Univalent foundations and the equivalence principle

Benedikt Ahrens

Vladimir Voevodsky Memorial Conference
Institute for Advanced Study, Princeton, NJ
2018-09-12

What is a foundation of mathematics?

Voevodsky: a foundation of mathematics is specified by

1. Syntax for mathematical objects
2. Notion of proposition and proof
3. Interpretation of the syntax into world of mathematical objects

This talk: introduction to Voevodsky's univalent foundations

- Its syntax
- Propositions and proofs
- The equivalence principle in univalent foundations

Outline

Outline

Indiscernability of identicals

Indiscernability of identicals

$$x = y \quad \rightarrow \quad \forall P\,(P(x) \leftrightarrow P(y))$$

- Reasoning **in logic** is invariant under equality
- **In mathematics**, reasoning should be invariant under weaker notion of sameness!

The equivalence principle

Equivalence principle

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

Notion of sameness depends on the objects under consideration:

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

Violating the equivalence principle

We can easily **violate** this principle:

Exercise

Find a statement about categories that is not invariant under the equivalence of categories

Violating the equivalence principle

We can easily **violate** this principle:

Exercise

Find a statement about categories that is not invariant under the equivalence of categories

A solution

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

Maybe this statement is simply silly!

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.*

Makkai's goal

to devise a language in which only invariant properties and constructions can be expressed

Voevodsky's homotopy lambda calculus

> *[. . .] My homotopy lambda calculus is an attempt to create a system which is very good at dealing with equivalences. In particular it is supposed to have the property that given any type expression $F(T)$ depending on a term subexpression t of type T and an equivalence $t \to t'$ (a term of the type $Eq(T; t, t')$) there is a mechanical way to create a new expression F' now depending on t' and an equivalence between $F(T)$ and $F'(T')$ (note that to get F' one can not just substitute t' for t in F – the resulting expression will most likely be syntactically incorrect).*

Email from Vladimir Voevodsky to Daniel R. Grayson, Sept 2006

Outline

Overview of type theory

(Dependent) Type theory

- Is a (functional programming) language of types and terms
- Has infrastructure to write mathematical statements and proofs

Write $a : A$ to say that term a has type A (e.g., $1 : \mathsf{Nat}$)

Writing well-typed programs

In type theory, both the activities of

- implementing an algorithm
- proving a mathematical statement

are done by writing well-typed programs.

Dependent types and functions

Examples of dependent types

- $\mathsf{Vect}(A, n)$ — type of vectors of length $n : \mathsf{Nat}$ of elements in type A
- $\mathsf{GrpStr}(X)$ — type of group structures on a set X

Examples of dependent functions

$$\mathsf{zero} : \prod_{m:\mathsf{Nat}} \mathsf{Vect}(\mathsf{Nat}, m)$$

$$\mathsf{tail} : \prod_{n:\mathsf{Nat}} \mathsf{Vect}(A, 1+n) \to \mathsf{Vect}(A, n)$$

$$\mathsf{GrpStrTransfer} : \prod_{X,Y:\mathsf{Set}} (X \cong Y) \times \mathsf{GrpStr}(X) \to \mathsf{GrpStr}(Y)$$

This section

In this section, I give a brief overview of type theory:

- judgements
- inference rules
- derivations
- overview of the type constructions available in type theory

Syntax of type theory

Fundamental: **judgment**

$$\text{context} \vdash \text{conclusion}$$

Contexts & judgments

Γ	sequence of variable declarations $(x_1 : A_1), (x_2 : A_2(x_1)), \ldots, (x_n : A_n(\vec{x}_i))$
$\Gamma \vdash A$	A is well–formed **type** in context Γ
$\Gamma \vdash a : A$	**term** a is well-formed and of type A
$\Gamma \vdash A \equiv B$	types A and B are **convertible**
$\Gamma \vdash a \equiv b : A$	a is convertible to b in type A

$$(x : \mathsf{Nat}), (f : \mathsf{Nat} \to \mathsf{Bool}) \vdash f@x : \mathsf{Bool}$$

