LECTURE 1: TYPE THEORY

ANDREJ BAUER

ABSTRACT. Lecture notes for the second part of the doctoral course "Homotopy (Type) Theory", taught by Jaka Smrekar and Andrej Bauer in the Spring 2019 at the Department of Mathematics, Faculty of Mathematics and Physics, University of Ljubljana.

1. WHAT IS TYPE THEORY?

In this lecture we give an introduction to type theory aimed at mathematicians who are possibly unacquainted with logic and theoretical computer science. In fact, the whole series of lectures will consciously avoid discussing the connections between type theory and computer science.

1.1. **Type theory as a foundation.** Type theory is a *foundational theory*. This means that it stands on its own, without any support from logic and set theory, and it aims to be general enough to encompass a great deal of mathematical activity. While this sounds a bit scary, it also means that type theory can in principle be understood and used without prior knowledge of algebra, category theory, analysis, etc. The situation is not unlike many others in mathematics, where we limit ourselves to a certain style of reasoning. For example, when doing planar geometry we only use the basic geometric objects (points, segments, lines, circles) and we adhere to the axioms of geometry.

Type theory is not a replacement, or an alternative, to logic and set theory. Rather, it is a *generalization* of existing foundations that *expands* the view of the mathematical universe. In later lectures we shall identify logic and sets as just two homotopy levels of the type-theoretic mathematical universe.

There are several variants of type theory. We shall focus on *homotopy type theory (HoTT)*, a recently developed version which has close connections to abstract and sophisticated topics, such as ∞-groupoids and $(\infty, 1)$-toposes. However, we will steer clear of a precise technical explanation on how type theory relates to these, as that would require a course on its own. Instead, we will explain types intuitively, relying on your prior mathematical knowledge and intuitions about geometry and space.

Of course, type theory itself will be formulated quite precisely and as rigorously as any other mathematics (and more so, because it has all been formalized and verified by computer programs known as proof assistants).

1.2. **A theory of constructions.** In type theory, we take as the central notion *construction* of mathematical objects. Constructions permeate mathematics, and are at least as important as proofs. In fact proofs themselves are a certain kind of constructions: we construct proofs from axioms using the logical rules of reasoning.

By taking constructions as primitive, we are in good company. Proposition 1 of the First book of Euclid's Elements [1] reads:

To construct an equilateral triangle on a given finite straight-line.

Mathematics started with a construction problem, not a statement of truth!

Euclid's Proposition 1 is not just any kind of construction but rather a *dependent* one: the constructed equilateral triangle depends on the line segment. Dependence is a basic phenomenon in mathematics, thus our theory will be a theory of *dependent constructions*.

2. The basics of type theory

Let us proceed with an overview of the basic concepts of type theory, namely *types* and *points* of types.

A type is a description of a certain kind of construction. For example, we will introduce the type of natural numbers N by explaining how to construct natural numbers, and how to use them. When an object is constructed according to these rules, we say that its type is N. In fact, every construction must always be made according to the rules of one of the types, and then it is of course of that type. It cannot happen that a construction is of two different types, for each type will prescribe a different way of building objects.

It is important to distinguish a type from its *extension*. A type describes a way of constructing objects. The extension of a type is the collection (we avoid saying "set" lest we suggest unwarranted intuitions) of all things that may be constructed according to the given rules. It may happen that two different types have the same extension. This already happens quite naturally in informal mathematics. Consider:

(P) the even primes larger than three,
(L) the straight lines in the plane that are perpendicular to themselves.

There are no even primes larger than three, and no straight lines perpendicular to themselves, so the extensions of P and L are both vacuous. However, it would be absurd to say that the meanings of "even prime larger than three" and "straigh line perpendicular to itself" coincide!

2.1. **Types, points, and equalities.**

If A is a type and t is a construction made according to the rules of A, then we write

$$t : A.$$

We call t an *element* or a *point* of A. The word "element" makes one think that A is a collection of things (which it is not, but its extension is), while "point" emphasizes the view that types are like spaces.

People close to logic, as well as theoretical computer scientists, sometimes say that t is a *term* of type A, as if constructions were just formal symbolic expressions. We eschew this view, because it is analogous to thinking that a function is a symbolic expression with the letter x appearing in it.

