, then by function extensionality, we only need to show $f(p) = g(p)$ for all $p : P$, but $Q$ is a proposition, so we are done.

*Notice that we only needed the **codomain** in the function type to be a proposition.*
This observations helps when we consider universal quantification.

# Logical connectives: universal quantification

With function extensionality, the type $\prod(x : A), P(x)$ is a proposition whenever $P(x)$ is a proposition for every $x : A$, even when $A$ is not a proposition.

We then encode $\forall$ as $\prod$.

E.g. the type $\prod(n, m : \mathbb{N}), n + m = m + n$ expressing the commutativity of addition on natural numbers is a proposition.
This needs that $x =_{\mathbb{N}} y$ is a proposition, i.e. that $\mathbb{N}$ is a *set* or $0$-type.

# Logical connectives: negation

As usual in intuitionistic/constructive logic, we *define* negation as implying false, so that we put

$$\neg A :\equiv A \to \mathbb{0}.$$

Since $\mathbb{0}$ is a proposition, this type is a proposition (under function extensionality).

# Disjunction?

One may expect to encode disjunction $\vee$ as a coproduct, but $P + Q$ need not be a proposition. The simplest example is $\mathbb{1} + \mathbb{1}$ which is the definition of $\mathbb{2}$ with distinct elements $0$ and $1$.

# Disjunction?

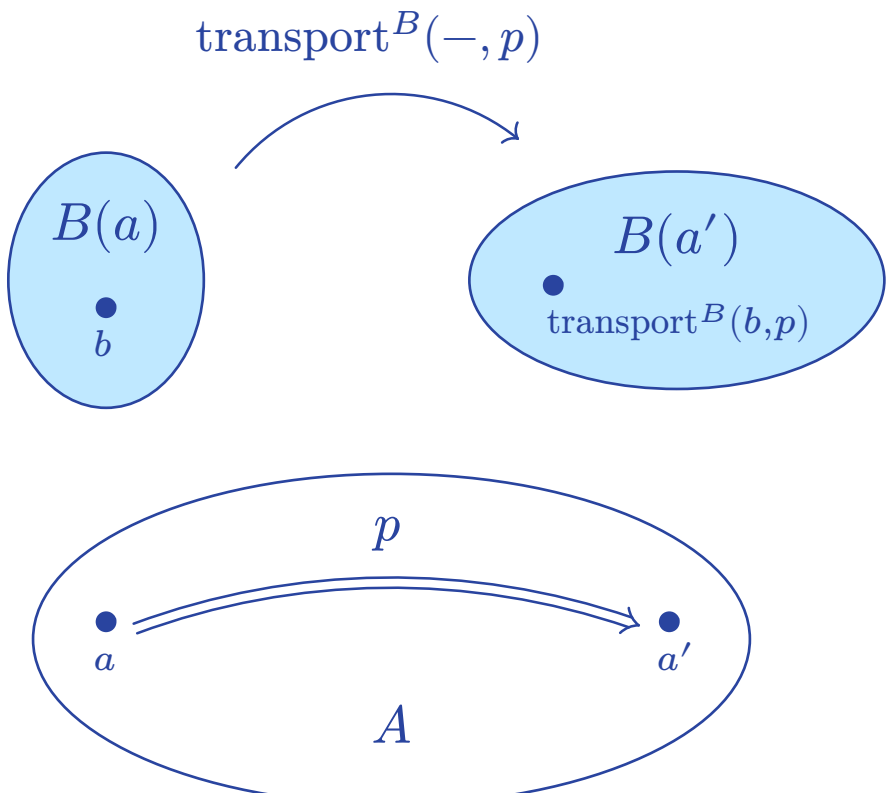
One may expect to encode disjunction $\vee$ as a coproduct, but $P + Q$ need not be a proposition. The simplest example is $\mathbb{1} + \mathbb{1}$ which is the definition of $\mathbb{2}$ with distinct elements $0$ and $1$.

This is a nice opportunity to show that $0 \neq 1$, i.e. that $0 =_{\mathbb{2}} 1 \longrightarrow \mathbb{0}$.
Suppose we have $p : 0 = 1$. We need to construct an element of $\mathbb{0}$.
Consider the function $C : \mathbb{2} \to \mathcal{U}_0$ given by $C(0) := \mathbb{0}$ and $C(1) := \mathbb{1}$.
Then we have $\star : C(1)$, so that $\mathrm{transport}^C(\star, p^{-1}) : C(0) \equiv \mathbb{0}$.

