Emily Riehl

Johns Hopkins University

How I became seduced by univalent foundations

Mathematisches Kolloquium im Wintersemester 2022/23

Plan

1. A reintroduction to proofs

2. On the art of giving the same name to different things

3. Contractibility as uniqueness

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$_1$: If $p \wedge q$ is true, then p is true.
- ${}^{\wedge}$elim$_2$: If $p \wedge q$ is true, then q is true.

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

- ${}^{\vee}$intro$_1$: If p is true, then $p \vee q$ is true.
- ${}^{\vee}$intro$_2$: 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, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

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, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

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

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, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

Proof: By $^{\Rightarrow}$intro we may assume that $(p \Rightarrow q) \wedge (q \Rightarrow r)$ is true; our goal is to prove $p \Rightarrow r$. By $^{\Rightarrow}$intro again, we may also assume p is true, so our goal is just to prove r.

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, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

Proof: By $^{\Rightarrow}$intro we may assume that $(p \Rightarrow q) \wedge (q \Rightarrow r)$ is true; our goal is to prove $p \Rightarrow r$. By $^{\Rightarrow}$intro again, we may also assume p is true, so our goal is just to prove r. By $^{\wedge}$elim$_1$ and $^{\wedge}$elim$_2$ it follows that $p \Rightarrow q$ and $q \Rightarrow r$ 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, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

Proof: By $^{\Rightarrow}$intro we may assume that $(p \Rightarrow q) \wedge (q \Rightarrow r)$ is true; our goal is to prove $p \Rightarrow r$. By $^{\Rightarrow}$intro again, we may also assume p is true, so our goal is just to prove r. By $^{\wedge}\text{elim}_1$ and $^{\wedge}\text{elim}_2$ it follows that $p \Rightarrow q$ and $q \Rightarrow r$ are true. By $^{\Rightarrow}$elim, from p and $p \Rightarrow q$, we may conclude that q is true. By $^{\Rightarrow}$elim again, from q and $q \Rightarrow r$, we may conclude r is true as desired. □

Universal and existential quantification

Let $p : X \to \{\bot, \top\}$ 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., $\mathbb{N}$, $\mathbb{Q}$, Group
- terms, e.g., $17 : \mathbb{N}$, $\sqrt{2} : \mathbb{R}$, K_4 : 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., $\mathbb{N}$, $\mathbb{Q}$, Group
- terms, e.g., $17 : \mathbb{N}$, $\sqrt{2} : \mathbb{R}$, K_4 : Group
- dependent types, e.g., $\mathbb{R}^\bullet : \mathbb{N} \to \mathrm{Type}$, $\text{is-prime} : \mathbb{N} \to \mathrm{Type}$, $\mathrm{Mat}_{\bullet \times \bullet} : \mathbb{N} \to \mathbb{N} \to \mathrm{Type}$

Dependent type theory

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

- types, e.g., $\mathbb{N}$, $\mathbb{Q}$, Group
- terms, e.g., $17 : \mathbb{N}$, $\sqrt{2} : \mathbb{R}$, $K_4 : \mathsf{Group}$
- dependent types, e.g., $\mathbb{R}^{\bullet} : \mathbb{N} \to \mathsf{Type}$, $\mathsf{is\text{-}prime} : \mathbb{N} \to \mathsf{Type}$, $\mathsf{Mat}_{\bullet\times\bullet} : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}$
- dependent terms, e.g., $\vec{0}^{\bullet} : \prod_{n:\mathbb{N}} \mathbb{R}^n$, $I_{\bullet} : \prod_{n:\mathbb{N}} \mathsf{Mat}_{n,n}$, $S_{\bullet} : \prod_{n:\mathbb{N}} \mathsf{Group}$

Dependent type theory

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

- types, e.g., $\mathbb{N}$, $\mathbb{Q}$, Group
- terms, e.g., $17 : \mathbb{N}$, $\sqrt{2} : \mathbb{R}$, K_4 : Group
- dependent types, e.g., $\mathbb{R}^\bullet : \mathbb{N} \to \mathrm{Type}$, is-prime $: \mathbb{N} \to \mathrm{Type}$, $\mathrm{Mat}_{\bullet \times \bullet} : \mathbb{N} \to \mathbb{N} \to \mathrm{Type}$
- dependent terms, e.g., $\vec{0}^\bullet : \prod_{n:\mathbb{N}} \mathbb{R}^n$, $I_\bullet : \prod_{n:\mathbb{N}} \mathrm{Mat}_{n,n}$, $S_\bullet : \prod_{n:\mathbb{N}} \mathrm{Group}$

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 $\mathrm{pr}_1 p : A$ and $\mathrm{pr}_2 p : B$

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 $\mathrm{inl}\, a : A + B$, and given a term $b : B$, there is a term $\mathrm{inr}\, b : A + B$
- ${}^{+}$elim: given a family of types $C : (A + B) \to \mathrm{Type}$, if there are dependent terms $c : \prod_{a:A} C(\mathrm{inl}\, a)$ and $d : \prod_{b:B} C(\mathrm{inr}\, b)$, 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 $f : X \to Y$ 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 $f : X \to Y$ 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 $A \to B$ is governed by the rules:

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

plus computation rules that relate λ-abstractions and evaluations.

