Emily Riehl

Johns Hopkins University

An Introduction to Homotopy Type Theory & Univalent Foundations

Google

Motivation I

CAHIERS DE TOPOLOGIE
ET GÉOMÉTRIE DIFFÉRENTIELLE
CATÉGORIQUES

VOL. XXXII-1 (1991)

∞-GROUPOIDS AND HOMOTOPY TYPES

by M.M. KAPRANOV and V.A. VOEVODSKY

RÉSUMÉ. Nous présentons une description de la categorie homotopique des CW-complexes en termes des ∞-groupoïdes. La possibilité d'une telle description a été suggérée par A. Grothendieck dans son mémoire "A la poursuite des champs".

It is well-known [GZ] that CW-complexes X such that $\pi_i(X,x) = 0$ for all $i \geq 2$, $x \in X$, are described, at the homotopy level, by groupoids. A. Grothendieck suggested, in his unpublished memoir [Gr], that this connection should have a higher-dimensional generalisation involving polycategories. viz. polycategorical analogues of groupoids. It is the purpose of this paper to establish such a generalisation.

- 15 statements =
  4 theorems
  + 9 propositions
  + 1 lemma
  + 1 corollary
- 5 short "obvious" proofs + 3 proofs

- Carlos Simpson's "Homotopy types of strict 3-groupoids" (1998) shows that the 3-type of S^2 can't be realized by a strict 3-groupoid — contradicting the last corollary
- But no explicit mistake was found. Voevodsky: "I was sure that we were right until the fall of 2013 (!!)"

A sociological problem

MATHEMATICS

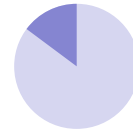
The Origins and Motivations of Univalent Foundations

A Personal Mission to Develop Computer Proof Verification to Avoid Mathematical Mistakes

By Vladimir Voevodsky • Published 2014

"A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail."

Motivation II



The Yoneda lemma. An object of a category is determined up to canonical isomorphism by the network of relationships that the object has with all the other objects in the category.

Corollary. All theorems in category theory.

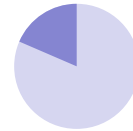
The Yoneda lemma for ordinary 1-categories is proven on:

- page 61/314 of *Categories for the Working Mathematician*
- page 57/240 of *Category Theory in Context*

The Yoneda lemma for ∞-categories is proven on:

- page 269/416 in a series of papers by Riehl–Verity
- page 47/78 of Riehl–Shulman, A type theory for synthetic ∞-categories, Higher Structures 1(1):116–193, 2017.

Motivation III



Why do I study category theory?

— I find category theoretic arguments to be beautiful.

What draws me to homotopy type theory?

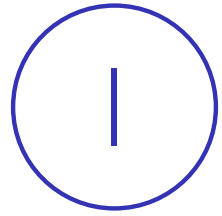
— I find homotopy type theoretic arguments to be beautiful.

Plan

1. Dependent type theory

2. The right way to think about equality

3. Homotopy type theory



Dependent type theory

Dependent type theory

Dependent type theory is:

- a formal system for mathematical constructions and proofs,
- which can be related to the conventional foundations of mathematics,
- but can also function independently.

For the purposes of this talk, it's best to imagine that you're learning what it means "to do mathematics" for the first time.

Alternatively, imagine you are a computer that needs to be taught how to recognize the statements and proofs of mathematical theorems.

Types, terms, and contexts

Dependent type theory has:

- types $\Gamma \vdash A$, $\Gamma \vdash B$
- terms $\Gamma \vdash x : A$, $\Gamma \vdash y : B$
- dependent types $\Gamma, x : A \vdash B(x)$, $\Gamma, x, y : A \vdash C(x, y)$
- dependent terms $\Gamma, x : A \vdash b(x){:}B(x)$, $\Gamma, x, y : A \vdash c(x, y){:}C(x, y)$

Types and terms can be defined in an arbitrary context of variables from previously-defined types, all of which are listed before the symbol "$\vdash$". Here Γ is shorthand for a generic context, which has the form

$$x_1 : A_1, x_2 : A_2(x_1), x_3 : A_3(x_1, x_2), \ldots, x_n : A_n(x_1, \ldots, x_{n-1})$$

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.

Type constructors

Type constructors build new types and terms from given ones:

- products $A \times B$, coproducts $A + B$, function types $A \to B$,
- dependent pairs $\sum_{x:A} B(x)$, dependent functions $\prod_{x:A} B(x)$.

