



Emily Riehl

Johns Hopkins University

## How I became seduced by univalent foundations

2022 Fields Medal Symposium: Akshay Venkatesh

# Plan



1. A reintroduction to proofs
2. On the art of giving the same name to different things
3. Contractibility as uniqueness
4. Computer proof assistants that compute



1

# A reintroduction to proofs

# Conjunction and disjunction



Compare a traditional introduction to the logical operators “and”  $\wedge$  and “or”  $\vee$  using truth tables with the following:

**Conjunction**  $\wedge$  is the logical operator defined by the rules:

- $\wedge$ intro: If  $p$  is true and  $q$  is true, then  $p \wedge q$  is true.
- $\wedge$ elim<sub>1</sub>: If  $p \wedge q$  is true, then  $p$  is true.
- $\wedge$ elim<sub>2</sub>: If  $p \wedge q$  is true, then  $q$  is true.

**Disjunction**  $\vee$  is the logical operator defined by the rules:

- $\vee$ intro<sub>1</sub>: If  $p$  is true, then  $p \vee q$  is true.
- $\vee$ intro<sub>2</sub>: If  $q$  is true, then  $p \vee q$  is true.
- $\vee$ elim: If  $p \vee q$  is true, and if  $r$  can be derived from  $p$  and from  $q$ , then  $r$  is true.

— from Clive Newstead’s *An Infinite Descent into Pure Mathematics*.

# Implication



The **introduction rules** explain how to prove a proposition involving a particular connective, while the **elimination rules** explain how to use a hypothesis involving a particular connective.

**Implication**  $\Rightarrow$  is the logical operator defined by the rules:

- $\Rightarrow$ **intro**: If  $q$  can be derived from the assumption that  $p$  is true, then  $p \Rightarrow q$  is true.
- $\Rightarrow$ **elim**: If  $p \Rightarrow q$  is true and  $p$  is true, then  $q$  is true.

# Implication



The **introduction rules** explain how to prove a proposition involving a particular connective, while the **elimination rules** explain how to use a hypothesis involving a particular connective.

**Implication**  $\Rightarrow$  is the logical operator defined by the rules:

- $\Rightarrow$ **intro**: If  $q$  can be derived from the assumption that  $p$  is true, then  $p \Rightarrow q$  is true.
- $\Rightarrow$ **elim**: If  $p \Rightarrow q$  is true and  $p$  is true, then  $q$  is true.

**Theorem.** For any propositions  $p$  and  $q$ ,  $(p \wedge (p \Rightarrow q)) \Rightarrow q$ .

# Implication



The **introduction rules** explain how to prove a proposition involving a particular connective, while the **elimination rules** explain how to use a hypothesis involving a particular connective.

**Implication**  $\Rightarrow$  is the logical operator defined by the rules:

- $\Rightarrow$ **intro**: If  $q$  can be derived from the assumption that  $p$  is true, then  $p \Rightarrow q$  is true.
- $\Rightarrow$ **elim**: If  $p \Rightarrow q$  is true and  $p$  is true, then  $q$  is true.

**Theorem.** For any propositions  $p$  and  $q$ ,  $(p \wedge (p \Rightarrow q)) \Rightarrow q$ .

**Proof:** By  $\Rightarrow$ **intro** we may assume that  $p \wedge (p \Rightarrow q)$  is true; our goal is to prove  $q$ .

# Implication



The **introduction rules** explain how to prove a proposition involving a particular connective, while the **elimination rules** explain how to use a hypothesis involving a particular connective.

**Implication**  $\Rightarrow$  is the logical operator defined by the rules:

- $\Rightarrow$ **intro**: If  $q$  can be derived from the assumption that  $p$  is true, then  $p \Rightarrow q$  is true.
- $\Rightarrow$ **elim**: If  $p \Rightarrow q$  is true and  $p$  is true, then  $q$  is true.

**Theorem.** For any propositions  $p$  and  $q$ ,  $(p \wedge (p \Rightarrow q)) \Rightarrow q$ .

**Proof:** By  $\Rightarrow$ **intro** we may assume that  $p \wedge (p \Rightarrow q)$  is true; our goal is to prove  $q$ . By  $\wedge$ **elim**<sub>1</sub> and  $\wedge$ **elim**<sub>2</sub> it follows that  $p$  and  $p \Rightarrow q$  are true.



# Implication



The **introduction rules** explain how to prove a proposition involving a particular connective, while the **elimination rules** explain how to use a hypothesis involving a particular connective.

**Implication**  $\Rightarrow$  is the logical operator defined by the rules:

- $\Rightarrow$ **intro**: If  $q$  can be derived from the assumption that  $p$  is true, then  $p \Rightarrow q$  is true.
- $\Rightarrow$ **elim**: If  $p \Rightarrow q$  is true and  $p$  is true, then  $q$  is true.

**Theorem.** For any propositions  $p$  and  $q$ ,  $(p \wedge (p \Rightarrow q)) \Rightarrow q$ .

**Proof:** By  $\Rightarrow$ **intro** we may assume that  $p \wedge (p \Rightarrow q)$  is true; our goal is to prove  $q$ . By  $\wedge$ **elim**<sub>1</sub> and  $\wedge$ **elim**<sub>2</sub> it follows that  $p$  and  $p \Rightarrow q$  are true. By  $\Rightarrow$ **elim** then we may conclude that  $q$  is true, as desired. □

# Universal and existential quantification



Let  $p : X \rightarrow \{\perp, \top\}$  be an  $X$ -indexed family of propositions, a **predicate** on  $X$ .

**Universal quantification**  $\forall x \in X, p(x)$  is the logical formula defined by the rules:

- **$\forall$ intro**: If  $p(x)$  can be derived from the assumption that  $x$  is an arbitrary element of  $X$ , then  $\forall x \in X, p(x)$  is true.
- **$\forall$ elim**: If  $\forall x \in X, p(x)$  is true and  $a \in X$ , then  $p(a)$  is true.

**Existential quantification**  $\exists x \in X, p(x)$  is the logical formula defined by the rules:

- **$\exists$ intro**: If  $a \in X$  and  $p(a)$  is true, then  $\exists x \in X, p(x)$  is true.
- **$\exists$ elim**: If  $\exists x \in X, p(x)$  is true and  $q$  can be derived from the assumption that  $p(a)$  is true for some  $a \in X$ , then  $q$  is true.

# Dependent type theory



Dependent type theory is a formal system for mathematical statements and proofs that has the following primitive notions:

- types, e.g.,  $\mathbb{N}$  ,  $\mathbb{Q}$  , Group

# Dependent type theory



Dependent type theory is a formal system for mathematical statements and proofs that has the following primitive notions:

- types, e.g.,  $\mathbb{N}$  ,  $\mathbb{Q}$  ,  $\text{Group}$
- terms, e.g.,  $17 : \mathbb{N}$  ,  $\sqrt{2} : \mathbb{R}$

# Dependent type theory



Dependent type theory is a formal system for mathematical statements and proofs that has the following primitive notions:

- types, e.g.,  $\mathbb{N}$  ,  $\mathbb{Q}$  ,  $\text{Group}$
- terms, e.g.,  $17 : \mathbb{N}$  ,  $\sqrt{2} : \mathbb{R}$
- dependent types, e.g.,  $\text{is-prime} : \mathbb{N} \rightarrow \text{Type}$ ,  $\mathbb{R}^\bullet : \mathbb{N} \rightarrow \text{Type}$ ,  $\text{Mat}_{\bullet \times \bullet} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Type}$

