Homotopy Type Theory

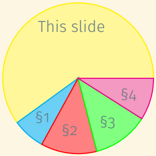
the confluence of logic and space

tslil

2018/11/30

The journey ahead

1. Foundations
2. Type Theory
3. Homotopy Type Theory
4. Univalence



some content copied with permission from a talk with not literally the same name by tslil

Foundations

Motivation

- Why should there be a 'the' foundation?
  - Euclid – who cares if l 'is' $\{(x, y) \in \mathbb{R} \times \mathbb{R} \mid \ldots\}$
  - Arithmetic – who cares if $2 \in 3$
- Set Theory™ does not understand *structures* and their *equalities*:
  - We can ask bad questions: $|7|$, $\mathtt{succ}(\mathbb{C})$, $\pi \cap e \in \mathbb{Q}$, …
  - groups $G \cong H \rightsquigarrow$ "everything that's true of G is true of H"
  - vs $\emptyset \in G$
- In Set Theoretic foundations like ZF:
  - Technically can't *construct* anything
  - Infeasible to actually work *in* the foundations
  - Nothing about *structures* to be gained by doing so

Our foundations should reflect how we do mathematics

The single most exciting thing about this

A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail.

$\sim$ V. V.

A foundation amenable to $\begin{cases} \text{computer verification} \\ \text{proof assistance} \end{cases}$

Proofs *are* programmes, and may be run!

Type Theory

You are already a type theorist

Instead of 'everything is set', why not have different types which reflect the *qualitatively* different partitions into which we naturally place things?

$\mathbb{N}$	type of natural numbers
$\mathbb{Z}$	type of integers
Mon	type of monoids
0	the 'empty' type
1	the 'singleton' type
$\mathcal{U}$	the 'universe' type $\leftarrow \mathrm{B}$

A typical solution

Basics

- Types have terms, $0 : \mathsf{N}, 1 : \mathsf{N}, \ldots$, and terms belong to a *unique* type.
- Given types A, B, we can form more types: $A \times B$, $A + B$, $A \to B$, ...
- These new types behave as one might expect, if we have $a : A, b : B$ then $(a, b) : A \times B$.
- Types come equipped with certain functions: $\mathsf{succ} : \mathsf{N} \to \mathsf{N}$, $\mathsf{pr_A} : A \times B \to A$, $\mathsf{app_{A,B}} : A \times (A \to B) \to B$.

Note: the statement '$a : A$' is not a proposition, it may not be proven or disproven, it is data.

An elaboration

Some types are *inductive*:

- To define a function from the type $A + B$ it suffices to perform case analysis,
- If 0 is the nullary sum then to define a function from 0 it suffices.
- $\mathsf{ind}_{\mathsf{C}}^{\mathsf{A+B}} : (A \to C) \times (B \to C) \to ((A + B) \to C)$
- $\mathsf{ind}_{\mathsf{C}}^{0} : 0 \to C$
- $\mathsf{ind}_{\mathsf{C}}^{\mathsf{N}} : C \times (C \to C) \to (\mathsf{N} \to C)$
- (Surprise?) To define a function from the type $A \times B$ it suffices to define it on all pairs (a, b).

What does it mean to work 'in' this system, and 'prove' things?

Propositions as types (P.A.T.)

Encode propositions as types P and proofs as terms $p : P$.

Logic	Types
P	P
proof of P	$p : P$
$P \wedge Q$	$P \times Q$
$P \vee Q$	$P + Q$
$P \implies Q$	$P \to Q$
$\neg P$	$P \to 0$

Let's implement one of the basic tools of propositional logic,

A DeMorgan Law

$$((\neg P) \wedge (\neg Q)) \implies \neg(P \vee Q)$$

The translation

$$\mathsf{DeMorgan} : ((\neg A) \times (\neg B)) \to \neg(A + B)$$

DeMorgan DeRivation

We want to build a term of type $((\neg A) \times (\neg B)) \to \neg(A + B)$

- It's enough to define it on input inp : $(\neg A) \times (\neg B)$, or (n_a, n_b) where $n_a :\equiv \pi_1(\mathsf{inp}) : \neg A$ and $n_b :\equiv \pi_2(\mathsf{inp}) : \neg B$
- Now DeMorgan$(n_a, n_b) : \neg(A + B) \equiv (A + B) \to 0$
- So again its enough to define it on input $z : A + B$, but by induction it's enough to define it on $a : A$ and $b : B$ separately
- Thus we must give DeMorgan$(n_a, n_b)(a) : 0$, so the puzzle is to inhabit 0 with the context

$$a : A, n_a : A \to 0, n_b : B \to 0$$

- Just apply n_a to a!

DeMorgan DeRivation DeUx

We want to build a term of type $((\neg A) \times (\neg B)) \to \neg(A + B)$

Putting everything together

$$\mathsf{DeMorgan} :\equiv (n_a, n_b) \mapsto \mathsf{case}_{\mathsf{A,B}}(a \mapsto n_a(a), b \mapsto n_b(b))$$

where case is the induction for $+$.

This is a small taste of what working in type theory is like.

How do we express a predicate over a type?

Take P.A.T. literally,

A first pass

A predicate P on A should be something that gives a type $P(a)$ for $a : A$ which is inhabited if *P(a) is true*.

Dependent types

A predicate on A should be a term $P : A \to \mathsf{U}$

Is that all? Well it depends...

Definition

We allow ourselves to form the type $A \to \mathsf{U}$. Terms B of this type are *dependent types* or *type families* varying over A.

