Univalent Type Theory and Modular Formalization of Mathematics

Thierry Coquand

Big Proof 2017, Newton Institute, Cambridge

Content

In the first part, I will try to compare

-set theory

-simple type theory

-(univalent) type theory

and especially the status of *extensionality* in these systems

Main point: *extensionality* is important for *modularity*

The second part will be about the «justification» of extensionality

⟪Simple⟫ type theory

1940 Church *A Formulation of the Simple Theory of Types*

Extremely simple and natural

A type $bool$ as a type of ⟪propositions⟫

A type I for ⟪individuals⟫

Function type $A \to B$ and we can add product type $A \times B$

Natural semantics of *types* as *sets*

Functions in simple type theory

In set theory, a function is a *functional graph*

In (this version of) type theory, a function is given by an *explicit definition*

If $t : B$, we can introduce f of type $A \to B$ by the definition

$$f\ x = t$$

$f\ a$ «reduces» to $(a/x)t$ if a is of type A

We use the notation $f\ a$ for $f(a)$, which comes from combinatory logic

The type $A \to B$ is «abstract»: we don't state that a function is a graph

Functions in simple type theory

We have two notions of function

-*functional graph*

-*function explicitly defined* by a term

What is the connection between these two notions?

Church introduces a special operation $\iota x.P(x)$ and the «axiom of description»

If $\exists! x : A.P(x)$ then $P(\iota x.P(x))$

Rules of equality

Equality can be specified by the following purely logical rules

(1) $a =_A a$

(2) if $a_0 =_A a_1$ and $P(a_0)$ then $P(a_1)$

Given (2), the law (1) is equivalent to the fact that $a_0 =_A a_1$ is implied by

$\forall(P : A \to bool)\ P(a_0) \to P(a_1)$

Church uses this to *define* $a_0 =_A a_1$

Equality in mathematics

The *first* axiom of set theory is the axiom of *extensionality* stating that two sets are equal if they have the same element

In Church's system we have two forms of the extensionality axiom

(1) two equivalent propositions are equal

$$((P \to Q) \wedge (Q \to P)) \;\to\; P =_{bool} Q$$

(2) two pointwise equal functions are equal

$$(\forall (x : A)\ f\ x =_B g\ x) \quad \to \quad f =_{A \to B} g$$

The univalence axiom is a generalization of (1)

Equality in simple type theory

Henkin and P. Andrews (generalizing some previous work of Tarski for propositional logic) reformulated the rules of simple type theory by taking equality as the only logical primitive

E.g. $\forall(x : A)\ P\ x$ is defined as

$$\lambda(x : A)\ P\ x \quad =_{A \to bool} \quad \lambda(x : A)\ \mathsf{true}$$

where true can be defined as

$$\lambda(x : bool)\ x \quad = \quad \lambda(x : bool)\ x$$

Equality in mathematics and modularity

These two axioms of extensionality are important for *modularity* of reasoning

If we prove something about a function f and another function g is pointwise equal to f we can replace f by g and conclude that the property also holds for g

Similarly if we prove P logically equivalent to Q, we can replace P by Q in any context

Equality in mathematics and modularity

Dependent type theory (without univalence) does not have these principles

However, propositional extensionality holds for propositions represented by Booleans (decidable propositions)

The formalization of the 4 color theorem in dependent type theory (and of Feit-Thompson) uses heavily such a restriction to decidable propositions

Also it relies on the fact that the objects are first-order, so that the problems coming from lack of function extensionality are not too serious

E.g. for functions of *finite* domain, one represents a function by its graph and equality of graphs corresponds to extensional equality of functions

Type theory and set theory

In his 1931 paper

On Formally Undecidable Propositions of Principia Mathematica and Related Systems I

Gödel uses a restriction of this system with only the types

$1 = I$

$2 = I \to bool$

$3 = (I \to bool) \to bool$

$\ldots$

Type theory and set theory

The *atomic formula* are of the form $b\ u$ where b of type $n+1$ and u of type n