# Dependent type theory



Dependent type theory is a formal system for mathematical statements and proofs that has the following primitive notions:

- types, e.g.,  $\mathbb{N}$  ,  $\mathbb{Q}$  ,  $\text{Group}$
- terms, e.g.,  $17 : \mathbb{N}$  ,  $\sqrt{2} : \mathbb{R}$
- dependent types, e.g.,  $\text{is-prime} : \mathbb{N} \rightarrow \text{Type}$ ,  $\mathbb{R}^\bullet : \mathbb{N} \rightarrow \text{Type}$ ,  $\text{Mat}_{\bullet \times \bullet} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Type}$
- dependent terms, e.g.,  $\vec{0}^\bullet : \prod_{n:\mathbb{N}} \mathbb{R}^n$  ,  $I_\bullet : \prod_{n:\mathbb{N}} \text{Mat}_{n,n}$  ,  $S_\bullet : \prod_{n:\mathbb{N}} \text{Group}$

# Dependent type theory



Dependent type theory is a formal system for mathematical statements and proofs that has the following primitive notions:

- types, e.g.,  $\mathbb{N}$  ,  $\mathbb{Q}$  ,  $\text{Group}$
- terms, e.g.,  $17 : \mathbb{N}$  ,  $\sqrt{2} : \mathbb{R}$
- dependent types, e.g.,  $\text{is-prime} : \mathbb{N} \rightarrow \text{Type}$ ,  $\mathbb{R}^\bullet : \mathbb{N} \rightarrow \text{Type}$ ,  $\text{Mat}_{\bullet \times \bullet} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \text{Type}$
- dependent terms, e.g.,  $\vec{0}^\bullet : \prod_{n:\mathbb{N}} \mathbb{R}^n$  ,  $I_\bullet : \prod_{n:\mathbb{N}} \text{Mat}_{n,n}$  ,  $S_\bullet : \prod_{n:\mathbb{N}} \text{Group}$

all of which can occur in an arbitrary context of variables from previously-defined types.

In a mathematical statement of the form “Let ...be ...then ...” The stuff following the “let” likely declares the names of the variables in the context described after the “be”, while the stuff after the “then” most likely describes a type or term in that context.

# Products and coproducts



Given types  $A$  and  $B$ , the **product type**  $A \times B$  is governed by the rules:

- $\times$ **intro**: given terms  $a : A$  and  $b : B$  there is a term  $(a, b) : A \times B$
- $\times$ **elim**: given a term  $p : A \times B$  there are terms  $\text{pr}_1 p : A$  and  $\text{pr}_2 p : B$

plus computation rules that relate pairings and projections.

Given types  $A$  and  $B$ , the **coproduct type**  $A + B$  is governed by the rules:

- $+$ **intro**: given a term  $a : A$ , there is a term  $\text{inl}a : A + B$ , and  
given a term  $b : B$ , there is a term  $\text{inr}b : A + B$
- $+$ **elim**: given a family of types  $C : (A + B) \rightarrow \text{Type}$ , if there are dependent terms  
 $c : \prod_{a:A} C(\text{inl}a)$  and  $d : \prod_{b:B} C(\text{inr}b)$ , then there is a term  $e : \prod_{x:A+B} C(x)$

plus computation rules that relate the inclusions and the elimination.



## Functions in set theory vs functions in type theory



In set theory, a function  $f: X \rightarrow Y$  is a subset  $\Gamma_f \subset X \times Y$  with the property that

$$\forall x \in X, \exists! y \in Y, (x, y) \in \Gamma_f.$$

# Functions in set theory vs functions in type theory



In set theory, a function  $f: X \rightarrow Y$  is a subset  $\Gamma_f \subset X \times Y$  with the property that

$$\forall x \in X, \exists! y \in Y, (x, y) \in \Gamma_f.$$

Given types  $A$  and  $B$ , the function type  $A \rightarrow B$  is governed by the rules:

- $\rightarrow$ intro: if in the context of a variable  $x: A$  there is a term  $b_x: B$ ,  
there is a term  $\lambda x. b_x: A \rightarrow B$
- $\rightarrow$ elim: given terms  $f: A \rightarrow B$  and  $a: A$ , there is a term  $f(a): B$

plus computation rules that relate  $\lambda$ -abstractions and evaluations.

# Dependent functions and dependent sums



Let  $B : A \rightarrow \text{Type}$  be a family of types over a type  $A$ .

The dependent function type  $\prod_{x:A} B(x)$  is governed by the rules:

- $\Pi_{\text{intro}}$ : if in the context of a variable  $x : A$  there is a term  $b_x : B$   
there is a term  $\lambda x. b_x : \prod_{x:A} B(x)$
- $\Pi_{\text{elim}}$ : given terms  $f : \prod_{x:A} B(x)$  and  $a : A$  there is a term  $f(a) : B(a)$

plus computation rules that relate  $\lambda$ -abstractions and evaluations.

The dependent sum type  $\sum_{x:A} B(x)$  is governed by the rules:

- $\Sigma_{\text{intro}}$ : if there are terms  $a : A$  and  $b : B(a)$ , there is a term  $(a, b) : \sum_{x:A} B(x)$
- $\Sigma_{\text{elim}}$ : given a type family  $C : \sum_{x:A} B(x) \rightarrow \text{Type}$ , if there is a term  $c : \prod_{a:A} \prod_{b:B(a)} C(a, b)$ , there is a term  $d : \prod_{z:\sum_{x:A} B(x)} C(z)$

plus computation rules that relate pairings and eliminations.

## The natural numbers in set theory



Recall the von Neumann and Zermelo constructions of the natural numbers in set theory:

$$3_{\text{vN}} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \quad 3_{\text{Z}} := \{\{\{\{\}\}\}\}$$

## The natural numbers in set theory



Recall the von Neumann and Zermelo constructions of the natural numbers in set theory:

$$3_{\text{vN}} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \quad 3_{\text{Z}} := \{\{\{\{\}\}\}\}$$

Q: Is 3 an element of 17?

— Paul Benacerraf “What numbers could not be”

# The natural numbers in set theory



Recall the von Neumann and Zermelo constructions of the natural numbers in set theory:

$$3_{\text{vN}} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \quad 3_{\text{Z}} := \{\{\{\{\}\}\}\}$$

Q: Is 3 an element of 17?

— Paul Benacerraf “What numbers could not be”

By **Dedekind's categoricity theorem**, the natural numbers  $\mathbb{N}$  are characterized by Peano's postulates:

- There is a natural number  $0 \in \mathbb{N}$ .
- Every natural number  $n \in \mathbb{N}$  has a successor  $\text{suc } n \in \mathbb{N}$ .
- $0$  is not the successor of any natural number.
- No two natural numbers have the same successor.
- The **principle of mathematical induction**: for all  $P : \mathbb{N} \rightarrow \{\perp, \top\}$

$$P(0) \rightarrow (\forall k \in \mathbb{N}, P(k) \rightarrow P(\text{suc } k)) \rightarrow (\forall n \in \mathbb{N}, P(n))$$

# The natural numbers in dependent type theory



The natural numbers type  $\mathbb{N}$  is governed by the rules:

- $\mathbb{N}_{\text{intro}}$ : there is a term  $0 : \mathbb{N}$  and for any term  $n : \mathbb{N}$  there is a term  $\text{succ } n : \mathbb{N}$

The elimination rule strengthens the principle of mathematical induction by replacing the predicate  $P : \mathbb{N} \rightarrow \{\perp, \top\}$  by an arbitrary family of types  $P : \mathbb{N} \rightarrow \text{Type}$ .