Type dependency

In particular: dependent type B over A

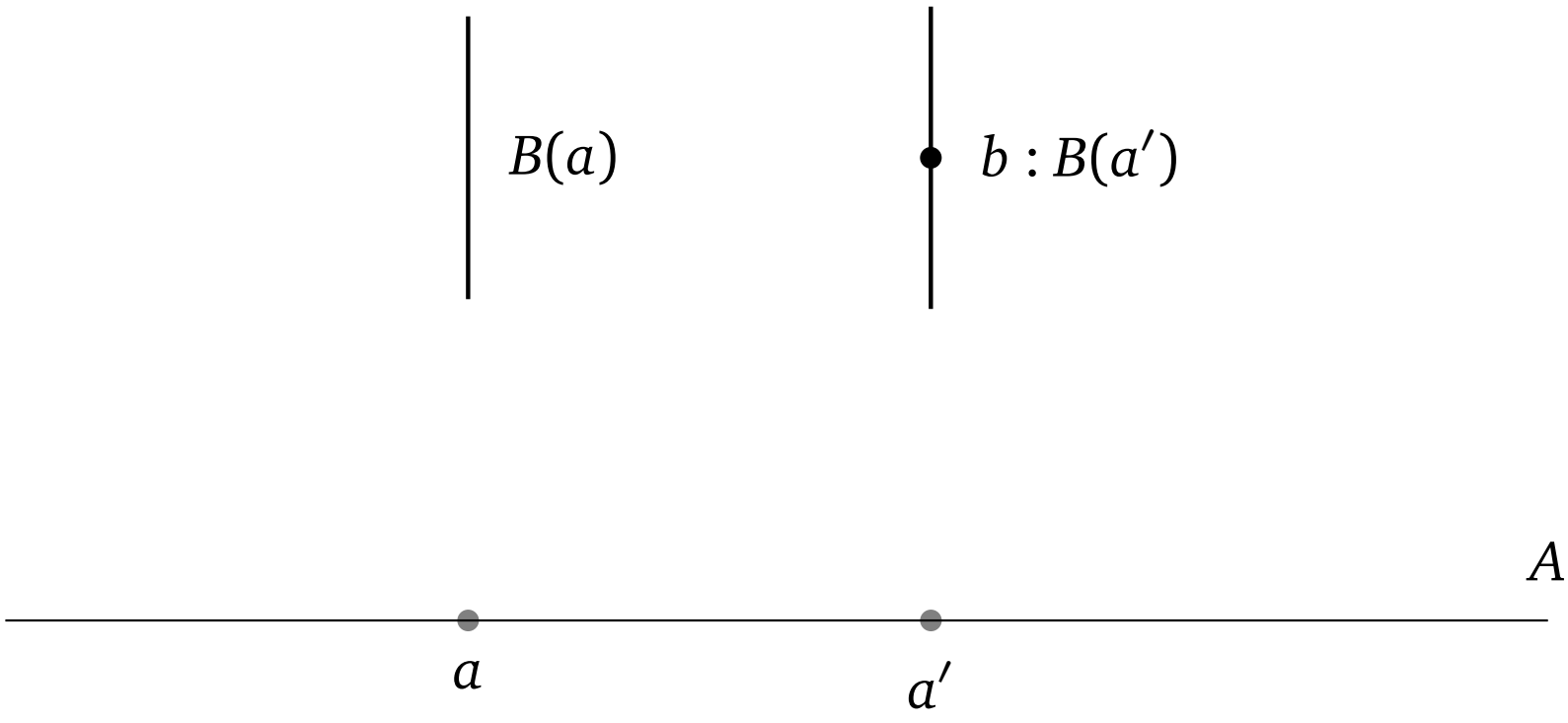
$$x : A \;\vdash\; B(x)$$

"family B of types indexed by A"

- A type can depend on several variables
- Example: type of vectors of length n with elements in type A

$$(A : \mathsf{Type}), (n : \mathsf{Nat}) \;\vdash\; \mathsf{Vect}(A, n)$$

Dependent types in pictures



Inference rules and derivations

An inference rule

is an implication of judgments,

$$\frac{J_1 \qquad J_2 \qquad \dots}{J}$$

e.g.,

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f@a : B} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash a : B}$$

Inference rules and derivations

An **inference rule**

is an implication of judgments,

$$\frac{J_1 \qquad J_2 \qquad \ldots}{J}$$

e.g.,

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f@a : B} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash a : B}$$

- Abbreviate rules to, e.g., " If $a : A$ and $A \equiv B$, then $a : B$".

Derivations

Example:

$$\dfrac{\dfrac{\dfrac{}{(f : A \to B) \vdash f : A \to B}}{(f : A \to B), (x : A) \vdash f : A \to B}}{(x : A), (f : A \to B) \vdash f : A \to B} \qquad \dfrac{\dfrac{}{(x : A) \vdash x : A}}{(x : A), (f : A \to B) \vdash x : A}$$

$$(x : A), (f : A \to B) \vdash f@x : B$$

Derivations

Example:

$$\dfrac{\dfrac{\dfrac{}{(f : A \to B) \;\vdash\; f : A \to B}}{(f : A \to B), (x : A) \;\vdash\; f : A \to B}}{(x : A), (f : A \to B) \;\vdash\; f : A \to B} \qquad \dfrac{\dfrac{}{(x : A) \;\vdash\; x : A}}{(x : A), (f : A \to B) \;\vdash\; x : A}$$

$$\dfrac{(x : A), (f : A \to B) \;\vdash\; f : A \to B \qquad (x : A), (f : A \to B) \;\vdash\; x : A}{(x : A), (f : A \to B) \;\vdash\; f@x : B}$$

Derivation

Term a is well-typed of type A in a context Γ if there is a derivation of the judgement $\Gamma \;\vdash\; a : A$

Interlude: Interpreting types as sets?

