Introduction to (Homotopy) Type Theory
or
Naïve Type Theory

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

April 6, 2018

In memoriam

Martin Hofmann (1965 - 2018)

Hochplatte 2017

Cameron, me, Annette, Martin

You guys are both my witnesses... He insinuated that ZFC set theory is superior to Type Theory!

Set Theory (ZFC)

- Formulated in classical predicate logic with equality.
- One relation $a \in A$ (a is an element of the set A).
- Axioms: extensionality, pairing, union, powerset, infinity, comprehension, regularity, replacement and choice.
- Can be used to represent most (all ?) mathematical concepts.

Naïve Set Theory

Use sets intuitively, don't refer to the axioms explicitely.

Type Theory (Martin-Löf)

- Basic judgments (static)

$$a : A \quad (a \text{ is an element of type } A),$$
$$a \equiv_A b \quad (a \text{ and } b \text{ are definitionally equal elements of type } A).$$

- Defined using typing rules defining e.g. $\Gamma \vdash a : A$, using contexts of assumptions $\Gamma = x_0 : A_0, x_1 : A_1 \ldots, x_n : A_n$.
- Basic type formers: Π-types, Σ-types, equality types, inductive types, universes, ...
- Uses the propositions as types translation. Intuitionistic logic.
- Different flavours: Intensional Type Theory (ITT), Extensional Type Theory (ETT), Homotopy Type Theory (HoTT)
- Implementations: NuPRL, Coq, Agda, Lean, Idris, ...

Naïve Type Theory

Use types intuitively, don't refer to the rules explicitely.

$$\frac{\text{Set Theory}}{\text{Type Theory}} = \frac{\text{Python}}{\text{Haskell}}$$

Simple types

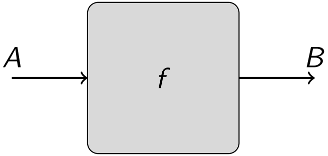
Given types A, B we can form

Products $(A \times B)$ The elements are tuples $(a, b) : A \times B$, where $a : A$ and $b : B$.

Sums $(A + B)$ The elements are injections $\mathrm{left}\, a, \mathrm{right}\, b : A + B$ where $a : A$ and $b : B$ respectively.

Unit (1), Empty type ($\emptyset$) Nullary product and sum. $() : 1$ but no elements in $\emptyset$.

Functions $(A \to B)$ A function $f : A \to B$ is a way to map elements $a : A$ to $f\ a : B$ (black box).



Propositions as types (propositional logic)

Given a proposition P we assign to it the type of its evidence $[\![P]\!]$:

$$\begin{aligned} [\![P \Rightarrow Q]\!] &:\equiv [\![P]\!] \to [\![Q]\!] \\ [\![P \wedge Q]\!] &:\equiv [\![P]\!] \times [\![Q]\!] \\ [\![\mathrm{True}]\!] &:\equiv \mathbf{1} \\ [\![P \vee Q]\!] &:\equiv [\![P]\!] + [\![Q]\!] \\ [\![\mathrm{False}]\!] &:\equiv \mathbf{0} \end{aligned}$$

Other connectives can be defined:

$$\begin{aligned} \neg P &:\equiv P \Rightarrow \mathrm{False} \\ P \Leftrightarrow Q &:\equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P) \end{aligned}$$

Example

To show that

$$P \wedge (Q \vee R) \Rightarrow (P \wedge Q) \vee (P \wedge R)$$

we need to find a function

$$f : P \times (Q + R) \to (P \times Q) + (P \times R)$$

which we define as follows:

$$\begin{aligned} f\,(p, \mathrm{left}\; q) &:\equiv \mathrm{left}\,(p, q) \\ f\,(p, \mathrm{right}\; r) &:\equiv \mathrm{right}\,(p, r) \end{aligned}$$

Exercise

Show that

$$P \wedge (Q \vee R) \Leftrightarrow (P \wedge Q) \vee (P \wedge R)$$

Dependent types

Given a type A a dependent type $B : A \to \mathbf{Type}$ assigns to every element $a : A$ a type $B\,a : \mathbf{Type}$.
Here **Type** is the universe of (small) types.