# Disjunction?

One may expect to encode disjunction $\vee$ as a coproduct, but $P + Q$ need not be a proposition. The simplest example is $\mathbb{1} + \mathbb{1}$ which is the definition of $\mathbb{2}$ with distinct elements $0$ and $1$.

This is a nice opportunity to show that $0 \neq 1$, i.e. that $0 =_{\mathbb{2}} 1 \longrightarrow \mathbb{0}$.
Suppose we have $p : 0 = 1$. We need to construct an element of $\mathbb{0}$.
Consider the function $C : \mathbb{2} \to \mathcal{U}_0$ given by $C(0) :\equiv \mathbb{0}$ and $C(1) :\equiv \mathbb{1}$.
Then we have $\star : C(1)$, so that $\mathrm{transport}^C(\star, p^{-1}) : C(0) \equiv \mathbb{0}$.



# Existential quantication?

We can show that if $P$ and $Q(p)$ are propositions, then so is $\sum(p : P), Q(p)$.

But this does not account for encoding statements of the form $\exists(n \in \mathbb{N}), Q(n)$. For example, $\sum(n : \mathbb{N}), n + 0 = n$ has $\mathbb{N}$-many elements.

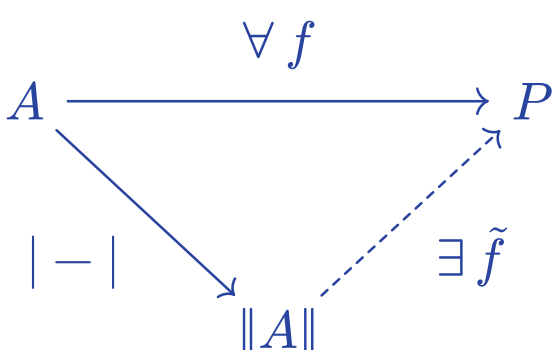
# Propositional truncation

# Turning types into propositions

Our solution will be to postulate a new operation called the **propositional truncation**.

It can be given as a higher inductive type, but we can just assume to be given

1. an operation $\| - \|$ on types such that $\|A\|$ is a proposition for all types $A$,
2. a function $| - | : A \to \|A\|$ for every type $A$ satisfying the **universal property**:
   for every *proposition* $P$ and every function $f : A \to P$,
   we have $\tilde{f} : \|A\| \to P$ such that $\tilde{f}(|a|) = f(a)$ for all $a : A$.



By function extensionality and the assumption that $P$ is a proposition, the map $\tilde{f} : \|A\| \to P$ is **necessarily unique**: any two maps with codomain $P$ must be equal.

The idea is that $\|A\|$ is like $A$ but we've identified all of its elements to make it a proposition. The universal property says that *to propositions*, the types $\|A\|$ and $A$ behave the same.

By function extensionality and the assumption that $P$ is a proposition, the map $\tilde{f} : \|A\| \to P$ is **necessarily unique**: any two maps with codomain $P$ must be equal.

> The idea is that $\|A\|$ is like $A$ but we've identified all of its elements to make it a proposition. The universal property says that *to propositions*, the types $\|A\|$ and $A$ behave the same.

We now define

$$P \vee Q := \|P + Q\| \quad \text{and} \quad \exists(x : A), P(x) := \left\|\sum(x : A), P(x)\right\|.$$

*Example*: Suppose we wanted to show that $(\exists(x : A), P(x)) \to Q$ where $Q$ is a proposition. By the universal property, it suffices to construct a function $(\sum(x : A), P(x)) \to Q$, so that we may assume to have an element $(a, p)$ with $a : A$ and $p : P(a)$, as familiar from logic.

# The truncation does *not* erase

Two examples show that the intuition that the propositional truncation erases information is not accurate.