- Can interpret types and terms as sets
- $a : A$ is interpreted as $[a] \in [A]$

Differences between $a : A$ and $a \in A$

- Judgement $\Gamma \vdash a : A$ is **not** a statement that can be proved or disproved
- Term a does not exist independently of its type A
- A term has exactly one type (up to $\equiv$)

Declaring types & terms

The inference rules of type theory consist of

infrastructural rules: substitution, weakening,. . .

logical rules: to build new types and terms

Declaring types & terms

The inference rules of type theory consist of

infrastructural rules: substitution, weakening,. . .

logical rules: to build new types and terms

The logical rules mostly come in groups of 4 rules:

Formation	way to construct a type
Introduction	way(s) to construct **canonical terms** of that type
Elimination	ways to use a term of the introduced type to construct other terms
Computation	what happens when one does Introduction followed by Elimination

Universes

Universes

- There is a type Type. Its elements are types, written $A : \mathsf{Type}$.
- Using dependent function types (cf. later), the dependent type $x : A \vdash B$ can be considered as a function

$$\lambda x.B : A \to \mathsf{Type}$$

Universes

Universes

- There is a type Type. Its elements are types, written $A : \mathsf{Type}$.
- Using dependent function types (cf. later), the dependent type $x : A \vdash B$ can be considered as a function

$$\lambda x.B : A \to \mathsf{Type}$$

- Actually, hierarchy $(\mathsf{Type}_i)_{i \in I}$ to avoid paradoxes.
- But we ignore this here, and only write Type.

$$(A : \mathsf{Type}), (n : \mathsf{Nat}) \vdash \mathsf{Vect}(A, n) : \mathsf{Type}$$

The singleton type

Formation	way to construct a type
Introduction	way(s) to construct **canonical terms** of that type
Elimination	ways to use a term of the introduced type to construct other terms
Computation	what happens when one does Introduction followed by Elimination

Singleton type

Formation	1 is a type
Introduction	$\mathsf{t} : 1$
Elimination	If $x : 1$ and C is a type and $c : C$, then $\mathsf{rec}_1(C, c, x) : C$
Computation	$\mathsf{rec}_1(C, c, \mathsf{t}) \equiv c$

The type of dependent pairs $\sum_{x:A} B$

Formation If $x : A \vdash B$, then $\sum_{x:A} B(x)$ is a type

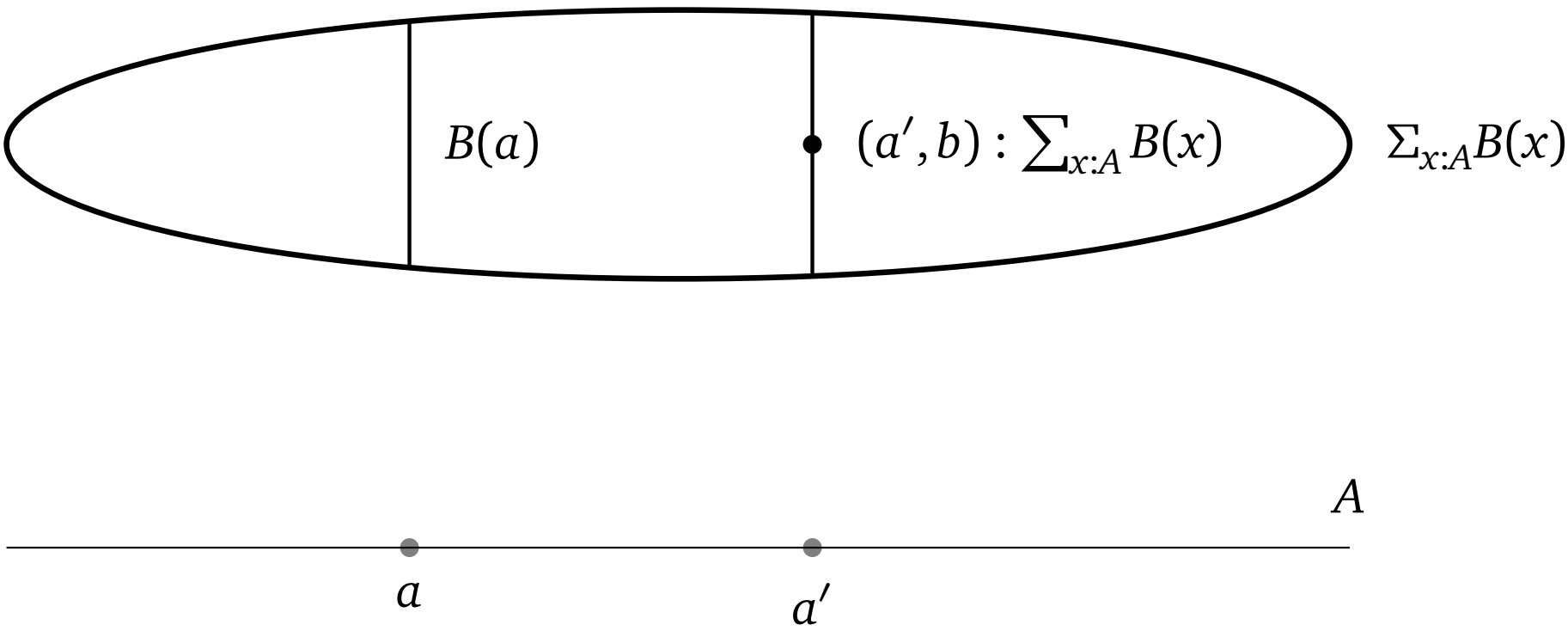
Introduction If $a : A$ and $b : B(a)$, then $\mathsf{pair}(a, b) : \sum_{x:A} B(x)$