- **Π-types** An element $f : \Pi x : A.B\,x$ is a function that maps elements $a : A$ to $f\,a : B\,a$.
- **Σ-types** Elements are tuples $(a, b) : \Sigma x : A.B\,x$ where $a : A$ and $b : B\,a$

$\to$ and $\times$ arise as special cases:

$$A \to B \equiv \Pi - : A.B$$
$$A \times B \equiv \Sigma - : A.B$$

Dependent types (Examples)

We write A^n for the type of n-tuples of elements of A.

$$\begin{aligned}\mathrm{zeroes} &: \Pi n : \mathbb{N}.\mathbb{N}^n \\ \mathrm{zeroes}\, n &:\equiv \underbrace{(0, 0, \dots, 0)}_{n}\end{aligned}$$

$(3, (1, 2, 3)) : \Sigma n : \mathbb{N}.\mathbb{N}^n$ because $(1, 2, 3) : \mathbb{N}^3$

Puzzle

What is a good name for $\Sigma n : \mathbb{N}.A^n$?

All type formers in one picture

We can derive binary operations from the dependent ones:

$$
\begin{aligned}
A + B &:\equiv \Sigma b : \mathrm{Bool}.\mathrm{if}\, b\, \mathrm{then}\, A\, \mathrm{else}\, B \\
A \times B &:\equiv \Pi b : \mathrm{Bool}.\mathrm{if}\, b\, \mathrm{then}\, A\, \mathrm{else}\, B
\end{aligned}
$$

$$
\begin{tikzcd}
& \Pi \arrow[dl] \arrow[dr, dashed] & & \Sigma \arrow[dl] \arrow[dr, dashed] & \\
\to & & \times & & +
\end{tikzcd}
$$

$\to$ goes from the dependent to the non-dependent version.
$\dashrightarrow$ goes from the indexed version to the binary one.

Propositions as types (predicate logic)

Given a type A a predicate over A is a dependent type $A \to \mathbf{Type}$

$$
\begin{aligned}
[\![\forall x : A.P(x)]\!] &:\equiv \Pi x : A.[\![P(x)]\!] \\
[\![\exists x : A.P(x)]\!] &:\equiv \Sigma x : A.[\![P(x)]\!]
\end{aligned}
$$

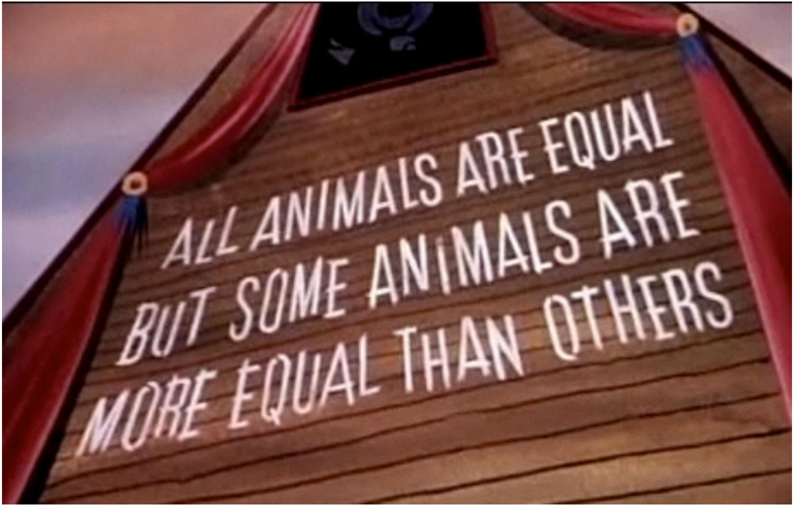
Example:

$$(\forall x : A.P\,x \wedge Q\,x) \to (\forall x : A.P\,x) \wedge (\forall x : A.Q\,x)$$

$$
\begin{aligned}
&f : ((\Pi x : A.P\,x \times Q\,x) \to (\Pi x : A.P\,x) \wedge (\Pi x : A.Q\,x)) \\
&f\ h :\equiv (\lambda x.\mathrm{fst}\,(h\,x), \lambda x.\mathrm{snd}\,(h\,x))
\end{aligned}
$$

where

$$
\begin{array}{ll}
\mathrm{fst} : A \times B \to A & \mathrm{snd} : A \times B \to B \\
\mathrm{fst}\,(a, b) :\equiv a & \mathrm{snd}\,(a, b) :\equiv b
\end{array}
$$


ALL ANIMALS ARE EQUAL
BUT SOME ANIMALS ARE
MORE EQUAL THAN OTHERS

Intensional equality

Equality is a type former, that is given $a, b : A$ we can form the type $a =_A b : \mathbf{Type}$.
Given $a : A$ we can construct $\mathrm{refl}_a : a =_A a$.
We can view this as a definition of equality.
We can derive for example:

$$\begin{aligned} &\mathrm{trans} : \Pi_{a,b,c:A} a =_A b \to b =_A c \to a =_A c \\ &\mathrm{trans}\,\mathrm{refl}_a\, p :\equiv p \end{aligned}$$