1. Let $P$ be a proposition. One can show (using a technique similar to the proof that $0 \neq 1$) that the type $P + \neg P$ is a proposition. This immediately gives a function

$$(P \vee \neg P) \longrightarrow (P + \neg P).$$

   Thus, given $P \vee \neg P$, we can really tell whether $P$ or $\neg P$ holds by inspecting whether the element of the type $P + \neg P$ starts with $\mathrm{inl}$ or $\mathrm{inr}$.

   In particular, we can formulate **excluded middle** as either

$$\prod(P : \mathcal{U}), \text{is-prop}(P) \to P + \neg P \quad \text{or} \quad \prod(P : \mathcal{U}), \text{is-prop}(P) \to P \vee \neg P.$$

2. (Escardó) Let $P$ be a dependent type on $\mathbb{N}$ such that
   - every $P(n)$ is a proposition,
   - and **decidable**, i.e. we have an element of $P(n) + \neg P(n)$.

   The type $\sum(n : \mathbb{N}), P(n)$ is not a proposition since there may be many $n$'s that satisfy $P$.

   But we can still define a function

$$(\exists(n : \mathbb{N}), P(n)) \longrightarrow \left(\sum(n : \mathbb{N}), P(n)\right)$$

   out of the truncation as follows.

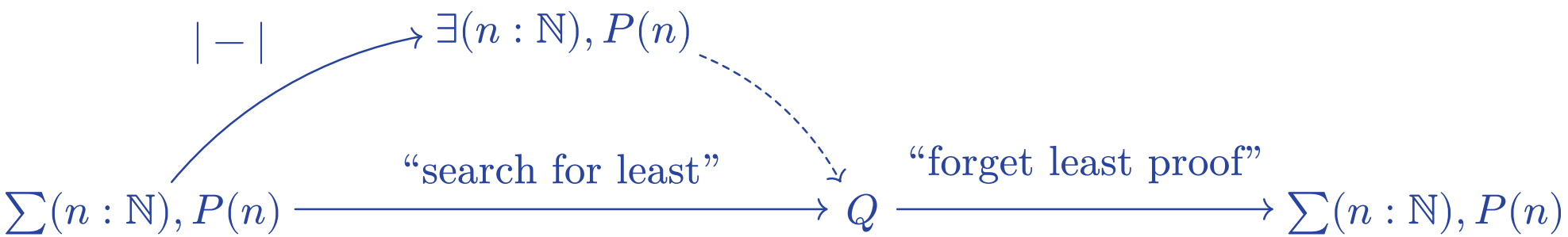
   We consider the type of *least* witnesses of $P$, i.e.

$$Q :\equiv \sum(n : \mathbb{N}), P(n) \times \left(\prod(k : \mathbb{N}), P(k) \to n \leq k\right).$$

   This type $Q$ of least witnesses can be shown to be a proposition.

Since $P$ is a decidable predicate, we can construct a "search" function $\sum(n : \mathbb{N}), P(n) \to Q$ that yields the least $n$ with $P(n)$.

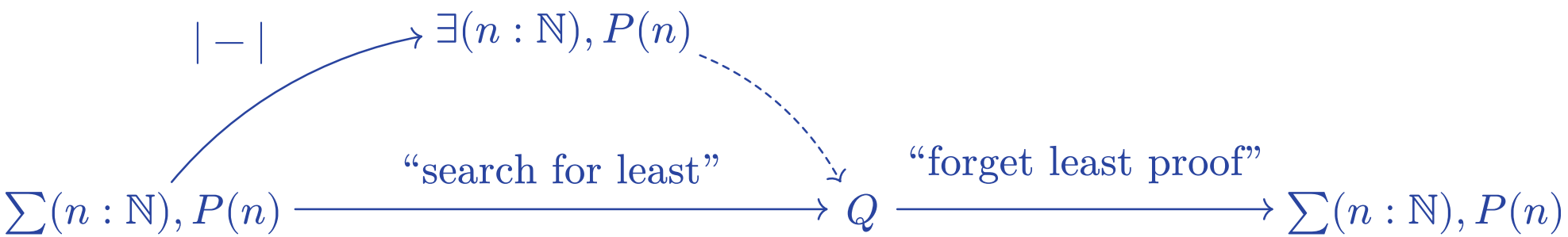
Hence, we get a commutative diagram




and our desired map $(\exists(n : \mathbb{N}), P(n)) \to (\sum(n : \mathbb{N}), P(n))$ by composing.

