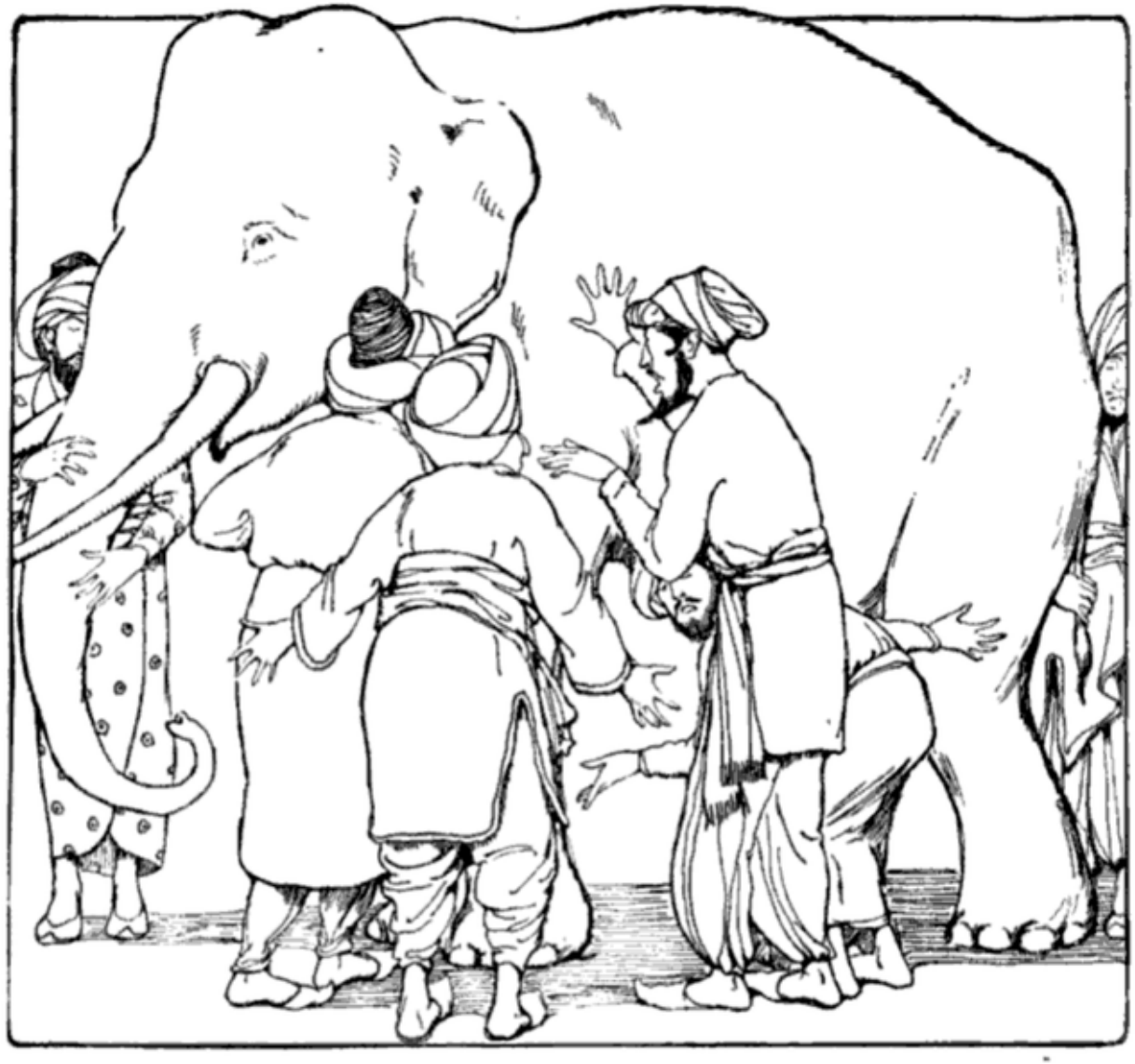
Homotopy type theory

A high-level language for invariant mathematics

Michael Shulman[1]

[1](University of San Diego)

March 8, 2019
Indiana University Bloomington



Homotopy Type Theory (HoTT) is...

- A framework for doing mathematics that can be "compiled" into any ∞-topos.
- A foundation for mathematics whose basic objects behave like ∞-groupoids.
- An enhancement of constructive type theory with a homotopical extensionality axiom.
- A dependently typed programming language that incorporates homotopical ideas.
- A way to formalize mathematics in a computer that is well-adapted to structural mathematics.
- A high-level abstract framework for working with sameness.

Homotopy Type Theory (HoTT) is...

- A framework for doing mathematics that can be "compiled" into any ∞-topos.
- A foundation for mathematics whose basic objects behave like ∞-groupoids.
- An enhancement of constructive type theory with a homotopical extensionality axiom.
- A dependently typed programming language that incorporates homotopical ideas.
- A way to formalize mathematics in a computer that is well-adapted to structural mathematics.
- A high-level abstract framework for working with sameness.

