Univalent foundations of mathematics

Benedikt Ahrens

Outline

Outline

What is a foundation of mathematics?

- syntax for mathematical objects
- notion of proposition and proof
- interpretation of the syntax into the world of mathematical objects

In this talk, I present one such foundation: univalent foundations (a.k.a univalent type theory)

Moving from classical foundations to univalent foundations

- Mathematics is the study of structures on sets and their higher analogs.
- Set-theoretic mathematics constitutes a subset of the mathematics that can be expressed in univalent foundations.
- Classical mathematics is a subset of univalent mathematics consisting of the results that require LEM and/or AC among their assumptions.

see Voevodsky, Talk at HLF, Sept 2016

Outline

Univalent foundations

Syntax/Language

- Language of **dependent types**, a.k.a., a type theory
- Logic and mathematical constructions are treated uniformly
- Developed by Per Martin-Löf starting from the 1970's

Semantics/Models

- Semantics in **spaces** (simplicial sets), developed by Vladimir Voevodsky from 2006 on
- Other interpretations are possible, e.g., in sets

Syntax of type theory

Fundamental: **judgment**

context $\vdash$ 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

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

Rules and derivations

- A **rule** is an implication of judgments,

  $$J_1 \wedge J_2 \wedge \ldots \wedge J_n \quad \Longrightarrow \quad J$$

  e.g.,

  $$\Gamma \vdash a \equiv b : A \quad \Longrightarrow \quad \Gamma \vdash b \equiv a : A$$

  (often written as inference rule with horizontal bar)

- We sometimes omit the context when writing judgments.

- We abbreviate the above to
  If $a \equiv b$, then $b \equiv a$.

- There is a different equality (**identity**) that can be proved or disproved, see later.

About well-typedness

There is no way to derive a typing judgment of the form

$$(b : \mathsf{Bool}), (f : \mathsf{Nat} \to A) \vdash f(b) : \ ??$$

We are hence not allowed to write $f(b)$ here.

Some differences to set-theoretic membership

- the judgment $a : A$ is **not** a statement that can be proved or disproved
- term a does not exist independently of the type A
- a valid term has exactly one type up to $\equiv$

Declaring types & terms

Any type (and corresponding term) construction is declared by giving 4 (groups of) rules:

Formation	a way to construct new types
Introduction	ways to construct terms of these types
Elimination	ways to use them to construct other terms
Computation	what happens when one does Introduction followed by Elimination

A singleton type

Formation $\mathtt{1}$ is a type

Introduction $\mathtt{t} : \mathtt{1}$

Elimination If A is a type and $a : A$ and $x : \mathtt{1}$, then $r(A, a, x) : A$

Computation $r(A, a, \mathtt{t}) \equiv a$

Interpretation in sets

a one-element set, $\mathtt{t} \in \mathtt{1}$

The type of pairs $A \times B$

Formation If A and B are types, then $A \times B$ is a type

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

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

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

Interpretation in sets

Cartesian product of sets A and B

The type of functions $A \to B$

Formation If A and B are types, then $A \to B$ is a type

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

Elimination If $f : A \to B$ and $a : A$, then $f(a) : B$

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

- $\lambda x.b$ corresponds to $x \mapsto b(x)$
- **Substitution** $b[x := a]$ is built-in
- Example: $\emptyset \vdash \lambda x.x^2 : \mathsf{Nat} \to \mathsf{Nat}$

Interpretation in sets

Set of functions from A to B

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 Booleans of length n

$$n : \mathsf{Nat} \vdash \mathsf{Vec}(n) \quad (= \mathsf{Bool}^n)$$

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

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

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

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

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

- The case $A \to B$ is a special case, where B does not depend on $x : A$

Interpretation in sets

The product $\prod_{(x:A)} B$

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

Formation If $x : A \vdash B(x)$, 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 $\mathsf{snd}(\mathsf{pair}(a, b)) \equiv b$

- The case $A \times B$ is a special case, where B does not depend on $x : A$

Interpretation in sets

The disjoint union $\coprod_{x:A} B$

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

Computation ...