This can be read as: u belongs to the class b

If we start from I empty collection, we get a cumulative hierarchy

Type theory and set theory

Extensionality axiom

$(\forall x_n\ (a\ x_n \leftrightarrow b\ x_n)) \ \rightarrow\ a =_{n+1} b$

《A class is completely determined by its elements》

Neither pairing, nor function type

-pairs can be defined (N. Wiener 1914, Kuratowski) 《variables for binary or n-ary functions (relations) are superflous as basic signs》

-functions can be defined as functional graphs

Type theory and set theory

In set theory, we usually start with I empty collection

Transfinite iteration and cumulativity

《Types》 become ordinals

We can quantify over variable ranging over all 《types》

An infinite type of individuals is obtained at stage ω

Type theory and set theory

«It is a pity that a system such as Zermelo-Fraenkel set theory is usually presented in a purely formal way, because the *conception* behind it is quite straightforwardly based on type theory. One has the concept of an arbitrary *subset of a given domain* and that the collection of *all subsets* of the given domain can form a new domain (of the next type!). Starting with a domain of individuals (possibly empty), this process of forming subsets is then iterated into the *transfinite*. Thus, each set has a type (or *rank*), given by the ordinal number of the stage at which it is first to be found in the iteration.»

Dana Scott, *A type-theoretical alternative to ISWIM, CUCH, OWHY*, 1969

Note that LCF was introduced to formalize the system described in this paper

Description of mathematical objects and collections

We will analyse next how to represent ⟪collections of collections⟫

Collections of mathematical structures

When two structures should be identified?

The answer provides a *new* and important modularity principle

This principle is closely connected to what is described in *Types, abstraction and parametric polymorphism*, J.R. Reynolds and works of Morris, Liskove and Zilles, ... on the importance of representation indenpendence results for modular development of programs

Description of mathematical objects and collections

Algebraic structures, ordered structures, e.g. groups, rings, lattices

Set equipped with some operations and/or relations satisfying some properties

An identification of two structures is provided by an *isomorphism*

At this level one can talk about ⟪initial structures⟫

Uniqueness up to isomorphism

This is the level considered by Bourbaki in his *théorie des structures*

Description of mathematical objects

Two isomorphic groups G and H satisfy the same «structural » properties

If G is abelian, so is H

If G is solvable, so is H

But we can have $\pi \in G$ and $\pi \notin H$

Transportable properties (Bourbaki)

«When two relations have the same structure, their logical properties are identical, except such as depend upon the membership of their fields»

Russell «relation arithmetic» (1900)

Description of mathematical objects

Next level: collection of all groups, or all sets (at a fixed universe)

When are two such collections considered to be the ⟪same⟫?

Description of mathematical objects

Let B be a given set

In mathematics the two collections SET^B and SET/B should be ⟪identified⟫

They satisfy the same ⟪structural⟫/transportables properties

SET^B contains families of sets $X_b,\ b \in B$

SET/B contains $Y, f \in B^Y$, functions of codomain B

Description of mathematical objects

We have two canonical maps $F : \mathsf{SET}^B \to \mathsf{SET}/B$ and $G : \mathsf{SET}/B \to \mathsf{SET}^B$

$F(X) = \Sigma(b : B)X_b, \pi_1$

$G(Y, f) = (f^{-1}(b))_{b \in X}$

$G(F(X))$ and X are only isomorphic (and not equal as sets in general)

$G(F(X))_b = \{b\} \times X_b$

The two collections (groupoids) SET^B and SET/B are *equivalent*

F and G do *not* define an isomorphism

Collapsing

We get a *new* way to identify collections

The collection of all linear orders with 27 elements in a given Grothendieck universe $\mathcal{U}$ is not an element of $\mathcal{U}$

But it can be *identified* with the groupoid with one object and one morphism

Mathematics is the art of giving the same name to different things. It is enough that these things, though differing in matter, should be similar in form, to permit of their being, so to speak, run in the same mould. When language has been well chosen, one is astonished to find that all demonstrations made for a known object apply immediately to many new objects: nothing requires to be changed, not even the terms, since the names have become the same.