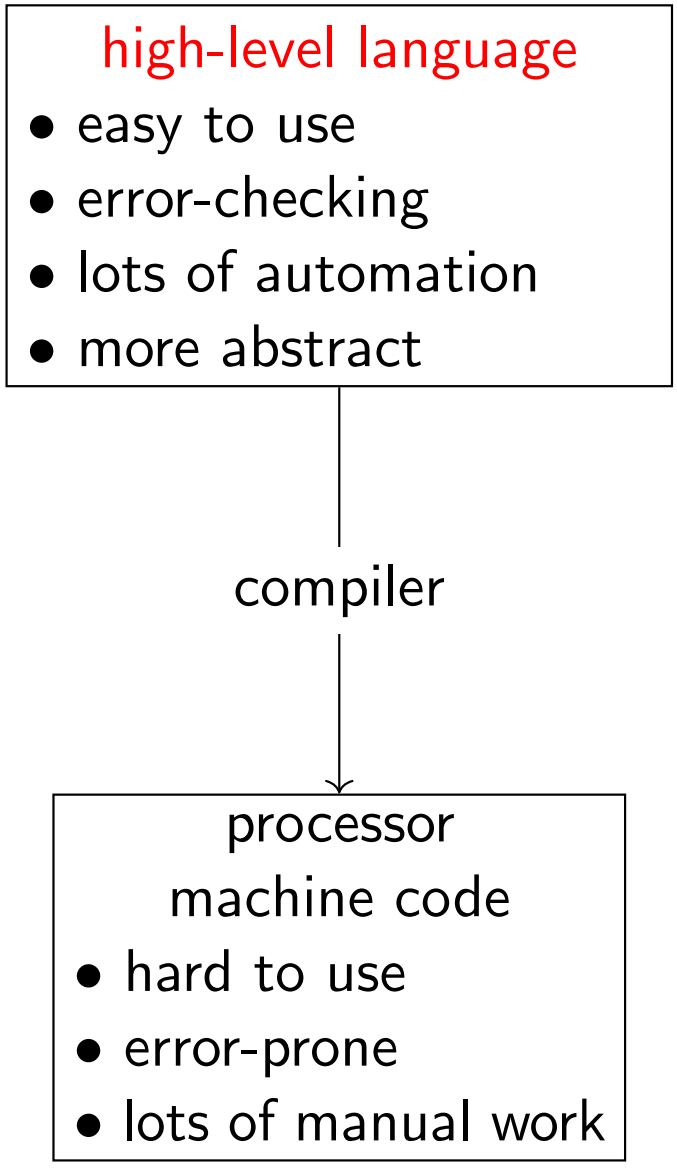
An analogy to programming

processor

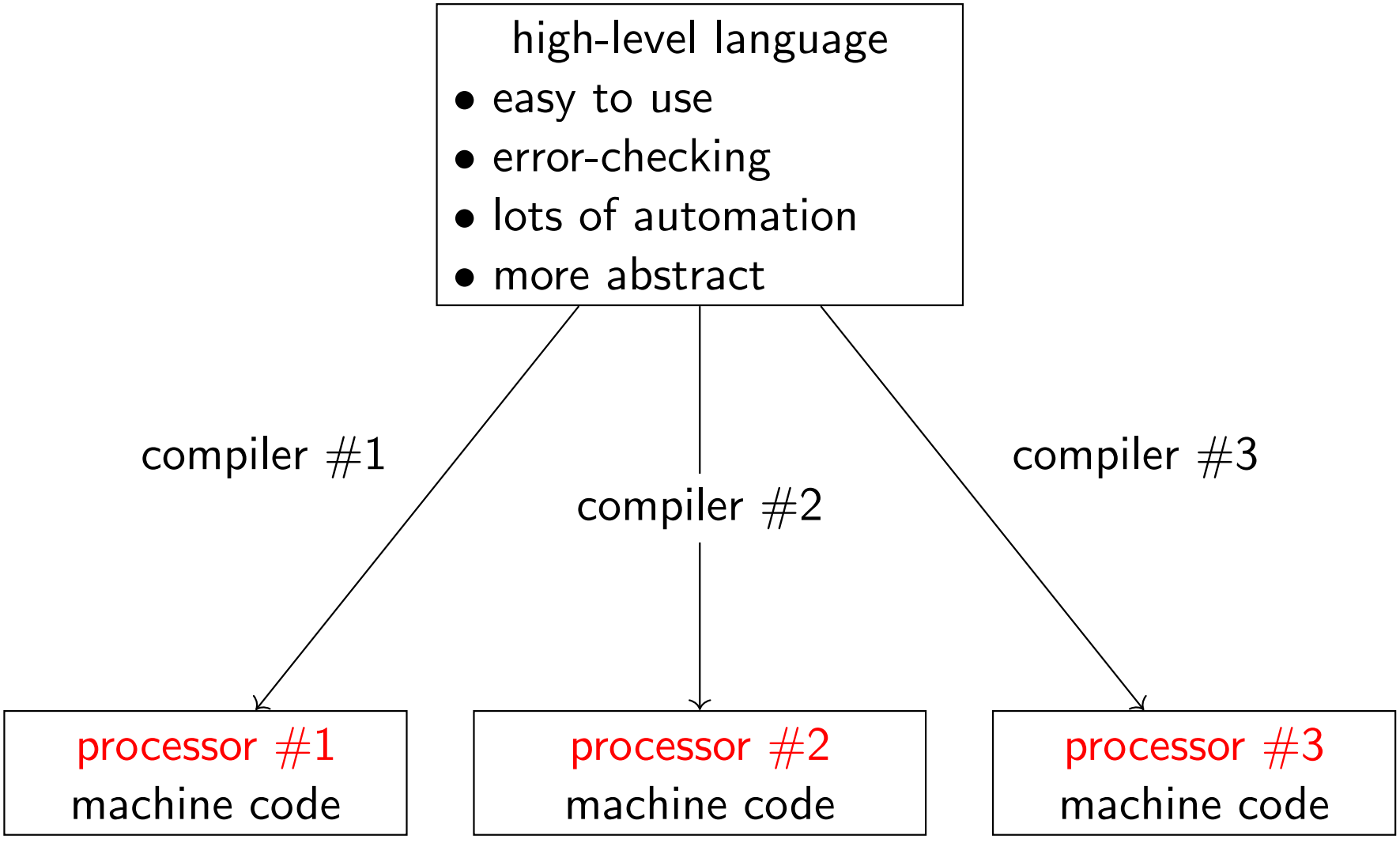
machine code

- hard to use
- error-prone
- lots of manual work

An analogy to programming



An analogy to programming

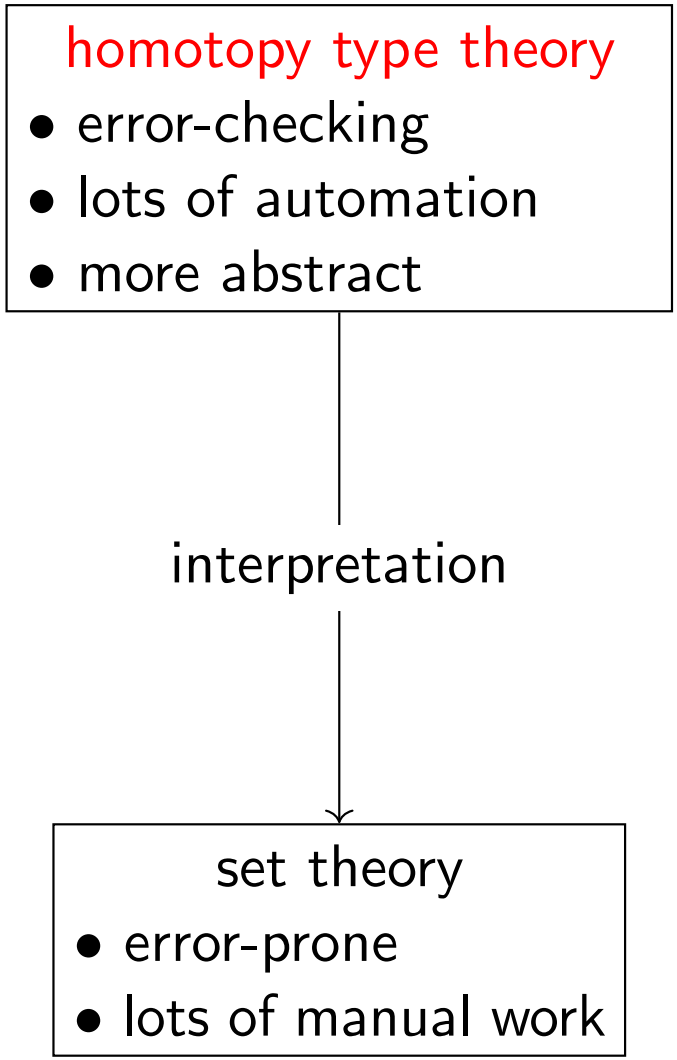


The other side of the analogy

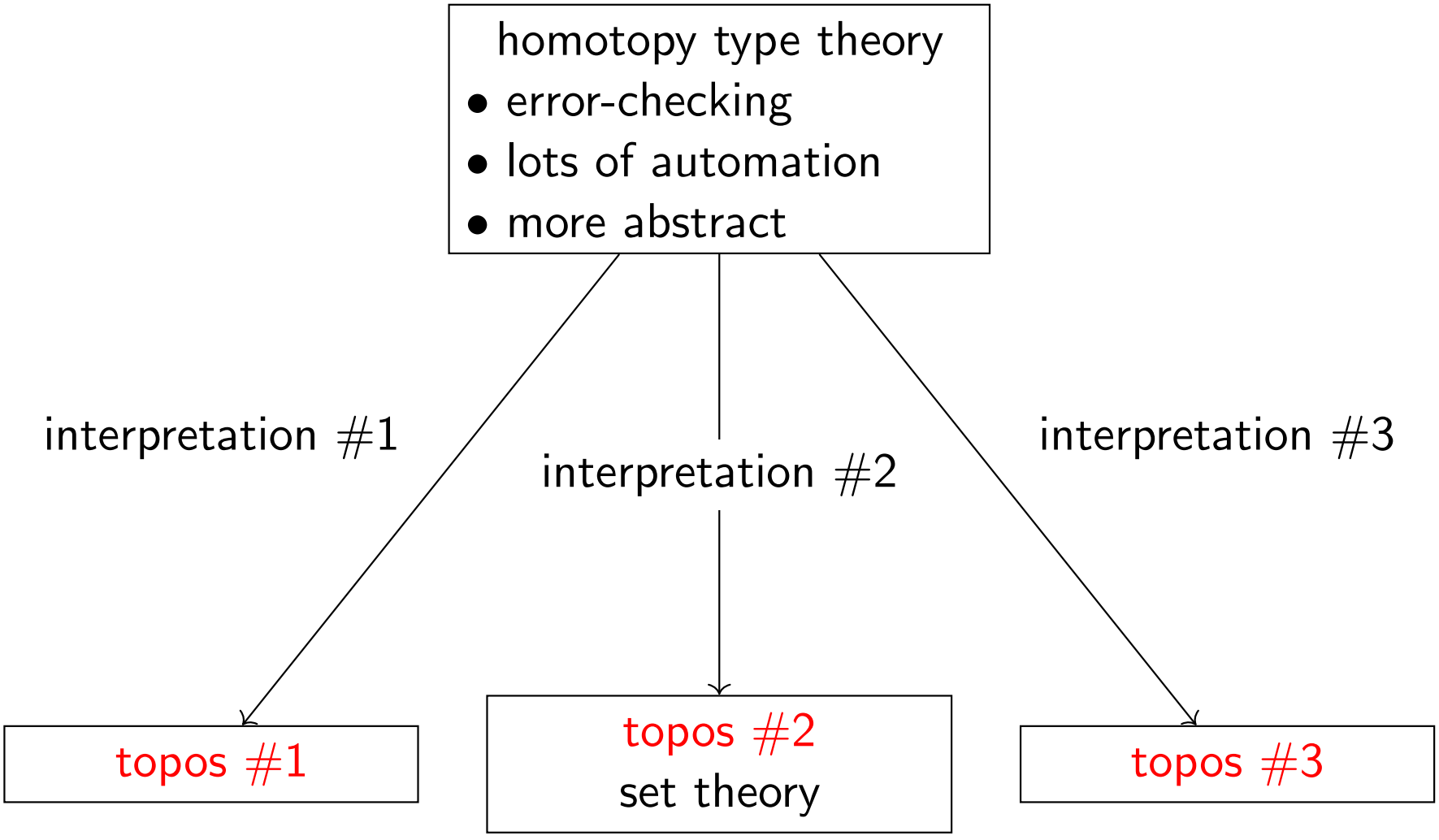
set theory

- error-prone
- lots of manual work

The other side of the analogy



The other side of the analogy



Homotopy type theory is
a high-level abstract framework for working with sameness.

Outline

Sameness can be tautological.

Example

$1 = 1$.

Sameness can be a highly nontrivial open problem.

Example

Is $P = NP$?

Sameness is. . .

- reflexive: everything is the same as itself.
- symmetric: if x is the same as y, then y is the same as x.
- transitive: if x is the same as y, and y is the same as z, then x is the same as z.

Two things that are the same have all the same properties.

Example

If x is the same as y, then any true statement about x is also a true statement about y. (The indiscernibility of identicals.)

Example

If x is the same as y, then $f(x)$ is the same as $f(y)$ for any f.

Two things can be the same even if
their representations are different.

Example

$0.5 = 0.499999\cdots$

Two things can be the same even if
their representations are different.

Example

$0.5 = 0.499999\cdots$

"Every interesting equation is a lie." (Gian-Carlo Rota?)

Example

We say $1 + 1 = 2$, but "$1 + 1$" and "2" are not the same expression.

For sameness to be non-tautological, we must be allowed to consider things the same even when their representations are different.