Elimination If $t : \sum_{x:A} B$, then $\mathsf{fst}(t) : A$ and $\mathsf{snd}(t) : B(\mathsf{fst}(t))$

Computation $\mathsf{fst}(\mathsf{pair}(a, b)) \equiv a$ and $\mathsf{snd}(\mathsf{pair}(a, b)) \equiv b$

- Special case: $A \times B$

Σ-type in pictures



The type of dependent functions $\prod_{x:A} B$

Formation If $x : A \vdash B$, then $\prod_{x:A} B$ is a type

Introduction If $x : A \vdash b : B$, then $\lambda(x : A).b : \prod_{x:A} B$

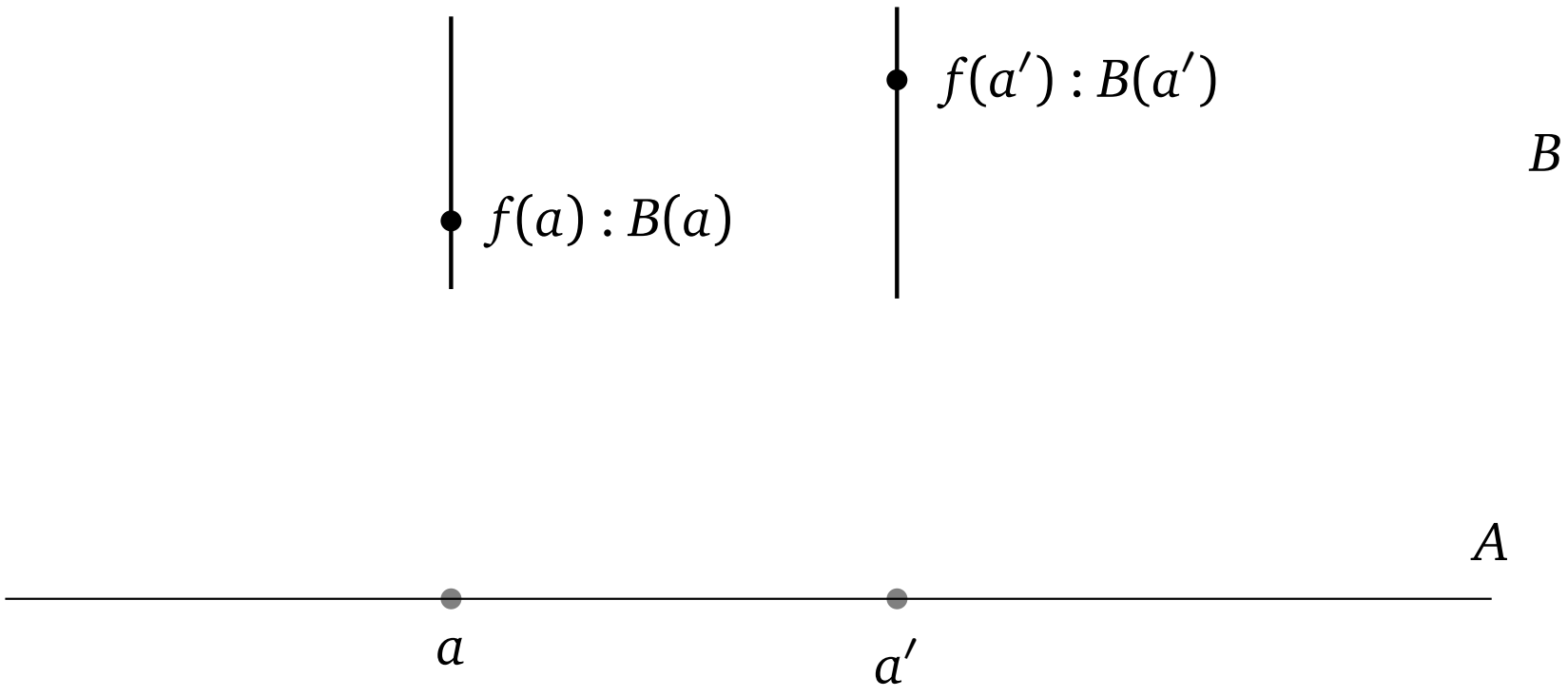
Elimination If $f : \prod_{x:A} B$ and $a : A$, then $f@a : B[x := a]$