We also extend the product forming operation:

Definition

Given $A : \mathsf{U}$ and $B : A \to \mathsf{U}$, we define

- the *dependent product*, $\prod(a : A), B(a)$, to be the type comprising terms $f :(a : A) \to B(a)$
- the *dependent sum*, $\sum(a : A), B(a)$, to be the type comprising terms $(a, b : B(a))$

Propositions as types, or, a pronunciation guide

Idea: encode logic statements as types, proofs as terms.

Logic	Types
Proposition on A	$A \to \mathsf{U}$
proof of $P(a)$	$p : P(a)$
$A \implies B$	$A \to B$
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
$\forall a[P(a)]$	$\prod(a : A), P(a)$
$\exists a[P(a)]$	$\sum(a : A), P(a)$

"Proof relevance"

The theorem of choice

Classically

$$\forall a \in A[\exists b \in B_a[R(a,b)]] \implies \exists c \in (\Pi_A B_a)[\forall a \in A[R(a, c(a))]]$$

Typically

$$\mathsf{ac} : \prod_{(a:A)} \sum_{(b:B)} R(a,b) \quad \to \quad \sum_{(c:\prod(a:A),B)} \prod_{(a:A)} R(a, c(a))$$

This is *true*!

In fact, the conclusion is *equivalent* to the antecedent.

Homotopy Type Theory

Identity crisis

How should we express equality?

P.A.T. $\rightsquigarrow$ for $a, b : A$ there is a type $a =_A b$.

Equality is detected by relations $R : A \to (A \to U)$, but these must be *reflexive*.

“All equations are lies…or useless”

Reflexivity: $\mathsf{refl}_a : a =_A a$

Universal property:

$\prod(a : A), R(a, a, \mathsf{refl}_a) \simeq \prod(a, b : A), \prod(p : a =_A b), R(a, b, p)$

It's ∞ all the way down

Definition

With $a, b, c : A$, $p : a =_A b$, $q : b =_A c$ we can construct

- $p^{-1} : b =_A a$
- $p \blacksquare q : a =_A c$

And we may give terms witnessing

- $\mathsf{lunit} : \mathsf{refl}_a \blacksquare p = p$
- $\mathsf{linv} : p^{-1} \blacksquare p = \mathsf{refl}_a$
- $\mathsf{assoc} : (p \blacksquare q) \blacksquare r = p \blacksquare (q \blacksquare r)$

Note: these are *not* 'strict' equalities, we can only prove them as propositions in our type theory.

The analogy

Type Theory	Category Theory	Topology
types A, B	∞-groupoids	spaces
functions $f : A \to B$	∞-functors	continuous maps
equality $p : a = b$	equivalence	path
reflexivity $\mathsf{refl}_a : a = a$	identity	constant path
symmetry $p^{-1} : b = a$	inverse	path reversal
transitivity $p \centerdot q : a = c$	composition	path concat.

Logic	Topology
propositions	spaces
implications	continuous maps
equalities	paths

Functions are ∞-functors

We think of types as ∞-groupoids.

A function $f : A \to B$ acts on morphisms:

$$\mathsf{ap}_f(a, b) : (a =_A b) \to (fa =_B fb)$$

Dependent types are fibrations

We think of $\mathsf{pr}_1 : (\sum(a : A), B) \to A$ as a fibration.

A path $p : a =_A a'$ in the base space acts on fibres:

$$\mathsf{transport}_{A,B}(a, a', p) : B(a) \to B(a')$$

Univalence

Function extensionality

Given $f, g : A \to B$, what should $f = g$ be?

Definition

Given $f, g : \prod(a : A), B$ define $f \sim g :\equiv \prod(a : A), (fa =_B ga)$ as the type of *homotopies* between f and g.

"a proof of a predicate is determined by those elements for which the predicate holds"

Definition

Function extensionality is the *axiom* that there is a term

$$\mathsf{funext} : f \sim g \to f = g.$$

The analogous question for types

Given $A, B : \mathsf{U}$, what should $A =_{\mathsf{U}} B$ be?

We have a notion of equivalence for types $A \simeq B$, *and* a canonical term idtoeqv $:(A =_U B) \to (A \simeq B)$

Definition

Univalence is the *axiom* that idtoeqv is an equivalence. We name its quasi-inverse

$$\mathsf{ua} : (A \simeq B) \to (A =_{\mathsf{U}} B).$$

Facts

- Univalence implies function extensionality
- Consistent to assume (Voevodsky's model in sSet)
- Consequently, for any proposition $P : \mathsf{U} \to \mathsf{U}$ and witness $A \simeq B$, one cannot show $P(A)$ but not $P(B)$.
- By the *structure identity principle*, this means that many algebraic things are univalent too.

To get a flavour of the S.I.P., let's pick our favourite toy algebraic structure, ~~affine schemes~~ 'sets' with a binary operation

$$\mathsf{Magma} :\equiv \sum(A : \mathsf{Set}), A \to (A \to A)$$

Audience Participation

Given $(A, \star), (B, \bullet) : \mathsf{Magma}$, what does equality mean?

$$\left(f : A \simeq B,\ t : \prod(a, b : A), f(a \star b) =_B (fa) \bullet (fb)\right)$$

$$\xleftrightarrow[\text{funext}]{\text{happly}} \quad \left(f : A \simeq B,\ s : (f \star =_{B \to (B \to B)} \bullet (f \times f))\right)$$

$$\xleftrightarrow[\text{ua}]{\text{idtoeqv}} \quad \left(q : A =_{\mathsf{U}} B,\ r : \mathsf{transport}(q, \star) =_{B \to (B \to B)} \bullet\right)$$

$$\xleftrightarrow[\text{lemmas}]{\text{generic}} \quad p : (A, \star) =_{\mathsf{Magma}} (B, \bullet)$$