The Equivalence Principle and Univalent Foundations

Paige Randall North

13 September 2021

Outline

The equivalence principle

Equivalence principle

Reasoning in mathematics should be **invariant under** the appropriate notion of **equivalence**.

The equivalence principle

Equivalence principle

Reasoning in mathematics should be **invariant under** the appropriate notion of **equivalence**.

Notion of equivalence depends on the objects under consideration:

- **equal** numbers, functions,. . .
- **isomorphic** sets, groups, rings,. . .
- **equivalent** categories
- **biequivalent** bicategories
- . . .

Non-examples: statements violating equivalence principle

We can easily **violate** this principle:

Exercise

Find a statement about sets that is not invariant under isomorphism:

$$\{\emptyset, \{\emptyset\}\} \cong \{\emptyset, \{\{\emptyset\}\}\}$$

Exercise

Find a statement about categories that is not invariant under equivalence:

$$\begin{tikzcd} \bullet \arrow[r, bend left] & \bullet \arrow[l, bend left] \end{tikzcd} \quad \simeq \quad \bullet$$

Non-examples: statements violating equivalence principle

We can easily **violate** this principle:

Exercise

Find a statement about sets that is not invariant under isomorphism:

$$\{\emptyset, \{\emptyset\}\} \cong \{\emptyset, \{\{\emptyset\}\}\}$$

$\{\emptyset\} \in X$

Exercise

Find a statement about categories that is not invariant under equivalence:

$$\begin{tikzcd} \bullet \arrow[r, bend left] & \bullet \arrow[l, bend left] & \simeq & \bullet \end{tikzcd}$$

$\mathscr{C}$ has exactly 1 object.

A language for invariant properties

Michael Makkai, *Towards a Categorical Foundation of Mathematics*:

> *"The basic character of the Principle of Isomorphism is that of a* ***constraint on the language*** *of Abstract Mathematics; a welcome one, since it provides for the separation of sense from nonsense."*

A language for invariant properties

Michael Makkai, *Towards a Categorical Foundation of Mathematics*:

> *"The basic character of the Principle of Isomorphism is that of a* ***constraint on the language*** *of Abstract Mathematics; a welcome one, since it provides for the separation of sense from nonsense."*

Goal

To have a **syntactic criterion** for properties and constructions that are invariant under equivalence

How to break the equivalence principle for categories. . .

- Recall: the statement

  The category $\mathscr{C}$ has exactly one object.

  is not invariant under equivalence of categories.
- In general, referring to **equality of objects** breaks invariance, but. . .

How to break the equivalence principle for categories. . .

- Recall: the statement

  The category $\mathscr{C}$ has exactly one object.

  is not invariant under equivalence of categories.
- In general, referring to **equality of objects** breaks invariance, but. . .
- even the **definition** of category refers to equality of objects:

Problem

"If $\mathrm{dom}(g)$ is **equal to** $\mathrm{cod}(f)$, then $g \circ f$ exists."

How to break the equivalence principle for categories. . .

- Recall: the statement

  The category $\mathscr{C}$ has exactly one object.

  is not invariant under equivalence of categories.
- In general, referring to **equality of objects** breaks invariance, but. . .
- even the **definition** of category refers to equality of objects:

Problem

"If $\mathrm{dom}(g)$ is **equal to** $\mathrm{cod}(f)$, then $g \circ f$ exists."

Can we give a definition of category without using equality of objects?

. . . and how to fix it.

Solution

Use a logic/language of **dependent sets**, in which $\mathrm{dom}(g) = \mathrm{cod}(f)$ is encoded by what type of thing f and g are.

. . . and how to fix it.

Solution

Use a logic/language of **dependent sets**, in which $\mathrm{dom}(g) = \mathrm{cod}(f)$ is encoded by what type of thing f and g are.

A category consists of

- a set O of objects
- for each $x, y \in O$, a type/set $A(x, y)$ of arrows
- for each $x, y, z \in O$ and each $f \in A(x, y)$ and $g \in A(y, z)$, a type/set $g \circ f \in A(x, z)$
- for each $x \in O$, an identity $\mathrm{id}_x \in A(x, x)$
- . . .

. . . and how to fix it.

Solution

Use a logic/language of **dependent sets**, in which $\mathrm{dom}(g) = \mathrm{cod}(f)$ is encoded by what type of thing f and g are.