Each type constructor comes with rules:

(i) formation: a way to construct new types

(ii) introduction: ways to construct terms of these types

(iii) elimination: ways to use them to construct other terms

(iv) computation: the way (ii) and (iii) relate

The rules suggest a logical naming for certain types:

$A \times B$	``A and B''	$\sum_{x:A} B(x)$	``$\exists x.B(x)$''
$A + B$	``A or B''	$\prod_{x:A} B(x)$	``$\forall x.B(x)$''
$A \to B$	``A implies B''	$x =_A y$	``x equals y''

Product types and function types

Product types are governed by the rules

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

plus computation rules that relate pairings and projections.

Function types are governed by the rules

- $\to$-form: given types A and B there is a type $A \to B$
- $\to$-intro: if in the context of a variable $x : A$ there is a term $b : B$ there is a term $\lambda x.b : A \to B$
- $\to$-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.

Mathematics in dependent type theory

$\times$-form: $A, B \rightsquigarrow A \times B$	$\to$-form: A and $B \rightsquigarrow A \to B$
$\times$-intro: $a : A, b : B \rightsquigarrow (a, b) : A \times B$	$\to$-intro: $x : A \vdash b : B \rightsquigarrow \lambda x.b : A \to B$
$\times$-elim: $p : A \times B \rightsquigarrow \mathrm{pr}_1 p : A, \mathrm{pr}_2 p : B$	$\to$-elim: $f : A \to B, a : A \rightsquigarrow f(a) : B$

To prove a mathematical proposition in dependent type theory, one constructs a term in the type that encodes its statement.

Prop. For any types A and B, $(A \times (A \to B)) \to B$.

Proof: By $\to$-intro, it suffices to assume given a term $p : (A \times (A \to B))$ and define a term of type B. By $\times$-elim, this provides terms $\mathrm{pr}_1 p : A$ and $\mathrm{pr}_2 p : A \to B$. By $\to$-elim, these combine to give a term $\mathrm{pr}_2 p(\mathrm{pr}_1 p) : B$. Thus we have

$$\lambda p.\mathrm{pr}_2 p(\mathrm{pr}_1 p) : (A \times (A \to B)) \to B. \quad \square$$

The natural numbers type

The natural numbers type is governed by the rules:

$\mathbb{N}$-form: $\mathbb{N}$ exists in the empty context

$\mathbb{N}$-intro: there is a term $0 : \mathbb{N}$ and
for any term $n : \mathbb{N}$ there is a term $\mathrm{succ}(n) : \mathbb{N}$

The elimination rule strengthens the principle of mathematical induction: for any predicate P on $\mathbb{N}$,

$$P(0) \to ((\forall n, P(n) \to P(\mathrm{succ}(n))) \to \forall n, P(n)).$$

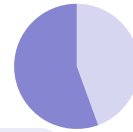
$\mathbb{N}$-elim: for any type family $n : \mathbb{N} \vdash P(n)$, if there are terms $p_0 : P(0)$ and $p_s : \prod_{n:\mathbb{N}} P(n) \to P(\mathrm{succ}(n))$ then there is a term $p : \prod_{n:\mathbb{N}} P(n)$

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

2

The right way to think about equality

What is the correct way to think about equality?



> "The heart and soul of much mathematics consists of the fact that the 'same' object can be presented to us in different ways."
> — Barry Mazur "When is one thing equal to some other thing?"

Not what it is but what it does.

Principle of substitution. To prove that every x, y with $x = y$ have property P, it suffices to:

- Prove that every pair x, x (for which $x = x$) has property P.

Example.

- To prove symmetry — for all x, y if $x = y$ then $y = x$ — apply the principle of substitution to the property $P(x, y) := y = x$. Now it's enough to show that for all x then $x = x$, which is true by reflexivity.
- Transitivity — for all x, y, z if $x = y$ and $y = z$ then $x = z$ — can be deduced similarly by applying the principle of substitution to $Q(x, y) := \forall z, (y = z \to x = z)$.

Identity types

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

$=$-form: given a type A and terms $x, y : A$, there is a type $x =_A y$

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

The elimination rule for the identity type is an enhanced version of the principle of substitution: to prove that every x, y with $x = y$ have property P, it suffices to prove that every pair x, x has property P.