Description of mathematical objects

This *natural* stratification of collection of mathematical objects is not *directly* represented in set theory

A (Grothendieck) universe is a set as any other set with its notion of equality

Similarly for a collection of structures

Bourbaki has a characterisation of transportable properties for his notion of structure

Such a description for the next level is much more complex

Description of mathematical objects

At the next level we have 2-*groupoids*

Then n-groupoids, then ∞-groupoids

More and more complex notions of equivalences/identifications

Less and less clear when a property is transportable along equivalences

This can be done in an uniform way in dependent type theory extended with the univalence axiom

Universes

Simple type theory is elegant but presents unnatural limitations

First, we cannot express the notion of ⟪arbitrary⟫ structures

⟪Let X be a type, then $\ldots$⟫

In set theory, we can quantify over all sets

In type theory, one can introduce variables ranging over types (already suggested in R. Gandy's 1952 PhD thesis)

But we cannot form the *type* of all given groups or rings

In set theory, *Grothendieck universes* are used to form a collection of mathematical structures as a set

Universes

In type theory, a universe is a type the elements of which are types

Notion already present in the system AUTOMATH (N.G. de Bruijn)

We can then use sigma types to represent a type of structures, e.g.

$\Sigma(X : U)\ \ X \times (X \to X \to X)$

This introduces dependent types, e.g. one can form the type $N \to U$

Universes

The same limitation of simple type theory holds for the notion of elementary topos

This limitation holds also for the notion of *sheaves*

As soon as one wants to describe a ≪sheaf≫ of *structures* one needs to replace the notion of sheaf by the notion of *stack*

For instance if we let $F(V)$ be the collection of sheaves over V

We have a natural restriction map $F(V) \to F(W)$ for $W \subseteq V$

This does *not* define a sheaf but a stack: 3 by 3 condition for glueing and glueing is only unique *up to isomorphism*

Dependent types

Logical operations = construction on types following the dictionnary

$A \wedge B$	$A \times B = \Sigma(x : A)B$
$A \to B$	$A \to B = \Pi(x : A)B$
$(\forall x : A)B(x)$	$\Pi(x : A)B(x)$
$(\exists x : A)B(x)$	$\Sigma(x : A)B(x)$
$A \vee B$	$A + B$

Dependent types

de Bruijn (1967) notices that this approach is suitable for representation of mathematical proofs on a computer (AUTOMATH)

Proving a proposition is reduced to building an element of a given type

«This reminds me of the very interesting language AUTOMATH, invented by Dijkstra's colleague (and next-door neighbor) N. G. de Bruijn. AUTOMATH is not a programming language, it is a language for expressing proofs of mathematical theorems. The interesting thing is that AUTOMATH works entirely by type declarations, without any need for traditional logic! I urge you to spend a couple of days looking at AUTOMATH, since it is the epitome of the concept of type.»

D. Knuth (1973, letter to Hoare)

Type of identifications

Introduce a new type $\mathsf{Id}\ A\ a_0\ a_1$ as the type of ways of *identifying* a_0 and a_1 (de Bruijn's notion of «book equality», Martin-Löf, 1973)

Different identifications for different A (not as in with set theory)

This type formation can be *iterated* since we can compare two given identifications

So we can form $\mathsf{Id}\ (\mathsf{Id}\ A\ a_0\ a_1)\ p_0\ p_1$

Note that this, like universes, introduces dependent types in the language

Type of identifications

We can then stratify types

-*proposition* $\quad \Pi(x_0\ x_1 : A)\ \mathsf{Id}\ A\ x_0\ x_1$

-*set* $\quad \Pi(x_0\ x_1 : A)\ \mathsf{isProp}\ (\mathsf{Id}\ A\ x_0\ x_1)$

-*groupoid* $\quad \Pi(x_0\ x_1 : A)\ \mathsf{isSet}\ (\mathsf{Id}\ A\ x_0\ x_1)$

