Higher Observational Type Theory

Michael Shulman

University of San Diego

j.w.w. Thorsten Altenkirch, Ambrus Kaposi, and Elif Uskuplu

Texas State University
April 21, 2025

Outline

1. From set theory to type theory
2. From type theory to HOTT
3. From HOTT to homotopy theory

What is this all about?

Higher observational type theory is a new formal framework for mathematics that

- Can represent all existing mathematics
- Is implementable in a computer proof assistant
- Is arguably more faithful to what mathematicians actually do
- Provides easier access to higher structure tools when needed
- Can be explained and justified intuitively

Set theory

The usual framework is set theory, according to which:

- All objects are sets.
- Numbers are built out of sets:

$$0 = \emptyset,\ 1 = \{0\},\ 2 = \{0, 1\},\ 3 = \{0, 1, 2\},\ \ldots$$

- Ordered pairs are built out of sets:

$$(a, b) = \{\{a\}, \{a, b\}\}.$$

- Functions are built out of sets:

$$f = \{(a, b) \mid b = f(a)\}$$

- Sets are equal when they have the same elements:

$$A = B \iff (\forall x \in A, x \in B) \text{ and } (\forall x \in B, x \in A).$$

Maybe not set theory?

These "definitions" of numbers, pairs, and functions are encodings in set theory, not unavoidable mandates.

- Instead of $0 = \emptyset,\ 1 = \{0\},\ 2 = \{0, 1\},\ 3 = \{0, 1, 2\},\ \ldots$ we could as well use $0 = \emptyset,\ 1 = \{0\},\ 2 = \{1\},\ 3 = \{2\},\ \ldots$
- A "working mathematician" doesn't care that $(a, b) = \{\{a\}, \{a, b\}\}$, only that ordered pairs behave correctly.

In addition to encoding things differently in set theory, we could also encode them in a different formal framework.

- Analogy: A high-level programming language can be compiled to many different machine-language architectures.

Why not set theory?

The problem with set theory is its untyped nature: everything is the same kind of thing (a set).

Examples

- We can ask meaningless questions like "is $2 \in \pi$?"
- We can make meaningless definitions like "a group is nice if its identity element is $\emptyset$".

Experienced human mathematicians simply ignore these possibilities, but students and computers can get tripped up on them.

We can design a computer proof assistant based on set theory, but the superstructure that makes it usable by humans requires a typed layer anyway, to infer missing information and detect mistakes. So why not use a typed framework in the first place?

- Analogy: A typed language like Java or Python is more practical for humans than untyped assembly language.

Why not set theory, II

More importantly, the uniform nature of equality in set theory does not match mathematical practice. In practice, we define equality separately for different kinds of objects:

- Two ordered pairs are equal if their components are:

$$(a, b) = (c, d) \in A \times B \iff a = c \text{ and } b = d.$$

- Two functions are equal if they take equal values:

$$f = g \in A \to B \iff \forall x \in A, f(x) = g(x).$$

- Two fractions are equal if their cross-multiplications are:

$$\tfrac{a}{b} = \tfrac{p}{q} \iff aq = bp$$

Why not set theory, III

We can get by, up to a point, by defining things carefully so that the uniform notion of equality for sets specializes to what we want:

- Defining $(a, b) = \{\{a\}, \{a, b\}\}$, since we can prove

$$\{\{a\}, \{a, b\}\} = \{\{c\}, \{c, d\}\} \iff a = c \text{ and } b = d.$$

- Defining functions as sets of ordered pairs.
- Defining fractions as equivalence classes:

$$\tfrac{p}{q} = \left\{(a, b) \;\middle|\; aq = bp\right\}$$

However:

- this is awkward, especially for students; and
- it eventually breaks down...

Sameness for structures

When are two groups equal?

Sameness for structures

When are two groups equal?

I claim the only sensible answer is "when they are isomorphic".

Stick a pin in your abstract algebra teacher's exhortations to distinguish isomorphism from equality, and hear me out.

Sameness for structures

When are two groups equal?

I claim the only sensible answer is “when they are isomorphic”.

Stick a pin in your abstract algebra teacher’s exhortations to distinguish isomorphism from equality, and hear me out.

- Isomorphic groups share all the same group-theoretic properties.

Sameness for structures

When are two groups equal?

I claim the only sensible answer is “when they are isomorphic”.

Stick a pin in your abstract algebra teacher’s exhortations to distinguish isomorphism from equality, and hear me out.

- Isomorphic groups share all the same group-theoretic properties.
- I claim this is completely analogous to how:
  - $\frac{1}{2}$ and $\frac{2}{4}$ share all the same numerical properties.
  - $\{x, y\}$ and $\{y, x\}$ share all the same set-theoretic properties.
  - $f(x) = (x+1)^2$ and $f(x) = x^2 + 2x + 1$ share all the same functional properties.