Since $P$ is a decidable predicate, we can construct a "search" function $\sum(n : \mathbb{N}), P(n) \to Q$ that yields the least $n$ with $P(n)$.

Hence, we get a commutative diagram




and our desired map $(\exists(n : \mathbb{N}), P(n)) \to (\sum(n : \mathbb{N}), P(n))$ by composing.

We see that making a type *more informative* can make it a proposition.

Propositions in HoTT are **very different** from the **syntactic** propositions in the Calculus of (Inductive) Constructions that you may know from Rocq or Lean.

## On $\Sigma$ vs $\exists$

The **image of a function** $f : A \to B$ is defined as $\{b \in B \mid \exists a \in A, f(a) = b\}$ in set theory.

In HoTT, we should similarly define it as $\sum(b : B), \exists(a : A), f(a) = b$ and **not** as $\sum(b : B), \sum(a : A), f(a) = b$.

This is not surprising: the latter type is akin to $\{(b, a) \in B \times A \mid f(a) = b\}$ which is in bijection to $A$, and so clearly not the image.

Similarly, $\sum(b : B), \sum(a : A), f(a) = b$ is equivalent to $A$, so clearly not the image.

## On $\Sigma$ vs $\exists$

The **image of a function** $f : A \to B$ is defined as $\{b \in B \mid \exists a \in A, f(a) = b\}$ in set theory.

In HoTT, we should similarly define it as $\sum(b : B), \exists(a : A), f(a) = b$ and **not** as $\sum(b : B), \sum(a : A), f(a) = b$.

This is not surprising: the latter type is akin to $\{(b, a) \in B \times A \mid f(a) = b\}$ which is in bijection to $A$, and so clearly not the image.

Similarly, $\sum(b : B), \sum(a : A), f(a) = b$ is equivalent to $A$, so clearly not the image.

---

Other times, the difference between $\Sigma$ and $\exists$ is immaterial.

For instance, in phrasing that a monoid has a **neutral element**, we can use either $\Sigma$ or $\exists$ since neutral elements happen to be unique.

# Truncations and higher types

# Truncations are tricky

Some examples illustrating the subtlety of the truncation in the presence of higher types:

1. The type $\prod(x, y : S^1), x = y$ has **no** elements.

   This type expresses that $S^1$ has at most one element, i.e. that $S^1$ is a proposition, which it isn't, because propositions have trivial identity types.

# Truncations are tricky

Some examples illustrating the subtlety of the truncation in the presence of higher types:

1. The type $\prod(x, y : S^1), x = y$ has **no** elements.

   This type expresses that $S^1$ has at most one element, i.e. that $S^1$ is a proposition, which it isn't, because propositions have trivial identity types.

   *However*, the type $\prod(x, y : S^1), \|x = y\|$ **does** have an element and it witnesses that $S^1$ is **(path) connected**.

# Truncations are tricky

Some examples illustrating the subtlety of the truncation in the presence of higher types:

1. The type $\prod(x, y : S^1), x = y$ has **no** elements.

   This type expresses that $S^1$ has at most one element, i.e. that $S^1$ is a proposition, which it isn't, because propositions have trivial identity types.

   *However*, the type $\prod(x, y : S^1), \|x = y\|$ **does** have an element and it witnesses that $S^1$ is **(path) connected**.

2. **Global choice** asserts that we can equip every inhabited type with a point. Formally,

$$\prod(X : \mathcal{U}), \|X\| \to X.$$

   This is **inconsistent** with univalence, but the variation $\prod(X : \mathcal{U}), \|\|X\| \to X\|$ is a **consistent** weak choice principle that follows from excluded middle.

3. The type

$$\sum(X : \mathcal{U}), \sum(f : X \to X), (X, f) = (\mathbb{Z}, \mathrm{succ})$$

is **contractible**, i.e. up to the identity type it has a unique element, namely the triple $(\mathbb{Z}, \mathrm{succ}, \mathrm{refl})$.

On the other hand, $\sum(X : \mathcal{U}), \sum(f : X \to X), \|(X, f) = (\mathbb{Z}, \mathrm{succ})\|$ is ???