Dependent functions and dependent sums

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

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

- $^{\Pi}$intro: if in the context of a variable $x : A$ there is a term $b_x : B(x)$ there is a term $\lambda x.b_x : \prod_{x:A} B(x)$
- $^{\Pi}$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 λ-abstractions and evaluations.

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

- $^{\Sigma}$intro: if there are terms $a : A$ and $b : B(a)$, there is a term $(a, b) : \sum_{x:A} B(x)$
- $^{\Sigma}$elim: given a type family $C : \sum_{x:A} B(x) \to \mathrm{Type}$, 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:

$$3_{\mathsf{vN}} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \qquad 3_{\mathsf{Z}} := \{\{\{\{\}\}\}\}$$

The natural numbers in set theory

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

$$3_{vN} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \qquad 3_{Z} := \{\{\{\{\}\}\}\}$$

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:

$$3_{\mathrm{vN}} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \qquad 3_{\mathrm{Z}} := \{\{\{\{\}\}\}\}$$

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 $\mathrm{suc}\, n \in \mathbb{N}$.
- 0 is not the successor of any natural number.
- No two natural numbers have the same successor.
- The principle of mathematical induction: for all $P : \mathbb{N} \to \{\bot, \top\}$

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

The natural numbers in dependent type theory

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

- $^{\mathbb{N}}$intro: there is a term $0 : \mathbb{N}$ and for any term $n : \mathbb{N}$ there is a term $\mathsf{suc}\, n : \mathbb{N}$

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

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

$$^{\mathbb{N}}\mathsf{ind} : P(0) \to \left(\prod_{k \in \mathbb{N}} P(k) \to P(\mathsf{suc}\, k)\right) \to \left(\prod_{n \in \mathbb{N}} P(n)\right)$$

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:

- $^{\mathbb{N}}$intro: there is a term $0 : \mathbb{N}$ and for any term $n : \mathbb{N}$ there is a term $\mathsf{suc}\, n : \mathbb{N}$

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

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

$$^{\mathbb{N}}\mathsf{ind} : P(0) \to \left(\prod_{k\in\mathbb{N}} P(k) \to P(\mathsf{suc}\, k)\right) \to \left(\prod_{n\in\mathbb{N}} P(n)\right)$$

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 $\mathsf{refl}_x : x =_A x$

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 refl_x. That is

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

A computation rule establishes that the proof of $P(x, x, \mathsf{refl}_x)$ 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 ∞-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 ∞-groupoid.

The ∞-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 ∞-groupoid.

The ∞-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 ∞-groupoid.

The ∞-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 refl_x. That is

$$^{=}\mathsf{ind} : \Big(\prod_{x:A} P(x,x,\mathsf{refl}_x)\Big) \to \Big(\prod_{x,y:A} \prod_{p:x=_A y} P(x,y,p)\Big)$$

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 refl_x. That is

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

In first-order logic, one 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 refl_x. That is

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

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

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

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

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 refl_x. That is

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

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

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

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

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

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 refl_x. That is

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

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

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

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

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

Corollary. For any $P : A \to \mathrm{Type}$, $x, y : A$, 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
$\emptyset, 1$	$\bot, \top$	$\{\}, \{\{\}\}$	$\emptyset, *$
$A \times B$	A and B	set of pairs	product space
$A + B$	A or B	disjoint union	coproduct
$A \to B$	A implies B	set of functions	function space
$P : A \to \mathrm{Type}$	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

The homotopy interpretation, developed by Awodey–Warren and Voevodsky, is justified by Voevodsky's model of types as Kan complexes and type families as Kan fibrations.

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 ${}^{\Sigma}$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 $x : A$.

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 $x : A$.

If A is contractible and $x, y : A$, then there is a term $p : x =_A y$, so x and y are indiscernible. Thus, contractible types behave as if they have a single term.