A category consists of

- a set O of objects
- for each $x, y \in O$, a type/set $A(x, y)$ of arrows
- for each $x, y, z \in O$ and each $f \in A(x, y)$ and $g \in A(y, z)$, a type/set $g \circ f \in A(x, z)$
- for each $x \in O$, an identity $\mathrm{id}_x \in A(x, x)$
- . . .

Gives rise to **dependently typed language** by adding logical connectors.

Invariance for statements

Theorem (Freyd '76, Blanc '78)

A property of categories (expressed in 2-sorted first order logic) is invariant under equivalence iff it can be expressed in this dependently typed language, using equality for arrows but not for objects.

Invariance for statements

Theorem (Freyd '76, Blanc '78)

*A **property** of categories (expressed in 2-sorted first order logic) is invariant under equivalence iff it can be expressed in this dependently typed language, using equality for arrows but not for objects.*

- What about **constructions** on categories?

Invariance for statements

Theorem (Freyd '76, Blanc '78)

*A **property** of categories (expressed in 2-sorted first order logic) is invariant under equivalence iff it can be expressed in this dependently typed language, using equality for arrows but not for objects.*

- What about **constructions** on categories?
- What about other mathematical structures?

Outline

What is type theory?

- Type theory is a language for mathematics, akin to category theory.
- Sentences are of the following form:
  - $a_1 : A_1, \ldots, a_n : A_n \vdash B(a_1, \ldots, a_n)$ type
  - $a_1 : A_1, \ldots, a_n : A_n \vdash b(a_1, \ldots, a_n) : B(a_1, \ldots, a_n)$
- e.g.
  - $x, y : \mathsf{ob}\mathscr{C} \vdash \hom_{\mathscr{C}}(x, y)$ type
  - $x : \mathsf{ob}\mathscr{C} \vdash 1_x : \hom_{\mathscr{C}}(x, x)$
- We conflate mathematical objects and mathematical statements.
  - $n : \mathbb{N} \vdash \mathsf{isEven}(n)$ type
  - $n : \mathbb{N} \vdash e(n) : \mathsf{isEven}(2n)$
  - $n : \mathbb{N} \vdash \mathsf{Vect_n}(\mathbb{N})$ type
  - $n : \mathbb{N} \vdash \mathrm{o}(n) : \mathsf{Vect_n}(\mathbb{N})$

Interpretations of type theory

- Examples:
  - $n : \mathbb{N} \vdash \mathsf{isEven}(n)\ \mathsf{type}$
  - $n : \mathbb{N} \vdash e(n) : \mathsf{isEven}(2n)$
  - $n : \mathbb{N} \vdash \mathsf{Vect}_{\mathsf{n}}(\mathbb{N})\ \mathsf{type}$
  - $n : \mathbb{N} \vdash \mathrm{o}(n) : \mathsf{Vect}_{\mathsf{n}}(\mathbb{N})$
- There are many interpretations of dependent type theory:

	Contexts	**Types**	**Terms**
Logical	hypotheses	predicates	proofs
Set theoretic	indices	indexed sets	sections
Homotopical	base space	total space	sections

Type formers

- We can define the natural numbers, booleans, the circle, and coproducts as initial objects in the following way. (Dependent) functions and (dependent) products are defined similarly.

Natural numbers

$$\frac{}{\vdash \mathbb{N} \text{ type}} \qquad \frac{}{\vdash 0 : \mathbb{N}} \qquad \frac{\vdash x : \mathbb{N}}{\vdash sx : \mathbb{N}}$$

$$\frac{x : \mathbb{N} \vdash D(x) \text{ type} \qquad \vdash z : D(0) \qquad x : \mathbb{N}, y : D(x) \vdash \sigma(y) : D(sx)}{\begin{array}{c} x : \mathbb{N} \vdash d(x) : D(x) \\ \vdash d(0) \equiv z : D(0) \qquad x : \mathbb{N} \vdash \sigma(d(x)) \equiv d(sx) : D(sx) \end{array}}$$

Type formers

Binary product

$$\frac{\vdash A \text{ type} \qquad \vdash B \text{ type}}{\vdash A \times B \text{ type}} \qquad \frac{\vdash a : A \qquad \vdash b : B}{\vdash \langle a, b \rangle : A \times B}$$

$$\frac{x : A \times B \vdash D(x) \text{ type} \qquad a : A, b : B \vdash \sigma(a, b) : D\langle a, b \rangle}{\begin{array}{c} x : A \times B \vdash d(x) : D(x) \\ a : A, b : B \vdash \sigma(a, b) \equiv d\langle a, b \rangle : D\langle a, b \rangle \end{array}}$$