3. The type

$$\sum(X : \mathcal{U}), \sum(f : X \to X), (X, f) = (\mathbb{Z}, \mathrm{succ})$$

is **contractible**, i.e. up to the identity type it has a unique element, namely the triple $(\mathbb{Z}, \mathrm{succ}, \mathrm{refl})$.

On the other hand, $\sum(X : \mathcal{U}), \sum(f : X \to X), \|(X, f) = (\mathbb{Z}, \mathrm{succ})\|$ is the type of $\mathbb{Z}$-*torsors* and **an alternative construction of the circle** $S^1$ (in the presence of univalence).

↳ Marc Bezem, Ulrik Buchholtz, Daniel R. Grayson and Michael Shulman. 'Construction of the circle in *UniMath*'. In: *Journal of Pure and Applied Algebra* (225.10), 106687 (2021). DOI: 10.1016/j.jpaa.2021.106687

# Bonus: proofs related to the second example

# Proof: $\prod(X : \mathcal{U}), \|\|X\| \to X\|$ follows from excluded middle

Let $X : \mathcal{U}$. We want an element of $T :\equiv \|\|X\| \to X\|$.

Since $\|X\|$ is a proposition, excluded middle gives us an element of

$$\|X\| + \neg \|X\|.$$

So there are two cases:

- if we have an element of $\|X\|$, then since $T$ is a proposition, we may assume to have $x : X$, and $|\_ \mapsto x|$ is an element of $T$;
- if we have $\nu : \neg\|X\| \equiv \|X\| \to \mathbb{0}$, then we can give an element of $T$ via the unique map $\mathbb{0} \to X$ ("ex-falso"). ■

# Proof: global choice $\prod(X : \mathcal{U}), \|X\| \to X$ is inconsistent with univalence

Any $p : A = B$, induces a function $\mathrm{idtofun}(p) : A \to B$ such that for any dependent function $f : \prod(X : \mathcal{U}), \|X\| \to X$, the following square commutes.

$$\begin{array}{ccc} \|A\| & \xrightarrow{f_A} & A \\ {\scriptstyle \|\mathrm{idtofun}(p)\|}\downarrow & & \downarrow{\scriptstyle \mathrm{idtofun}(p)} \\ \|B\| & \xrightarrow{f_B} & B \end{array}$$

**Proof.** We only need to consider the case where $p$ is refl in which case $\mathrm{idtofun}(p)$ is just the identity.

# Proof: global choice $\prod(X : \mathcal{U}), \|X\| \to X$ is inconsistent with univalence

Any $p : A = B$, induces a function $\mathrm{idtofun}(p) : A \to B$ such that for any dependent function $f : \prod(X : \mathcal{U}), \|X\| \to X$, the following square commutes.

$$\begin{array}{ccc} \|A\| & \xrightarrow{f_A} & A \\ {\scriptstyle \|\mathrm{idtofun}(p)\|}\downarrow & & \downarrow{\scriptstyle \mathrm{idtofun}(p)} \\ \|B\| & \xrightarrow{f_B} & B \end{array}$$

**Proof.** We only need to consider the case where $p$ is refl in which case $\mathrm{idtofun}(p)$ is just the identity.

We now specialize to the case where

- $A$ and $B$ are both $2$, and
- $p$ is the identification induced by $\mathrm{swap} : 2 \simeq 2$ — this is where we use univalence — so that $\mathrm{idtofun}(p)$ is the swap map.

Since $\|2\|$ is a proposition, we have $|0| = |1|$ and hence $f_2(|0|) = f_2(|1|)$.

But now the commutativity of the square

$$\begin{array}{ccc} \|2\| & \xrightarrow{f_2} & 2 \\ {\scriptstyle \|\mathrm{swap}\|}\downarrow & & \downarrow{\scriptstyle \mathrm{swap}} \\ \|2\| & \xrightarrow{f_2} & 2 \end{array}$$

yields

$$\mathrm{swap}(f_2(|0|)) = f_2(\|\mathrm{swap}\|(|0|)) = f_2(|\mathrm{swap}(0)|) = f_2(|1|).$$

Combining the equations gives $\mathrm{swap}(f_2(|0|)) = f_2(|0|)$, yielding a fixed point of $\mathrm{swap}$ which is impossible. ∎