Homotopy type theory

Nicola Gambino

School of Mathematics
University of Leeds

Young Set Theory
Copenhagen
June 13th, 2016

Note

- To simplify the presentation, I did not specify attibution of ideas and results.
- A list of references is given at the end.

Homotopy type theory

Motivation

- to understand better equality in type theory
- to facilitate computer-assisted verification of proofs

Main feature

- it combines ideas from type theory and homotopy theory

Plan of the lectures

Lecture I: Type theory

Lecture II: Homotopical algebra

Lecture III: Homotopical models of type theory

Lecture IV: Univalent foundations

Outline of lecture 1

Part I: Syntax for type theory

Part II: The type theory **T**

Part III: Type theory vs set theory

Part I: Syntax for type theory

Type theory

A type theory is a formal system for deriving judgements

$$\Gamma \vdash A : \text{type}$$
$$\Gamma \vdash a : A$$
$$\Gamma \vdash A = B : \text{type}$$
$$\Gamma \vdash a = b : A$$

Here, $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ is a context.

Key aspect. Types are allowed to depend on terms

$$x : A \vdash B(x) : \text{type}$$

For example, $n : \text{Nat} \vdash \text{List}_n(\text{Bool}) : \text{type}$.

A type theory is given by a set of deduction rules (cf. sequent calculus).

The type theory **T**

- Basic types

$$0\,, \quad 1\,, \quad \mathsf{Bool}\,, \quad \mathsf{Nat}$$

- Simple types

$$A \times B\,, \quad A + B\,, \quad A \to B\,, \quad \mathsf{List}(A)$$

- Dependent types

$$\mathsf{Id}_A(a, b)\,, \quad (\Sigma x : A)B(x)\,, \quad (\Pi x : A)B(x)\,, \quad (\mathsf{W}x : A)B(x)$$

- Universe type

$$\mathsf{U}$$

Syntactic issues (I)

Setting up a type theory is more complex than setting up a first-order theory.

Recall that to set up a first-order theory

(1) fix a language L
(2) define inductively the set of well-formed terms
(3) define inductively the set of well-formed formulas
(4) give the axioms for the theory
(5) define inductively the set of theorems of the theory

Note

- Each step depends only on the previous ones (e.g. the axioms of a theory do not change the set of well-formed terms).

Syntactic issues (II)

Problem. This approach does not work for type theories

- The deduction rules specify how the well-formed term expressions are built.

  Example:

$$\frac{c : A \times B}{\pi_1(c) : A}$$

- The sets of term expressions and of type expressions cannot be defined independently.

  Examples:

$$\mathrm{Id}_A(a, b)\,, \quad \mathrm{refl}_A(a)$$

Setting up a type theory

Solution

(1) fix a signature (operations and arities)

(2) define inductively the set of raw expressions

(3) give the deduction rules of the type theory

(4) define inductively the set of theorems of the theory

(5) the theorems isolate the well-formed expressions

Note. We need a more complex notion of arity than for algebraic theories, in order to account for **variable binding** operations, e.g.

$$(\Pi x : A)B(x)\,, \quad (\lambda x : A)b(x)$$

Signatures

Definition. An **arity** is a tuple

$$\left((n_1, \delta_1), \ldots, (n_k, \delta_k), \delta \right)$$

where $k \in \mathbb{N}$, $n_1, \ldots, n_k \in \mathbb{N}$, and $\delta_1, \ldots, \delta_k, \delta \in \{0, 1\}$.

When $k = 0$, we write (δ).

Definition

- A **signature** S is a set of pairs (s, α) where α is an arity.
- If $(s, \alpha) \in S$, we call s a **symbol of arity** α.

Idea

- An operation of the arity above takes k arguments and binds n_i variables in the i-th argument.
- $\delta_1, \ldots, \delta_k, \delta$ distinguish term expressions (0-expressions) from type expressions (1-expressions).