Computation $(\lambda(x : A).b)@a \equiv b[x := a]$

- Special case: $A \to B$

A dependent function in pictures

$f : \prod_{x:A} B(x)$



The identity type

Formation If $a : A$ and $b : A$, then $\mathsf{Id}_A(a, b)$ is a type

Introduction If $a : A$, then $\mathsf{refl}(a) : \mathsf{Id}_A(a, a)$

Elimination If

$$(x, y : A), (p : \mathsf{Id}_A(x, y)) \vdash C(x, y, p)$$

and

$$(x : A) \vdash t(x) : C(x, x, \mathsf{refl}(x))$$

then

$$(x, y : A), (p : \mathsf{Id}_A(x, y) \vdash \mathsf{ind}_{\mathsf{Id}}(t; x, y, p) : C(x, y, p)$$

Computation ...

Some terms that can be defined

- $\mathsf{Id}_{\mathsf{Bool}}(\mathsf{false}, \mathsf{false})$
- $\mathsf{Id}_{\mathsf{Bool}}(\mathsf{snd}(\mathsf{t}, \mathsf{false}), \mathsf{false})$
- $\mathsf{sym} : \prod_{x,y:A} \mathsf{Id}(x,y) \;\to\; \mathsf{Id}(y,x)$
- $\mathsf{trans} : \prod_{x,y,z:A} \mathsf{Id}(x,y) \times \mathsf{Id}(y,z) \to \mathsf{Id}(x,z)$

Transport

$$\mathsf{transport}_{x,y} : \mathsf{Id}(x,y) \to \prod_{B:A\to\mathsf{Type}} \big(B(x) \longleftrightarrow B(y)\big)$$

Interpretation of identities as paths

Inhabitants of $\mathsf{Id}(a, a')$ behave like equality in many ways

- reflexivity, symmetry, transitivity
- transport

Inhabitants of $\mathsf{Id}(a, a')$ behave **un**like equality

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

Lack of uniqueness motivates interpretation of elements of $\mathsf{Id}_X(x, y)$ as **paths from** x **to** y **in** X.
A new notation, to reflect the new interpretation:

$$x \rightsquigarrow_X y$$

Overview of types in Martin-Löf type theory

Type former	Notation	(special case)
Inhabitant	$a : A$	
Dependent type	$x : A \vdash B(x)$	
Sigma type	$\sum_{x:A} B(x)$	$A \times B$
Product type	$\prod_{x:A} B(x)$	$A \to B$
Coproduct type	$A + B$	
Identity type	$a \rightsquigarrow_A b$	
Universe	Type	
Base types	Nat, Bool, $\mathsf{1}$, $\mathsf{0}$	

Outline

Contractible types, propositions and sets

- A is **contractible**

$$\mathsf{isContr}(A) :\equiv \sum_{x:A} \prod_{y:A} y \rightsquigarrow x$$

- A is a **proposition**

$$\mathsf{isProp}(A) :\equiv \prod_{x,y:A} x \rightsquigarrow y$$

- A is a **set**

$$\mathsf{isSet}(A) :\equiv \prod_{x,y:A} \mathsf{isProp}(x \rightsquigarrow y)$$

$$\mathsf{Prop} :\equiv \sum_{X:\mathsf{Type}} \mathsf{isProp}(X) \qquad \mathsf{Set} :\equiv \sum_{X:\mathsf{Type}} \mathsf{isSet}(X)$$

Equivalences

Definition

A map $f : A \to B$ is an **equivalence** if it has contractible fibers, i.e.,

$$\mathsf{isequiv}(f) \quad :\equiv \quad \prod_{b:B} \mathsf{isContr}\left(\sum_{a:A} f(a) \rightsquigarrow b\right)$$