- $\mathbb{N}_{\text{elim}}$ : for any type family  $P : \mathbb{N} \rightarrow \text{Type}$ , to prove  $p : \prod_{n:\mathbb{N}} P(n)$  it suffices to prove  $p_0 : P(0)$  and  $p_s : \prod_{k:\mathbb{N}} P(k) \rightarrow P(\text{succ } k)$ . That is

$$\mathbb{N}_{\text{ind}} : P(0) \rightarrow \left( \prod_{k \in \mathbb{N}} P(k) \rightarrow P(\text{succ } k) \right) \rightarrow \left( \prod_{n \in \mathbb{N}} P(n) \right)$$

Computation rules establish that  $p$  is defined recursively from  $p_0$  and  $p_s$ .

# The natural numbers in dependent type theory



The natural numbers type  $\mathbb{N}$  is governed by the rules:

- $\mathbb{N}_{\text{intro}}$ : there is a term  $0 : \mathbb{N}$  and for any term  $n : \mathbb{N}$  there is a term  $\text{succ } n : \mathbb{N}$

The elimination rule strengthens the principle of mathematical induction by replacing the predicate  $P : \mathbb{N} \rightarrow \{\perp, \top\}$  by an arbitrary family of types  $P : \mathbb{N} \rightarrow \text{Type}$ .

- $\mathbb{N}_{\text{elim}}$ : for any type family  $P : \mathbb{N} \rightarrow \text{Type}$ , to prove  $p : \prod_{n:\mathbb{N}} P(n)$  it suffices to prove  $p_0 : P(0)$  and  $p_s : \prod_{k:\mathbb{N}} P(k) \rightarrow P(\text{succ } k)$ . That is

$$\mathbb{N}_{\text{ind}} : P(0) \rightarrow \left( \prod_{k \in \mathbb{N}} P(k) \rightarrow P(\text{succ } k) \right) \rightarrow \left( \prod_{n \in \mathbb{N}} P(n) \right)$$

Computation rules establish that  $p$  is defined recursively from  $p_0$  and  $p_s$ .

Note the other two Peano postulates are missing because they are provable!





2

On the art of giving the same name to different things

## Identity types



The following rules for **identity types** were developed by Martin-Löf:

Given a type  $A$  and terms  $x, y : A$ , the **identity type**  $x =_A y$  is governed by the rules:

- **=intro**: given a type  $A$  and term  $x : A$  there is a term  $\text{refl}_x : x =_A x$

The elimination rule for the identity type defines an induction principle analogous to recursion over the natural numbers: it provides sufficient conditions for which to define a dependent function out of the identity type family.

- **=elim**: for any type family  $P(x, y, p)$  over  $x, y : A$  and  $p : x =_A y$ , to prove  $P(x, y, p)$  for all  $x, y, p$  it suffices to assume  $y$  is  $x$  and  $p$  is  $\text{refl}_x$ . That is

$$\text{=ind} : \left( \prod_{x:A} P(x, x, \text{refl}_x) \right) \rightarrow \left( \prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p) \right)$$

A computation rule establishes that the proof of  $P(x, x, \text{refl}_x)$  is the given one.

# The homotopical interpretation of dependent type theory



Identity types can be iterated: given  $x, y : A$  and  $p, q : x =_A y$  there is a type  $p =_{x=_A y} q$ .

Does this type always have a term? In other words, are identity proofs unique?

# The homotopical interpretation of dependent type theory



Identity types can be iterated: given  $x, y : A$  and  $p, q : x =_A y$  there is a type  $p =_{x=_A y} q$ .  
Does this type always have a term? In other words, are identity proofs unique? No!

**Theorem** (Lumsdaine, Garner–van den Berg after Hofmann–Streicher). The terms belonging to the iterated identity types of any type  $A$  form an  $\infty$ -groupoid.

# The homotopical interpretation of dependent type theory



Identity types can be iterated: given  $x, y : A$  and  $p, q : x =_A y$  there is a type  $p =_{x=_A y} q$ . Does this type always have a term? In other words, are identity proofs unique? No!

**Theorem** (Lumsdaine, Garner–van den Berg after Hofmann–Streicher). The terms belonging to the iterated identity types of any type  $A$  form an  $\infty$ -groupoid.

The  $\infty$ -groupoid structure of  $A$  has

- terms  $x : A$  as objects,
- identifications  $p : x =_A y$  as 1-morphisms aka **paths**,
- higher identifications  $h : p =_{x=_A y} q$  as 2-morphisms aka **homotopies**, ...

# The homotopical interpretation of dependent type theory



Identity types can be iterated: given  $x, y : A$  and  $p, q : x =_A y$  there is a type  $p =_{x=_A y} q$ . Does this type always have a term? In other words, are identity proofs unique? **No!**

**Theorem** (Lumsdaine, Garner–van den Berg after Hofmann–Streicher). The terms belonging to the iterated identity types of any type  $A$  form an  $\infty$ -groupoid.

The  $\infty$ -groupoid structure of  $A$  has

- terms  $x : A$  as objects,
- identifications  $p : x =_A y$  as 1-morphisms aka **paths**,
- higher identifications  $h : p =_{x=_A y} q$  as 2-morphisms aka **homotopies**, ...

The required structures are proven from  $=$ -elim:

- **constant paths** (reflexivity)  $\text{refl}_x : x = x$
- **reversal** (symmetry)  $p : x = y$  yields  $p^{-1} : y = x$
- **concatenation** (transitivity)  $p : x = y$  and  $q : y = z$  yield  $p * q : x = z$

# The homotopical interpretation of dependent type theory



Identity types can be iterated: given  $x, y : A$  and  $p, q : x =_A y$  there is a type  $p =_{x=_A y} q$ . Does this type always have a term? In other words, are identity proofs unique? No!

**Theorem** (Lumsdaine, Garner–van den Berg after Hofmann–Streicher). The terms belonging to the iterated identity types of any type  $A$  form an  $\infty$ -groupoid.

The  $\infty$ -groupoid structure of  $A$  has

- terms  $x : A$  as objects,
- identifications  $p : x =_A y$  as 1-morphisms aka **paths**,
- higher identifications  $h : p =_{x=_A y} q$  as 2-morphisms aka **homotopies**, ...

The required structures are proven from  $=$ elim:

- **constant paths** (reflexivity)  $\text{refl}_x : x = x$
- **reversal** (symmetry)  $p : x = y$  yields  $p^{-1} : y = x$
- **concatenation** (transitivity)  $p : x = y$  and  $q : y = z$  yield  $p * q : x = z$

and furthermore concatenation is associative and unital, the associators are coherent ...

## Leibniz' indiscernibility of identicals as path lifting



The elimination rule for the identity types is also called **path induction**:

- **=elim**: for any type family  $P(x, y, p)$  over  $x, y : A$  and  $p : x =_A y$ , to prove  $P(x, y, p)$  for all  $x, y, p$  it suffices to assume  $y$  is  $x$  and  $p$  is  $\text{refl}_x$ . That is

$$\text{=ind} : \left( \prod_{x:A} P(x, x, \text{refl}_x) \right) \rightarrow \left( \prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p) \right)$$



## Leibniz' indiscernibility of identicals as path lifting



The elimination rule for the identity types is also called **path induction**:

- **=elim**: for any type family  $P(x, y, p)$  over  $x, y : A$  and  $p : x =_A y$ , to prove  $P(x, y, p)$  for all  $x, y, p$  it suffices to assume  $y$  is  $x$  and  $p$  is  $\text{refl}_x$ . That is

$$\text{=ind} : \left( \prod_{x:A} P(x, x, \text{refl}_x) \right) \rightarrow \left( \prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p) \right)$$

In first-order logic, an axiom for the equality relation is **indiscernibility of identicals**:

$x = y$  implies that for all predicates  $P$ ,  $P(x) \Leftrightarrow P(y)$ .

