Type Theory and Constructive Mathematics

Type Theory and Constructive Mathematics

Thierry Coquand

University of Gothenburg

Content

An introduction to Voevodsky's Univalent Foundations of Mathematics

The theory of types with which we shall be concerned is intended to be a full scale system for formalizing intuitionistic mathematics as developed, for example, in the book of Bishop 1967 (Martin-Löf, 1975)

Problems with *equality* in type theory. Existing implementations work well for representing mathematics in special domain (Gonthier's work: combinatorics, decidable equality)

Univalent foundations provide a new view of equality and new concepts that seem essential for representing mathematical notions in type theory

New foundation, non set theoretic, of mathematics?

References

Papers/book by Martin-Löf

Voevodsky's home page on Univalent Foundations of Mathematics

Mike Shulman's minicourse on Homotopy Type Theory

Homotopy type theory web site

History

Successive refinements, each bringing new notions

de Bruijn (1967) context, proof irrelevance, definitional equality

Howard (1969) strong existence

Martin-Löf (1970-1975) strong notion of existence (axiom of choice), of disjunction of absurdity, of identity, refinement of Gentzen's introduction/elimination rules

Martin-Löf (1970-1975) universe, first a type of a all types, then as a reflection principle to express something like Grothendieck universes

Notations

The system is based on λ-calculus

free and bound variables

$\lambda x.t$ for lambda abstraction

$t(x = a)$ for substitution, or even $t(a)$ if x is clear

$f\ a$ for application of f to a and $f\ a_1\ a_2$ for $(f\ a_1)\ a_2$

$A_0 \to A_1 \to A_2$ for $A_0 \to (A_1 \to A_2)$

Type Theory

First version (1971)

Terms $t, u, A, B \ ::= \ x \mid V \mid \lambda x.t \mid t\ u \mid (\Pi x : A)B$

$$\frac{}{\vdash} \qquad \frac{\Gamma \vdash A : V}{\Gamma, x : A \vdash}$$

$$\frac{\Gamma, x : A \vdash B : V}{\Gamma \vdash (\Pi x : A)B : V} \qquad \frac{\Gamma \vdash}{\Gamma \vdash V : V}$$

$$\frac{\Gamma \vdash \quad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash t : (\Pi x : A)B \quad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B(u)} \qquad \frac{\Gamma, x : A \vdash v : B}{\Gamma \vdash \lambda x.v : (\Pi x : A)B}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : V \qquad A = B}{\Gamma \vdash t : B}$$

Type Theory

Some differences with the original presentation

1. domain-free presentation $\lambda x.v$ versus $\lambda x : A.v$
2. notation $a : A$ instead of $a \in A$
3. the notion of *context* was implicit in MLTT (until 1991). Comes from programming language theory

Equality is β-conversion. The proof of Church-Rosser based on parallel reductions appeared first for this system (Tait/Martin-Löf)!

Type = Proposition

$A \to B$ defined as $(\Pi x : A)B$ where x is not free in B

Terms and types (propositions) are treated uniformely

Type Theory

The symbol $\vdash$ comes from Frege's Begriffsschrift (1879)

$\vdash A$ meant that A is valid

The relation $\vdash a : A$ is more fundamental in intuitionism than simply asserting that A is provable

The relation $\vdash a : A$ should be decidable

"We recognize a proof of a proposition when we see one" (Kreisel)

Type Theory

If $\Gamma \vdash t : A$ then $\Gamma \vdash A : V$

If $\Gamma \vdash t : A$ and $t \rightarrow_\beta t'$ then $\Gamma \vdash t' : A$

We cannot (directly) reproduce Russell's paradox of the set of all sets that do not contain themselves : we cannot form the proposition/type that x is not of type x

$t : A$ is a *judgement* and not a *proposition*

No term t in normal form such that

$$X : V \vdash t : X$$

This can be proved in a purely combinatorial way, and expresses some form of (weak) consistency