Sameness is compositional:
two structures are the same if all their pieces are.

Example

$(x_1, y_1) = (x_2, y_2)$ if and only if $x_1 = x_2$ and $y_1 = y_2$.

Example

For functions $f, g : A \to B$, we have $f = g$ if and only if $f(x) = g(x)$ for all $x \in A$.

Sameness is not always the same as equality.

Example

Two groups are the same if they are isomorphic.

Example

Two topological spaces are the same if they are homeomorphic.

Example

Two categories are the same if they are equivalent.

(etc.)

Sameness is context-dependent.

The word for a thing carries a presumption of the context for sameness.

Example

0.5 and $0.499999\ldots$ are different decimal expansions, but represent the same real number.

Example

3 and 7 are not the same integer, but represent the same integer modulo 4.

Example

$M_2(\mathbb{R})$ (2×2 matrices over $\mathbb{R}$) and $\mathbb{H}$ (the quaternions) are not the same algebra, but they are the same vector space.

It only makes sense to compare things for sameness
that have the same context.

Example

"$e^{i\pi}$ is the same as -1" is a meaningful (true) sentence; both are complex numbers.

Example

"$\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ is the same as $\mathbb{Z}/4\mathbb{Z}$" is a meaningful (false) sentence; both are abelian groups.

Example

"$e^{i\pi}$ is the same as $\mathbb{Z}/4\mathbb{Z}$" is not a meaningful sentence.
Like "colorless green ideas sleep furiously", it is a "category mistake" (in the philosophical sense).

Nontrivial sameness occurs outside pure mathematics too.

Example