## Leibniz' indiscernibility of identicals as path lifting



The elimination rule for the identity types is also called **path induction**:

- **=elim**: for any type family  $P(x, y, p)$  over  $x, y : A$  and  $p : x =_A y$ , to prove  $P(x, y, p)$  for all  $x, y, p$  it suffices to assume  $y$  is  $x$  and  $p$  is  $\text{refl}_x$ . That is

$$=_{\text{ind}} : \left( \prod_{x:A} P(x, x, \text{refl}_x) \right) \rightarrow \left( \prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p) \right)$$

In first-order logic, an axiom for the equality relation is **indiscernibility of identicals**:

$x = y$  implies that for all predicates  $P$ ,  $P(x) \Leftrightarrow P(y)$ .

**Proposition.** Let  $P : A \rightarrow \text{Type}$  be any family of types. For any  $x, y : A$  and  $p : x =_A y$ , there is a transport function  $\text{tr}_{P,p} : P(x) \rightarrow P(y)$ .

## Leibniz' indiscernibility of identicals as path lifting



The elimination rule for the identity types is also called **path induction**:

- **=elim**: for any type family  $P(x, y, p)$  over  $x, y : A$  and  $p : x =_A y$ , to prove  $P(x, y, p)$  for all  $x, y, p$  it suffices to assume  $y$  is  $x$  and  $p$  is  $\text{refl}_x$ . That is

$$=_{\text{ind}} : \left( \prod_{x:A} P(x, x, \text{refl}_x) \right) \rightarrow \left( \prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p) \right)$$

In first-order logic, an axiom for the equality relation is **indiscernibility of identicals**:

$x = y$  implies that for all predicates  $P$ ,  $P(x) \Leftrightarrow P(y)$ .

**Proposition.** Let  $P : A \rightarrow \text{Type}$  be any family of types. For any  $x, y : A$  and  $p : x =_A y$ , there is a transport function  $\text{tr}_{P,p} : P(x) \rightarrow P(y)$ .

**Proof:** By **=elim**, to define  $\text{tr}_P : \prod_{x,y:A} \prod_{p:x=_A y} P(x) \rightarrow P(y)$  it suffices to define a term of type  $\prod_{x:A} P(x) \rightarrow P(x)$ , for which we take the identity function  $\lambda x. \lambda q. q$ .  $\square$

## Leibniz' indiscernibility of identicals as path lifting



The elimination rule for the identity types is also called **path induction**:

- **=elim**: for any type family  $P(x, y, p)$  over  $x, y : A$  and  $p : x =_A y$ , to prove  $P(x, y, p)$  for all  $x, y, p$  it suffices to assume  $y$  is  $x$  and  $p$  is  $\text{refl}_x$ . That is

$$\text{=ind} : \left( \prod_{x:A} P(x, x, \text{refl}_x) \right) \rightarrow \left( \prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p) \right)$$

In first-order logic, an axiom for the equality relation is **indiscernibility of identicals**:

$x = y$  implies that for all predicates  $P$ ,  $P(x) \Leftrightarrow P(y)$ .

**Proposition.** Let  $P : A \rightarrow \text{Type}$  be any family of types. For any  $x, y : A$  and  $p : x =_A y$ , there is a transport function  $\text{tr}_{P,p} : P(x) \rightarrow P(y)$ .

**Proof:** By **=elim**, to define  $\text{tr}_P : \prod_{x,y:A} \prod_{p:x=_A y} P(x) \rightarrow P(y)$  it suffices to define a term of type  $\prod_{x:A} P(x) \rightarrow P(x)$ , for which we take the identity function  $\lambda x. \lambda q. q$ .  $\square$

**Corollary.** For any  $P : A \rightarrow \text{Type}$ ,  $x, y : A$ , and  $p : x =_A y$ , then  $P(x) \simeq P(y)$ .

# The homotopy type theoretic Rosetta stone



type theory	logic	set theory	homotopy theory
$A$	proposition	set	space
$x : A$	proof	element	point
$\emptyset, 1$	$\perp, \top$	$\{\}, \{\{\}\}$	$\emptyset, *$
$A \times B$	$A$ and $B$	set of pairs	product space
$A + B$	$A$ or $B$	disjoint union	coproduct
$A \rightarrow B$	$A$ implies $B$	set of functions	function space
$P : A \rightarrow \text{Type}$	predicate	family of sets	fibration
$f : \prod_{x:A} P(x)$	conditional proof	family of elements	section
$\prod_{x:A} P(x)$	$\forall x. P(x)$	product	space of sections
$\sum_{x:A} P(x)$	$\exists x. P(x)$	disjoint union	total space
$p : x =_A y$	proof of equality	$x = y$	path from $x$ to $y$
$\sum_{x,y:A} x =_A y$	equality relation	diagonal	path space for $A$

# Contractible types



The homotopical perspective on type theory suggests new definitions:

A type  $A$  is **contractible** if it comes with a term of type

$$\text{is-contr}(A) := \sum_{a:A} \prod_{x:A} a =_A x$$

By  $\Sigma_{\text{elim}}$ , a proof of contractibility provides:

- a term  $c : A$  called the **center of contraction** and
- a dependent function  $h : \prod_{x:A} c =_A x$  called the **contracting homotopy**, which can be thought of as a continuous choice of paths  $h_x : c =_A x$  for each  $x : A$ .

# The hierarchy of types



Contractible types, those types  $A$  for which the type

$$\text{is-contr}(A) := \sum_{a:A} \prod_{x:A} a =_A x$$

has a term, form the bottom level of Voevodsky's hierarchy of types.

# The hierarchy of types



Contractible types, those types  $A$  for which the type

$$\text{is-contr}(A) := \sum_{a:A} \prod_{x:A} a =_A x$$

has a term, form the bottom level of Voevodsky's hierarchy of types.

A type  $A$  is

- a proposition if

$$\text{is-prop}(A) := \prod_{x,y:A} \text{is-contr}(x =_A y)$$



# The hierarchy of types



**Contractible types**, those types  $A$  for which the type

$$\text{is-contr}(A) := \sum_{a:A} \prod_{x:A} a =_A x$$

has a term, form the bottom level of **Voevodsky's hierarchy of types**.

A type  $A$  is

- a **proposition** if

$$\text{is-prop}(A) := \prod_{x,y:A} \text{is-contr}(x =_A y)$$

- a **set** or **0-type** if

$$\text{is-set}(A) := \prod_{x,y:A} \text{is-prop}(x =_A y)$$

# The hierarchy of types



**Contractible types**, those types  $A$  for which the type

$$\text{is-contr}(A) := \sum_{a:A} \prod_{x:A} a =_A x$$

has a term, form the bottom level of **Voevodsky's hierarchy of types**.