In the same way we can derive

$$\begin{aligned} \mathrm{sym} \quad & : \Pi_{a,b:A} a =_A b \to b =_A a \\ \mathrm{resp} \quad & : \Pi f : A \to B.\Pi_{a,b:A} a =_A b \to f\ a =_B f\ b \\ \mathrm{sym}\,\mathrm{refl}_a \quad & :\equiv \mathrm{refl}_A \\ \mathrm{resp}\, f\, \mathrm{refl}_a & :\equiv \mathrm{refl}_{f\ a} \end{aligned}$$

Uniqueness of equality proofs

Can we prove

$$\mathrm{uep} : \Pi_{a,b:A}\Pi p, q : a =_A b.p =_{a=_A b} q$$

It seems yes:

$$\mathrm{uep}\,\mathrm{refl}_a\,\mathrm{refl}_a :\equiv \mathrm{refl}_{\mathrm{refl}_A}$$

The J-eliminator

However, in intensional Martin-Löf Type Theory, dependent functions out of an equality type can only be defined by J.
To define a function

$$f : \Pi x, y : A.\Pi p : x = y.C\, x\, y\, p$$

where $C : \Pi x, y : A, x = y \to \mathbf{Type}$, it is sufficient to supply:

$$f\, x\, x\, \mathrm{refl}_x :\equiv g\, x$$

where $g : \Pi x : A.C\, x\, x\, \mathrm{refl}_x$.
Formally, we write $f :\equiv J\, C\, g$.

Hofmann-Streicher's Groupoid model

Martin Hofmann and Thomas Streicher have shown that uep is not derivable from J using the groupoid model of Type Theory.

Adding K ?

This can be fixed by adding a 2nd eliminator.
To define a function

$$f : \Pi x : A . \Pi p : x = x . C\, x\, p$$

where $C : \Pi x : A, x = x \to \textbf{Type}$, it is sufficient to define

$$f\, x\, \mathrm{refl}_x :\equiv g\, x$$

where $g : \Pi x : A . C\, x\, \mathrm{refl}_x$.
Formally, we write $f :\equiv K\, C\, g$.

Univalence ?

However, K is incompatible with Voevodsky's univalence principle which is the cornerstone of Homotopy Type Theory (HoTT).

Vladimir Voevodsky (1966 - 2017)

What is equality of types?

- Easier question: What is equality of propositions?
- Follow up: What is a proposition?

What is a proposition?

classical

$$\textbf{Prop} = \mathrm{Bool}$$

$$\text{Propositional extensionality}: \; P = Q \Leftrightarrow (P \Leftrightarrow Q)$$

Type Theory (naive)

$$\textbf{Prop} = \textbf{Type}$$

- Axiom of choice (AC) is provable.

$$(\forall x : A.\exists y : B.R\,x\,y) \to \exists f : A \to B.\forall x : A.R\,x\,(f\,x)$$

- Subset inclusion may not be injective.

$$\{x : A \mid P\,x\} = \Sigma x : A.P\,x$$

What is a proposition?

Type Theory (HoTT)

$$
\begin{aligned}
\mathbf{Prop} &:\equiv \{A : \mathbf{Type} \mid \forall x, y : A. x = y\} \\
&\equiv \Sigma A : \mathbf{Type}. \Pi x, y : A. x = y
\end{aligned}
$$

- AC not provable, implies excluded middle (Diaconescu)
- Subset inclusion injective.
- Retain propositional extensionality.
  $(P = Q) \Leftrightarrow (P \Leftrightarrow Q)$
- Subobject classifier in a (predicative) Topos

Propositions as Types (HoTT)

Goal : $[\![P]\!]$: **Prop**

$$
\begin{aligned}
[\![P \implies Q]\!] &:\equiv [\![P]\!] \to [\![Q]\!] \\
[\![P \land Q]\!] &:\equiv [\![P]\!] \times [\![Q]\!] \\
[\![\mathrm{True}]\!] &:\equiv \mathbf{1} \\
[\![P \lor Q]\!] &:\equiv \|[\![P]\!] + [\![Q]\!]\| \\
[\![\mathrm{False}]\!] &:\equiv \mathbf{0} \\
[\![\forall x : A.P(x)]\!] &:\equiv \Pi x : A.[\![P(x)]\!] \\
[\![\exists x : A.P(x)]\!] &:\equiv \|\Sigma x : A.[\![P]\!]\|
\end{aligned}
$$

where $\|\_\|$: **Type** $\to$ **Prop** such that

$$\|A\| \to P \simeq A \to P \qquad \text{for } P : \mathbf{Prop}$$

What is a set?