Type formers

Binary product

$$\frac{\vdash A \text{ type} \qquad \vdash B \text{ type}}{\vdash A \times B \text{ type}} \qquad \frac{\vdash a : A \qquad \vdash b : B}{\vdash \langle a, b \rangle : A \times B}$$

$$\frac{x : A \times B \vdash D(x) \text{ type} \qquad a : A, b : B \vdash \sigma(a, b) : D\langle a, b \rangle}{\begin{array}{c} x : A \times B \vdash d(x) : D(x) \\ a : A, b : B \vdash \sigma(a, b) \equiv d\langle a, b \rangle : D\langle a, b \rangle \end{array}}$$

- Set interpretation: $\times$
- Logical interpretation: $\wedge$

Type formers

Dependent sums

$$\frac{a : A \vdash B(a) \text{ type}}{\vdash \Sigma_{a:A} B(a) \text{ type}} \qquad \frac{\vdash a : A \qquad \vdash b : B(a)}{\vdash \langle a, b \rangle : \Sigma_{a:A} B(a)}$$

$$\frac{x : \Sigma_{a:A} B(a) \vdash D(x) \text{ type} \qquad a : A, b : B(a) \vdash \sigma(a,b) : D\langle a, b \rangle}{\begin{array}{c} x : \Sigma_{a:A} B(a) \vdash d(x) : D(x) \\ a : A, b : B(a) \vdash \sigma(a,b) \equiv d\langle a, b \rangle : D\langle a, b \rangle \end{array}}$$

- Set interpretation: $\cup_{a:A} B(a)$
- Logical interpretation: $\exists_{a:A} B(a)$

The surprising type former

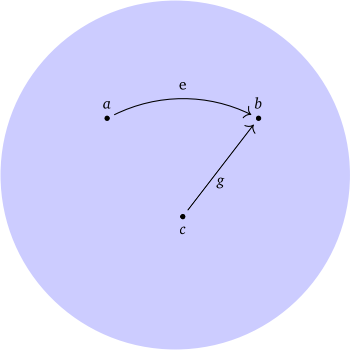
Identity type

$$\frac{\vdash A \text{ type} \qquad \vdash a, b : A}{\vdash a =_A b} \qquad \frac{\vdash A \text{ type} \qquad \vdash a : A}{\vdash \mathrm{refl}_a : a =_A a}$$

$$\frac{\vdash A \text{ type} \qquad x, y : A,\ p : x =_A y \vdash D(p) \text{ type} \qquad x : A \vdash \rho(x) : D(\mathrm{refl}_x)}{\begin{array}{c} x, y : A,\ p : x =_A y \vdash d(p) : D(p) \\ \qquad\qquad x : A \vdash \rho(x) \equiv d(\mathrm{refl}_x) : D(\mathrm{refl}_x) \end{array}}$$

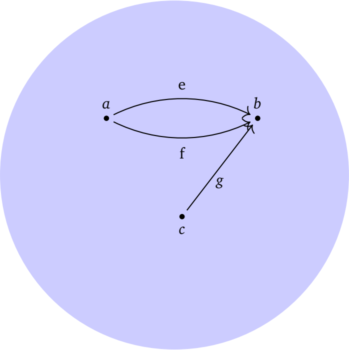
Homotopy type theory

- Equality is given inductively, just like the natural numbers.
- The **equality type** $a = b$ (for two terms $a, b : A$) is generated inductively by the *canonical term* $\mathrm{refl}_a : a = a$ for each term $a : A$.
  - Just as $\mathbb{N}$ is generated by the canonical elements $0 : \mathbb{N}$ and $Sn : \mathbb{N}$ for each $n : \mathbb{N}$.