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 suc$n$-type for $n : \mathbb{N}$ if
$$\text{is-suc}n\text{-type}(A) := \prod_{x,y:A} \text{is-}n\text{-type}(x =_A y)$$

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 suc$n$-type for $n : \mathbb{N}$ if

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

Traditional set theory and logic are encoded by layers formed by the 0-types and -1-types.

Equivalences

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

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

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

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

- functions $f : A \to B$ and $g, h : B \to A$ and
- homotopies α and β 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 $f : A \to B$ is an equivalence:

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

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

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

- functions $f : A \to B$ and $g, h : B \to A$ and
- homotopies α and β 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 $f : A \to B$ is an equivalence:

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

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

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

- functions $f : A \to B$ and $g, h : B \to A$ and
- homotopies α and β 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 $f : A \to B$ is an equivalence just when:

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

Equivalences

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

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

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

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

- functions $f : A \to B$ and $g, h : B \to A$ and
- homotopies α and β 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 $f : A \to B$ is an equivalence just when:

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

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 $\quad\rightsquigarrow\quad A, B : \mathcal{U}$.

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 $\quad\rightsquigarrow\quad A, B : \mathcal{U}$.

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

By $^{=}$elim, there is a canonical function

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

defined by sending refl_A to the identity equivalence 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 $^{=}$elim, there is a canonical function

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

defined by sending refl_A to the identity equivalence id_A.

Univalence Axiom: $\text{id-to-equiv} : (A =_{\mathcal{U}} B) \to (A \simeq B)$ 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 $f, g : A \to B$, the canonical function defines an equivalence between the identity type and the type of homotopies:

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

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 $f, g : A \to B$, the canonical function defines an equivalence between the identity type and the type of homotopies:

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

- By indiscernibility of identicals, if $x, y : A$ and $p : x =_A y$ then $P(x) \simeq P(y)$ for any $P : A \to \text{Type}$. 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*

∞-categories in set theory

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

$$\text{sets} :: \infty\text{-groupoids}$$
$$\text{categories} :: \infty\text{-categories}$$

∞-categories in set theory

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

$$\text{sets} :: \infty\text{-groupoids}$$
$$\text{categories} :: \infty\text{-categories}$$

Where

- categories have sets of objects, ∞-categories have ∞-groupoids of objects, and
- categories have hom-sets, ∞-categories have ∞-groupoidal mapping spaces.

∞-categories in set theory

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

$$\text{sets} :: \infty\text{-groupoids}$$
$$\text{categories} :: \infty\text{-categories}$$

Where

- categories have sets of objects, ∞-categories have ∞-groupoids of objects, and
- categories have hom-sets, ∞-categories have ∞-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

∞-categories in set theory

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

$$\text{sets} :: \infty\text{-groupoids}$$
$$\text{categories} :: \infty\text{-categories}$$

Where

- categories have sets of objects, ∞-categories have ∞-groupoids of objects, and
- categories have hom-sets, ∞-categories have ∞-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 ∞-category, as a morphism between ∞-groupoids, isn't a "function" in the traditional sense (since homotopy types do not have underlying sets of points).

∞-categories in set theory

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

sets :: ∞-groupoids
categories :: ∞-categories

Where

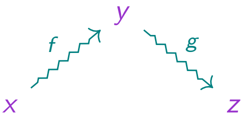
- categories have sets of objects, ∞-categories have ∞-groupoids of objects, and
- categories have hom-sets, ∞-categories have ∞-groupoidal mapping spaces.

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 ∞-category, as a morphism between ∞-groupoids, isn't a "function" in the traditional sense (since homotopy types do not have underlying sets of points).

This is why ∞-categories are so difficult to model within set theory.

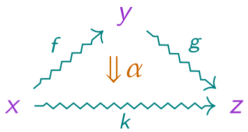
Composing paths

In the total singular complex aka the fundamental ∞-groupoid aka the anima or "soul" of a space X, composites of paths are witnessed by higher paths:



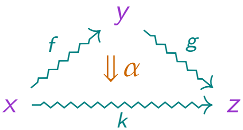
Composing paths

In the total singular complex aka the fundamental ∞-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 ∞-groupoid aka the anima or "soul" of a space X, composites of paths are witnessed by higher paths:

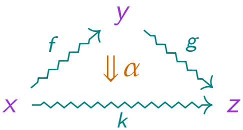


Theorem. The space of composites of two paths f and g in X is contractible.

Composing paths

In the total singular complex aka the fundamental ∞-groupoid aka the anima or "soul"

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



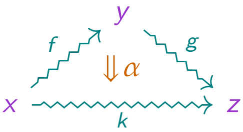
Theorem. The space of composites of two paths f and g in X is contractible.