The type of equivalences:

$$A \simeq B \quad :\equiv \quad \sum_{f:A\to B} \mathsf{isequiv}(f)$$

Transport revisited

$$\mathsf{transport}_{x,y} : (x \rightsquigarrow y) \to \prod_{B:A\to \mathsf{Type}} \big(B(x) \simeq B(y)\big)$$

The path type of pairs

Can construct equivalences

- for $s, t : A \times B$

$$\left(s \rightsquigarrow t\right) \simeq \left(\left(\mathsf{fst}(s) \rightsquigarrow \mathsf{fst}(t)\right) \times \left(\mathsf{snd}(s) \rightsquigarrow \mathsf{snd}(t)\right)\right)$$

- for $s, t : \sum_{x:A} B(x)$

$$\left(s \rightsquigarrow t\right) \simeq \left(\sum_{e:\mathsf{fst}(s) \rightsquigarrow \mathsf{fst}(t)} \mathsf{transport}^{B}\left(e, \mathsf{snd}(s)\right) \rightsquigarrow \mathsf{snd}(t)\right)$$

The path type of pairs

Can construct equivalences

- for $s, t : A \times B$

$$\left(s \rightsquigarrow t\right) \simeq \left(\left(\mathsf{fst}(s) \rightsquigarrow \mathsf{fst}(t)\right) \times \left(\mathsf{snd}(s) \rightsquigarrow \mathsf{snd}(t)\right)\right)$$

- for $s, t : \sum_{x:A} B(x)$

$$\left(s \rightsquigarrow t\right) \simeq \left(\sum_{e:\mathsf{fst}(s) \rightsquigarrow \mathsf{fst}(t)} \mathsf{transport}^{B}\left(e, \mathsf{snd}(s)\right) \rightsquigarrow \mathsf{snd}(t)\right)$$

Can other path types be characterized similarly?

Axioms to characterize some path types

Axiom of function extensionality for $f, g : A \to B$

$$\left(f \rightsquigarrow g\right) \simeq \left(\prod_{a:A} f(a) \rightsquigarrow g(a)\right)$$

Univalence axiom for $A, B : \mathsf{Type}$

$$\left(A \rightsquigarrow B\right) \simeq \left(A \simeq B\right)$$

More precisely: canonical map $\to$ is an equivalence.

Theorem (Voevodsky)

Univalence axiom implies function extensionality

Outline

Transport along isomorphism

Have:

$$\mathsf{transport}_{x,y} : (x \rightsquigarrow y) \to \prod_{B:A\to\mathsf{Type}} \big(B(x) \simeq B(y)\big)$$

Want:

$$\mathsf{transport}_{x,y} : (x \cong y) \to \prod_{B:A\to\mathsf{Type}} \big(B(x) \simeq B(y)\big)$$

Suffices:

$$(x \rightsquigarrow y) \;\simeq\; (x \cong y)$$

Transport along isomorphism

Have:

$$\mathsf{transport}_{x,y} : (x \rightsquigarrow y) \to \prod_{B:A\to\mathsf{Type}} \big(B(x) \simeq B(y)\big)$$

Want:

$$\mathsf{transport}_{x,y} : (x \cong y) \to \prod_{B:A\to\mathsf{Type}} \big(B(x) \simeq B(y)\big)$$

Suffices:

$$(x \rightsquigarrow y) \xrightarrow{\simeq} (x \cong y)$$

Monoids

Traditionally (in set theory), a monoid is a triple (M, μ, e) of

- a set M
- a multiplication $\mu : M \times M \to M$
- a unit $e \in M$

subject to the usual axioms: associativity, and left and right neutrality.

Monoids in type theory

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

1. $M : \mathsf{Set}$
2. $\mu : M \times M \to M$ (multiplication)
3. $e : M$ (neutral element)
4. $\alpha : \Pi_{(a,b,c:M)} \mu(\mu(a,b),c) \rightsquigarrow \mu(a,\mu(b,c))$ (associativity)
5. $\lambda : \Pi_{(a:M)} \mu(e,a) \rightsquigarrow a$ (left neutrality)
6. $\rho : \Pi_{(a:M)} \mu(a,e) \rightsquigarrow a$ (right neutrality)

Monoids in type theory

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

1. $M : \mathsf{Set}$
2. $\mu : M \times M \to M$ (multiplication)
3. $e : M$ (neutral element)
4. $\alpha : \Pi_{(a,b,c:M)} \mu(\mu(a,b),c) \rightsquigarrow \mu(a,\mu(b,c))$ (associativity)
5. $\lambda : \Pi_{(a:M)} \mu(e,a) \rightsquigarrow a$ (left neutrality)
6. $\rho : \Pi_{(a:M)} \mu(a,e) \rightsquigarrow a$ (right neutrality)

Why $M : \mathsf{Set}$?