An electric potential is a function φ whose negative gradient, $-\nabla\varphi$, is the (static) electric field. Two potentials are the same if $\varphi_1 - \varphi_2$ is a constant.

Example

A magnetic potential is a vector field $\mathbf{A}$ whose curl, $\nabla \times \mathbf{A}$, is the (static) magnetic field. Two magnetic potentials are the same if $\mathbf{A}_1 - \mathbf{A}_2$ is irrotational.

At least arguably, potentials have a real existence independently of fields: in the quantum-mechanical *Aharonov-Bohm effect* a particle is affected by a potential whose field is zero.

Sameness is compositional. . . ?

Example

A group is a triple $(G, *, e)$ where G is a set, $*$ is a binary operation $G \times G \to G$, and e is an element of G, satisfying axioms.

So two groups $(G, *, e)$ and $(H, \star, i)$ should be the same if

- The set G is the same as the set H,
- The operation $*$ is the same as the operation $\star$, and
- The element e is the same as the element i.

Sameness is compositional. . . ?

Example

A group is a triple $(G, *, e)$ where G is a set, $*$ is a binary operation $G \times G \to G$, and e is an element of G, satisfying axioms.

So two groups $(G, *, e)$ and $(H, \star, i)$ should be the same if

- The set G is the same as the set H,
- The operation $*$ is the same as the operation $\star$, and
- The element e is the same as the element i.

What does it mean for two sets to be the same?

Two different answers! Maybe we should use different words.

Definition (Extensionality)

Two material sets A and B are the same if they have the same elements: for all x we have $x \in A$ if and only if $x \in B$.