Just making constructions without ever doing anything with them would lead to pretty boring mathematics. We should have the ability to *deconstruct* and *use* constructed objects. Thus in type theory there are generally two kinds of operations, the constructors and the deconstructors.

In addition, there may be several ways of constructing the same object, for example by undoing some construction steps (using deconstructors), and then redoing them. Consequently, we should be able to compare the constructed objects. For this purpose we introduce *equations*. An equation

$$s \equiv_A t,$$

states that points s and t of type A are equal. Similarly, we may state that two types A and B are the same, written as

$$A \equiv B.$$

It only makes sense to compare points of the same type. That is, if $s : A$ and $t : B$ then $s \equiv_A t$ is not even false — it is senseless, unless we can establish $A \equiv B$.

Equality satisfies the usual rules of manipulation: reflexivity, symmetry, transitivity, equals may be subsituted for equals. But since there is no logic, we cannot deny an equality: there is no such thing as $\neg(A \equiv B)$ or $A \not\equiv B$. The situation is similar to universal algebra, where structures are described in terms of operations and equations only.

2.2. **Dependent types and points.** There is one last phenomenon that requires out attention, namely *dependency*. Everywhere in mathematics we build objects and spaces which depend on parameters. For instance, the space of real-valued continuous maps on a closed interval, $\mathcal{C}([a,b],\mathbb{R})$, is not really a single space. It is a *family* of spaces, one for each pair (a,b) of endpoints of the interval $[a,b]$. We say that $\mathcal{C}([a,b],\mathbb{R})$ *depends* on the parameters a and b. Let us also note that $\mathbb{R}$ is *not* a parameter, but instead a fixed space that has been previously constructed.

Not only do types depend on parameters, but so do their points. The midpoint $(a+b)/2$ of the closed interaval $[a,b]$ is not a fixed point, because it and its type both depend on the parameters a and b.

Formal type theory provides an explicit notation for dependency, namely

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

indicates that $B(x)$ is a type which depends on a parameter $x : A$, and similarly

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

that $t(x)$ is a point of $B(x)$ which depends on $x : A$. Such formalism is necessary if one studies the meta-theory of type theory, but for us a more familiar notation will suffice.

In §2.3.7 we shall introduce the *universe* U, which is the type of all (small) types. It captures the idea that types themselves are constructions. Then in §2.3.4 we will learn how to form the *function type* $A \to B$ of all maps from A to B. Using these, we may express the fact that A is a type by stating that A is a point of U,

$$A : \mathsf{U}.$$

You should be asking whether $\mathsf{U} : \mathsf{U}$, a question that is addressed in §2.3.7.

A type B depending on a parameter of type A is simply a map from A to U,

$$B : A \to \mathsf{U}.$$

We can manage without a special notation for dependent points.

To summarize, the basic building blocks of type theory are as follows:

CONCEPT	NOTATION
type	$A : \mathsf{U}$
point	$t : A$
dependent type	$B : A \to \mathsf{U}$
dependent point	$x : A \vdash t(x) : B(x)$
equal types	$A \equiv B$
equal points	$s \equiv_A t$

There is nothing else. For instance, stating that t does *not* have type A, or that an equation does not hold, falls outside of the realm of type theory. You will have to take it on faith that these restrictions exist for good reasons, and that eventually they will allow us to reap some benefits. (Compare the situation to universal algebra: restricting the axioms to be just equalities is, well, restrictive, but the resulting theory is still very rich.)

2.3. **Basic type-theoretic constructions.**

There is an established method for introducing a new type:

- *formation:* we postulate rules which explain how the new type is constructed,
- *construction:* we postulate rules for constructing the points of the new type,
- *deconstruction:* we postulate rules for deconstructing or using the points of the new type,
- *equations:* we postulate equations which explain what happens when a construction is followed by a deconstruction, or vice versa.

Pay attention to the difference between the formation and the construction rules: latter explain how to form the type itself, while the latter explain how to form the points of the type.

2.3.1. *The unit type* $\mathsf{1}$.

Our first example is the *unit* type. Intuitively speaking, it corresponds to a one-point space. Here are the rules:

- formation: $\mathsf{1}$ is a type,
- construction: $\star$ is a point of $\mathsf{1}$,
- deconstruction: there are no deconstructors,
- equations: any two points s and t of type $\mathsf{1}$ are equal.