Sameness for structures

When are two groups equal?

I claim the only sensible answer is "when they are isomorphic".

Stick a pin in your abstract algebra teacher's exhortations to distinguish isomorphism from equality, and hear me out.

- Isomorphic groups share all the same group-theoretic properties.
- I claim this is completely analogous to how:
  - $\frac{1}{2}$ and $\frac{2}{4}$ share all the same numerical properties.
  - $\{x, y\}$ and $\{y, x\}$ share all the same set-theoretic properties.
  - $f(x) = (x+1)^2$ and $f(x) = x^2 + 2x + 1$ share all the same functional properties.
- Mathematical equality is extensional, not intensional: it doesn't matter how something is defined, only how it behaves.
  - $\frac{1}{2}$ and $\frac{2}{4}$ are different presentations of the same number.
  - $\mathbb{Z}_{/2}$ and S_2 as different presentations of the same group.

Sameness for structures

When are two groups equal?

I claim the only sensible answer is "when they are isomorphic".

Stick a pin in your abstract algebra teacher's exhortations to distinguish isomorphism from equality, and hear me out.

- Isomorphic groups share all the same group-theoretic properties.
- I claim this is completely analogous to how:
  - $\frac{1}{2}$ and $\frac{2}{4}$ share all the same numerical properties.
  - $\{x, y\}$ and $\{y, x\}$ share all the same set-theoretic properties.
  - $f(x) = (x+1)^2$ and $f(x) = x^2 + 2x + 1$ share all the same functional properties.
- Mathematical equality is extensional, not intensional: it doesn't matter how something is defined, only how it behaves.
  - $\frac{1}{2}$ and $\frac{2}{4}$ are different presentations of the same number.
  - $\mathbb{Z}_{/2}$ and S_2 as different presentations of the same group.
- In practice, mathematicians blithely replace groups by isomorphic ones all the time and think nothing of it.

So what's the problem?

In set theory, the only way we could get isomorphic groups to be equal would be to use equivalence classes, like for fractions:

$$[S_2] = \left\{ G \;\middle|\; S_2 \cong G \right\}$$

But in this case, that would destroy too much information.

Example

What is a homomorphism between isomorphism classes, $[G] \to [H]$?

- Should be something to do with a homomorphism $G \to H$.
- But if $[H] = [H']$, when should $\varphi : G \to H$ equal $\varphi' : G \to H'$ as homomorphisms $[G] \to [H] = [H']$?
- If we pick some $\psi : H \cong H'$, we can ask if $\varphi' = \psi \circ \varphi$. But the answer depends on ψ.
- Worse, how can we compose $\varphi : G \to H$ with $\theta : H' \to K$? The obvious $\theta \circ \psi \circ \varphi$ also depends on the choice of ψ.

Towards type theory

Type theory is an alternative framework for mathematics.

- Basic objects called types act (mostly) like sets.
- Each type A has elements, written $x : A$.
- Each element has a unique type that is intrinsic to its nature. We never "prove" that $x : A$; we can't ever *have* an x without knowing A. No two distinct types share any elements.*

Examples

- If A and B are types, then $A \times B$ is a type whose elements are pairs (a, b) where $a : A$ and $b : B$. They are not defined in terms of anything else; they are primitive objects.
- If A and B are types, then $A \to B$ is a type whose elements are functions from A to B. These are also primitive objects. (The notation $f : A \to B$ should be familiar.)
- $\mathbb{N}$ is a type whose elements are $0, 1, 2, 3, \ldots$.

But I need sets!

We can still introduce sets as long as they are local and well-typed.

If A is a type, then $\mathcal{P}A$ is a type whose elements are subsets of A.

- For $a : A$ and $X : \mathcal{P}A$ we can prove or disprove $a \in X$.
- For $X : \mathcal{P}A$ and $Y : \mathcal{P}A$ can prove or disprove $X \subseteq Y$.
- All the elements of a set have the same type.
- Can't compare two sets whose elements have different types.

Examples

- We can use $\mathcal{P}A$ to define point-set topologies on A.
- We can use $\mathcal{P}\mathbb{Q}$ to define real numbers as Dedekind cuts.

But I need sets of sets!

Well, $\mathcal{P}\mathcal{P}A$ contains sets of subsets of A (e.g. ultrafilters on A).

But often what you need instead is:

- There is a type $\mathcal{U}$ (a universe) whose elements are types[1].

For example, an I-indexed family of types is a function $B : I \to \mathcal{U}$.
Then we can talk about its product and its disjoint union:

$$\prod_{i:I} B(i) \qquad \coprod_{i:I} B(i).$$

- $\prod_{i:I} B(i)$ contains functions f such that $f(i) : B(i)$ for all $i : I$.
- $\coprod_{i:I} B(i)$ contains pairs (i, b) where $i : I$ and $b : B(i)$.