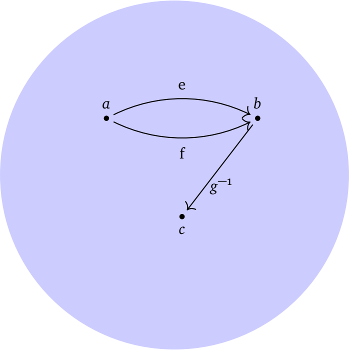
Homotopy type theory

- Equality is given inductively, just like the natural numbers.
- The **equality type** $a = b$ (for two terms $a, b : A$) is generated inductively by the *canonical term* $\mathrm{refl}_a : a = a$ for each term $a : A$.
  - Just as $\mathbb{N}$ is generated by the canonical elements $0 : \mathbb{N}$ and $Sn : \mathbb{N}$ for each $n : \mathbb{N}$.
- We can have equalities $e, f : a = b$.



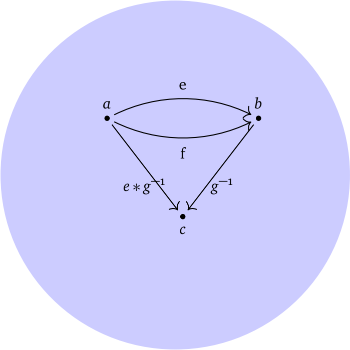
Homotopy type theory

- Equality is given inductively, just like the natural numbers.
- The **equality type** $a = b$ (for two terms $a, b : A$) is generated inductively by the *canonical term* $\mathrm{refl}_a : a = a$ for each term $a : A$.
  - Just as $\mathbb{N}$ is generated by the canonical elements $0 : \mathbb{N}$ and $Sn : \mathbb{N}$ for each $n : \mathbb{N}$.
- We can have equalities $e, f : a = b$.
- Equalities are invertible.



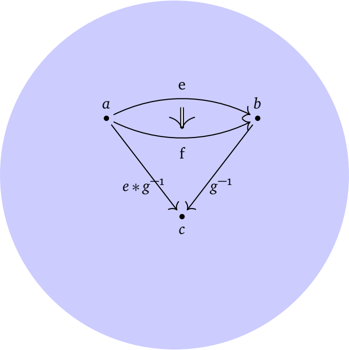
Homotopy type theory

- Equality is given inductively, just like the natural numbers.
- The **equality type** $a = b$ (for two terms $a, b : A$) is generated inductively by the *canonical term* $\mathrm{refl}_a : a = a$ for each term $a : A$.
  - Just as $\mathbb{N}$ is generated by the canonical elements $0 : \mathbb{N}$ and $Sn : \mathbb{N}$ for each $n : \mathbb{N}$.
- We can have equalities $e, f : a = b$.
- Equalities are invertible.
- Equalities are composable.



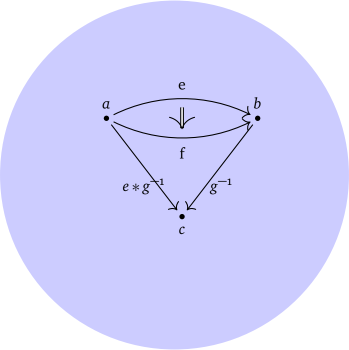
Homotopy type theory

- Equality is given inductively, just like the natural numbers.
- The **equality type** $a = b$ (for two terms $a, b : A$) is generated inductively by the *canonical term* $\mathrm{refl}_a : a = a$ for each term $a : A$.
  - Just as $\mathbb{N}$ is generated by the canonical elements $0 : \mathbb{N}$ and $Sn : \mathbb{N}$ for each $n : \mathbb{N}$.
- We can have equalities $e, f : a = b$.
- Equalities are invertible.
- Equalities are composable.
- There can be "higher" equalities.



Homotopy type theory

- Equality is given inductively, just like the natural numbers.
- The **equality type** $a = b$ (for two terms $a, b : A$) is generated inductively by the *canonical term* $\mathrm{refl}_a : a = a$ for each term $a : A$.
  - Just as $\mathbb{N}$ is generated by the canonical elements $0 : \mathbb{N}$ and $Sn : \mathbb{N}$ for each $n : \mathbb{N}$.
- We can have equalities $e, f : a = b$.
- Equalities are invertible.
- Equalities are composable.
- There can be "higher" equalities.
- This makes types behave like homotopy types or spaces.