Definition (Bijection)

Two structural sets A and B are the same if we have functions $f : A \to B$ and $g : B \to A$ such that $f \circ g = \mathtt{id}_B$ and $g \circ f = \mathtt{id}_A$.

Material sets (which are used in ZFC) can be considered "representations" of structural sets (which are used in most mathematics), like integers are representations of integers-mod-4.

Example

Two groups $(G, *, e)$ and $(H, \star, i)$ are the same if

- The set G is bijective to the set H,
- The operation $*$ is the same as the operation $\star$ under this bijection, and
- The element e is the same as the element i under this bijection.

Example

Two topological spaces $(X, \mathcal{O}_X)$ and $(Y, \mathcal{O}_Y)$ are the same if

- The set X is bijective to the set Y, and
- The open sets $U \in \mathcal{O}_X$ are the same as the open sets $V \in \mathcal{O}_Y$ under this bijection.

(etc.)

Sameness is transportive.

Example

Two groups $(G, *, e)$ and $(H, \star, i)$ are the same if

- The structural set G is the same as (i.e. bijective to) the set H,
- The operation $*$ is the same as the operation $\star$ under this bijection, and
- The element e is the same as the element i under this bijection.

In other words, if $f : G \xrightarrow{\sim} H$ is the bijection, then

$$f(x_1 * x_2) = f(x_1) \star f(x_2) \qquad \text{and} \qquad f(e) = i.$$

Sameness can be structure rather than a mere relation.

Example

Two groups can be isomorphic in more than one way.
In fact a group can be isomorphic to itself in more than one way!

Example

Similarly, two sets can be bijective in more than one way, etc.

The "ways in which x is the same as y" (isomorphisms, bijections) have their own context for sameness.

If sameness is structure, then so are its "properties".

- reflexivity: everything is the same as itself *in a specified way*.
  (the identity isomorphism $\mathtt{id}_G : G \cong G$)
- symmetry: for any way in which x is the same as y, there is a *specified* way in which y is the same as x.
  (the inverse $f^{-1} : H \xrightarrow{\sim} G$ of an isomorphism $f : G \xrightarrow{\sim} H$)
- transitive: for any way in which x is the same as y, and any way in which y is the same as z, there is a specified way in which x is the same as z.
  (the composite of two isomorphisms $G \xrightarrow{f} H \xrightarrow{g} K$)

We must remember the structure of sameness
even outside pure mathematics.

Example

- In Einstein's general relativity, spacetime is represented by a 4-manifold M with a Lorentzian metric.
- Two spacetimes are the same if there is a diffeomorphism $\phi : M_1 \xrightarrow{\sim} M_2$ that preserves the metrics. Then $x \in M_1$ and $\phi(x) \in M_2$ represent the same point in spacetime.

We must remember the structure of sameness
even outside pure mathematics.

Example

- In Einstein's general relativity, spacetime is represented by a 4-manifold M with a Lorentzian metric.
- Two spacetimes are the same if there is a diffeomorphism $\phi : M_1 \xrightarrow{\sim} M_2$ that preserves the metrics. Then $x \in M_1$ and $\phi(x) \in M_2$ represent the same point in spacetime.
- If $\psi : M_1 \xrightarrow{\sim} M_2$ is a different diffeomorphism that doesn't preserve the metrics, then $x \in M_1$ and $\psi(x) \in M_2$ do not represent the same point in spacetime.

We must remember the structure of sameness
even outside pure mathematics.

Example

- In Einstein's general relativity, spacetime is represented by a 4-manifold M with a Lorentzian metric.
- Two spacetimes are the same if there is a diffeomorphism $\phi : M_1 \xrightarrow{\sim} M_2$ that preserves the metrics. Then $x \in M_1$ and $\phi(x) \in M_2$ represent the same point in spacetime.
- If $\psi : M_1 \xrightarrow{\sim} M_2$ is a different diffeomorphism that doesn't preserve the metrics, then $x \in M_1$ and $\psi(x) \in M_2$ do not represent the same point in spacetime.
- This is true even when ψ is $\mathtt{id}_M : M \cong M$. But this confused Einstein for a long time (his hole argument).

Outline

Problem #1

Two things that are the same have all the same properties. . . ?

Example

Define a group G to be splanzy if $G \subseteq \mathbb{R}$.

- $\mathbb{Z}$ is splanzy.
- $\mathbb{Z}i = \{ai \mid a \in \mathbb{Z}\} \subseteq \mathbb{C}$ is not splanzy.
- Yet, $\mathbb{Z}$ is isomorphic to $\mathbb{Z}i$.

Splanziness is not meaningful in the "sameness context" of groups.

Problem #1

Two things that are the same have all the same properties. . . ?

Example

Define a group G to be splanzy if $G \subseteq \mathbb{R}$.

- $\mathbb{Z}$ is splanzy.
- $\mathbb{Z}i = \{ai \mid a \in \mathbb{Z}\} \subseteq \mathbb{C}$ is not splanzy.
- Yet, $\mathbb{Z}$ is isomorphic to $\mathbb{Z}i$.

Splanziness is not meaningful in the "sameness context" of groups.

- Question: Which properties are meaningful in the sameness context of groups?
- Answer #1: Um. . . those that are invariant under isomorphism?

Problem #1

- Question: Which properties are meaningful in the sameness context of groups?
- Answer #2:

I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description, and perhaps I could never succeed in intelligibly doing so. But I know it when I see it.

– Justice Potter Stewart, Jacobellis v. Ohio

Problem #1

Why is this not good enough?