[1] Though not all of them, for Russellian paradox reasons.

But I need subsets as types!

Often we want to treat some subset of a type as a new type in its own right, e.g.

$$\mathbb{S}^1 = \{(x, y) : \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 = 1\}$$

Given a subset $B : \mathcal{P}A$, let $\chi_B : A \to \mathcal{U}$ be its type-valued characteristic function:

$$\chi_B(a) = \begin{cases} \{*\} & \text{if } a \in B \\ \emptyset & \text{if } a \notin B. \end{cases}$$

Then the type $\coprod_{a:A} \chi_B(a)$ consists of pairs (a, z) where $a : A$ and $z : \chi_B(a)$, or equivalently elements $a : A$ such that $a \in B$.

Type theory and computation

Type theories are also programming languages.

- Makes it easier to implement proof assistants: verifying a proof is the same as typechecking a program.
- If a proof is constructive (doesn't use excluded middle and choice), it can be executed as a program.

Nearly all modern proof assistants are based on type theories: Rocq, Agda, HOL, Isabelle, Lean, . . .

Outline

1. From set theory to type theory
2. From type theory to HOTT
3. From HOTT to homotopy theory

So far so good

Recall: equality in set theory is uniform, but we want to define equality separately for each type. However:

- Martin-Löf's original type theory (1972) defined equality uniformly! Each type has a separate equality, but all defined the same way, as the "smallest reflexive binary relation". This is underspecified; it can be proven to behave correctly on some types, but not all.
- "Book" homotopy type theory ("Book HoTT") postulates that Martin-Löf's equality behaves appropriately on the other types (function-types, infinite products, and the universe).
  (Hofmann–Streicher 1998, Awodey–Warren and Voevodsky 2009)
- "Cubical type theory" introduces an equality that can be proven to behave appropriately on each type, but is still defined uniformly.
  (Cohen–Coquand–Huber–Mörtberg 2015, Angiuli–Brunerie–Coquand–Favonia–Harper–Licata 2021)
- Higher observational type theory (HOTT) finally introduces an equality that is defined separately for each type.
  (Altenkirch–Kaposi–Shulman–Uskuplu 2025+)

I will explain HOTT intuitively based on five simple principles.

Identity types

As type theorists, we like to work with types. Thus, rather than axiomatize the relation "$x = y \in A$", we will axiomatize its type-valued characteristic function. This gives us the

First principle of equality

For any type A and any $x, y : A$, there is an identity type $\mathrm{Id}_A(x, y)$. That is, $\mathrm{Id}_A : A \times A \to \mathcal{U}$. We define Id_A separately for each A.

At this point, we can think intuitively of

$$\mathrm{Id}_A(x, y) = \begin{cases} \{*\} & \text{if } x = y \\ \emptyset & \text{if } x \neq y \end{cases}$$

Defining identity types

One immediate nice consequence is that we can define the identity types of most types to be another type of the same kind.

Example

$$\mathrm{Id}_{A\times B}((a_0,b_0),(a_1,b_1)) = \mathrm{Id}_A(a_0,a_1)\times \mathrm{Id}_B(b_0,b_1).$$

This is the type-valued-characteristic-function way of saying that

$$(a_0,b_0)=(a_1,b_1) \iff a_0=a_1 \text{ and } b_0=b_1.$$

Example

$$\mathrm{Id}_{\coprod_{i:I} B(i)}((i_0,b_0),(i_1,b_1)) = \coprod_{i_2:\mathrm{Id}_I(i_0,i_1)} \mathrm{Id}_B(b_0,b_1).$$

"Two elements of a disjoint union are equal iff they come from the same summand and are equal* there."

Properties of identity types

Equality ought to satisfy:

- Reflexivity: $x = x$.
- Symmetry: if $x = y$ then $y = x$.
- Transitivity: if $x = y$ and $y = z$ then $x = z$.
- Congruence: if $f : A \to B$ and $x = y \in A$, then $f(x) = f(y) \in B$.
- Substitution: if $x = y$ and $P(x)$ is true, then $P(y)$ is true.

The first is important enough to be the

Second principle of equality

For any type A and any element $x : A$, there is a reflexivity element $\mathsf{refl}_x : \mathsf{Id}_A(x, x)$. We define refl_x separately for each x.

The definitions of refl match those of Id, e.g. $\mathsf{refl}_{(a,b)} = (\mathsf{refl}_a, \mathsf{refl}_b)$.