A type  $A$  is

- a **proposition** if

$$\text{is-prop}(A) := \prod_{x,y:A} \text{is-contr}(x =_A y)$$

- a **set** or **0-type** if

$$\text{is-set}(A) := \prod_{x,y:A} \text{is-prop}(x =_A y)$$

- a **sucn-type** for  $n : \mathbb{N}$  if

$$\text{is-sucn-type}(A) := \prod_{x,y:A} \text{is-}n\text{-type}(x =_A y)$$

# Equivalences



Similarly, homotopy theory suggests definitions of when two types  $A$  and  $B$  are **equivalent** or when a function  $f : A \rightarrow B$  is an **equivalence**:

An **equivalence** between types  $A$  and  $B$  is a term of type:

$$A \simeq B := \sum_{f:A \rightarrow B} \left( \sum_{g:B \rightarrow A} \prod_{a:A} g(f(a)) =_A a \right) \times \left( \sum_{h:B \rightarrow A} \prod_{b:B} f(h(b)) =_B b \right)$$

By  $\Sigma_{\text{elim}}$  and  $\times_{\text{elim}}$ , a term of type  $A \simeq B$  provides:

- functions  $f : A \rightarrow B$  and  $g, h : B \rightarrow A$  and
- homotopies  $\alpha$  and  $\beta$  relating  $g \circ f$  and  $f \circ h$  to the identity functions.

## Equivalences



Similarly, homotopy theory suggests definitions of when two types  $A$  and  $B$  are **equivalent** or when a function  $f : A \rightarrow B$  is an **equivalence**:

An **equivalence** between types  $A$  and  $B$  is a term of type:

$$A \simeq B := \sum_{f:A \rightarrow B} \left( \sum_{g:B \rightarrow A} \prod_{a:A} g(f(a)) =_A a \right) \times \left( \sum_{h:B \rightarrow A} \prod_{b:B} f(h(b)) =_B b \right)$$

By  $\Sigma$ elim and  $\times$ elim, a term of type  $A \simeq B$  provides:

- functions  $f : A \rightarrow B$  and  $g, h : B \rightarrow A$  and
- homotopies  $\alpha$  and  $\beta$  relating  $g \circ f$  and  $f \circ h$  to the identity functions.

Using this data, one can define a homotopy from  $g$  to  $h$ .

## Equivalences



Similarly, homotopy theory suggests definitions of when two types  $A$  and  $B$  are **equivalent** or when a function  $f : A \rightarrow B$  is an **equivalence**:

An **equivalence** between types  $A$  and  $B$  is a term of type:

$$A \simeq B := \sum_{f:A \rightarrow B} \left( \sum_{g:B \rightarrow A} \prod_{a:A} g(f(a)) =_A a \right) \times \left( \sum_{h:B \rightarrow A} \prod_{b:B} f(h(b)) =_B b \right)$$

By  $\Sigma$ **elim** and  $\times$ **elim**, a term of type  $A \simeq B$  provides:

- functions  $f : A \rightarrow B$  and  $g, h : B \rightarrow A$  and
- homotopies  $\alpha$  and  $\beta$  relating  $g \circ f$  and  $f \circ h$  to the identity functions.

Using this data, one can define a homotopy from  $g$  to  $h$ .

So why not say  $f : A \rightarrow B$  is an equivalence just when:

$$\sum_{g:B \rightarrow A} \left( \prod_{a:A} g(f(a)) =_A a \right) \times \left( \prod_{b:B} f(g(b)) =_B b \right)?$$

# Equivalences



Similarly, homotopy theory suggests definitions of when two types  $A$  and  $B$  are **equivalent** or when a function  $f : A \rightarrow B$  is an **equivalence**:

An **equivalence** between types  $A$  and  $B$  is a term of type:

$$A \simeq B := \sum_{f:A \rightarrow B} \left( \sum_{g:B \rightarrow A} \prod_{a:A} g(f(a)) =_A a \right) \times \left( \sum_{h:B \rightarrow A} \prod_{b:B} f(h(b)) =_B b \right)$$

By  $\Sigma_{\text{elim}}$  and  $\times_{\text{elim}}$ , a term of type  $A \simeq B$  provides:

- functions  $f : A \rightarrow B$  and  $g, h : B \rightarrow A$  and
- homotopies  $\alpha$  and  $\beta$  relating  $g \circ f$  and  $f \circ h$  to the identity functions.

Using this data, one can define a homotopy from  $g$  to  $h$ .

So why not say  $f : A \rightarrow B$  is an equivalence just when:

$$\sum_{g:B \rightarrow A} \left( \prod_{a:A} g(f(a)) =_A a \right) \times \left( \prod_{b:B} f(g(b)) =_B b \right)?$$

This type is not a **proposition** and may have non-trivial higher structure.

## The univalence axiom



Another notion of sameness between types is provided by the **universe**  $\mathcal{U}$  of types, which has (small) types  $A, B$  as its terms  $\rightsquigarrow A, B : \mathcal{U}$ .

Q: How do the types  $A =_{\mathcal{U}} B$  and  $A \simeq B$  compare?

## The univalence axiom



Another notion of sameness between types is provided by the **universe**  $\mathcal{U}$  of types, which has (small) types  $A, B$  as its terms  $\rightsquigarrow A, B : \mathcal{U}$ .

Q: How do the types  $A =_{\mathcal{U}} B$  and  $A \simeq B$  compare?

By **=elim**, there is a canonical function

$$\text{id-to-equiv} : \prod_{A, B : \mathcal{U}} (A =_{\mathcal{U}} B) \rightarrow (A \simeq B)$$

defined by sending  $\text{refl}_A$  to the identity equivalence  $\text{id}_A$ .



## The univalence axiom



Another notion of sameness between types is provided by the **universe**  $\mathcal{U}$  of types, which has (small) types  $A, B$  as its terms  $\rightsquigarrow A, B : \mathcal{U}$ .

Q: How do the types  $A =_{\mathcal{U}} B$  and  $A \simeq B$  compare?

By  $=_{\text{elim}}$ , there is a canonical function

$$\text{id-to-equiv} : \prod_{A, B : \mathcal{U}} (A =_{\mathcal{U}} B) \rightarrow (A \simeq B)$$

defined by sending  $\text{refl}_A$  to the identity equivalence  $\text{id}_A$ .

**Univalence Axiom:**  $\text{id-to-equiv} : (A =_{\mathcal{U}} B) \rightarrow (A \simeq B)$  is an equivalence for all  $A, B : \mathcal{U}$ .

“Identity is equivalent to equivalence.”

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B)$$

## Consequences of univalence

There are myriad consequences of the univalence axiom  $(A =_{\mathcal{U}} B) \simeq (A \simeq B)$ :



## Consequences of univalence



There are myriad consequences of the univalence axiom  $(A =_{\mathcal{U}} B) \simeq (A \simeq B)$ :

- The **structure-identity principle**, which specializes to the statement that for set-based structures (monoids, groups, rings) **isomorphic** structures are **identical**.