1. We are mathematicians, not lawyers: we hold ourselves to a higher standard.
2. The *expert* may know it when they see it, but a novice can be tripped up.
3. Computers can't (yet) learn to "know it when they see it".
4. Actually *proving* invariance is mind-numbingly tedious.

Problem #2

Whenever we define some structure (say widgets), we must

1. Define when two widgets are the same.
2. Show that sameness for widgets is reflexive, symmetric, and transitive.

Often, this all means means defining structure, about which we must then prove things. That might also mean defining more structure, etc.

Problem #2

The structure of fully-structural sameness is highly complicated.

- If $f : x \xrightarrow{\sim} y$ and $g : y \xrightarrow{\sim} z$ are structures for sameness of x and y and of y and z, we have a specified $g \circ f : x \xrightarrow{\sim} z$.
- If also $h : z \xrightarrow{\sim} u$, we have $(h \circ g) \circ f$ and $h \circ (g \circ f)$, which should be the same as structures for sameness of x and u.
- If also $k : u \xrightarrow{\sim} v$, we have two structures for sameness of $((k \circ h) \circ g) \circ f$ and $k \circ (h \circ (g \circ f))$ as structures for sameness of x and v, which should be the same.
- ...

There are techniques for handling all of this that work in many cases, but not all; and we have to learn and apply them.

Problem #3

Sameness is compositional. . . ?

Example

A category is a set C_0 of objects, a set C_1 of morphisms, . . .

So two categories should be the same if

- The set C_0 is the same as the set D_0,
- The set C_1 is the same as the set D_1,
- . . .

Problem #3

Sameness is compositional. . . ?

Example

A category is a set C_0 of objects, a set C_1 of morphisms, . . .

So two categories should be the same if

- The set C_0 is bijective to the set D_0,
- The set C_1 is bijective to the set D_1,
- . . .

But this defines an "isomorphism of categories", not an equivalence!

For ordinary structures like groups, compositionality based on bijections gave the right notion of sameness. But it seems like the objects of a category need to be something other than "sets".

Outline

Homotopy type theory is an abstract framework for working with sameness that:

1. automatically tracks contexts for sameness.
2. automatically builds notions of sameness compositionally.
3. automatically transports anything along a sameness.
4. automatically generates the complicated structure of sameness.
5. admits structured sameness and remembers the structure.
6. allows introducing new notions of sameness on old objects.
7. treats sets structurally, but subsets materially.
8. can treat categories and similar structures up to equivalence.

Think of it like a "high-level programming language" (Python, Ruby, Java), with features and guarantees making it easier to use and less error-prone, that gets automatically "compiled" to the low-level "assembly language" of mathematical objects.

Definition

We will say type rather than "context for sameness".

Thus a "type" specifies a collection of things (its "elements" or "objects") together with the relevant notion of sameness for them.

Definition

If x is an element of the type A we write $x : A$.

If $x : A$ and $y : A$ for the same type A, we can ask whether they are the same.

Definition

For $x : A$ and $y : A$, we write $x = y$ for "x is the same as y".

There is no existing notation encompassing all notions of sameness. But equality is the most common kind of sameness.

Recall we sometimes ask whether two "structures for sameness" are themselves the same. Thus "$x = y$" must itself be a type.

Definition

We call the elements of $x = y$ identifications of x with y.

Example

If G, H are groups (i.e. $G : \texttt{Groups}$ and $H : \texttt{Groups}$), then $G = H$ is the type of isomorphisms from G to H.

Example

If A, B are (structural) sets (i.e. $A : \texttt{Sets}$ and $B : \texttt{Sets}$), then $A = B$ is the type of bijections from A to B.

Often there is no "structure" to sameness (i.e. simple equality). But we don't want $x = y$ to sometimes be a type and sometimes not. Instead, sometimes it is a type with only one element.

Definition

A proposition is a type in which any two elements are the same.

If a proposition has at least one (hence exactly one) element, we call it true. If it has no elements, we call it false.

Example

- $1 + 1 = 2$ is (a type that is) a proposition, which is true.
- $\sqrt{2} = \pi$ is (a type that is) a proposition, which is false.

Definition

A set is a type A such that for any $x, y : A$ the type $x = y$ is a proposition.

But what is a type, really?

But what is a type, really?

Well, that's a question for whoever programs the "compiler".
All we as the "user" need to know is that there are these things called "types" that we can do certain things with.

But what is a type, really?

Well, that's a question for whoever programs the "compiler".
All we as the "user" need to know is that there are these things called "types" that we can do certain things with.

In fact, there are lots of different compilers!

- In the "reference implementation" by Voevodsky, types are compiled to Kan complexes, a simplicial notion of ∞-groupoid.
- They can also be fibrant objects in a Quillen model category that presents any $(\infty, 1)$-topos. (We'll come back to this later.)

For now, we'll just focus on how types behave.

Sameness is compositional.

Example

For types A and B, there is a product type $A \times B$ whose elements are ordered pairs (x, y) with $x : A$ and $y : B$. The type $(x_1, y_1) = (x_2, y_2)$ is the type $(x_1 = x_2) \times (y_1 = y_2)$.

Example

For an indexed family of types $\{A_j\}$, there is a product type $\prod_j A_j$ whose elements are tuples $(x_j)_j$ with each $x_j : A_j$. The type $(x_j)_j = (y_j)_j$ is the type $\prod_j (x_j = y_j)$.

Example

For types A and B, there is a function type "$A \to B$" whose elements are functions from A to B. For $f, g : A \to B$ the type $f = g$ is the type $\prod_x (f(x) = g(x))$.