Congruence

We'll come back to symmetry, transitivity, and substitution later. But congruence can be generalized to the

Third principle of equality

All constructions respect equality: if their inputs are all replaced by something equal, the outputs will also be equal. We prove/define this separately for each construction.

It suffices to prove/define this for the primitive constructions that come with each type, out of which which everything is built.

- Pairing: given $a : A$ and $b : B$, we have $(a, b) : A \times B$.
  - Given $a_2 : \mathsf{Id}_A(a_0, a_1)$ and $b_2 : \mathsf{Id}_B(b_0, b_1)$, we have $(a_2, b_2) : \mathsf{Id}_{A\times B}((a_0, b_0), (a_1, b_1))$.
- Projection: given $p : A \times B$, we have $\pi_A(p) : A$ and $\pi_B(p) : B$.
  - Given $p_2 : \mathsf{Id}_{A\times B}(p_0, p_1)$, we have $\pi_A(p_2) : \mathsf{Id}_A(\pi_A(p_0), \pi_A(p_1))$ and $\pi_B(p_2) : \mathsf{Id}_B(\pi_B(p_0), \pi_B(p_1))$.

Identity of functions

One of the primitive constructions of the function-type $A \to B$ is:

- Application: given $f : A \to B$ and $a : A$, we have $f(a) : B$.

So we need

- Given $f_2 : \mathsf{Id}_{A\to B}(f_0, f_1)$ and $a_2 : \mathsf{Id}_A(a_0, a_1)$ we have $f_2(a_2) : \mathsf{Id}_B(f_0(a_0), f_1(a_1))$.

This means we have to define

$$\mathsf{Id}_{A\to B}(f_0, f_1) = \prod_{a_0, a_1 : A} \Big(\mathsf{Id}_A(a_0, a_1) \to \mathsf{Id}_B(f_0(a_0), f_1(a_1))\Big).$$

"Two functions are equal if they jointly map
equal pairs of elements to equal pairs of elements."

This is equivalent to $\prod_{a:A} \mathsf{Id}_B(f_0(a), f_1(a))$, but more convenient.
In particular, it gives us the usual congruence rule:

$$\mathsf{refl}_f : \prod_{a_0, a_1 : A} \Big(\mathsf{Id}_A(a_0, a_1) \to \mathsf{Id}_B(f(a_0), f(a_1))\Big).$$

Identity of types

Recall the type $\mathcal{U}$ whose elements are types. What is $\mathsf{Id}_{\mathcal{U}}(A, B)$?

Remember in set theory we have

$$A = B \iff (\forall x \in A, x \in B) \text{ and } (\forall x \in B, x \in A).$$

How close can we come to importing this into type theory?

1. An element of one type A can't itself also be an element of another type B. So as a first step, let's rewrite this as

$$(\forall x \in A, \exists y \in B, x = y) \text{ and } (\forall y \in B, \exists x \in A, x = y).$$

2. An element of one type also can't be equal to an element of another type, so we replace equality by some other relation:

$$(\forall x \in A, \exists y \in B, x \sim y) \text{ and } (\forall y \in B, \exists x \in A, x \sim y).$$

Heterogeneous equality

Thus, just as each type separately comes with an equality relation on its elements, each equality of types comes with an "equality relation" relating elements of the two types. As always, we represent this by its type-valued characteristic function, giving the

Fourth principle of equality

Any $E : \mathsf{Id}_{\mathcal{U}}(A, B)$ gives rise to:

- A function $E : A \times B \to \mathcal{U}$.
- For any $a : A$, an element $\overrightarrow{E}(a) : B$.
- For any $a : A$, an element $\overrightarrow{\overrightarrow{E}}(a) : E(a, \overrightarrow{E}(a))$.
- For any $b : B$, an element $\overleftarrow{E}(b) : A$.
- For any $b : B$, an element $\overleftarrow{\overleftarrow{E}}(b) : E(\overleftarrow{E}(b), b)$.

Moreover, for $A : \mathcal{U}$, we have $\mathsf{refl}_A(a_0, a_1) = \mathsf{Id}_A(a_0, a_1)$.

We can think of $E(a, b)$ as saying a and b are equal modulo E.

Example: rational numbers

An equality of types gives "two presentations of the same notion".

Example

- $\mathbb{Q}_{\mathsf{f}}$ = integer fractions $\frac{1}{2}$, $-\frac{4}{3}$, $\frac{3}{7}$, ...
- $\mathbb{Q}_{\mathsf{d}}$ = finite or repeating decimals 0.5, $-1.\overline{3}$, $0.\overline{428571}$,...