New connectives

The language of the theory is richer than the language of first-order predicate logic. This makes it possible to strengthen the axioms for existence and disjunction. (1972)

The language of the theory is richer than the language of traditional systems in permitting proofs to appear as parts of the propositions so that the propositions can express properties of proofs (and not only individuals, like in first-order logic). This makes it possible to strengthen the axioms for existence, disjunction, absurdity and identity. (1975)

Dependent sum

$$\frac{\Gamma, x : A \vdash B : V}{\Gamma \vdash (\Sigma x : A)B : V}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B(a)}{\Gamma \vdash (a, b) : (\Sigma x : A)B}$$

$$\frac{\Gamma \vdash c : (\Sigma x : A)B}{\Gamma \vdash c.1 : A} \qquad \frac{\Gamma \vdash c : (\Sigma x : A)B}{\Gamma \vdash c.2 : B(c.1)}$$

$$(a, b).1 = a \qquad (a, b).2 = b$$

Dependent sum

The new quantification operation $(\Sigma x : A)B$ was introduced by Howard (1969)

$(\Sigma x : A)B$ can be read as the proposition: there *exists* an object x of type A such that $B(x)$

Another interpretation is the type of all objects x of type A *such that* $B(x)$ *because, from the intuitionistic point of view, to give an object* x *of type* A *such that* $B(x)$ *is to give* x *together with a proof of the proposition* $B(x)$

Disjoint union

Usual rule for disjunction

introduction rules

$$\frac{A}{A \vee B} \qquad \frac{B}{A \vee B}$$

elimination rule

$$\frac{A \to C \qquad B \to C}{A \vee B \to C}$$

Gentzen natural deduction: *introduction rules* give the meaning of logical connectives, the *elimination rules* are then justified in term of the introduction rules by suitable *normalization rules*

Disjoint union

In type theory $A + B : V$ if $A \quad B : V$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{Inl}\ t : A + B} \qquad \frac{\Gamma \vdash u : B}{\Gamma \vdash \mathsf{Inr}\ u : A + B}$$

$$\frac{\Gamma \vdash\vdash a : (\Pi x : A) C(\mathsf{Inl}\ x) \qquad \Gamma \vdash b : (\Pi y : B) C(\mathsf{Inr}\ y)}{\Gamma \vdash \mathsf{Elim}\ a\ b : (\Pi z : A + B) C}$$

$$\mathsf{Elim}\ a\ b\ (\mathsf{Inl}\ t) = a\ t \qquad \mathsf{Elim}\ a\ b\ (\mathsf{Inr}\ u) = b\ u$$

This is a *refinement* of Gentzen's elimination rules

Inductive definitions

All these constructions follow the same pattern

constructors correspond to primitive operations and to introduction rules in natural deduction

defined functions correspond to elimination rules that are justified by

computation rules which correspond to normalization in natural deduction

Computation rules correspond to the process of understanding a proof by eliminating lemmas

Natural numbers

N is a type of *constructors* $0 : N$ and $\mathsf{S}\ x : N$ if $x : N$

Let $C(n)$ be a function which to an arbitrary natural number $n : N$ assigns a type

Given $d : C(0)$ and $e : (\Pi n : N)\ (C(n) \to C(\mathsf{S}\ n))$

we may introduce a function $f : (\Pi n : N)\ C(n)$ by the equations

$$f\ 0 = d \qquad\qquad f\ (\mathsf{S}\ n) = e\ n\ (f\ n)$$

Inductive definitions

We can introduce the type N_1 with only one constructor $0 : N_1$

If $C(x)$ type for $x : N_1$ and we have $a : C(0)$

we can introduce $f : (\Pi x : N_1)\ C(x)$ by the equation

$$f\ 0 = a$$

Inductive definitions

We can introduce the type N_2 with two constructors $0 : N_2$ and $1 : N_2$

If $C(x)$ type for $x : N_2$ and we have $a_0 : C(0)$ and $a_1 : C(1)$

we can introduce $f : (\Pi x : N_2)\ C(x)$ by the equations

$$f\ 0 = a_0 \qquad\qquad f\ 1 = a_1$$

Inductive definitions

We introduce the type N_0 with *no* constructor

If $C(x)$ type for $x : N_0$

we have $f : (\Pi x : N_0)\ C(x)$

This is a refinement of the usual elimination rule $N_0 \rightarrow A$ for any type A