Type theory appears as a *generalization* of set theory

We also define $\mathsf{isContr}\ A$ to be $A \times \mathsf{isProp}\ A$

Laws of identifications

The *logical* laws of identifications are

refl a : Id $A\ a\ a$

Id $A\ a_0\ a_1 \to B(a_0) \to B(a_1)$

Laws formally similar to the ones of equality in simple type theory

The third law is quite mysterious and was found by Martin-Löf in 1973

$\Pi(a : A)$ isContr $(\Sigma(x : A)$ Id $A\ a\ x)$

Laws of identifications

The *mathematical* law of identification is the univalence axiom

The canonical map Id $U\ A\ B \to$ Equiv $A\ B$ *is an equivalence*

where

Equiv $A\ B = \Sigma(f : A \to B)$isEquiv f

isEquiv $f = \Pi(b : B)$isContr$(\Sigma(x : A)$Id $B\ (f\ x)\ b)$

This formally generalizes propositional extensionality

Equiv $A\ B$ generalizes uniformly logical equivalence (between propositions), bijection (between sets), categorical equivalence (between groupoids), ...

Abstract data types

The type of groups (in a given universe) will be

$\Sigma(X : U)$ isSet $X \times X \times (X \times X \times X) \times \dots$

Given a group $(G, p, e, f, \dots)$ it is impossible to have access to the «internal» of G and form the statement "does 0 belong to G"

We «hide» the internal structure of the group

Invariance by isomorphisms (Tarski-Lindenbaum, Gandy)

Representation independence (Morris, Reynolds, ...)

Type theory is more abstract/modular than set theory

Extensionality

In simple type theory, one extensionality axiom for each possible type formation (proposition or function type)

Univalence axiom corresponds to the case of equality for universe

Generalization of the propositional extensionality axiom of simple type theory

It *implies* function extensionality (Voevodsky, 2010)

Extensionality

In simple type theory, one can give an *interpretation* of the system with extensionality in the system *without* extensionality

Russell proves that logical connectives preserve logical equivalence in

«The Theory of Implications» American Journal of Mathematics, Vol. 28, 2, p. 159-202, 1906.

Gandy generalizes this to simple type theory in

«On axiomatic systems in mathematics and theories in physics» PhD thesis, University of Cambridge, 1953.

Extensionality

For instance, quantifiers are functionals $(I \to bool) \to bool$

They are *extensional* operations since if P and Q are two extensionaly equal predicates on I then $\forall(x : I)\ P\ x$ (resp. $\exists(x : I)\ P\ x$) and $\forall(x : I)\ Q\ x$ (resp. $\exists(x : I)\ Q\ x$) are logically equivalent

The starting point of Gandy's relative consistency proof for extensionality is then that each constant is an extensional operation and to be extensional is preserved by abstraction and application

Extensionality

Can we do the same for dependent type theory?

It does not *seem* possible to do it in one step like in simple type theory

Instead, we proceed in two steps: first a ⟪relational⟫ model, which can be seen as a higher-order generalization of the reflexive graph model of type theory (cubical sets)

Extensionality

In this model, the type of « relations » on a type A can be seen as an exponential $A^{\mathbb{I}}$ where $\mathbb{I}$ is an *interval* type

We can *internalize* an approximation of the identification type Path $A\ a_0\ a_1$

Almost all rules of the identification type are satisfied but the transport function

Path $A\ a_0\ a_1 \to P(a_0) \to P(a_1)$

Extensionality

This is a *presheaf* model

Γ is interpreted by a presheaf

$\Gamma \vdash A$ is interpreted by a presheaf on the category of elements of Γ

$\Gamma \vdash M : A$ is interpretation by a global section of this presheaf

$\mathsf{U}(X)$: collection of $\mathcal{U}$-presheafs on the category of elements of X

Equivalence relation

From homotopy theory, we get a suitable generalization of equivalence relation

Lemma: *If* A *subpolyhedra of* B *then* $B \times 0 \cup A \times \mathbb{I}$ *is a retract of* $B \times \mathbb{I}$

where $\mathbb{I}$ is now the closed unit interval $[0, 1]$

On the relation between the fundamental group of a space and the higher homotopy groups
Eilenberg 1939, Fund. Math.