These are two representations of the rational numbers: we have

$$E : \mathsf{Id}_{\mathcal{U}}(\mathbb{Q}_{\mathsf{f}}, \mathbb{Q}_{\mathsf{d}})$$

recording which elements of $\mathbb{Q}_{\mathsf{f}}$ and $\mathbb{Q}_{\mathsf{d}}$ correspond to each other:

$$E\left(\tfrac{1}{2}, 0.5\right) \qquad E\left(-\tfrac{4}{3}, -1.\overline{3}\right) \qquad E\left(\tfrac{3}{7}, 0.\overline{428571}\right)$$

The other four pieces of E say that every fraction corresponds to some decimal, and every decimal corresponds to some fraction.

The missing properties of equality

These now come for free!

- Substitution: Represent a property of elements of A by its type-valued characteristic function $P : A \to \mathcal{U}$.

  If $a_2 : \mathrm{Id}_A(a_0, a_1)$, then $\mathrm{refl}_P(a_2) : \mathrm{Id}_{\mathcal{U}}(P(a_0), P(a_1))$, so

  $$\overrightarrow{\mathrm{refl}_P(a_2)} : P(a_0) \to P(a_1).$$

- Symmetry: Given $x : A$, we have $\mathrm{Id}_A(-, x) : A \to \mathcal{U}$.

  If $e : \mathrm{Id}_A(x, y)$, then $\mathrm{refl}_{\mathrm{Id}_A(-,x)}(e) : \mathrm{Id}_{\mathcal{U}}(\mathrm{Id}_A(x, x), \mathrm{Id}_A(y, x))$, so

  $$\overrightarrow{\mathrm{refl}_{\mathrm{Id}_A(-,x)}(e)}(\mathrm{refl}_x) : \mathrm{Id}_A(y, x).$$

- Transitivity: Given $z : A$, we have $\mathrm{Id}_A(x, -) : A \to \mathcal{U}$.

  If $e : \mathrm{Id}_A(y, z)$, then $\mathrm{refl}_{\mathrm{Id}_A(x,-)}(e) : \mathrm{Id}_{\mathcal{U}}(\mathrm{Id}_A(x, y), \mathrm{Id}_A(x, z))$, so

  $$\overrightarrow{\mathrm{refl}_{\mathrm{Id}_A(x,-)}(e)} : \mathrm{Id}_A(x, y) \to \mathrm{Id}_A(x, z).$$

The missing definitions of equality

Example

$$¿ \quad \mathsf{Id}_{\coprod_{i:I} B(i)}((i_0, b_0), (i_1, b_1)) = \coprod_{i_2 : \mathsf{Id}_I(i_0, i_1)} \mathsf{Id}_B(b_0, b_1) \quad ?$$

This doesn't quite make sense, since $b_0 : B(i_0)$ and $b_1 : B(i_1)$, and B isn't a single type. The correct thing to write is

$$\mathsf{Id}_{\coprod_{i:I} B(i)}((i_0, b_0), (i_1, b_1)) = \coprod_{i_2 : \mathsf{Id}_I(i_0, i_1)} \mathsf{refl}_B(i_2)(b_0, b_1).$$

Example

$$\mathsf{Id}_{\prod_{i:I} B(i)}(f_0, f_1) = \prod_{i_0, i_1 : I} \prod_{i_2 : \mathsf{Id}_I(i_0, i_1)} \mathsf{refl}_B(i_2)(f_0(i_0), f_1(i_1)).$$

Is something else missing?

Given $E : \mathsf{Id}_{\mathcal{U}}(A, B)$, we have functions

$$\overrightarrow{E} : A \to B \qquad \qquad \overleftarrow{E} : B \to A.$$

You may think we should require these to be inverses.

But in fact this also follows from our four principles already!

Type equalities are isomorphisms

Theorem

Given $E : \mathrm{Id}_{\mathcal{U}}(A, B)$, the functions $\overrightarrow{E}$ and $\overleftarrow{E}$ are inverses.

Proof.

1. For any type X and $x_0 : X$ and $x_1 : X$, there is a type $\mathrm{Id}_X(x_0, x_1)$.
3. All constructions respect equality.

Therefore,

- For any $E : \mathrm{Id}_{\mathcal{U}}(A, B)$ and $e_0 : E(a_0, b_0)$ and $e_1 : E(a_1, b_1)$, we have $\mathrm{Id}_E(e_0, e_1) : \mathrm{Id}_{\mathcal{U}}(\mathrm{Id}_A(a_0, a_1), \mathrm{Id}_B(b_0, b_1))$.

