Univalent foundations and the equivalence principle

Paige Randall North

16 October 2020
Outline

1. The equivalence principle
2. Univalent foundations
3. The equivalence principle in univalent foundations
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, \ldots
- *isomorphic* sets, groups, rings, \ldots
- *equivalent* categories
- *biequivalent* bicategories
- \ldots
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:

\[
\bullet \cong \bullet
\]
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:

\[
\begin{array}{ccc}
\bullet & \cong & \bullet \\
\end{array}
\]

\(\mathcal{C}\) has exactly 1 object.
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
  
  \( \text{The category } \mathcal{C} \text{ has exactly one object.} \)

  is not invariant under equivalence of categories.

- In general, referring to \textbf{equality of objects} breaks invariance, but...
How to break the equivalence principle for categories. . .

• Recall: the statement

\[ \text{The category } \mathcal{C} \text{ has exactly one object.} \]

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 dom(g) is equal to cod(f), then g \circ f exists.”
How to break the equivalence principle for categories. . .

• Recall: the statement

   The category $\mathcal{C}$ 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 $\text{dom}(g)$ is equal to $\text{cod}(f)$, 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 \( \text{dom}(g) = \text{cod}(f) \) 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 \(\text{dom}(g) = \text{cod}(f)\) 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 \(\text{id}_x \in A(x, x)\)
- ...
... and how to fix it.

Solution

Use a logic/language of **dependent sets**, in which \( \text{dom}(g) = \text{cod}(f) \) 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 \( \text{id}_x \in A(x, x) \)
- ...

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

1. The equivalence principle
2. Univalent foundations
3. The equivalence principle in univalent foundations
## Overview of types 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        | $\text{Bool}$  | $\top, \bot$  |
| Natural numbers type| $\text{Nat}$   | $0, sx$        |
| Sum type            | $\sum_{x:A} B(x)$ | $(a, b)$       |
| Product type        | $\prod_{x:A} B(x)$ | $\lambda(x : A).b$ |
| Identity type       | $x : A, y : A \vdash x = y$ | $\text{refl}(x) : x = x$ |
| Universe            | $\text{Type}$  |               |

### Curry-Howard Correspondence

We can interpret these types as propositions or sets.
Properties of the identity type

Induction principle for $a = b$

To define a function

$$f : \prod_{x,y:A} \prod_{p:x=y} C(x,y,p)$$

it suffices to specify its image on $(x, x, \text{refl}_x)$.

- **sym**: $\prod_{x,y:A} (x = y) \to (y = x)$
- **trans**: $\prod_{x,y,z:A} (x = y) \times (y = z) \to (x = z)$
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 \( \text{Grp} \to \text{Prop} \).
- Considering the lattice of subgroups of any group \( G \) produces a function \( \text{Grp} \to \text{Latt} \).

Equality principle

\[
\prod_{x,y:A} (x = y) \twoheadrightarrow \prod_{f:A \to B} (f(x) = f(y))
\]
Space interpretation

The identity type behaves like equality:

- reflexivity, symmetry, transitivity
- Everything respects equality

but more like paths in a space:

- Can iterate identity type
- Cannot show that any two identities are identical

Voevodsky Correspondence

We can interpret

- a type $K$ as a Kan complex $[K]$
- 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 of $[e]$ of $[p]$
- a term $p : a \to_K b$ as a path from $a$ to $b$ in $K$
The Univalence Axiom

There are two notions of ‘sameness’ between types:

- $A = B$
- $A \simeq B$ (functions $f : A \leftrightarrow B : g$ such that $fg = 1$ and $gf = 1$)

There is always a function

$$(A = B) \rightarrow (A \simeq B)$$

which is an equivalence in Kan complexes.
Outline

1. The equivalence principle
2. Univalent foundations
3. The equivalence principle in univalent foundations
Strategy

We always have a version of the equivalence principle:

Equality principle

\[ \prod_{x,y:A} (x = y) \to \prod_{f:A \to B} (f(x) = f(y)) \]

but we want better ones where we replace the ‘synthetic’ equality \( x = y \) with an ‘analytic’ equality \( x \cong y \) which depends on the type.