Monoids in type theory

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

1. $M : \mathsf{Set}$
2. $\mu : M \times M \to M$ (multiplication)
3. $e : M$ (neutral element)
4. $\alpha : \Pi_{(a,b,c:M)} \mu(\mu(a,b),c) \rightsquigarrow \mu(a,\mu(b,c))$ (associativity)
5. $\lambda : \Pi_{(a:M)} \mu(e,a) \rightsquigarrow a$ (left neutrality)
6. $\rho : \Pi_{(a:M)} \mu(a,e) \rightsquigarrow a$ (right neutrality)

Why $M : \mathsf{Set}$?

Abstractly, a monoid is a (dependent) pair $(data, proof)$ where

- $data$ is a triple (M, μ, e) as above
- $proof$ is a triple (α, λ, ρ) saying that $(data)$ satisfy the usual axioms.

The type of monoids

- We want two monoids $(data, proof)$ and $(data', proof')$ to be the same if $data$ is the same as $data'$.
- This requires the type of $proof$ and $proof'$ to be a **proposition**.
- This is guaranteed when the underlying type M is a **set**.

Summarily:

$$\mathsf{Monoid} :\equiv \sum_{(M:\mathsf{Set})} \sum_{(\mu,e):\mathsf{MonoidStr}(M)} \mathsf{MonoidAxioms}(M,(\mu,e))$$

Can show

$$\mathsf{isProp}(\mathsf{MonoidAxioms}(M,(\mu,e)))$$

Monoid isomorphisms

Given monoids $\mathbf{M} \equiv (M, \mu, e, \alpha, \lambda, \rho)$ and $\mathbf{M}' \equiv (M', \mu', e', \alpha', \lambda', \rho')$, a **monoid isomorphism** is a bijection $f : M \cong M'$ preserving multiplication and neutral element.

Monoid isomorphisms

Given monoids $\mathbf{M} \equiv (M, \mu, e, \alpha, \lambda, \rho)$ and $\mathbf{M}' \equiv (M', \mu', e', \alpha', \lambda', \rho')$, a **monoid isomorphism** is a bijection $f : M \cong M'$ preserving multiplication and neutral element.

$$
\begin{aligned}
\mathbf{M} \rightsquigarrow \mathbf{M}' &\simeq (M, \mu, e) \rightsquigarrow (M', \mu', e') \\
&\simeq \sum_{p : M \rightsquigarrow M'} (\mathsf{transport}^{Y \mapsto (Y \times Y \to Y)}(p, \mu) \rightsquigarrow \mu') \\
&\qquad \times (\mathsf{transport}^{Y \mapsto Y}(p, e) \rightsquigarrow e') \\
&\simeq \sum_{f : M \cong M'} \left(f \circ m \circ (f^{-1} \times f^{-1}) \rightsquigarrow m'\right) \\
&\qquad \times (f \circ e \rightsquigarrow e') \\
&\simeq \mathbf{M} \cong \mathbf{M}'
\end{aligned}
$$

Transport along monoid isomorphism

We now have two ingredients:

1. $$(\mathbf{M} \rightsquigarrow \mathbf{M}') \simeq (\mathbf{M} \cong \mathbf{M}')$$

2. $$\mathsf{transport}_{\mathbf{M},\mathbf{M}'} : (\mathbf{M} \rightsquigarrow \mathbf{M}') \to \prod_{B:\mathsf{Monoid}\to\mathsf{Type}} \left(B(\mathbf{M}) \simeq B(\mathbf{M}')\right)$$

Composing these, we get

$$\mathsf{transport}_{\mathbf{M},\mathbf{M}'} : (\mathbf{M} \cong \mathbf{M}') \to \prod_{B:\mathsf{Monoid}\to\mathsf{Type}} \left(B(\mathbf{M}) \simeq B(\mathbf{M}')\right)$$

I.e., any structure on monoids that can be expressed in univalent type theory can be transported along isomorphism of monoids.

Equivalence principle for set-level structures

EP for set-level structures (Coquand&Danielsson)

For many set-level structures in univalent foundations, paths are isomorphisms.

Examples include:

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

What about **categories**?

Outline

EP for categories

Conjecture

For categories $\mathscr{C}$ and $\mathscr{D}$, the canonical map

$$(\mathscr{C} \rightsquigarrow \mathscr{D}) \;\to\; \mathsf{AdjEquiv}(\mathscr{C}, \mathscr{D})$$

is an equivalence.

EP for categories

Conjecture

For categories $\mathscr{C}$ and $\mathscr{D}$, the canonical map

$$(\mathscr{C} \rightsquigarrow \mathscr{D}) \quad \to \quad \mathsf{AdjEquiv}(\mathscr{C}, \mathscr{D})$$

is an equivalence.