There are no deconstructors because nothing useful can be extracted from $\star$. Or to put it another way, $\star$ has no internal structure, hence there is nothing to deconstruct.

Type theorists would write the rules for the unit type as follows:

$$\frac{}{\mathsf{1} : \mathsf{U}} \qquad\qquad \frac{}{\star : \mathsf{1}} \qquad\qquad \frac{s : \mathsf{1} \qquad t : \mathsf{1}}{s \equiv_{\mathsf{1}} t}$$

Such "fractions" should be read as inference rules: above the line are the *premises*, and below the line is the *conclusion*. We only occasionally use inference rules, but be warned that hard-core type theorists thrive on writing pages and pages of inference rules, and they can read them really fast, too.

2.3.2. *The empty type* $\mathsf{0}$.

Next we have the empty type $\mathsf{0}$, which intuitively corresponds to the empty space. Thus, not suprisingly, it will have no constructors. But how do we say that there is nothing in it? Here are the rules:

- formation: $\mathsf{0}$ is a type,
- construction: there are no constructors
- deconstruction: for every $e : \mathsf{0}$ and a type A there is a point $\mathsf{absurd}_A(e) : A$.
- equations: if $e : \mathsf{0}$, then any two points of any type are equal.

Written as inference rules, these are:

$$\frac{}{\mathsf{0} : \mathsf{U}} \qquad\qquad \frac{e : \mathsf{0} \qquad A : \mathsf{U}}{\mathsf{absurd}_A(e) : A} \qquad\qquad \frac{e : \mathsf{0} \qquad s : A \qquad t : A}{s \equiv_A t}$$

If you know a little bit of category theory you should draw an analogy with the initial object: the deconstructor $\mathsf{absurd}_A(t)$ gives us a way of constructing a point of any type A from a point $e : \mathsf{0}$. This is the type-theoretic way of saying that "$\mathsf{0}$ is as empty as any other type". The equation rule expresses the absurdity of having a point in $\mathsf{0}$, at the level of equations.

2.3.3. *The booleans* 2. The type of 2 is a space with two distinct points. We thus have:

- formation: 2 is a type,
- constructors: 1_2 and 0_2 are points of 2.

We carefully subscripted the points 1_2 and 0_2 with $_2$ in order to distinguish them from the empty and the unit type, and from the natural numbers zero and one. We must do so because a point can only ever have one type. When such formalism becomes cumbersome we shall lapse into using the same symbol for several different points, but until then let us be very explicit.

The deconstructor and the equations for 2 are as follows: given a dependent type $A : 2 \to \mathsf{U}$, points $s : A(0_2)$ and $t : A(1_2)$, and a point $b : 2$, there is a point

$$\mathsf{ind}_2^A(b, s, t) : A(b) \tag{1}$$

such that

$$\mathsf{ind}_2^A(0_2, s, t) \equiv_{A(0_2)} s,$$
$$\mathsf{ind}_2^A(1_2, s, t) \equiv_{A(1_2)} t.$$

Two alternative, but less explicit notations for (1) are

$$\mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t \qquad\qquad \text{and} \qquad\qquad \begin{cases} s & \text{if } b \equiv_2 0_2, \\ t & \text{if } b \equiv_2 1_2. \end{cases}$$

The deconstructor gets its symbol "ind" because it can be read as an induction principle: there are two base cases, 0_2 and 1_2, and no induction step.

2.3.4. *Dependent products* $\Pi(x : A) . B(x)$ *and exponentials* $A \to B$. The dependent product is our first example of a type-theoretic construction that has a somewhat interesting formation rule: if $A : \mathsf{U}$ and $B : A \to \mathsf{U}$ then $\Pi(x : A) . B(x) : \mathsf{U}$.

The points of $\Pi(x : A) . B(x)$ are *(dependent) maps*, which take $x : A$ to a point of $B(x)$. These are formed using the so-called λ*-abstraction*: given a dependent point

$$x : A \vdash t(x) : B(x),$$

there is a point

$$\lambda(x : A) . t(x) : \Pi(x : A) . B(x).$$

A more familiar notation for $\lambda(x : A) . t(x)$ is

$$x \mapsto t(x),$$

which reads "x maps to $t(x)$". The variable x is bound in the λ-abstraction. When the type of x is clear we shall abbreviate the λ-abstraction as $\lambda x . t(x)$. Nested λ-abstractions

$$\lambda(x : A) . \lambda(y : B) . \lambda(z : C) . t(x, y, z)$$

may be abbreviated as $\lambda x\, y\, z . t(x, y, z)$.