Raw expressions

Fix

- an infinite set of variables $\{x_0, x_1, \ldots\}$,
- a signature S

Define the sets of 0-expressions and 1-expressions inductively:

- every variable is a 0-expression
- if $s \in S$ has arity $((n_1, \delta_1), \ldots, (n_k, \delta_k), \delta)$ and M_i is an δ_i-expression and $\vec{x}_i$ is a vector of n_i distinct variables, for $i = 1, \ldots, k$, then

$$s\big((\vec{x}_1)M_1, \ldots, (\vec{x}_k)M_k\big)$$

  is an δ-expression.

Notation

- When $k = 0$, we write s rather than $s(\ )$.
- If some $n_i = 0$ we write M_i rather than $(\ )M_i$.

Judgements

A **judgement** has one of the following four forms

(1)	$A : \text{type}$	"A is a type"
(2)	$A = B : \text{type}$	"A and B are definitionally equal types"
(3)	$a : A$	"a is a term of type A"
(4)	$a = b : A$	"a and b are definitionally equal terms of type A"

Contexts and hypothetical judgements

- A **context** is a sequence of the form

$$x_0 : A_0 \, , x_1 : A_1 \, , \ldots , x_n : A_n$$

where $x_0, \ldots, x_n$ are distinct variables and $A_1, \ldots, A_n$ are 1-expressions.

When $n = 0$, we write $()$.

- A **hypothetical judgement** has the form

$$\Gamma \vdash J$$

where Γ is a context and J is a judgement.

We write J instead of $() \vdash J$

Deduction rules

- A **deduction rule** has the form

$$\frac{\Gamma_1 \vdash J_1 \qquad \cdots \qquad \Gamma_n \vdash J_n}{\Gamma \vdash J}$$

  where $\Gamma_1 \vdash J_1 , \ldots , \Gamma_n \vdash J_n , \Gamma \vdash J$ are hypothetical judgements.

  When $n = 0$, we simply write $\Gamma \vdash J$.

- A **type theory** is given by a signature and a set of deduction rules.

Derivability is defined in the standard way.

The type theory **T**

- Basic types

$$0\,, \quad 1\,, \quad \mathsf{Bool}\,, \quad \mathsf{Nat}$$

- Simple types

$$A \times B\,, \quad A + B\,, \quad A \to B\,, \quad \mathsf{List}(A)$$

- Dependent types

$$\mathsf{Id}_A(a, b)\,, \quad (\Sigma x : A)B(x)\,, \quad (\Pi x : A)B(x)\,, \quad (\mathsf{W}x : A)B(x)$$

- Universe types

$$\mathsf{U}$$

Types-as-propositions

Types can be used to represent formulas:

$\mathrm{Id}_A(a, b)$	$a = b$
0	$\bot$
1	$\top$
$A \times B$	$A \wedge B$
$A + B$	$A \vee B$
$A \to B$	$A \to B$
$(\Pi x : A)B(x)$	$(\forall x \in A)B(x)$
$(\Sigma x : A)B(x)$	$(\exists x \in A)B(x)$

Idea

$$a : A \quad \Longleftrightarrow \quad \text{“}a \text{ is a proof of } A\text{”}$$

The deduction rules of **T**

For each form of type we have:

- formation rules
- introduction rules
- elimination rules
- computation rules

Note

- Some elimination rules can be read as induction principles

The type of natural numbers (I)

Formation rule

$$\mathsf{Nat} : \mathsf{type}$$

Introduction rules

$$0 : \mathsf{Nat} \qquad \frac{x : \mathsf{Nat}}{\mathsf{succ}(x) : \mathsf{Nat}}$$

The type of natural numbers (II)

Elimination rule

$$\frac{x : \mathsf{Nat} \vdash E(x) : \mathsf{type} \qquad d : E(0) \qquad x : \mathsf{Nat}, y : E(x) \vdash e(x, y) : E(\mathsf{succ}(x))}{x : \mathsf{Nat} \vdash \mathsf{natrec}(x, d, e) : E(x)}$$