Counter-example

Categories in univalent foundations

Definition

A **category** $\mathscr{C}$ is given by

- a **type** $\mathscr{C}_0$: Type of **objects**
- for any $a, b : \mathscr{C}_0$, a **set** $\mathscr{C}(a,b)$: Set of **morphisms**
- operations: identity & composition

$$1_a : \mathscr{C}(a,a)$$
$$(\circ)_{a,b,c} : \mathscr{C}(b,c) \times \mathscr{C}(a,b) \to \mathscr{C}(a,c)$$

- axioms: unitality & associativity

$$1 \circ f \rightsquigarrow f \qquad f \circ 1 \rightsquigarrow f \qquad (h \circ g) \circ f \rightsquigarrow h \circ (g \circ f)$$

From paths to isomorphisms

Definition (univalent category)

For a category $\mathscr{C}$ we define

$$\mathsf{idtoiso} : \prod_{a,b:\mathscr{C}_0} (a \rightsquigarrow b) \to \mathsf{iso}(a,b)$$

$$\mathsf{idtoiso}(a,a,\mathsf{refl}(a)) \quad :\equiv \quad 1_a$$

We call the category $\mathscr{C}$ **univalent** if, for any objects $a, b : \mathscr{C}_0$,

$$\mathsf{idtoiso}_{a,b} : (a \rightsquigarrow b) \to \mathsf{iso}(a,b)$$

is an equivalence of types.

Examples of univalent categories

- $\underline{\mathrm{Set}}$
- Groups, rings, . . . (Structure Identity Principle)
- Functor category $[\mathscr{C}, \mathscr{D}]$, if $\mathscr{D}$ is univalent
- Full subcategories of univalent categories

More examples of univalent categories

- A preorder is univalent iff it is antisymmetric
- If X is of h-level 3, then there is a univalent category with X as objects and $\hom(x,y) \;:\equiv\; (x \rightsquigarrow y)$
- If $\mathscr{C}$ is univalent, then the category of cones of shape $F : \mathscr{J} \to \mathscr{C}$ is
  - $\rightsquigarrow$ limits (limiting cones) in a univalent category are unique **up to paths**

Non-univalent categories

- Any "chaotic" category $\mathscr{C}$ with $\mathscr{C}(x,y) \simeq 1$, for $\mathscr{C}_0$ not a proposition

- Any chaotic category $\mathscr{C}$ with an object $c : \mathscr{C}_0$ is **equivalent** to the terminal category $\mathbf{1}$
  - $\rightsquigarrow$ a category can be equivalent to a univalent one without being univalent itself

(Adjoint) equivalence of categories

An **equivalence** $\mathscr{C} \simeq \mathscr{D}$ is given by

- a functor $F : \mathscr{C} \to \mathscr{D}$
- a functor $G : \mathscr{D} \to \mathscr{C}$
- a natural isomorphism $\eta : 1_{\mathscr{C}} \xrightarrow{\cong} GF$
- a natural isomorphism $\epsilon : FG \xrightarrow{\cong} 1_{\mathscr{D}}$

(F, G, η, ϵ) is an adjoint equivalence if F and G form an adjunction.

(Adjoint) equivalence of categories

An **equivalence** $\mathscr{C} \simeq \mathscr{D}$ is given by

- a functor $F : \mathscr{C} \to \mathscr{D}$
- a functor $G : \mathscr{D} \to \mathscr{C}$
- a natural isomorphism $\eta : 1_{\mathscr{C}} \xrightarrow{\cong} GF$
- a natural isomorphism $\epsilon : FG \xrightarrow{\cong} 1_{\mathscr{D}}$

(F, G, η, ϵ) is an adjoint equivalence if F and G form an adjunction.

Theorem

*For **univalent** categories $\mathscr{C}$ and $\mathscr{D}$, the canonical map*

$$(\mathscr{C} \rightsquigarrow \mathscr{D}) \;\to\; \mathsf{AdjEquiv}(\mathscr{C}, \mathscr{D})$$

is an equivalence.

Conclusions

- In univalent type theory, EP can be proved for various set-level mathematical structures
- EP only holds for *univalent* categories
- Pseudo-conjecture inspired by the case of categories: EP holds for higher-categorical "univalent" structures
- Voevodsky's Homotopy Type System, and variants of it (e.g., Capriotti et al.'s Two-Level Type Theory) allow leaving the univalent world

Conclusions

- In univalent type theory, EP can be proved for various set-level mathematical structures
- EP only holds for *univalent* categories
- Pseudo-conjecture inspired by the case of categories: EP holds for higher-categorical "univalent" structures
- Voevodsky's Homotopy Type System, and variants of it (e.g., Capriotti et al.'s Two-Level Type Theory) allow leaving the univalent world

Thanks for your attention!