We also write $a =_A b$ and $a = b$ for $\mathsf{Id}_A(a, b)$

Interpretation in sets

Equality $a = b$

More about identities

Can construct terms of type

- $(a = b) \;\to\; (b = a)$
- $(a = b) \times (b = c) \;\to\; (a = c)$
- $B(a) \times (a = b) \;\to\; B(b)$

Can **not** construct a term UIP of type

$$(x : A), (p : x = x) \vdash UIP : p =_{(x=x)} \mathsf{refl}_x$$

but can construct a term

$$(x : A), (p : \sum_{y:A} x = y) \vdash contr : p = \mathsf{pair}(x, \mathsf{refl}_x)$$

Interpreting types as sets

Syntax	Set interpretation
A	set A
$a : A$	$a \in A$
$A \times B$	cartesian product
$A \to B$	set of functions $A \to B$
$A + B$	disjoint union $A \amalg B$
$x : A \vdash B(x)$	family B of sets indexed by A
$\sum_{(x:A)} B(x)$	disjoint union $\amalg_{x:A} B(x)$
$\prod_{(x:A)} B(x)$	dependent function
$\mathsf{Id}_A(a, b)$	equality $a = b$

Outline

Interpreting types as propositions

Syntax	Logic
A	proposition A
$a : A$	a is a proof of A
$A \times B$	$A \wedge B$
$A \to B$	$A \Rightarrow B$
$A + B$	$A \vee B$
$x : A \vdash B(x)$	predicate B on A
$\sum_{(x:A)} B(x)$	$\exists x \in A, B(x)$
$\prod_{(x:A)} B(x)$	$\forall x \in A, B(x)$
$\mathsf{Id}_A(a, b)$	equality $a = b$

- The connectives $\vee$ and $\exists$ thus obtained behave constructively, not classically.
- One can also obtain the classical variants, see HoTT book (references)

Logic in type theory

Curry-Howard isomorphism resp. Brouwer-Heyting-Kolmogorov interpretation:

- propositions are types
- proofs of P are terms of type P

Hence

- In principle, all types could be called propositions.
- To prove a proposition P means to construct a term of type P.
- In UF, only some types are called ‘propositions’, cf later.

Outline

Types are ω-groupoids

Garner, van den Berg

$$(A, \mathsf{Id}_A, \mathsf{Id}_{\mathsf{Id}_A}, \ldots)$$

forms ω-groupoid, i.e., groupoid laws hold up to "higher" identities

- gives rise to model of type theory in simplicial sets (Voevodsky)
- this model motivates (and justifies) the univalence axiom, cf later

Interpreting types as simplicial sets

Syntax	Simpl. set interpretation
$(A, \mathsf{Id}_A, \mathsf{Id}_{\mathsf{Id}_A}, \ldots)$	Kan complex A
$a : A$	$a \in A_0$
$A \times B$	binary product
$A \to B$	space of maps
$A + B$	binary coproduct
$x : A \vdash B(x)$	fibration $B \to A$ with fibers $B(x)$
$\sum_{(x:A)} B(x)$	total space of fibration $B \to A$
$\prod_{(x:A)} B(x)$	space of sections of fibration $B \to A$

Interpreting types as topological spaces?

Intuition

The Quillen equivalence between simplicial sets and topological spaces gives rise to an intuition of 'types as (topological) spaces'.

- $a =_A b$ as the space of paths from a to b in A
- Failure of UIP says that one can have non-trivial loop spaces
- The space of paths in A with one endpoint fixed to $a \in A$ is contractible: any such path is homotopic to the constant path on a

No interpretation in topological spaces

It seems impossible to give a formal interpretation of type theory in the category of topological spaces.

Contractible types, propositions and sets

Definitions:

- A is **contractible** if we can construct a term of type

$$\mathsf{isContr}(A) \overset{\text{def}}{=} \sum_{(x:A)} \prod_{(y:A)} y = x$$

- A is a **proposition** if $\prod_{(x\ y:A)} x = y$ is inhabited
- A is a **set** if, for any $x, y : A$, $x = y$ is a proposition

These are just the first instances of a recursive definition of **homotopy level of a type**.