Types as Kan complexes

We can interpret

- a type K as a *Kan complex* (space) $[K]$
- a term $k : K$ as a *point* of K
- a dependent type $x : B \vdash E(b)$ as a *Kan fibration* $[p] : [E] \to [B]$
- a dependent term $x : B \vdash e(b) : E(b)$ as a *section* $[e]$ of $[p]$
- a term $p : a =_K b$ as a path from a to b in K

Type formers in Martin-Löf type theory

Type former	Notation	canonical term
Dependent type	$x : A \;\vdash\; B(x)$	
Dependent term	$x : A \;\vdash\; b(x) : B(x)$	
Boolean type	Bool	$\top, \bot$
Natural numbers type	Nat	$0, sx$
Sum type	$\sum_{x:A} B(x)$	(a, b)
Product type	$\prod_{x:A} B(x)$	$\lambda(x : A).b$
Identity type	$x : A, y : A \vdash x = y$	$\mathrm{refl}_x : x = x$
Universe	Type	

Outline

Characterizing equalities

We can characterize the equalities in many type formers.

Theorem about equalities in $\mathbb{N}$

For $n, m : \mathbb{N}$, if $n \equiv m$, then $(n =_{\mathbb{N}} m) \simeq *$; otherwise $(n =_{\mathbb{N}} m) \simeq \emptyset$.

Theorem about equalities in $A \times B$

For $p, q : A \times B$,

$$(p =_{A \times B} q) \simeq (\pi_1 p =_A \pi_1 q) \times (\pi_2 p =_B \pi_2 q).$$

Theorem about equalities in $\Sigma_{a:A} B$

For $p, q : \Sigma_{a:A} B$,

$$(p =_{\Sigma_{a:A} B} q) \simeq \Sigma_{\alpha : \pi_1 p =_A \pi_1 q} \alpha^* \pi_2 p =_{B(\pi_1 q)} \pi_2 q.$$

Under-determined equalities

We could postulate:

Function extensionality

For $f, g : A \to B$, the function

$$(f =_{A \to B} g) \to \Pi_{a:A} f(a) =_B g(a)$$

is an equivalence.

Uniqueness of identity proofs

For $p, q : a =_A b$, we have a term of

$$p =_{a =_A b} q.$$

Univalence

For $A, B : U$, the function

$$(A =_U B) \to (A \simeq_U B)$$

is an equivalence.

Univalent foundations

- *Function extensionality* holds both in the set and the space models (and most other ones).
- *Uniqueness of identity proofs* holds in the set model, but not the space model.
- *Univalence* holds in the space model, but not in the set model.

Univalent foundations admits *univalence* as an axiom (which implies function extensionality).

The equality principle in type theory

Any predicate or construction that can be defined on terms of a type A is of the form $f : A \to B$.

- The predicate "G is an abelian group" is a function $Grp \to Prop$.
- Considering the lattice of subgroups of any group G produces a function $Grp \to Latt$.

Equality principle

We can prove:

$$\prod_{x,y:A} (x = y) \to \prod_{f:A\to B} \big(f(x) = f(y)\big)$$

The equality principle in type theory

Any predicate or construction that can be defined on terms of a type A is of the form $f : A \to B$.

- The predicate "G is an abelian group" is a function $Grp \to Prop$.
- Considering the lattice of subgroups of any group G produces a function $Grp \to Latt$.

Equality principle

We can prove:

$$\prod_{x,y:A}(x = y) \to \prod_{f:A\to B}\big(f(x) = f(y)\big)$$

Everything respects equality.

Back to the equivalence principle

Using univalence, we have:

Equality to equivalence principle

$$\prod_{x,y:A}(x = y) \to \prod_{B:A\to U}\big(B(x) \simeq B(y)\big)$$

For example:

- The predicate "G is an abelian group" is a function $Grp \to Prop$ which we can compose with the inclusion $Prop \hookrightarrow U$.
- Considering the lattice of subgroups of any group G produces a function $Grp \to Latt$, which we can compose with a forgetful functor $Latt \to U$.

Next time

We would like to prove an equivalence principle like

$$\prod_{G,H:Grp} (G \cong H) \to \prod_{B:Grp \to Latt} B(G) \cong B(H)$$

where $G \cong H$ is group isomorphism and $B(G) \cong B(H)$ is lattice isomorphism.

Next time

We would like to prove an equivalence principle like

$$\prod_{G,H:Grp} (G \cong H) \to \prod_{B:Grp\to Latt} B(G) \cong B(H)$$

where $G \cong H$ is group isomorphism and $B(G) \cong B(H)$ is lattice isomorphism.

To be continued...

Thank you!