Inductive definitions

N, N_2, N_1, N_0 and $A + B$ are examples of type introduced by *ordinary inductive definition*

Hilbert 1926 considers the type *Ord* of *ordinal numbers*

$0 : Ord$
if $x : Ord$ then $\mathsf{S}\ x : Ord$
if $u : N \to Ord$ then its limit $L\ u : Ord$

One can write elimination rules for this type as well (Martin-Löf, 1971)

Type theory

In the formal theory the abstract entities (natural numbers, ordinals, functions, types, and so on) become represented by certain symbol configurations, called terms, *and the definitional schema, read from the left to the right, become mechanical* reduction rules *for these symbol configurations.*
Type theory *effectuates the computerization of abstract intuitionistic mathematics that above all Bishop has asked for* (1971)

It provides a framework in which we can express *conceptual* mathematics in a *computational* way.

Type theory

The law of excluded middle EM becomes

$$(\Pi X : V)(X + \neg\, X)$$

where $\neg\, X$ is $X \to N_0$

There is no term in normal form of this type

We represent *intuitionistic logic* and type theory can be seen as a concrete representation of Kolmogorov 1930 interpretation of this logic

Type theory

The underlying computation system of type theory extended with data types is the following

Terms of type theory: λ-calculus with constants

Two kind of constants: *constructors* or *defined functions*

Constructors $c(\vec{t})$

A *data type* is of this form, e.g. type of list

$f(x_1, \ldots, x_n)\ x$ parameters and main argument, function defined by case on the main argument

Type theory

Let $C(n)$ be a function which to an arbitrary natural number $n : N$ assigns a type

Given $d : C(0)$ and $e : (\Pi n : N)\ (C(n) \rightarrow C(\mathsf{S}(n)))$ we introduce $f(d,e) : (\Pi n : N)\ C(n)$ with computation rules

$$f(d,e)\ 0 = d \qquad\qquad f(d,e)\ (S(n)) = e\ n\ (f(d,e)\ n)$$

Type theory

The language of type theory

$$t \ ::= \ x \mid c(\vec{t}) \mid f(\vec{t}) \mid \lambda x.t \mid t \ t$$

has a simple operational and denotational semantics

We have computation rules of the form $f(\vec{x}) \ c(\vec{y}) = t(\vec{x},\vec{y})$ e.g. $f(d,e) \ 0 = d$ or $f(d,e) \ (S(n)) = e \ n \ (f(d,e) \ n)$

If $f(\vec{t}) \ u$ is well-typed then the type of u is a data type and the type of a term $c(\vec{t})$ is a data type

Type theory can be seen as a functional programming language with dependent types

Type theory

Syntax of normal forms

$$v ::= k \mid c(\vec{v}) \mid f(\vec{v}) \mid \lambda x.v$$

$$k ::= x \mid f(\vec{v})\ k \mid k\ v$$

Lemma: *Any well-typed closed term in normal form has to be in canonical form. If it is of the form $f(\vec{v})$ or $\lambda x.v$ its type has to be a dependent product*

Type theory

A term is *normal* if it cannot be further reduced

The closed normal term of type N_2 are exactly 0 and 1

The closed normal term of type N have the form $0,\ \mathsf{S}(0),\ \mathsf{S}(\mathsf{S}(0)),\ldots$

There is no normal closed term of type N_0, since there is no canonical term of type N_0 and normal closed term are canonical