Computation rules

$$\frac{x : \mathsf{Nat} \vdash E(x) : \mathsf{type} \qquad d : E(0) \qquad x : \mathsf{Nat}, y : E(x) \vdash e(x, y) : E(\mathit{succ}(x))}{\mathsf{natrec}(0, d, e) = d : E(0)}$$

$$\frac{x : \mathsf{Nat} \vdash E(x) : \mathsf{type} \qquad d : E(0) \qquad x : \mathsf{Nat}, y : E \vdash e(x, y) : E(\mathsf{succ}(x))}{x : \mathsf{Nat} \vdash \mathsf{natrec}(\mathsf{succ}(x), d, e) = e(x, \mathsf{natrec}(x, d, e)) : E(\mathsf{succ}(x))}$$

Product types (I)

Formation rule

$$\frac{A : \text{type} \quad B : \text{type}}{A \times B : \text{type}}$$

Introduction rules

$$\frac{a : A \qquad b : B}{\text{pair}(a, b) : A \times B}$$

Product types (II)

Elimination rule

$$\frac{u : A \times B \vdash E(u) : \mathsf{type} \qquad x : A, y : B \vdash e(x, y) : E(\mathsf{pair}(x, y))}{u : A \times B \vdash \mathsf{split}(u, e) : E(u)}$$

Computation rule

$$\frac{u : A \times B \vdash E(u) : \mathsf{type} \qquad x : A, y : B \vdash e(x, y) : E(\mathsf{pair}(x, y))}{x : A, y : B \vdash \mathsf{split}(\mathsf{pair}(x, y), e) = e(x, y) : E(\mathsf{pair}(x, y))}$$

Function types (I)

Formation rule

$$\frac{A : \text{type} \quad B : \text{type}}{A \to B : \text{type}}$$

Introduction rule

$$\frac{x : A \vdash b(x) : B}{(\lambda x : A) b(x) : A \to B}$$

Function types (II)

Elimination rule

$$\frac{f : A \to B \qquad a : A}{\mathsf{app}(f, a) : B}$$

Computation rule

$$\frac{x : A \vdash b(x) : B \qquad a : A}{\mathsf{app}((\lambda x : A)b(x), a) = b(a) : B}$$

Σ-types (I)

Formation rule

$$\frac{x : A \vdash B(x) : \text{type}}{(\Sigma x : A)B(x) : \text{type}}$$

Introduction rule

$$\frac{a : A \qquad b : B(a)}{\text{pair}(a, b) : (\Sigma x : A)B(x) : \text{type}}$$

Σ-types (II)

Elimination rule

$$\frac{u : (\Sigma x : A)B(x) \vdash E(u) : \mathsf{type} \qquad x : A, y : B(x) \vdash e(x, y) : E(\mathsf{pair}(x, y))}{u : (\Sigma x : A)B(x) \vdash \mathsf{split}(u, e) : E(u)}$$

Computation rule

$$\frac{u : (\Sigma x : A)B(x) \vdash E(u) : \mathsf{type} \qquad x : A, y : B(x) \vdash e(x, y) : E(\mathsf{pair}(x, y))}{x : A, y : B(x) \vdash \mathsf{split}(\mathsf{pair}(x, y), e) = e(x, y) : E(\mathsf{pair}(x, y))}$$

Exercise

Prove that the following judgements are derivable:

$$u : (\Sigma x : A) B(x) \vdash \pi_1(u) : A$$

$$u : (\Sigma x : A) B(x) \vdash \pi_2(u) : B(\pi_1(u))$$

Π-types (I)

Formation rule

$$\frac{x : A \vdash B(x) : \text{type}}{(\Pi x : A)B(x) : \text{type}}$$

Introduction rule

$$\frac{x : A \vdash b(x) : B(x)}{(\lambda x : A)b(x) : (\Pi x : A)B(x)}$$

Π-types (II)

Elimination rule

$$\frac{f : (\Pi x : A)B(x) \qquad a : A}{\mathsf{app}(f, a) : B(a)}$$

Computation rule

$$\frac{x : A \vdash b(x) : B(x) \qquad a : A}{\mathsf{app}((\lambda x : A)b(x), a) = b(a) : B(a)}$$

