Type theory: Basic ideas

Ansten Klev
Czech Academy of Sciences

Logic department at Celetná

What is a type?

A type is a kind, sort, category of object.

Examples:

- human being, horse, dandelion (pampeliška), ...
- mountain, river, dessert, ...
- city, ice hockey team, primary school, ...
- knife, violin, pencil, ...
- natural number, integer, real number, set, function, ...
- proposition, propositional connective, quantifier, ...

What is type theory?

Type theory is the theory of types from logic and mathematics.

Empirical types are not part of the motivating examples.

This is not to say that type theory cannot be applied to the empirical domain.

What is it good for?

As a philosopher I find that type theory provides a rich (unmatched, in fact) conceptual toolbox for thinking about logic and the foundations of mathematics.

In the practice of computer programming, types entail greater safety.

For a mathematician, the close connection to category theory is perhaps the strongest selling point.

The general form of type theory

Type theory makes judgements of two forms:

$a : \alpha$	"a is an object of type α"
$a = b : \alpha$	"a and b are equal objects of type α"

There is great flexibility in what can take the place of α here.

But it needs to be clear what an object of type α is,

- its criterion of application,

and what it is for objects of type α to be equal,

- its criterion of identity.

Inductive definition

A general method for defining types (inductive datatypes).

We define a type α by laying down how the elements of α are generated.

$$0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{\mathbf{s}(n) : \mathbb{N}}$$

Not every element of $\mathbb{N}$ is formed in this way, e.g. $2 + 2$.

> *To be an element of* $\mathbb{N}$ *is to evaluate either to 0 or to* $\mathbf{s}(n)$*, where* $n : \mathbb{N}$*.*

This is the criterion of application for $\mathbb{N}$.

We stipulate, quite generally, that an element of an inductively defined type is something that can be evaluated (a program).

Another example: Well-formed formulas

The well-formed formulas of propositional logic, $\mathbb{P}$:

$$\frac{n : \mathbb{N}}{p_n : \mathbb{P}} \qquad \frac{A : \mathbb{P}}{\neg A : \mathbb{P}} \qquad \frac{A : \mathbb{P} \qquad B : \mathbb{P}}{A \wedge B : \mathbb{P}}$$

$$\frac{A : \mathbb{P} \qquad B : \mathbb{P}}{A \vee B : \mathbb{P}} \qquad \frac{A : \mathbb{P} \qquad B : \mathbb{P}}{A \supset B : \mathbb{P}}$$

Criterion of application:

> *To be an element of* $\mathbb{P}$ *is to evaluate to* p_n, $\neg A$, $A \wedge B$, $A \vee B$, *or* $A \supset B$.

Final examples: booleans (truth values) and empty type

The type $\mathbb{B}$ is defined by listing its two elements:

$$\top : \mathbb{B} \qquad\qquad \bot : \mathbb{B}$$

Criterion of application:

To be an element of $\mathbb{B}$ *is to evaluate either to* $\top$ *or to* $\bot$*.*

The empty type $\mathbb{O}$ is defined by its having no defining clauses.

Criterion of application:

There is no element of $\mathbb{O}$*.*

Type theory vs set theory: Membership

One must distinguish the internal relation between an object and its type,

$$a : \alpha$$

from the binary propositional function of membership in set theory,

$$a \in b$$

Examples to illustrate the difference

1. Let $\mathsf{id}_{\mathbb{N}}$ be the identity function on $\mathbb{N}$. Whereas

$$\mathsf{id}_{\mathbb{N}} \in \mathbb{N}$$

is well formed (albeit false),

$$\mathsf{id}_{\mathbb{N}} : \mathbb{N}$$

is ill formed (nonsense).

2. Whereas

$$\emptyset : \mathbb{V}$$

is well formed (and therefore correct),

$$\emptyset \in \mathbb{V}$$

is not well formed, since $\mathbb{V}$ is not a set.

A basic type theory

I will now present a basic type theory.

We assume certain ground types, for instance $\mathbb{N}$ and $\mathbb{B}$.

The composite types will be function types and product types.

Types

Ground types: $\mathbb{N}$, $\mathbb{B}$, $\mathbb{O}$, ...