Extensionality

If A subpolyhedra of B

Proposition: *Given two homotopic functions f_0 $f_1 : A \to X$ and an extension $f'_0 : B \to X$ of f_0 there is an extension f'_1 of f_1 homotopic to f_0*

Proofs of basic results about homotopy "can be obtained quite neatly by repeated, and sometimes tricky, use of this general lemma" (Bourbaki's notes on homotopy by Eilenberg, 1951)

If we think of $\Gamma \vdash A$ as a higher-order reflexive relation, to satisfy Eilenberg's condition can be thought of as expressing that this relation is an equivalence relation

Extensionality

Eilenberg's condition can be expressed internally (using the notion of partial element to represent subpolyhedra)

If we have $\gamma \in \Gamma^{\mathbb{I}}$ and $a_0 \in A\gamma(0)$ and a *partial* section $a(i) \in A\gamma(i),\ i \in \mathbb{I}$ with $a(0) = a_0$ then can extend this to a *total* section $\tilde{a}(i) \in A\gamma(i)$

For empty extent we get a transport function $A\gamma(0) \to A\gamma(1)$

But this condition is expressed in the context $\Gamma^{\mathbb{I}}$ and *not* in the context Γ

Extensionality

For *cubical sets* we can find a type $R(T)$ such that

the set of terms in $\Gamma^{\mathbb{I}} \vdash T$ is in *bijection* with the set of terms in $\Gamma \vdash R(T)$

This is a dependent version of a *right Kan extension* (thanks to discussions with Christian Sattler)

$\Gamma^{\mathbb{I}} \to \Delta$ naturally isomorphic to $\Gamma \to R(\Delta)$

For cubical sets, the path functor, which is the right adjoint of the cylinder functor, has itself *a right adjoint*

This follows from the fact that the cylinder functor preserves representables

Extensionality

(A. Pitts) Note that for nominal sets, the function $[\mathbb{A}]\_$ has a right adjoint (Theorem 4.13) but this adjunction does not internalize (Exercise 4.7)

The same probably holds for cubical sets

Extensionality

Hence, for *cubical sets* we can find a type $C(A)$ defined in the *same* context as A which expresses that A is a higher-order equivalence relation

This defines $C : \mathsf{U}_i \to \mathsf{U}_i$ for each universe U_i

One can build elements

$$\pi_C : C(A) \to (\Pi(x : A)C(B)) \to C(\Pi(x : A)B)$$

$$\sigma_C : C(A) \to (\Pi(x : A)C(B)) \to C(\Sigma(x : A)B)$$

$$u^i_C : C(\Sigma(X : \mathsf{U}_i)C(X))$$

Interpretation of Extensionality

$$
\begin{array}{lcl}
[x] & = & x \\
[M\ N] & = & [M]\ [N] \\
[\lambda(x : A)\ M] & = & \lambda(x : [\![A]\!])[M] \\
 & & \\
[\Pi(x : A)\ B] & = & (\Pi(x : [\![A]\!])[\![B]\!],\ \ \pi_C\ [A].2\ (\lambda(x : [\![A]\!])[B].2)) \\
[\Sigma(x : A)\ B] & = & (\Sigma(x : [\![A]\!])[\![B]\!],\ \ \sigma_C\ [A].2\ (\lambda(x : [\![A]\!])[B].2)) \\
[\mathsf{U}_i] & = & (\Sigma(X : \mathsf{U}_i)C(X),\ \ u^i_C) \\
 & & \\
[\![A]\!] & = & [A].1 \\
 & & \\
[\![x_1 : A_1, \ldots, x_n : A_n]\!] & = & x_1 : [\![A_1]\!], \ldots, x_n : [\![A_n]\!]
\end{array}
$$

Theorem: *If* $\Gamma \vdash M : A$ *then* $[\![\Gamma]\!] \vdash [M] : [\![A]\!]$

Interpretation of Extensionality

$[A].2$ computes by induction on A the proof that $[A].1$ is an equivalence relation, i.e. a proof of $C([A].1)$

We see that, *for getting a model*, the *exact* form of the constants π_C, σ_C, u_C does *not* matter

However the exact definition of these constants might be important if we want elegant/efficient computations (and there seems indeed to be significant possible simplifications w.r.t. our published version)

In the simplicial set model, to be an equivalence is a *property* and not a *structure*