In particular, since

$$\overrightarrow{\overrightarrow{E}}(a) : E\left(a, \overrightarrow{E}(a)\right) \qquad \overleftarrow{\overleftarrow{E}}\left(\overrightarrow{E}(a)\right) : E\left(\overleftarrow{E}\left(\overrightarrow{E}(a)\right), \overrightarrow{E}(a)\right)$$

we have $\overleftarrow{\mathrm{Id}_E(e_0, e_1)}(\mathrm{refl}_{\overrightarrow{E}(a)}) : \mathrm{Id}_A\left(a, \overleftarrow{E}\left(\overrightarrow{E}(a)\right)\right)$ (and dually). □

Isomorphisms are type equalities

Conversely:

Theorem

Given an isomorphism of types $f : A \rightleftarrows B : g$, if we define

$$E(a,b) = \mathsf{Id}_B(f(a), b)$$

we can construct all the other necessary data for $\mathsf{Id}_{\mathcal{U}}(A,B)$, including the congruence of Id.

Corollary (Voevodsky's Univalence Principle)

$\mathsf{Id}_{\mathcal{U}}(A,B)$ is equivalent to the type of isomorphisms $A \cong B$.

In particular, $\mathsf{Id}_{\mathcal{U}}(A,B)$ is not just the characteristic function of some relation. There could be more than one isomorphism $A \cong B$, and hence more than one element of $\mathsf{Id}_{\mathcal{U}}(A,B)$.

Equality for structures

It may seem weird for A and B to be "equal in more than one way". But this is a virtue: compared to the equality proposition of set theory, the identity type can represent wider notions of "sameness".

Example

Let Group denote the type of groups, i.e. tuples $\mathbf{G} = (G, m, e, i, \dots)$ where $G : \mathcal{U}$ and $m : G \times G \to G$ and $e : G$ and so on.
Then $\mathsf{Id}_{\mathsf{Group}}(\mathbf{G}, \mathbf{H})$ is the type of group isomorphisms $\mathbf{G} \cong \mathbf{H}$.

The problems we had in set theory go away:

- Equality is defined correctly, not encoded as set-equality with isomorphism classes, so we don't need to write $[\mathbf{G}]$.
- We define the type $\mathsf{Hom}(\mathbf{G}, \mathbf{H})$ of isomorphisms as usual. If $e : \mathsf{Id}_{\mathsf{Group}}(\mathbf{H}, \mathbf{H}')$, we get $\mathsf{Id}_{\mathsf{Group}}(\mathsf{Hom}(\mathbf{G}, \mathbf{H}), \mathsf{Hom}(\mathbf{G}, \mathbf{H}'))$, but what we get (hence which φ correspond to which φ') depends on e, and similarly for composition.

The structure identity principle

Moreover, we can no longer ask meaningless questions:

Theorem

For any property P of groups, if $P(\mathbf{G})$ and $\mathbf{G} \cong \mathbf{H}$, then $P(\mathbf{H})$.

Proof.

Represent P by its characteristic function $P : \mathsf{Group} \to \mathcal{U}$. Since $\mathbf{G} \cong \mathbf{H}$, we have $E : \mathsf{Id}_{\mathsf{Group}}(\mathbf{G}, \mathbf{H})$, hence $P(\mathbf{G}) \to P(\mathbf{H})$. □

The mathematician's habit of replacing groups by isomorphic ones is now fully rigorous, and not just for groups but all structures.

Outline

1. From set theory to type theory
2. From type theory to HOTT
3. From HOTT to homotopy theory

Groupoids

Similarly:

- $\mathsf{Id}_{\mathsf{Top}}(\mathbf{X},\mathbf{Y})$ consists of homeomorphisms $\mathbf{X} \cong \mathbf{Y}$.
- $\mathsf{Id}_{\mathsf{Vect}}(\mathbf{V},\mathbf{W})$ consists of linear isomorphisms $\mathbf{V} \cong \mathbf{W}$.
- $\mathsf{Id}_{\mathsf{Manifold}}(\mathbf{M},\mathbf{N})$ consists of diffeomorphisms $\mathbf{M} \cong \mathbf{N}$.

  $\vdots$

Here, transitivity of equality becomes composition of isomorphisms:

$$\mathsf{Id}_A(x,y) \times \mathsf{Id}_A(y,z) \to \mathsf{Id}_A(x,z)$$

Just like we could prove transitivity from our basic principles, we can prove this is associative and has identities and inverses.

Thus, the types Top, Vect, ... act like groupoids rather than sets.

Anima

More generally, we can prove that any type A comes with

- Elements $a : A$
- "Equalities" or "isomorphisms" or "paths" $a_2 : \mathsf{Id}_A(a_0, a_1)$.
- "2-equalities" or "2-morphisms" $a_{12} : \mathsf{Id}_{\mathsf{Id}_A(a_{00}, a_{01})}(a_{10}, a_{11})$.
- "3-morphisms" $a_{22} : \mathsf{Id}_{\mathsf{Id}_{\mathsf{Id}_A(a_{00}, a_{01})}(a_{10}, a_{11})}(a_{20}, a_{21})$
- and so on. . .

with composition, identities, associativity, etc., at all levels, coherently up to equalities at all higher levels.