Strategy: prove that the function \( (x = y) \to (x \cong y) \) is an equivalence

Univalence principle

\( (x =_T y) \cong (x \cong_T y) \)

for a type \( T \) and appropriate \( \cong_T \). Then we will get:

Equivalence principle

\[ \prod_{x,y:A} (x \cong y) \to \prod_{f:A \to B} (f(x) = f(y)) \]
Contractible types, propositions and sets

- **A is contractible**

\[
isContra(A) \equiv \sum_{x:A} \prod_{y:A} y = x
\]

- **A is a proposition**

\[
isProp(A) \equiv \prod_{x,y:A} x = y
\]

- **A is a set**

\[
isSet(A) \equiv \prod_{x,y:A} isProp(x = y)
\]

\[
\text{Prop} \equiv \sum_{X: \text{Type}} isProp(X) \quad \text{Set} \equiv \sum_{X: \text{Type}} isSet(X)
\]
Contractible types, propositions and sets

- **$A$ is contractible**

  \[
  \text{isContr}(A) \equiv \sum_{x:A} \prod_{y:A} y = x
  \]

- **$A$ is a proposition**

  \[
  \text{isProp}(A) \equiv \prod_{x,y:A} \text{isContr}(x = y)
  \]

- **$A$ is a set**

  \[
  \text{isSet}(A) \equiv \prod_{x,y:A} \text{isProp}(x = y)
  \]

\[
\text{Prop} \equiv \sum_{X: \text{Type}} \text{isProp}(X) \quad \text{Set} \equiv \sum_{X: \text{Type}} \text{isSet}(X)
\]
### Univalence for Propositions and Sets

**Univalence for propositions**

\[ P \equiv_{\text{Prop}} Q \simeq P \leftrightarrow Q \]

**Univalence for sets**

\[ P \equiv_{\text{Set}} Q \simeq P \cong Q \]
In type theory, a monoid is a tuple \((M, \mu, e, \alpha, \lambda, \rho)\) where

1. \(M : \text{Set}\)
2. \(\mu : M \times M \rightarrow M\)
3. \(e : M\)
4. \(\alpha : \Pi_{(a,b,c:M)} \mu(\mu(a,b),c) = \mu(a,\mu(b,c))\)
5. \(\lambda : \Pi_{(a:M)} \mu(e,a) = a\)
6. \(\rho : \Pi_{(a:M)} \mu(a,e) = a\)
Monoids in type theory

In type theory, a monoid is a tuple \((M, \mu, e, \alpha, \lambda, \rho)\) where

1. \(M : \text{Set}\)
2. \(\mu : M \times M \to M\)
3. \(e : M\)
4. \(\alpha : \Pi_{(a,b,c:M)} \mu(\mu(a,b),c) = \mu(a,\mu(b,c))\)
5. \(\lambda : \Pi_{(a:M)} \mu(e,a) = a\)
6. \(\rho : \Pi_{(a:M)} \mu(a,e) = a\)

Why \(M : \text{Set}\)?
Monoids in type theory

In type theory, a monoid is a tuple \((M, \mu, e, \alpha, \lambda, \rho)\) where

1. \(M : \text{Set}\)
2. \(\mu : M \times M \to M\)
3. \(e : M\)
4. \(\alpha : \Pi_{(a,b,c:M)} \mu(\mu(a,b),c) = \mu(a,\mu(b,c))\)
5. \(\lambda : \Pi_{(a:M)} \mu(e,a) = a\)
6. \(\rho : \Pi_{(a:M)} \mu(a,e) = a\)

Why \(M : \text{Set}\)?

Abstractly, a monoid is a (dependent) pair \((\text{data},\text{proof})\) where

- \(\text{data}\) is 1.–3.
- \(\text{proof}\) is 4.–6.
Structure Identity Principle

Univalence for monoids

\[ M =_{\text{Monoid}} N \cong M \cong N \]

We also have univalence for other set-level structures (Coquand-Danielsson):

- groups, rings
- posets
- discrete fields
- sets with fixpoint operator
### Structure Identity Principle

#### Univalence for monoids

\[ M =_{\text{Monoid}} N \simeq M \cong N \]

We also have univalence for other set-level structures (Coquand-Danielsson):

- groups, rings
- posets
- discrete fields
- sets with fixpoint operator

What about **categories**?
Univalence for categories

We only have univalence for **univalent** categories: ones where the canonical function $A = B \rightarrow A \simeq B$ for objects $A, B : \mathcal{C}$ is an equivalence.

Here, the homsets are sets, and the type of objects will be groupoids.

Univalence for univalent categories

$\mathcal{C} =_{\text{UCat}} \mathcal{D} \simeq C \simeq D$

We also have univalence for other higher structures (Ahrens-North-Shulman-Tsementzis):

- bicategories, tricategories, etc
- double categories
- dagger categories
Further resources

- HoTT Reading Group, 10:30-12 on Wednesdays
- HoTT Book
  - [https://homotopytypetheory.org/book/](https://homotopytypetheory.org/book/)
- Learn how to write proofs in a computer!
  - [https://leanprover-community.github.io/learn.html](https://leanprover-community.github.io/learn.html)
  - (Number Game)
Thank you!