Function types:

$$\frac{\alpha : \textbf{type} \qquad \beta : \textbf{type}}{\alpha \to \beta : \textbf{type}}$$

Product types:

$$\frac{\alpha : \textbf{type} \qquad \beta : \textbf{type}}{\alpha \times \beta : \textbf{type}}$$

Objects I

Natural numbers

$$0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{\mathbf{s}(n) : \mathbb{N}}$$

Functions may be defined on $\mathbb{N}$ by recursion, e.g.

$$m + 0 = m : \mathbb{N}$$
$$m + \mathbf{s}(n) = \mathbf{s}(m + n) : \mathbb{N}$$

Such a function need not have $\mathbb{N}$ as its co-domain, e.g.

$$\mathsf{sg}(0) = \top : \mathbb{B}$$
$$\mathsf{sg}(\mathbf{s}(n)) = \bot : \mathbb{B}$$

This is a function from $\mathbb{N}$ to $\mathbb{B}$.

Objects II

Booleans

$$\top : \mathbb{B} \qquad\qquad \bot : \mathbb{B}$$

Functions may be defined on $\mathbb{B}$ by case distinction, e.g.

$$\textbf{if } \top \textbf{ then } a \textbf{ else } b \;=\; a : \alpha$$
$$\textbf{if } \bot \textbf{ then } a \textbf{ else } b \;=\; b : \alpha$$

Empty type

There are no objects of type $\mathbb{O}$, but for every type α, there is a function R_α from $\mathbb{O}$ to α.

(This is "the empty function" with co-domain α.)

Objects III

Variables For each type α, there are enough variables of type α.

Function type

$$\frac{\begin{array}{c} x : \alpha \\ | \\ b : \beta \end{array}}{\lambda x.b : \alpha \to \beta} \qquad \text{(Abstraction)}$$

$$\frac{f : \alpha \to \beta \qquad a : \alpha}{f(a) : \beta} \qquad \text{(Application)}$$

Defining equation

$$(\lambda x.b)(a) = b[a/x] : \beta$$

Objects IV

Product type

$$\frac{a : \alpha \qquad b : \beta}{\langle a, b \rangle : \alpha \times \beta} \quad \text{(Pairing)}$$

$$\frac{c : \alpha \times \beta}{\mathbf{fst}(c) : \alpha} \qquad \frac{c : \alpha \times \beta}{\mathbf{snd}(c) : \beta} \quad \text{(Projection)}$$

Defining equations

$$\mathbf{fst}(\langle a, b \rangle) = a : \alpha$$
$$\mathbf{snd}(\langle a, b \rangle) = b : \beta$$

Derivations

We derive judgements, $a : \alpha$, in natural deduction style.

$$\frac{(x : \alpha)}{\lambda x.x : \alpha \to \alpha}$$

$$\frac{\dfrac{(x : \alpha)}{\lambda y.x : \beta \to \alpha}}{\lambda x.\lambda y.x : \alpha \to (\beta \to \alpha)}$$

$$\frac{\dfrac{(x : \alpha \times \beta)}{\mathbf{fst}(x) : \alpha}}{\lambda x.\mathbf{fst}(x) : (\alpha \times \beta) \to \alpha}$$

$$\frac{\dfrac{\dfrac{(x : \alpha) \qquad (y : \beta)}{\langle x, y \rangle : \alpha \times \beta}}{\lambda y.\langle x, y \rangle : \beta \to (\alpha \times \beta)}}{\lambda x.\lambda y.\langle xy \rangle : \alpha \to (\beta \to (\alpha \times \beta))}$$

A mathematical question

Can we give sufficient conditions on the structure of a type for it to be inhabited?

For instance, $\alpha \to \alpha$ is always inhabited, no matter which type α is.

What about

$$((\alpha \to \beta) \to \alpha) \to \alpha \ ?$$

It turns out that this is not in general inhabited.

Computation

A definitional equation becomes a rule of computation when directed from left to right.

For example, the definitional equation

$$m + 0 = m$$

is turned into the computation rule

$$m + 0 \twoheadrightarrow m$$

and the equation

$$(\lambda x.b)(a) = b[a/x]$$

is turned into the computation rule

$$(\lambda x.b)(a) \twoheadrightarrow b[a/x]$$

Normalization

Do all computations halt?

This computation does not halt:

$$(\lambda x.x(x))(\lambda x.x(x)) \twoheadrightarrow (\lambda x.x(x))(\lambda x.x(x)) \twoheadrightarrow \cdots$$

But the term $(\lambda x.x(x))(\lambda x.x(x))$ cannot be typed.

With types present, all computations halt – normalization.