This structure is called an ∞-groupoid or homotopy type or anima.

These are notoriously fiddly even to define in set theory, but in HOTT we don't have to define them: our simple principles automatically give every type this structure.

The homotopy hierarchy

Usually, all that higher structure can be ignored.

- -1 A type A is a proposition or (-1)-type if it has at most one element, i.e. any two elements are equal: $\prod_{x,y:A} \mathsf{Id}_A(x,y)$. This implies all the higher Id-types are trivial also.
- 0 A type A is a set or 0-type if all its identity types are propositions: $\prod_{x,y:A} \prod_{p,q:\mathsf{Id}_A(x,y)} \mathsf{Id}_{\mathsf{Id}_A(x,y)}(p,q)$.
  This is the case in which Id_A is the type-valued characteristic function of some relation.
- 1 A is a 1-type if all its identity types are 0-types.
- 2 A is a 2-type if all its identity types are 1-types. . . .

Most mathematics uses 0-types, like $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$.

But the higher types are there, "waiting in the wings" until we need them. They arise naturally, e.g., in category theory: the type of groups is a 1-type, the type of categories is a 2-type, etc.

Quotient types

Given a type A and a family $R : A \times A \to \mathcal{U}$, we can build a type A/R freely generated by A and such that R becomes equality.

$$[-] : A \to A/R \qquad \llbracket - \rrbracket : R(a_0, a_1) \to \mathsf{Id}_{A/R}([a_0], [a_1]).$$

For any $f : A \to B$ with maps $R(a_0, a_1) \to \mathsf{Id}_B(f(a_0), f(a_1))$, there is a unique compatible $g : A/R \to B$.

If A is a 0-type, R is the type-valued characteristic function of an equivalence relation, and we also force A/R to be a 0-type, we get an encoding of quotient sets.

Higher types

If we don't force A/R to be a 0-type, it isn't generally one.

Example

Suppose $A = \{a\}$ and $R(a, a) = \{r\}$. Then a map $g : A/R \to B$ is uniquely determined by

$$f : \{a\} \to B \qquad \text{and} \qquad \{r\} \to \mathsf{Id}_B(f(a), f(a));$$

that is, by

$$b : B \qquad \text{and} \qquad \ell : \mathsf{Id}_B(b, b).$$

For instance, a map $A/R \to \mathsf{Group}$ is determined by a group G and an automorphism $G \cong G$.

The circle

This type A/R, for $A = \{a\}$ and $R(a,a) = \{r\}$, is called the homotopy-theoretic circle S^1.

- It contains $[\![r]\!] : \mathrm{Id}_{A/R}([a],[a])$, not equal to $\mathrm{refl}_{[a]}$.
- And also $[\![r]\!] \circ [\![r]\!]$, and $[\![r]\!]^{-1}$, etc.

Theorem

$\mathrm{Id}_{A/R}([a],[a]) \cong \mathbb{Z}$.

Sketch of proof.

1. Define $C : A/R \to \mathscr{U}$ by $C([a]) = \mathbb{Z}$ and $\mathrm{refl}_C([\![r]\!]) = S$, where $S : \mathrm{Id}_{\mathscr{U}}(\mathbb{Z},\mathbb{Z})$ is the successor isomorphism $n \mapsto n+1$.
2. Given $p : \mathrm{Id}_{A/R}([a],[a])$, we have $\overrightarrow{\mathrm{refl}_C(p)}(0) : \mathbb{Z}$.
3. Given $n : \mathbb{Z}$, we have $[\![r]\!]^n : \mathrm{Id}_{A/R}([a],[a])$.
4. We can extend these to $\mathrm{Id}_{A/R}([a],x) \cong C(x)$, hence in particular $\mathrm{Id}_{A/R}([a],[a]) \cong \mathbb{Z}$. □

Synthetic homotopy theory

We have analogues of the basic objects and theorems of homotopy theory, a.k.a. ∞-groupoid theory:

- Spheres S^n for all $n : \mathbb{N}$
- Homotopy groups $\pi_n(X)$.
- Homology groups $H_n(X)$ and cohomology groups $H^n(X)$.
- Fibrations, long exact sequences, spectral sequences, cup products, Steenrod operations, ...
- $\pi_n(S^n) = \mathbb{Z}$, $\pi_3(S^2) = \mathbb{Z}$, $\pi_4(S^3) = \mathbb{Z}_{/2}$, ...
- Freudenthal suspension theorem, Blakers–Massey theorem, ...

Thus our simple basic principles, which are (I claim) the most logical way to implement typed equality in a formal framework, ineluctably lead to all the structure of homotopy theory.

Homotopy theory is implicit in the concept of equality!