## Consequences of univalence



There are myriad consequences of the univalence axiom  $(A =_{\mathcal{U}} B) \simeq (A \simeq B)$ :

- The **structure-identity principle**, which specializes to the statement that for set-based structures (monoids, groups, rings) **isomorphic** structures are **identical**.
- **Function extensionality**: for any  $f, g : A \rightarrow B$ , the canonical function defines an equivalence between the identity type and the type of homotopies:

$$\text{id-to-htpy} : (f =_{A \rightarrow B} g) \rightarrow \left( \prod_{a:A} f(a) =_B g(a) \right)$$

# Consequences of univalence



There are myriad consequences of the univalence axiom  $(A =_{\mathcal{U}} B) \simeq (A \simeq B)$ :

- The **structure-identity principle**, which specializes to the statement that for set-based structures (monoids, groups, rings) **isomorphic** structures are **identical**.
- **Function extensionality**: for any  $f, g : A \rightarrow B$ , the canonical function defines an equivalence between the identity type and the type of homotopies:

$$\text{id-to-htpy} : (f =_{A \rightarrow B} g) \rightarrow \left( \prod_{a:A} f(a) =_B g(a) \right)$$

- By **indiscernibility of identicals**, if  $x, y : A$  and  $p : x =_A y$  then  $P(x) \simeq P(y)$  for any  $P : A \rightarrow \mathbf{Type}$ . By univalence, whenever  $X \simeq Y$  then  $X =_{\mathcal{U}} Y$  and thus any type constructed from  $X$  is equivalent to the corresponding type constructed from  $Y$ .

Voevodsky's univalence axiom — which is justified by the homotopical model of type theory — captures the common mathematical practice of transporting results proven about one object to any other object that is equivalent to it!



3

Contractibility as uniqueness

## Rebuilding the pragmatic foundations for higher structures



*I am pretty strongly convinced that there is an ongoing reversal in the collective consciousness of mathematicians: the homotopical picture of the world becomes the basic intuition, and if you want to get a discrete set, then you pass to the set of connected components of a space defined only up to homotopy ... Cantor's problems of the infinite recede to the background: from the very start, our images are so infinite that if you want to make something finite out of them, you must divide them by another infinity.*

— Yuri Manin “We do not choose mathematics as our profession, it chooses us: Interview with Yuri Manin” by Mikhail Gelfand

## $\infty$ -categories in set theory



Essentially,  $\infty$ -categories are 1-categories in which all the **sets** have been replaced by  **$\infty$ -groupoids** aka **homotopy types**:

sets  $:: \infty$ -groupoids  
categories  $:: \infty$ -categories



## $\infty$ -categories in set theory



Essentially,  $\infty$ -categories are 1-categories in which all the **sets** have been replaced by  **$\infty$ -groupoids** aka **homotopy types**:

sets ::  $\infty$ -groupoids  
categories ::  $\infty$ -categories

Where

- categories have sets of objects,  $\infty$ -categories have  $\infty$ -groupoids of objects, and
- categories have hom-sets,  $\infty$ -categories have  $\infty$ -groupoidal mapping spaces.

## $\infty$ -categories in set theory



Essentially,  $\infty$ -categories are 1-categories in which all the **sets** have been replaced by  **$\infty$ -groupoids** aka **homotopy types**:

sets ::  $\infty$ -groupoids  
categories ::  $\infty$ -categories

Where

- categories have sets of objects,  $\infty$ -categories have  $\infty$ -groupoids of objects, and
- categories have hom-sets,  $\infty$ -categories have  $\infty$ -groupoidal mapping spaces.

While the axioms that turn a directed graph into a category are expressed in the language of set theory — a category has a composition function satisfying axioms expressed in first-order logic with equality

## $\infty$ -categories in set theory



Essentially,  $\infty$ -categories are 1-categories in which all the **sets** have been replaced by  **$\infty$ -groupoids** aka **homotopy types**:

sets ::  $\infty$ -groupoids  
categories ::  $\infty$ -categories

Where

- categories have sets of objects,  $\infty$ -categories have  $\infty$ -groupoids of objects, and
- categories have hom-sets,  $\infty$ -categories have  $\infty$ -groupoidal mapping spaces.

While the axioms that turn a directed graph into a category are expressed in the language of set theory — a category has a composition function satisfying axioms expressed in first-order logic with equality — composition in an  $\infty$ -category, as a morphism between  $\infty$ -groupoids, isn't a “function” in the traditional sense (since homotopy types do not have underlying sets of points).

## $\infty$ -categories in set theory



Essentially,  $\infty$ -categories are 1-categories in which all the **sets** have been replaced by  **$\infty$ -groupoids** aka **homotopy types**:

sets ::  $\infty$ -groupoids  
categories ::  $\infty$ -categories

Where

- categories have sets of objects,  $\infty$ -categories have  $\infty$ -groupoids of objects, and
- categories have hom-sets,  $\infty$ -categories have  $\infty$ -groupoidal mapping spaces.

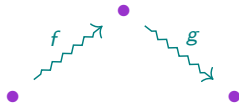
While the axioms that turn a directed graph into a category are expressed in the language of set theory — a category has a composition function satisfying axioms expressed in first-order logic with equality — composition in an  $\infty$ -category, as a morphism between  $\infty$ -groupoids, isn't a “function” in the traditional sense (since homotopy types do not have underlying sets of points).

This is why  $\infty$ -categories are so difficult to model within set theory.

## Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

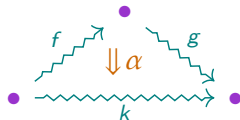
of a space  $X$ , composites of paths are witnessed by higher paths:



## Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

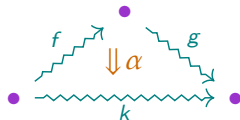
of a space  $X$ , composites of paths are witnessed by higher paths:



## Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

of a space  $X$ , composites of paths are witnessed by higher paths:

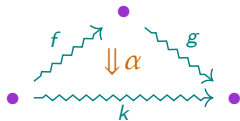


**Theorem.** The space of composites of two paths  $f$  and  $g$  in  $X$  is contractible.

## Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

of a space  $X$ , composites of paths are witnessed by higher paths:



**Theorem.** The space of composites of two paths  $f$  and  $g$  in  $X$  is contractible.

**Proof:** The **space of composites** of paths  $f$  and  $g$  in  $A$  is defined by the pullback:

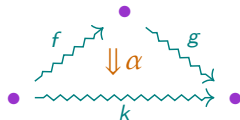
$$\begin{array}{ccc} \text{Comp}(f, g) & \hookrightarrow & \text{Map}(\Delta, X) \\ \downarrow & \lrcorner & \downarrow \text{restrict} \\ * & \xrightarrow{f \wedge g} & \text{Map}(\Lambda, X) \end{array}$$



# Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

of a space  $X$ , composites of paths are witnessed by higher paths:



**Theorem.** The space of composites of two paths  $f$  and  $g$  in  $X$  is contractible.

**Proof:** The **space of composites** of paths  $f$  and  $g$  in  $A$  is defined by the pullback:

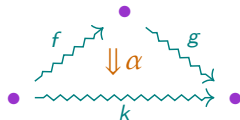
$$\begin{array}{ccccc}
 S^{n-1} & \longrightarrow & \text{Comp}(f, g) & \hookrightarrow & \text{Map}(\Delta, X) \\
 \downarrow & \nearrow \text{dashed} & \downarrow & \lrcorner & \downarrow \text{restrict} \\
 D^n & \longrightarrow & * & \xrightarrow{f \wedge g} & \text{Map}(\Lambda, X)
 \end{array}$$

A space is **contractible** just when any sphere  $S^{n-1}$  can be filled to a disk  $D^n$ .

# Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

of a space  $X$ , composites of paths are witnessed by higher paths:



**Theorem.** The space of composites of two paths  $f$  and  $g$  in  $X$  is contractible.

**Proof:** The **space of composites** of paths  $f$  and  $g$  in  $A$  is defined by the pullback:

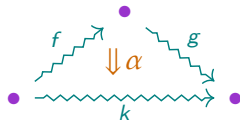
$$\begin{array}{ccccc}
 S^{n-1} & \longrightarrow & \text{Comp}(f, g) & \hookrightarrow & \text{Map}(\Delta, X) \\
 \downarrow & \nearrow & \downarrow & \lrcorner & \downarrow \text{restrict} \\
 D^n & \xrightarrow{\quad} & * & \xrightarrow{f \wedge g} & \text{Map}(\Lambda, X)
 \end{array}$$

A space is **contractible** just when any sphere  $S^{n-1}$  can be filled to a disk  $D^n$ .