A set is a type whose equalities are propositions.

$$\mathbf{Set} \equiv \{A : \mathbf{Type} \mid \forall x, y : A.\mathrm{isProp}\,(x = y)\}$$

where $\mathrm{isProp}\, A \equiv \forall x, y : A.x = y$

Propositional extensionality becomes set extensionality.

$$A = B \cong (A \cong B) \qquad (A, B : \mathbf{Set})$$

$$\begin{aligned} A \cong B :\equiv \Sigma f &: A \to B \\ g &: B \to A \\ \eta &: \Pi x : B.f\,(g\,x) = x \\ \epsilon &: \Pi x : A.g\,(f\,x) = x \end{aligned}$$

- $\mathrm{Bool} = \mathrm{Bool}$ is not a proposition.
- Hence **Set** is not a set (not just due to size)!
- Correct statement: the canonical map $A = B \to (A \cong B)$ is an isomorphism.

Equality of types (univalence)

Fir general types we need to modify the previous definition, replacing isomorphism by equivalence.

$$\begin{aligned} A \simeq B :\equiv \Sigma f : A \to B \\ g : B \to A \\ \eta : \Pi y : B . f\,(g\,y) = y \\ \epsilon : \Pi x : A . g\,(f\,x) = x \\ \delta : \Pi x : A . \eta\,(f\,x) = f\,(\epsilon\,x) \end{aligned}$$

$$A = B \simeq (A \simeq B) \qquad (A, B : \mathbf{Type})$$

- I write $f\,(\epsilon\,x)$ for $\mathrm{resp}\, f\,(\epsilon\,x)$
- Asymmetric

$$\tau : \Pi y : B . \epsilon\,(g\,x) = g\,(\eta\,y) ???$$

- Correct statement: the canonical map $A = B \to (A \simeq B)$ is an equivalence.

Extensionality

Should mathematical objects be considered equal, if they are defined the same way (intensional) or if they behave the same way (extensional)?

Examples

functional extensionality the functions $\lambda x.x + 0$ and $\lambda y.y + 0$ are intensionally different but extensionally equal.

propositional extensionality the propositions True and $\neg\mathrm{False}$ are intensionally different but extensionally equal.

set extensionality The sets $\mathbb{N}$ (Peano numbers) and $\mathrm{List\,Bool}$ (binary numbers) are intensionally different but extensionally equal.

Sets vs Types

- Set theory has *functional extensionality* and *propositional extensionality*.
- But it lacks *set extensionality*.
- Indeed we can distinguish isomorphic sets (e.g. von Neuman numerals and Zermelo numerals).
- Intensional Type Theory lacks all extensionality principles.
- However, we cannot distinguish isomorphic types.
- Extensional Type Theory has the same extensionality principles as set theory.
- It also requires uniqueness of equality proofs, hence is inconsistent with set extensionality.
- Homotopy Type Theory has all extensionality prinicples (consequence of univalence).

Higher Groupoids

- While we cannot prove univalence from J we can show that every type is a groupoid:

$$\mathrm{trans}\,p\,\mathrm{refl} = p \qquad \mathrm{trans}\,p\,(\mathrm{sym}\,p) = \mathrm{refl}$$
$$\mathrm{trans}\,\mathrm{refl}\,p = p \qquad \mathrm{trans}\,(\mathrm{sym}\,p)\,p = \mathrm{refl}$$
$$\mathrm{trans}\,(\mathrm{trans}\,p\,q)\,r = \mathrm{trans}\,p\,(\mathrm{trans}\,q\,r)$$

- Indeed to model types in HoTT we need ω-groupoids.
- Voevodsky was using Kan simplicial sets to model HoTT, including the univalence principle.
- However, the metatheory was classical. It was believed but not known wether univalence is constructive.
- This was resolved by Coquand et als work on cubical type theory which uses cubical sets to interpret univalence constructively.
- This gives rise to implementations of HoTT.
- Higher groupoids also lead to Higher Inductive Types (HITs) which are extremely useful when representing mathematical concepts choice-free.

Thesis

To build towers of abstractions that can withstand the rigorous demands of formal (computer-aided) Mathematics we need foundations that support extensional reasoning and structural Mathematics in their core. Homotopy Type Theory is currently the only foundational calculus that fits this bill.