Exercise

Let

$$A : \text{type}$$
$$B : \text{type}$$
$$x : A, y : B \vdash C(x, y) : \text{type}$$

Define functions

$$f : (\Sigma u : A \to B)(\Pi x : A) C(x, \text{app}(u\, x)) \to (\Pi x : A)(\Sigma y : B) C(x, y)$$

$$g : (\Pi x : A)(\Sigma y : B) C(x, y) \to (\Sigma u : A \to B)(\Pi x : A) C(x, \text{app}(u\, x))$$

Identity types (I)

Formation rule

$$\frac{A : \mathsf{type} \qquad a : A \qquad b : A}{\mathsf{Id}_A(a, b) : \mathsf{type}}$$

Idea. $p : \mathsf{Id}_A(a, b) \Leftrightarrow$ "p is a proof that $a = b$"

Introduction rule

$$\frac{a : A}{\mathsf{refl}(a) : \mathsf{Id}_A(a, a)}$$

Identity types (II)

Elimination rule

$$\frac{x\,,y : A\,,u : \mathsf{Id}_A(x,y) \vdash E(x,y,u) : \mathsf{type} \qquad x : A \vdash d(x) : E(x,x,\mathsf{refl}(x))}{x\,,y : A\,,u : \mathsf{Id}_A(x,y) \vdash \mathsf{J}(x,y,u,d) : E(x,y,u)}$$

Computation rule

$$\frac{x\,,y : A\,,u : \mathsf{Id}_A(x,y) \vdash E(x,y,u) : \mathsf{type} \qquad x : A \vdash d(x) : E(x,x,\mathsf{refl}(x))}{x : A \vdash \mathsf{J}(x,x,\mathsf{refl}(x),d) = d(x) : E(x,x,\mathsf{refl}(x))}$$

Exercise

Let $x, y, z : A$.

Prove that the following judgements are derivable:

$$u : \mathrm{Id}_A(x, y) \vdash u^{-1} : \mathrm{Id}_A(y, x)$$

$$u : \mathrm{Id}_A(x, y), v : \mathrm{Id}_A(y, z) \vdash v \circ u : \mathrm{Id}_A(x, z)$$

The type universe

Formation rule

$$\mathsf{U} : \mathsf{type}$$

Elimination rule

$$\frac{a : \mathsf{U}}{\mathsf{El}(a) : \mathsf{type}}$$

Plus rules to express that:

- U contains 'names' for all the basic types
- U is closed under all type operations.

Part III: Type theory vs set theory

From type theory to set theory

Type theories such as **T** have (straightforward) set-theoretic models:

- types-as-sets
- terms-as-elements

Theorem. **T** $\hookrightarrow$ ZFC + 1 inaccessible cardinal.

Note. There are many refinements of this result, e.g.

- Types-as-sets interpretations into constructive set theories
- Realizability models into variants of Kripke-Platek set theory

From set theory to type theory

Within **T** one can construct a 'set-theoretic universe'

$$V = (\mathsf{W}x : \mathsf{U})\, \mathsf{El}(x)$$

Idea

- sets are interpreted as well-founded trees,

$$\frac{a : \mathsf{U} \qquad f : \mathsf{El}(a) \to V}{\sup(a, f) : V}$$

- extensional equality is given by bisimulation.

Theorem. V is a model of Constructive Zermelo-Frankel set theory (CZF)

- Extensionality, Set Induction, Pairing, Union, Infinity, Δ_0-separation, Strong Collection, Subset Collection.

Lots of work to match precisely type theories and set theories.

Type theory vs set theory

Key difference. In type theory, there are two notions of equality:

$$a = b : A \qquad \text{and} \qquad p : \mathrm{Id}_A(a, b)$$

Extensional type theories

- ✓ Two notions of equality coincide
- ✓ Convenient to work with (e.g. function extensionality holds)
- × Type-checking is decidable

Intensional type theories

- × Two notions of equality coincide
- × Convenient to work with (e.g. function extensionality holds)
- ✓ Type-checking is decidable

Significant efforts to find a compromise, mostly syntactic.

Links with homotopy theory offer a new perspective.