# Composing paths

In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

of a space  $X$ , composites of paths are witnessed by higher paths:



**Theorem.** The space of composites of two paths  $f$  and  $g$  in  $X$  is contractible.

**Proof:** The **space of composites** of paths  $f$  and  $g$  in  $A$  is defined by the pullback:

$$\begin{array}{ccc}
 S^{n-1} & \longrightarrow & \text{Comp}(f, g) \hookrightarrow \text{Map}(\Delta, X) \\
 \downarrow & \nearrow & \downarrow \text{restrict} \\
 D^n & \xrightarrow{\quad} & * \xrightarrow{f \wedge g} \text{Map}(\Lambda, X)
 \end{array}
 \quad \Leftrightarrow \quad
 \begin{array}{ccc}
 S^{n-1} \times \Delta \cup_{S^{n-1} \times \Lambda} D^n \times \Lambda & \longrightarrow & X \\
 \downarrow & \nearrow & \\
 D^n \times \Delta & &
 \end{array}$$

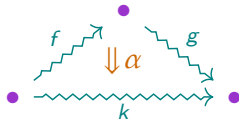
A space is **contractible** just when any sphere  $S^{n-1}$  can be filled to a disk  $D^n$ .

# Composing paths



In the **total singular complex** aka the **fundamental  $\infty$ -groupoid** aka the **anima** or “soul”

of a space  $X$ , composites of paths are witnessed by higher paths:



**Theorem.** The space of composites of two paths  $f$  and  $g$  in  $X$  is contractible.

**Proof:** The **space of composites** of paths  $f$  and  $g$  in  $A$  is defined by the pullback:

$$\begin{array}{ccc}
 S^{n-1} & \longrightarrow & \text{Comp}(f, g) \hookrightarrow \text{Map}(\Delta, X) \\
 \downarrow & \nearrow & \downarrow \text{restrict} \\
 D^n & \xrightarrow{\quad} & * \xrightarrow{f \wedge g} \text{Map}(\Lambda, X)
 \end{array}
 \quad \Leftrightarrow \quad
 \begin{array}{ccc}
 S^{n-1} \times \Delta \cup_{S^{n-1} \times \Lambda} D^n \times \Lambda & \longrightarrow & X \\
 \downarrow & \nearrow & \\
 D^n \times \Delta & &
 \end{array}$$

A space is **contractible** just when any sphere  $S^{n-1}$  can be filled to a disk  $D^n$ . The extension exists since the inclusion admits a continuous deformation retract. □

## $\infty$ -categories in homotopy type theory



The identity type family gives each type the structure of an  $\infty$ -groupoid: each type  $A$  has a family of identity types  $x =_A y$  over  $x, y : A$  whose terms  $p : x =_A y$  are called **paths**.

## $\infty$ -categories in homotopy type theory



The identity type family gives each type the structure of an  $\infty$ -groupoid: each type  $A$  has a family of identity types  $x =_A y$  over  $x, y : A$  whose terms  $p : x =_A y$  are called **paths**.

In a “directed” extension of homotopy type theory introduced in

Emily Riehl and Michael Shulman, *A type theory for synthetic  $\infty$ -categories*,  
Higher Structures 1(1):116–193, 2017

each type  $A$  also has a family of hom types  $\text{Hom}_A(x, y)$  over  $x, y : A$  whose terms  $f : \text{Hom}_A(x, y)$  are called **arrows**.

## $\infty$ -categories in homotopy type theory



The identity type family gives each type the structure of an  $\infty$ -groupoid: each type  $A$  has a family of identity types  $x =_A y$  over  $x, y : A$  whose terms  $p : x =_A y$  are called **paths**.

In a “directed” extension of homotopy type theory introduced in

Emily Riehl and Michael Shulman, *A type theory for synthetic  $\infty$ -categories*,  
Higher Structures 1(1):116–193, 2017

each type  $A$  also has a family of hom types  $\text{Hom}_A(x, y)$  over  $x, y : A$  whose terms  $f : \text{Hom}_A(x, y)$  are called **arrows**.

**Definition (RS).** A type  $A$  is an  $\infty$ -category if:

## $\infty$ -categories in homotopy type theory



The identity type family gives each type the structure of an  $\infty$ -groupoid: each type  $A$  has a family of identity types  $x =_A y$  over  $x, y : A$  whose terms  $p : x =_A y$  are called **paths**.

In a “directed” extension of homotopy type theory introduced in

Emily Riehl and Michael Shulman, *A type theory for synthetic  $\infty$ -categories*,  
Higher Structures 1(1):116–193, 2017

each type  $A$  also has a family of hom types  $\text{Hom}_A(x, y)$  over  $x, y : A$  whose terms  $f : \text{Hom}_A(x, y)$  are called **arrows**.

**Definition (RS).** A type  $A$  is an  $\infty$ -category if:

- Every pair of arrows  $f : \text{Hom}_A(x, y)$  and  $g : \text{Hom}_A(y, z)$  has a **unique composite**, defining a term  $g \circ f : \text{Hom}_A(x, z)$ .



## $\infty$ -categories in homotopy type theory



The identity type family gives each type the structure of an  $\infty$ -groupoid: each type  $A$  has a family of identity types  $x =_A y$  over  $x, y : A$  whose terms  $p : x =_A y$  are called **paths**.

In a “directed” extension of homotopy type theory introduced in

Emily Riehl and Michael Shulman, *A type theory for synthetic  $\infty$ -categories*,  
Higher Structures 1(1):116–193, 2017

each type  $A$  also has a family of hom types  $\text{Hom}_A(x, y)$  over  $x, y : A$  whose terms  $f : \text{Hom}_A(x, y)$  are called **arrows**.

**Definition (RS).** A type  $A$  is an  $\infty$ -category if:

- Every pair of arrows  $f : \text{Hom}_A(x, y)$  and  $g : \text{Hom}_A(y, z)$  has a **unique composite**, defining a term  $g \circ f : \text{Hom}_A(x, z)$ .
- Paths in  $A$  are equivalent to **isomorphisms** in  $A$ .

## $\infty$ -categories in homotopy type theory



The identity type family gives each type the structure of an  $\infty$ -groupoid: each type  $A$  has a family of identity types  $x =_A y$  over  $x, y : A$  whose terms  $p : x =_A y$  are called **paths**.

In a “directed” extension of homotopy type theory introduced in

Emily Riehl and Michael Shulman, *A type theory for synthetic  $\infty$ -categories*,  
Higher Structures 1(1):116–193, 2017

each type  $A$  also has a family of hom types  $\text{Hom}_A(x, y)$  over  $x, y : A$  whose terms  $f : \text{Hom}_A(x, y)$  are called **arrows**.

**Definition (RS).** A type  $A$  is an  $\infty$ -category if:

- Every pair of arrows  $f : \text{Hom}_A(x, y)$  and  $g : \text{Hom}_A(y, z)$  has a **unique composite**, defining a term  $g \circ f : \text{Hom}_A(x, z)$ .
- Paths in  $A$  are equivalent to **isomorphisms** in  $A$ .