All functions respect sameness: if $f : A \to B$ and $x, y : A$ with $e : x = y$, we have $\mathrm{ap}_f(e) : f(x) = f(y)$.

Example

There is a type $\emptyset$ that has no elements, and a type $\mathbf{1}$ that has exactly one element.

Example

For types A and B, there is a disjoint union type $A \amalg B$ whose elements are either $\texttt{inl}(a)$ for $a : A$ or $\texttt{inr}(b)$ for $b : B$.

- The type $\texttt{inl}(a_1) = \texttt{inl}(a_2)$ is the type $a_1 = a_2$.
- The type $\texttt{inl}(b_1) = \texttt{inl}(b_2)$ is the type $b_1 = b_2$.
- The type $\texttt{inl}(a) = \texttt{inr}(b)$ is the type $\emptyset$.

Example

For types A and B, there is a disjoint union type $A \amalg B$ whose elements are either $\texttt{inl}(a)$ for $a : A$ or $\texttt{inr}(b)$ for $b : B$.

- The type $\texttt{inl}(a_1) = \texttt{inl}(a_2)$ is the type $a_1 = a_2$.
- The type $\texttt{inl}(b_1) = \texttt{inl}(b_2)$ is the type $b_1 = b_2$.
- The type $\texttt{inl}(a) = \texttt{inr}(b)$ is the type $\emptyset$.

Example

For an indexed family of types $\{A_j\}$, there is a disjoint union type $\coprod_j A_j$, or $\sum_j A_j$, whose elements are pairs (j, x) with $x : A_j$. The type $(i, x) = (j, y)$ is. . . ?

We want to say $(i = j) \times (x = y)$, but that doesn't make sense since x and y don't belong to the same type.

But if $i = j$, then A_i and A_j ought to be "the same". . .

The way to compare an element of A with an element of B is to be given a structure for sameness of A and B as types, i.e. an identification $A = B$ in the type of types.

Thus $E : A = B$ should have, for any $a : A$ and $b : B$ a type $E(a, b)$ generalizing "$x = y$", with "sameness-like" properties:

- If $a_1 = a_2$ and $E(a_2, b)$ then $E(a_1, b)$.
- If $E(a, b_1)$ and $b_1 = b_2$ then $E(a, b_2)$.
- If $E(a_1, b)$ and $E(a_2, b)$ then $a_1 = a_2$.

- If $E(a, b_1)$ and $E(a, b_2)$ then $b_1 = b_2$.

Also every element of A should be representable by some element of B and vice versa:

- For any $a : A$ there exists a $b : B$ such that $E(a, b)$.
- For any $b : B$ there exists a $a : A$ such that $E(a, b)$.

The way to compare an element of A with an element of B is to be given a structure for sameness of A and B as types, i.e. an identification $A = B$ in the type of types.

Thus $E : A = B$ should have, for any $a : A$ and $b : B$ a type $E(a, b)$ generalizing "$x = y$", with "sameness-like" properties:

- ~~If $a_1 = a_2$ and $E(a_2, b)$ then $E(a_1, b)$.~~ $\leftarrow$ automatic
- ~~If $E(a, b_1)$ and $b_1 = b_2$ then $E(a, b_2)$.~~ $\leftarrow$ automatic
- If $e_1 : E(a_1, b)$ and $e_2 : E(a_2, b)$ then $(a_1, e_1) = (a_2, e_2)$ in $\sum_x E(x, b)$.
- If $e_1 : E(a, b_1)$ and $e_2 : E(a, b_2)$ then $(b_1, e_1) = (b_2, e_2)$ in $\sum_y E(a, y)$.

Also every element of A should be representable by some element of B and vice versa:

- For any $a : A$ there exists a $b : B$ such that $E(a, b)$.
- For any $b : B$ there exists a $a : A$ such that $E(a, b)$.

Definition

There is a universe type "Type" whose elements are types. For types A, B, the type $A = B$ is the type of equivalences.
(This is Voevodsky's univalence axiom.)

Definition

An equivalence between types A, B consists of a type $E(a, b)$ for each $a : A$ and $b : B$ such that

1. If $e_1 : E(a_1, b)$ and $e_2 : E(a_2, b)$ then $(a_1, e_1) = (a_2, e_2)$.
2. If $e_1 : E(a, b_1)$ and $e_2 : E(a, b_2)$ then $(b_1, e_1) = (b_2, e_2)$.
3. For each $a : A$, there is a $b : B$ such that $E(a, b)$.
4. For each $b : B$, there is an $a : A$ such that $E(a, b)$.

- There are also many other equivalent definitions of equivalences; this is not Voevodsky's original one.
- When A and B are sets, this reduces to a bijection.

Now we can fill in some gaps.

Definition

For a type A, an A-indexed type family is a function $P : A \to \mathtt{Type}$.

Example

Given $P : A \to \mathtt{Type}$, and $e : x = y$ in A, we have an equivalence $\mathrm{ap}_P(e) : P(x) = P(y)$. This gives indiscernibility of identicals: for any $u : P(x)$ there is $v : P(y)$ (such that $\mathrm{ap}_P(e)(u, v)$).

In the types $\prod_{j:J} A_j$ and $\sum_{j:J} A_j$, actually A is an indexed type family $A : J \to \mathtt{Type}$.

Definition (Identifications in $\sum$-types)

If $e : i = j$ in J, we have $\mathrm{ap}_A(e) : A_i = A_j$, an equivalence. Hence for $x : A_i$ and $y : A_j$ we have a type $\mathrm{ap}_A(e)(x, y)$; we define $(i, x) = (j, y)$ in $\sum_{j:J} A_j$ to be the type $\sum_{e:i=j} \mathrm{ap}_A(e)(x, y)$.