Hence *if* we have normalization there is *no* closed term of type N_0 since types are preserved by reduction

Since $\bot = N_0$ this expresses the *logical consistency* of type theory

The type of types

The idea of the type of types is forced upon us by accepting simultaneously each of the following three principles.
First, quantification over propositions as in second order logic.
Second, Russell's doctrine of types according to which the ranges of significance of propositional functions form types so that, in particular, it is only meaningful to quantify over all objects of a certain type.
Third, the identification of propositions and types.
Suppose namely that quantification over propositions is meaningful. Then by the doctrine of types, the propositions must form a type. But, if propositions and types are identified, then this type is at the same time the type of types.

The type of types

The type of types introduces a strong kind of selfreference which, as pointed out by Gödel 1964, transcends the cumulative hierarchy notion of set and may seem to verge on the paradoxes, but which is actually being used in category theory, notably, in the construction of the category of all categories

The type of types

We can define types by recursion, for instance $T : N \to V$
$T\ 0 = N \qquad T\ (n+1) = (T\ n) \to N$

but also $Eq : N \to N \to V$

$Eq\ 0\ 0 = N_1 \qquad Eq\ (n+1)\ (m+1) = Eq\ n\ m$

$Eq\ 0\ (m+1) = Eq\ (n+1)\ 0 = N_0$

Girard's paradox

It does not seem possible to translate directly Russell's paradox

We can form $(\Sigma X : V)B(X)$ but $B(X)$ cannot express that X is not of type X

The judgement $a : A$ is not a proposition that can be negated

Girard's paradox

We can however translate Burali-Forti's paradox (Girard, 1971)

Alexandre Miquel found a variation of this paradox by interpreting *set theory* in *type theory* and translating Russell's paradox

We have a closed term t such that

$$\vdash t : \quad \bot \quad = \quad N_0$$

and this closed term t is *not* normalizable

Use of inconsistent systems

In this domain-free presentation we can build a fixed point combinator: there is a term $L : (\Pi X : V)(X \to X) \to X$ such that

$$L\ X\ f = f\ (L\ X\ f) \quad [X : V, f : X \to X]$$

(A. Meyer and M. Reinhold, 1986; T. Urkens, 1994)

In the typed version $\lambda x : A.t$ we can only state the existence of a family of looping combinators L_n

$$L_n\ X\ f = f\ (L_{n+1}\ X\ f) \quad [X : V, f : X \to X]$$

In both cases, this implies that the typing relation $a : A$ is *undecidable*

Open problem (since 1985): is there a fixed point combinator in the typed version?

Use of inconsistent systems: parametricity

Theorems for free (Ph. Wadler 1990, J.P. Bernardy et al. 2009)

The only definable function of type $(\Pi X : V)(X \to X)$ is the identity

This can be described purely syntactically

Use of inconsistent systems: parametricity

Given $t(x_1, \ldots, x_n)$ define $[t]_{(x_1,\ldots,x_n,x_1',\ldots,x_n')}$ by induction on t

$$[V] = \lambda X.X \to V$$

$$[(\Pi x : A)B]_{(\vec{x},\vec{x}')} = \lambda f.(\Pi x : A(\vec{x}))(\Pi x' : [A]_{(\vec{x},\vec{x}')}\ x)[B]_{(\vec{x},x,\vec{x}',x')}\ (f\ x)$$

$$[x_i]_{(\vec{x},\vec{x}')} = x_i'$$

$$[t\ u]_{(\vec{x},\vec{x}')} = [t]_{(\vec{x},\vec{x}')}\ t\ [u]_{(\vec{x},\vec{x}')}$$

$$[\lambda x.v]_{(\vec{x},\vec{x}')} = \lambda x.\lambda x'.[v]_{(\vec{x},x,\vec{x}',x')}$$

Use of inconsistent systems

This can be extended to data types

Theorem: *If* $\vdash A : V$ *we have* $\vdash [A] : A \to V$ *and in general if* $\vdash t : A$ *then* $\vdash [t] : [A]\ t$