The anima-tion of mathematics

. . . after Cantor and Bourbaki . . . set theoretic mathematics resides in our brains. When I first start talking about something, I explain it in terms of Bourbaki-like structures . . . we start with the discrete sets of Cantor, upon which we impose something more in the style of Bourbaki.

But fundamental psychological changes also occur. . . . the place of old forms and structures . . . is taken by some geometric, right-brain objects.

. . . 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. . .

From "Interview with Yuri Manin" (by Mikhail Gelfand), AMS Notices, October 2009

HOTT is a framework for 21st century mathematics!

Towards an implementation

Follow development of a proof assistant for HOTT here:

`https://github.com/gwaithimirdain/narya`

Thanks!

Outline

Squares

For a type A we have $\mathsf{Id}_A : A \times A \to \mathscr{U}$.

Therefore, $\mathsf{refl}_{\mathsf{Id}_A}$ takes $a_{02} : \mathsf{Id}_A(a_{00}, a_{01})$ and $a_{12} : \mathsf{Id}_A(a_{10}, a_{11})$ to

$$\mathsf{refl}_{\mathsf{Id}_A}(a_{02}, a_{12}) : \mathsf{Id}_{\mathscr{U}}\left(\mathsf{Id}_A(a_{00}, a_{10}), \mathsf{Id}_A(a_{01}, a_{11})\right)$$

So if we also have $a_{20} : \mathsf{Id}_A(a_{00}, a_{10})$ and $a_{21} : \mathsf{Id}_A(a_{01}, a_{11})$, we get

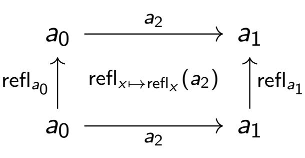
$$\mathsf{refl}_{\mathsf{Id}_A}(a_{02}, a_{12})(a_{20}, a_{21}) : \mathscr{U}$$

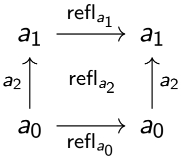
whose elements we can picture as squares:

$$\begin{array}{ccc} a_{10} & \xrightarrow{a_{12}} & a_{11} \\ {\scriptstyle a_{20}}\uparrow & a_{22} & \uparrow{\scriptstyle a_{21}} \\ a_{00} & \xrightarrow[a_{02}]{} & a_{01} \end{array}$$

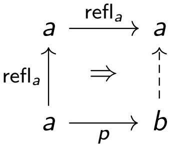
Problems involving squares

1. For $a_2 : \mathsf{Id}_A(a_0, a_1)$, there is nothing to define the degenerate square $\mathsf{refl}_{x \mapsto \mathsf{refl}_x}(a_2)$ to equal (refl_{a_2} has the wrong boundary).

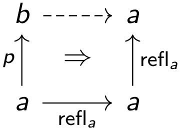




2. The subset $\{a\} = \coprod_{x:A} \mathsf{Id}_A(a, x)$ should be a singleton. Given $(b, p) : \coprod_{x:A} \mathsf{Id}_A(a, x)$, to show $\mathsf{Id}_A((b, p), (a, \mathsf{refl}_a))$, with $\overrightarrow{\mathsf{refl}_{\mathsf{Id}_A}(p, \mathsf{refl}_a)}(\mathsf{refl}_a)$ and $\overrightarrow{\overrightarrow{\mathsf{refl}_{\mathsf{Id}_A}(p, \mathsf{refl}_a)}}(\mathsf{refl}_a)$ we get



but we need



The fifth principle

Fifth principle of equality

Every square has an associated symmetric/transposed square:

$$
\begin{array}{ccc}
a_{10} & \xrightarrow{a_{12}} & a_{11} \\
{\scriptstyle a_{20}}\uparrow & a_{22} & \uparrow{\scriptstyle a_{21}} \\
a_{00} & \xrightarrow[a_{02}]{} & a_{01}
\end{array}
\quad \rightsquigarrow \quad
\begin{array}{ccc}
a_{01} & \xrightarrow{a_{21}} & a_{11} \\
{\scriptstyle a_{02}}\uparrow & \mathsf{sym}(a_{22}) & \uparrow{\scriptstyle a_{12}} \\
a_{00} & \xrightarrow[a_{20}]{} & a_{10}
\end{array}
$$

We define these separately for each construction of squares.

1. Now we can define $\mathsf{refl}_{x \mapsto \mathsf{refl}_x}(a_2) = \mathsf{sym}(\mathsf{refl}_{a_2})$.
2. And $\mathsf{sym}(\overset{\Longrightarrow}{\mathsf{refl}_{\mathsf{Id}_A}(p, \mathsf{refl}_a)}(\mathsf{refl}_a))$ proves that $\{a\}$ is a singleton.