That is, i and j are the same, and under this identification x and y are the same.

The type of groups is

$$\sum_{G:\mathsf{Set}} \sum_{m:G\times G\to G} \sum_{e:G} \Big(\prod_{x,y,z:G} m(m(x,y),z) = m(x,m(y,z)) \Big) \times \Big(\prod_{x:G} ((m(x,e) = x) \times (m(e,x) = x)) \Big) \times \Big(\prod_{x:G} \sum_{y:G} ((m(x,y) = e) \times (m(y,x) = e)) \Big)$$

Thus, by definition of sameness in $\sum$, two groups are the same if

- The sets G and H are the same (i.e. bijective).
- The multiplications are the same under this bijection.
- The units are the same under this bijection.
- (No more conditions; the other components are propositions.)

Definition

A category is

- A *type* C_0 of objects,
- A family of sets $\hom_C : C_0 \times C_0 \to \mathtt{Set}$
- Compositions, identities, axioms, ...
- For objects $x, y : C_0$, the canonical map from $x = y$ to the type $\mathtt{iso}_C(x, y)$ of isomorphisms in C is an equivalence.

Now two categories are the same (compositionally!) if

- The types C_0 and D_0 are the same in $\mathtt{Type}$ (i.e. equivalent).
- $\hom_C$ and $\hom_D$ are the same under this equivalence.
- These bijections respect composition and identities.

This is the same as an equivalence of categories, because the sameness of C_0 and D_0 is "up to" their identifications, which are equivalent to the isomorphisms in C and D.

Outline

Let G be a group (i.e. a set with a multiplication and unit s.t.. . .).

Definition

A principal G-bundle over a set X is a set P with free G-action $G \times P \to P$ such that $P/G = X$.

Example

The twisted double cover of a circle is a principal $\mathbb{Z}/2\mathbb{Z}$-bundle.

The type of principal G-bundles over X is

$$G\text{-}\mathtt{Bun}(X) = \sum_{P:\mathtt{Set}} \sum_{a:G\times P\to P} (\mathtt{free\text{-}action}(a) \times (P/G = X)).$$

Definition

A G-torsor is a nonempty set Z with a free and transitive G-action.

The type of G-torsors is

$$G\text{-}\mathtt{Tors} = \sum_{Z:\mathrm{Set}} \sum_{a:G\times P\to P} (\mathtt{free\text{-}trans}(a) \times \|Z\|).$$

Definition

A G-torsor is a nonempty set Z with a free and transitive G-action.

The type of G-torsors is

$$G\text{-}\mathtt{Tors} = \sum_{Z:\mathrm{Set}} \sum_{a:G\times P\to P} (\mathtt{free\text{-}trans}(a) \times \|Z\|).$$

Theorem (in homotopy type theory)

G-$\mathtt{Tors}$ is a classifying type for principal G-bundles:

$$G\text{-}\mathtt{Bun}(X) = (X \to G\text{-}\mathtt{Tors}).$$

Note the "=" referring to sameness of types, i.e. equivalence. The notion of sameness supplied by the automatic machinery of homotopy type theory is almost always the "right" one for whatever you're doing.

Usually, principal bundles and torsors have topology on them, but these definitions and theorem don't say anything explicitly about topology. It looks like they're only talking about discrete sets (where every principal bundle is trivial).

And in the "reference implementation", with types interpreted by simplicial sets, that's exactly what they come out to mean.

But we can also "compile" the same theorem in lots of other interpretations!

Example

We can interpret types as topological spaces*. Then G is a topological group, and G-`Tors` is a topological groupoid classifying topological principal bundles.

Example

We can interpret types as smooth spaces*. Then G is a Lie group, and G-`Tors` is a Lie groupoid classifying smooth principal bundles.

Example

We can interpret types as algebraic spaces*. Then G is an algebraic group, and G-`Tors` is an algebraic groupoid classifying algebraic principal bundles.

Example

We can interpret types as sheaves on some space Y. Then G is a sheaf of groups, and G-$\mathtt{Tors}$ is a sheaf of groupoids classifying sheaf principal bundles.

Example

We can interpret types as computable sets*. Then G is a computable group, and G-$\mathtt{Tors}$ is a computable groupoid classifying computable principal bundles.

Example

We can interpret types as sets or spaces with an Γ-action, for some fixed group Γ. Then G is an extension of Γ, and G-$\mathtt{Tors}$ is an Γ-groupoid classifying (G, Γ)-principal bundles.

Other "compilers" exist that interpret types as:

- Super-geometric spaces
- Sets with "a compatible action of all groups at once"
- Sets with time-varying dynamical behavior
- Classically observable aspects of quantum systems
- Nonstandard sets containing infinitesimal objects
- Parametrized spectra or excisive homotopy functors
- ∞-categories or (∞, n)-categories

For a theorem to compile to all these models, it has to avoid making special "system calls" that don't exist in all of them. The most familiar such is the law of excluded middle, that every proposition is either true or false.

However, this is often surprisingly easy to do without.

Some open problems

1. No existing version of homotopy type theory makes all the characterizations of identification types hold by definition; some of them hold only up to equivalence.
2. Most of the "compilers" that I mentioned are not yet completely implemented. (But come back tomorrow to hear about some recent progress!)

Further reading

- The *Homotopy Type Theory* book: `https://homotopytypetheory.org/book/`
- Other surveys and introductions: `https://ncatlab.org/homotopytypetheory/show/References#surveys`