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

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

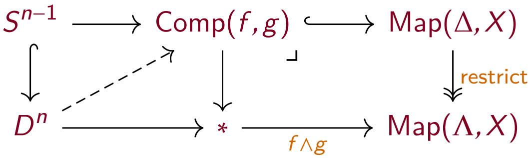
Composing paths

In the total singular complex aka the fundamental ∞-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.

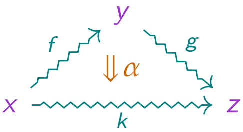
Proof: The space of composites of paths f and g in X is defined by the pullback:



A space is contractible just when any sphere S^{n-1} can be filled to a disk D^n for $n \geq 0$.

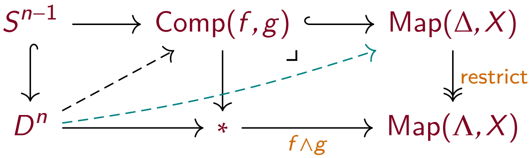
Composing paths

In the total singular complex aka the fundamental ∞-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.

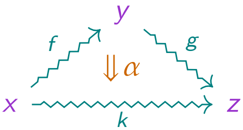
Proof: The space of composites of paths f and g in X is defined by the pullback:



A space is contractible just when any sphere S^{n-1} can be filled to a disk D^n for $n \geq 0$.

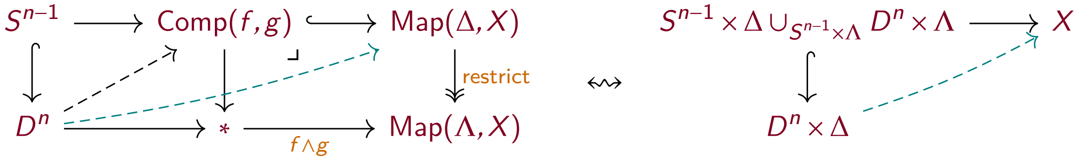
Composing paths

In the total singular complex aka the fundamental ∞-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.

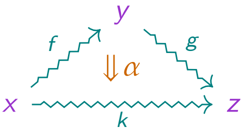
Proof: The space of composites of paths f and g in X is defined by the pullback:



A space is contractible just when any sphere S^{n-1} can be filled to a disk D^n for $n \geq 0$.

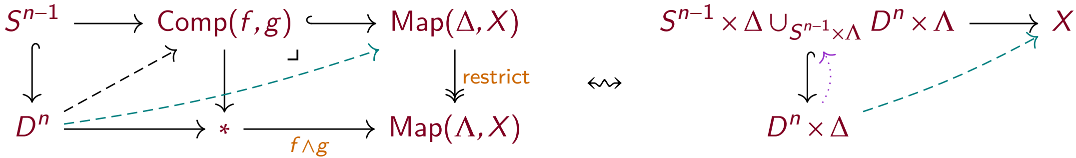
Composing paths

In the total singular complex aka the fundamental ∞-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 X is defined by the pullback:



A space is contractible just when any sphere S^{n-1} can be filled to a disk D^n for $n \geq 0$.
The extension exists since the inclusion admits a continuous deformation retract. □

∞-categories in homotopy type theory

The identity type family gives each type the structure of an ∞-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.

∞-categories in homotopy type theory

The identity type family gives each type the structure of an ∞-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 ∞-categories, Higher Structures 1(1):116–193, 2017

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

∞-categories in homotopy type theory

The identity type family gives each type the structure of an ∞-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 ∞-categories, Higher Structures 1(1):116–193, 2017

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

Definition (Riehl–Shulman). A type A is an ∞-category if:

∞-categories in homotopy type theory

The identity type family gives each type the structure of an ∞-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 ∞-categories, Higher Structures 1(1):116–193, 2017

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

Definition (Riehl–Shulman). A type A is an ∞-category if:

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

∞-categories in homotopy type theory

The identity type family gives each type the structure of an ∞-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 ∞-categories, Higher Structures 1(1):116–193, 2017

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

Definition (Riehl–Shulman). A type A is an ∞-category if:

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

∞-categories in homotopy type theory

The identity type family gives each type the structure of an ∞-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 ∞-categories, Higher Structures 1(1):116–193, 2017

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

Definition (Riehl–Shulman). A type A is an ∞-category if:

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

With more of the work being done by the foundation system, perhaps someday ∞-category theory will be easy enough to teach to undergraduates?

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 ∞-categories
- Jonathan Weinberger, A Synthetic Perspective on $(\infty,1)$-Category Theory: Fibrational and Semantic Aspects, PhD thesis TU Darmstadt 2022.

Danke!