Perhaps someday  $\infty$ -category theory will be easy enough to teach to undergraduates.



4

Computer proof assistants that compute

## Formalizing univalent foundations



*Today we face a problem that involves two difficult to satisfy conditions. On the one hand we have to find a way for computer assisted verification of mathematical proofs. This is necessary, first of all, because we have to stop the dissolution of the concept of proof in mathematics. On the other hand we have to preserve the intimate connection between mathematics and the world of human intuition. This connection is what moves mathematics forward and what we often experience as the beauty of mathematics.*

*— Vladimir Voevodsky, Heidelberg Laureate Forum, September 2016*

## Formalizing univalent foundations



*Today we face a problem that involves two difficult to satisfy conditions. On the one hand we have to find a way for computer assisted verification of mathematical proofs. This is necessary, first of all, because we have to stop the dissolution of the concept of proof in mathematics. On the other hand we have to preserve the intimate connection between mathematics and the world of human intuition. This connection is what moves mathematics forward and what we often experience as the beauty of mathematics.*

*— Vladimir Voevodsky, Heidelberg Laureate Forum, September 2016*

Lean is a computer proof assistant based on dependent type theory, but its identity types are propositions so all types are sets, which is not compatible with univalence!

## Formalizing univalent foundations



*Today we face a problem that involves two difficult to satisfy conditions. On the one hand we have to find a way for computer assisted verification of mathematical proofs. This is necessary, first of all, because we have to stop the dissolution of the concept of proof in mathematics. On the other hand we have to preserve the intimate connection between mathematics and the world of human intuition. This connection is what moves mathematics forward and what we often experience as the beauty of mathematics.*

— Vladimir Voevodsky, Heidelberg Laureate Forum, September 2016

Lean is a computer proof assistant based on dependent type theory, but its identity types are propositions so all types are sets, which is not compatible with univalence!

Thus, the homotopy type theory libraries have been built in **Coq**, **Agda** (`--without-K`), **Lean v2**, and new experimental systems ... though much of the work thusfar has been metatheoretic, developing univalent formal systems and their semantics.

# Computing the Brunerie number



Guillaume Brunerie's 2016 PhD thesis contained a proof in homotopy type theory that  
there exists  $n : \mathbb{Z}$  such that  $\pi_4(S^3) \simeq \mathbb{Z}/n\mathbb{Z}$

using the Hopf fibration, the long exact sequence of homotopy groups of a fibration, the Freudenthal suspension theorem, the James construction, the Blakers-Massey theorem, and Whitehead products — a substantial part of synthetic homotopy theory!

# Computing the Brunerie number



Guillaume Brunerie's 2016 PhD thesis contained a proof in homotopy type theory that  
there exists  $n : \mathbb{Z}$  such that  $\pi_4(S^3) \simeq \mathbb{Z}/n\mathbb{Z}$

using the Hopf fibration, the long exact sequence of homotopy groups of a fibration, the Freudenthal suspension theorem, the James construction, the Blakers-Massey theorem, and Whitehead products — a substantial part of synthetic homotopy theory!

*This result is quite remarkable in that even though it is a constructive proof, it is not at all obvious how to actually compute this  $n$ . At the time of writing, we still haven't managed to extract its value from its definition. A complete and concise definition of this number  $n$  is presented in appendix B, for the benefit of someone wanting to implement it in a prospective proof assistant.*



# Computing the Brunerie number



Guillaume Brunerie's 2016 PhD thesis contained a proof in homotopy type theory that  
there exists  $n : \mathbb{Z}$  such that  $\pi_4(S^3) \simeq \mathbb{Z}/n\mathbb{Z}$

using the Hopf fibration, the long exact sequence of homotopy groups of a fibration, the Freudenthal suspension theorem, the James construction, the Blakers-Massey theorem, and Whitehead products — a substantial part of synthetic homotopy theory!

*This result is quite remarkable in that even though it is a constructive proof, it is not at all obvious how to actually compute this  $n$ . At the time of writing, we still haven't managed to extract its value from its definition. A complete and concise definition of this number  $n$  is presented in appendix B, for the benefit of someone wanting to implement it in a prospective proof assistant.*

When dependent type theory is a foundation for **constructive mathematics**, the univalence axiom posits an inverse equivalence to **id-to-equiv** which is “stuck.” Voevodsky's simplicial set based model proves that it is consistent to assume univalence but does not provide a construction of the inverse equivalence.

# A constructive foundation for univalent mathematics



In the last decade, constructive proofs of univalence have been discovered in **cubical** variants of homotopy type theory with semantics in various categories of cubical sets:

Bezem-Coquand-Huber, Cohen-Coquand-Huber-Mörtberg, Awodey,  
Angiuli-Brunerie-Coquand-Favonia-Harper-Licata,  
Awodey-Cavallo-Coquand-Riehl-Sattler,...

In parallel, a variety of experimental **cubical proof assistants** have been developed —  
cubical, cubicaltt, yacctl, redtt, Cubical Agda,...

# A constructive foundation for univalent mathematics



In the last decade, constructive proofs of univalence have been discovered in **cubical** variants of homotopy type theory with semantics in various categories of cubical sets:

Bezem-Coquand-Huber, Cohen-Coquand-Huber-Mörtberg, Awodey,  
Angiuli-Brunerie-Coquand-Favonia-Harper-Licata,  
Awodey-Cavallo-Coquand-Riehl-Sattler,...

In parallel, a variety of experimental **cubical proof assistants** have been developed — cubical, cubicaltt, yacctl, redtt, Cubical Agda,... — but attempts to compute the Brunerie number in each of these during 2014-2021 ran out of memory.

**Breakthrough (Ljungström 2022):** after discovering simplifications in the constructive proof, Brunerie's number  $n$  normalizes in seconds in Cubical Agda

# A constructive foundation for univalent mathematics



In the last decade, constructive proofs of univalence have been discovered in **cubical** variants of homotopy type theory with semantics in various categories of cubical sets:

Bezem-Coquand-Huber, Cohen-Coquand-Huber-Mörtberg, Awodey,  
Angiuli-Brunerie-Coquand-Favonia-Harper-Licata,  
Awodey-Cavallo-Coquand-Riehl-Sattler,...

In parallel, a variety of experimental **cubical proof assistants** have been developed — cubical, cubicaltt, yacctl, redtt, Cubical Agda,... — but attempts to compute the Brunerie number in each of these during 2014-2021 ran out of memory.

**Breakthrough (Ljungström 2022):** after discovering simplifications in the constructive proof, Brunerie's number  $n$  normalizes in seconds in Cubical Agda to **-2!**

# References



A reintroduction to proofs

- Clive Newstead, [An Infinite Descent into Pure Mathematics](#)

On the art of giving the same name to different things

- [Homotopy Type Theory: Univalent Foundations of Mathematics](#)
- Egbert Rijke, [Introduction to Homotopy Type Theory](#), forthcoming from *CUP*

Contractibility as uniqueness

- Emily Riehl, Mike Shulman, [A type theory for synthetic  \$\infty\$ -categories](#)
- Emily Riehl,  [\$\infty\$ -category theory for undergraduates](#), forthcoming in the *Notices*

Computer proof assistants that compute:

- Axel Ljungström, [The Brunerie Number Is -2](#)
- Anders Mörtberg (j.w.w. Axel Ljungström), [Formalizing  \$\pi\_4\(S^3\)\$  and computing a Brunerie number in Cubical Agda](#)

Thank you!