Given a dependent map $f : \Pi(x : A) . B(x)$ and a point $a : A$, we may form the *application* $f(a) : B(a)$. We also write $f\, a$ instead of $f(a)$.

Nested dependent products are not uncommon. A map

$$f : \Pi(x : A) . \Pi(y : B(x)) . C(x, y)$$

takes a point $a : A$ to a dependent map $f\, a : \Pi(y : B(a)) . C(a, y)$. The map $f\, a$ can further be applied to a point $b : B(a)$ to give a point $(f\, a)\, b : C(a, b)$. Let us agree that application is right-associative, i.e., $f\, a\, b = (f\, a)\, b$.

2.3.5. *Dependent sums* $\Sigma(x : A) . B(x)$.

2.3.6. *The natural numbers* N. The type of natural number N has two constructors:

- $\mathsf{0} : \mathsf{N}$,
- if $t : \mathsf{N}$ then $\mathsf{S}\,t : \mathsf{N}$.

The deconstructor is a form of induction principle. Suppose $A : \mathsf{N} \to \mathsf{U}$ is a type which depends on N. Given $a : A(\mathsf{0})$,

$$f : \Pi(k : \mathsf{N}) \,.\, A(k) \to A(\mathsf{S}\,k),$$

and $n : \mathsf{N}$, there is a point

$$\mathsf{ind}_{\mathsf{N}}^{A}(a, f, n) : A(n),$$

such that

$$\begin{aligned} \mathsf{ind}_{\mathsf{N}}^{A}(a, f, \mathsf{0}) &\equiv_{A(\mathsf{0})} a, \\ \mathsf{ind}_{\mathsf{N}}^{A}(a, f, \mathsf{S}\,n) &\equiv_{A(\mathsf{S}\,n)} f\,n\,(\mathsf{ind}_{\mathsf{N}}^{A}(a, f, n)). \end{aligned}$$

2.3.7. *The universe* U.

2.4. **Derived constructions.**

2.4.1. *Simple products* $A \times B$.

2.4.2. *Sums* $A + B$.

2.5. **What about $\cap$ and $\cup$?**

3. The homotopy-theoretic reading of types

So far we have been dealing mostly with pure formalism. Is there a way to explain what the types really are? Yes, there are several ways! Type theory is so general that it can be understood in many ways: types are datatypes in a programming language, types are sets, types are symmetric transitive relations on algebraic lattices, types are spaces. It is this last explanation which we will rely on and that we now briefly outline.

Keep in mind that what follows is an informal explanation of how to imagine types. We are not defining anything, nor are we introducing any new mathematics. We are just going to draw helpful pictures.

Let us augment our table of basic concepts with the homotopy-theoretic explanation:

Concept	Notation	Homotopy theory
type	A type	space
point	$t : A$	point
dependent type	$x : A \vdash B(x)$	fibration B over A
dependent point	$x : A \vdash s(x) : B(x)$	section of B
equal types	$A \equiv B$	equal spaces
equal points	$s \equiv_A t$	equal points

The most revealing is the interpretation of dependent type: a type $B(x)$ depending on $x : A$ is a fibration with base A whose fiber at x is the space $B(x)$.

We may proceed with the basic constructions:

CONCEPT	NOTATION	HOMOTOPY THEORY
unit type	$\mathsf{1}$	one-point space
empty type	$\mathsf{0}$	empty space
boolean type	$\mathsf{2}$	two-point discrete space
natural numbers	N	discrete space $\mathbb{N}$
dependent sum	$\Sigma(x : A) \,.\, B(x)$	total space
dependent product	$\Pi(x : A) \,.\, B(x)$	space of sections

All of these should be accompanied with nice pictures. For instance, the induction principle for N can be seen as giving a way of producing a section of a fibration over N.

You will notice that we have not given a homotopy-theoretic interpretation of the universe U. What would that be? A space of all spaces, presumably, but have we ever seen such a thing in homotopy theory? Things are going to get interesting.

REFERENCES

[1] Richard Fitzpatrick. *Euclid's Elements*. Self-published, 2009. Available at `https://farside.ph.utexas.edu/Books/Euclid/Elements.html`.