$=$-elim: If $P(x, y, p)$ is a type family dependent on $x, y : A$ and $p : x =_A y$, then to prove $P(x, y, p)$ it suffices to assume y is x and p is refl_x.

Note that 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?

3

Homotopy type theory

Homotopy type theory

Homotopy type theory is:

- a formal system for mathematical constructions and proofs,
- in which the basic objects, types, may be regarded as "spaces" or ∞-groupoids,
- and all constructions are automatically "continuous" or equivalence-invariant.

Homotopy type theory is $\begin{cases} \text{homotopy (type theory)} \\ \text{(homotopy type) theory} \end{cases}$

Types A can be regarded simultaneously as both mathematical constructions and mathematical assertions; accordingly, a term $a : A$ can be regarded as a proof of the proposition A. But it's somewhat misleading to think of propositions as types, because types may have non-trivial higher dimensional structure.

The Curry-Howard-Voevodsky correspondence

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

Path induction

Now that terms $p : x =_A y$ are called paths, we rebrand $=$-elim as:

Path induction. If $x, y : A, p : x =_A y \vdash P(x, y, p)$ is a type family then to prove $P(x, y, p)$ it suffices to assume y is x and p is refl_x. I.e.:

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

By path induction, paths can be reversed and concatenated:

$$(-)^{-1} : x =_A y \to y =_A x \qquad * : x =_A y \to (y =_A z \to x =_A z).$$

To define both terms, we may assume $p : x =_A y$ and then define terms in the types $P(x, y, p) :\equiv y =_A x$ and $Q(x, y, p) :\equiv y =_A z \to x =_A z$. By path induction, we may reduce to the cases $P(x, x, \mathrm{refl}_x) :\equiv x =_A x$ and $Q(x, x, \mathrm{refl}_x) :\equiv x =_A z \to x =_A z$, for which we have terms $\mathrm{refl}_x : x =_A x$ and $\lambda q.q : x =_A z \to x =_A z$.

The ∞-groupoid of paths

Theorem (Lumsdaine,Garner–van den Berg). 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
- paths $p : x =_A y$ as 1-morphisms
- paths of paths $h : p =_{x=_A y} q$ as 2-morphisms, ...

The required structures are proven from the path induction principle:

- constant paths (reflexivity) $\mathrm{refl}_x : x = x$
- 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 ...

Contractible types

The homotopical perspective on type theory suggests new definitions:

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

$$\sum_{a:A} \prod_{x:A} a =_A x$$

By Σ-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.

The contracting homotopy 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

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

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

A type A

- is a proposition if

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

- is a set or 0-type if

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

- is an succ(n)-type for $n : \mathbb{N}$ if

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

Equivalences

Similarly, homotopy theory suggests 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:

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

A term of type $A \simeq B$ provides functions $f : A \to B$, $g, h : B \to A$ and homotopies α and β relating the composite functions $g \circ f$ and $f \circ h$ to the identities. 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, C as its terms.

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

By path induction, there is a canonical function

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

defined by sending refl_A to the identity equivalence id_A.

The univalence axiom, which is justified by the homotopical model of type theory, asserts that id-to-equiv is an equivalence.

"Identity is equivalent to equivalence."

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

This axiom justifies the common mathematical practice of applying results proven about one object to any other object that is equivalent to it.

Consequences of univalence

There are myriad consequences of the univalence axiom:

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

- the structure-identity principle, which for set-based structures (monoids, groups, rings) says that isomorphic structures are identical
- function extensionality, which says that for any $f, g : A \to B$, the canonical function defines an equivalence

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

between the identity type and the type of homotopies

Homotopy type theory satisfies the principle of substitution: if $x, y : A$ and $x =_A y$ then $P(x) \simeq P(y)$ for any family of types $a : A \vdash P(a)$.

Thus, by univalence, whenever $A \simeq B$, any type constructed from A is equivalent to the corresponding type constructed from B.

References

Egbert Rijke, Introduction to Homotopy Type Theory,

- for the current draft: `hott.zulipchat.com`
- agda formalization: `github.com/HoTT-Intro/Agda`

Homotopy Type Theory: Univalent Foundations of Mathematics,
`https://homotopytypetheory.org/book/`

Michael Shulman, Homotopy type theory: the logic of space,
`arXiv:1703.03007`

Thank you!