This transformation is the purely syntactic essence of parametricity results

Use of inconsistent systems: parametricity

If $T = (\Pi X : V)(X \to X)$ then

$$[T] = \lambda f.(\Pi X : V)(\Pi X' : X \to V)(\Pi x : X)(X'\ x \to X'\ (f\ X\ x))$$

If $t : T$ we have $[t] : [T]\ t$ and hence, if $A : V, a : A$ we get a proof of

$$(\Pi P : A \to V)(P\ a \to P\ (t\ A\ a))$$

which states that a and $t\ A\ a$ are equal

Hence t has to be the identity function

Use of inconsistent systems

There is no proof in normal form of $\perp = N_0$, since a closed normal term is in canonical form

Captures elegantly the essence of consistency arguments in proof theory

Use of inconsistent systems: completeness

For a fixed first-order language, say a predicate symbol $P : X \to V$ and a constant $a : X$ and a binary function symbol $f : X \to X \to X$ one can translate any first-order formula of this language using only implication and universal quantification in the context

$$X : V, P : X \to V, a : X, f : X \to X \to X$$

For instance $\varphi = \forall x.P(x) \to P(f(x, a))$ becomes

$$\varphi^* = (\Pi x : X)(P\ x \to P\ (f\ x\ a))$$

Use of inconsistent systems: completeness

Theorem: (Martin-Löf, 1971) *A formula φ is provable in first-order logic iff there is a normal term of type φ^* in the context* $X : V, P : X \rightarrow V, a : X, f : X \rightarrow X \rightarrow X$. *More precisely, there is a one-to-one correspondance between normal term of type φ^* and normal proof of φ in natural deduction*

For instance $\lambda z.z\ a$ is a normal proof of $((\forall x.P(x)) \rightarrow P(a))^*$

If we build a proof of φ^* in the system *and* the proof can be normalized then we can read a standard first-order proof from this normal proof

This is one possible use of an inconsistent system

Universe

The incoherence of the idea of a type of all types whatsoever made it necessary to distinguish, like in category theory, between small and large types. Thus the universe U appears, not as the type of all types, but as the type of small types, whereas U itself and all types which are built up from it are large. This makes the types wellfounded and the theory predicative.

The situation is reminiscent of the situation in set theory after Russell's paradox

Remark: one can prove normalization of the inconsistent system (or give parametricity arguments) in the same way as one proves normalization of a consistent system!

Universe

This is reminiscent of Grothendieck's notion of universe

If $A : U$ and $B : U$ for $x : A$ then we have

$$(\Pi x : A)\ B : U \qquad\qquad (\Sigma x : A) B : U$$

$N_0,\ N_1,\ N_2, N$ are all of types U
We can define small types by recursion

Type Theory with one universe (1972)

Terms $t, u, A, B \ ::= \ x \mid U \mid \lambda x.t \mid t\ u \mid (\Pi x : A)B$

$$\frac{}{\vdash} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash}$$

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash (\Pi x : A)B} \qquad \frac{\Gamma \vdash}{\Gamma \vdash U}$$

$$\frac{\Gamma \vdash \quad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash t : (\Pi x : A)B \quad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B(u)} \qquad \frac{\Gamma, x : A \vdash v : B}{\Gamma \vdash \lambda x.v : (\Pi x : A)B}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B \qquad A = B}{\Gamma \vdash t : B}$$

$$\frac{\Gamma \vdash A : U \quad \Gamma, x : A \vdash B : U}{\Gamma \vdash (\Pi x : A)B : U} \qquad \frac{\Gamma \vdash A : U}{\Gamma \vdash A}$$

Type Theory with one universe (1972)

We can define $F : N \to U$

$$F\ 0 = N \qquad F\ (n+1) = (F\ n) \to N$$

In ZF set theory, this function can only be defined with the *axiom schema of replacement*