Equivalences

Definition

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

$$\mathsf{isequiv}(f) \overset{\text{def}}{=} \prod_{b:B} \mathsf{isContr}\left(\sum_{a:A} f(a) = b\right)$$

The type of equivalences:

$$A \simeq B \overset{\text{def}}{=} \sum_{f:A\to B} \mathsf{isequiv}(f)$$

Characterizing some identity types

Can construct equivalences (i.e., terms of type)

- for $\mathsf{pair}(a, b) : A \times B$,

$$\Big(\mathsf{pair}(a, b) = \mathsf{pair}(a', b')\Big) \simeq \Big((a = a') \times (b = b')\Big)$$

- for $f, g : A \to B$

$$\Big(f = g\Big) \simeq \Big(\prod_{a:A} f(a) = g(a)\Big)$$

- $\ldots$

Universes

Universes

There is also a type $\mathcal{U}$ that contains all types, i.e., $A : \mathcal{U}$.

- Actually, hierarchy $(\mathcal{U}_i)_{i \in I}$ to avoid paradoxes.
- a dependent type
  $$x : A \vdash B : \mathcal{U}$$
  can be considered as a function
  $$\lambda x.B : A \to \mathcal{U}$$

Question

What is

$$\mathsf{Id}_{\mathcal{U}}(A, B) \quad ?$$

Voevodsky's Univalence Axiom

Answer 1

$$\mathsf{univalence} : (A =_{\mathcal{U}} B) \simeq (A \simeq B)$$

More controlled:

Answer 2

Define

$$\mathsf{idtoeqv} : \prod_{A,B:\mathcal{U}} (A = B) \to (A \simeq B)$$

$$\mathsf{refl}_A \mapsto \mathsf{id}_A$$

$$\mathsf{univalence} : \prod_{A,B:\mathcal{U}} \mathsf{isequiv}(\mathsf{idtoeqv}_{A,B})$$

Summary: Univalent Foundations

- A language of **dependent types**, a.k.a. a **type theory**
- With an interpretation in spaces (precisely: Kan complexes)

Type theory	Interpretation
A type	space A
$a : A$ (term a of type A)	point a in space A
$f : A \to B$	map from A to B
$p : a =_A b$	path (1-morphism) from a to b in A
$\alpha : p =_{a =_A b} q$	homotopy from p to q in A

- Universe of **sets** given by **discrete spaces**
- Logic and mathematical constructions are treated uniformly in UF

Outline

Computer theorem proving

- Type theory is particularly well-suited as a basis for computer proof assistants.
- Several such proof assistants based on type theory exist (Coq, Agda).
- A new proof assistant is developed specifically to integrate Voevodsky's Univalence Axiom natively. It is based on an interpretation of type theory in **cubical** sets (Bezem, Coquand, and Huber).

Higher Inductive Types

- The type constructors presented so far do not allow the construction of a type with ‘non-trivial homotopy’.
- The extension of type theory with ‘Higher Inductive Types’ (homotopy pushouts) allows the construction of such types, for instance the circle and the torus.
- Assuming the existence of such types, one can reason about their homotopy groups. Various homotopy groups have been computed in HoTT.

Equivalence principle

- The equivalence principle (EP) says that reasoning in mathematics should be invariant under an appropriate notion of equivalence, e.g., under isomorphism of groups, equivalence of categories, etc.
- EP is generally false in set-theoretic foundations, e.g., $1 \in \mathsf{Nat}$ is not invariant under isomorphism of sets.
- EP has been proved in UF for many mathematical structures, such as groups, fields, categories,…

Some references

Syntax of type theory and the Univalence Axiom

- Univalent Foundations Program: Homotopy Type Theory—Univalent Foundations of Mathematics (freely—as in speech—available online)

Interpretation of type theory:

- Bezem, Coquand, Huber: *A model of type theory in cubical sets*
- Gambino, van den Berg: *Types are weak ω-groupoids*
- Kapulkin, Lumsdaine: *The Simplicial Model of Univalent Foundations (after Voevodsky)*

Proof assistants:

- `http://github.com/UniMath/UniMath`
- `http://github.com/HoTT/HoTT`
- `http://github.com/mortberg/cubicaltt`

More on
`https://ncatlab.org/homotopytypetheory/show/References`