Homotopy Type Theory
the confluence of logic and space

tslil
2018/11/30
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: $|7|$, $\text{succ}(\mathbb{C})$, $\pi \cap e \in \mathbb{Q}$, ...
  - groups $G \cong H \rightsquigarrow \text{“everything that’s true of } G \text{ is true of } H\text{”}$
  - 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.

~ V. V.

A foundation amenable to \{computer verification, proof assistance\}

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?

| N   | type of natural numbers  |
| Z   | type of integers         |
| Mon | type of monoids          |
| 0   | the ‘empty’ type         |
| 1   | the ‘singleton’ type     |
| U   | the ‘universe’ type      | ← B
Basics

- Types have terms, $0 : \mathbb{N}$, $1 : \mathbb{N}$, ..., 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:
  \[
  \text{succ} : \mathbb{N} \to \mathbb{N}, \text{pr}_A : A \times B \to A, \text{app}_{A,B} : A \times (A \to B) \to B.
  \]

Note: the statement ‘$a : A$’ is not a proposition, it may not be proven or disproven, it is data.
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.
- $\text{ind}_{C}^{A+B} : (A \to C) \times (B \to C) \to ((A + B) \to C)$
- $\text{ind}_{C}^{0} : 0 \to C$
- $\text{ind}_{C}^{N} : C \times (C \to C) \to (N \to C)$
- (Surprise?) To define a function from the type $A \times B$ it suffices to define it on all pairs $(a, b)$. 

An elaboration
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 $p : P$.

<table>
<thead>
<tr>
<th>Logic</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$P$</td>
</tr>
<tr>
<td>proof of $P$</td>
<td>$p : P$</td>
</tr>
<tr>
<td>$P \land Q$</td>
<td>$P \times Q$</td>
</tr>
<tr>
<td>$P \lor Q$</td>
<td>$P + Q$</td>
</tr>
<tr>
<td>$P \Rightarrow Q$</td>
<td>$P \rightarrow Q$</td>
</tr>
<tr>
<td>$\neg P$</td>
<td>$P \rightarrow 0$</td>
</tr>
</tbody>
</table>
Let’s implement one of the basic tools of propositional logic,

**A DeMorgan Law**

\[ ((\neg P) \land (\neg Q)) \implies \neg(P \lor Q) \]

**The translation**

\[ \text{DeMorgan} : ((\neg A) \times (\neg B)) \rightarrow \neg(A + B) \]
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 \( \text{inp} : (\neg A) \times (\neg B) \), or \( (n_a, n_b) \) where \( n_a :\equiv \pi_1(\text{inp}) : \neg A \) and \( n_b :\equiv \pi_2(\text{inp}) : \neg B \)
- Now \( \text{DeMorgan}(n_a, n_b) : \neg (A + B) \equiv (A + B) \to 0 \)
- 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 \( \text{DeMorgan}(n_a, n_b)(a) : 0 \), 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 \( a \)! 
We want to build a term of type \(((\neg A) \times (\neg B)) \rightarrow \neg (A + B)\)

Putting everything together

\[
\text{DeMorgan} \equiv (n_a, n_b) \mapsto \text{case}_{A,B}(a \mapsto n_a(a), b \mapsto n_b(b))
\]

where \(\text{case}\) 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 $P : A \to U$
Definition
We allow ourselves to form the type $A \rightarrow U$. Terms $B$ of this type are dependent types or type families varying over $A$.

We also extend the product forming operation:

Definition
Given $A : U$ and $B : A \rightarrow U$, we define

- the dependent product, $\prod (a : A), B(a)$, to be the type comprising terms $f : (a : A) \rightarrow B(a)$
- the dependent sum, $\sum (a : A), B(a)$, to be the type comprising terms $(a, b : B(a))$
Idea: encode logic statements as types, proofs as terms.

<table>
<thead>
<tr>
<th>Logic</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposition on $A$</td>
<td>$A \rightarrow U$</td>
</tr>
<tr>
<td>proof of $P(a)$</td>
<td>$p : P(a)$</td>
</tr>
<tr>
<td>$A \implies B$</td>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>$A \land B$</td>
<td>$A \times B$</td>
</tr>
<tr>
<td>$A \lor B$</td>
<td>$A + B$</td>
</tr>
<tr>
<td>$\forall a[P(a)]$</td>
<td>$\prod (a : A), P(a)$</td>
</tr>
<tr>
<td>$\exists a[P(a)]$</td>
<td>$\sum (a : A), P(a)$</td>
</tr>
</tbody>
</table>

“Proof relevance”
The theorem of choice

Classically

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

Typically

$$\text{ac} : \prod_{(a : A)} \sum_{b : B} R(a, b) \rightarrow \sum_{(c : \prod (a : A), B)} \prod_{a : A} R(a, c(a))$$

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. \( \leadsto \) 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: \( \text{refl}_a : a =_A a \)

Universal property:

\[
\prod (a : A), R(a, a, \text{refl}_a) \simeq \prod (a, b : A), \prod (p : a =_A b), R(a, b, p)
\]
Definition

With \( a, b, c : A, p : a =_A b, q : b =_A c \) we can construct

- \( p^{-1} : b =_A a \)
- \( p \cdot q : a =_A c \)

And we may give terms witnessing

- \( \text{lunit} : \text{refl}_a \cdot p = p \)
- \( \text{linv} : p^{-1} \cdot p = \text{refl}_a \)
- \( \text{assoc} : (p \cdot q) \cdot r = p \cdot (q \cdot r) \)

Note: these are not ‘strict’ equalities, we can only prove them as propositions in our type theory.
The analogy

<table>
<thead>
<tr>
<th>Type Theory</th>
<th>Category Theory</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>types $A, B$</td>
<td>$\infty$-groupoids</td>
<td>spaces</td>
</tr>
<tr>
<td>functions $f : A \to B$</td>
<td>$\infty$-functors</td>
<td>continuous maps</td>
</tr>
<tr>
<td>equality $p : a = b$</td>
<td>equivalence</td>
<td>path</td>
</tr>
<tr>
<td>reflexivity $\text{refl}_a : a = a$</td>
<td>identity</td>
<td>constant path</td>
</tr>
<tr>
<td>symmetry $p^{-1} : b = a$</td>
<td>inverse</td>
<td>path reversal</td>
</tr>
<tr>
<td>transitivity $p \cdot q : a = c$</td>
<td>composition</td>
<td>path concat.</td>
</tr>
</tbody>
</table>

Logic

<table>
<thead>
<tr>
<th>propositions</th>
<th>Topology</th>
<th>spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>implications</td>
<td>continuous maps</td>
<td>paths</td>
</tr>
<tr>
<td>equalities</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instances of the analogy

**Functions are \(\infty\)-functors**

We think of types as \(\infty\)-groupoids.

A function \(f : A \rightarrow B\) acts on morphisms:

\[
ap_f(a, b) : (a =_A b) \rightarrow (fa =_B fb)
\]

**Dependent types are fibrations**

We think of \(\text{pr}_1 : (\sum (a : A), B) \rightarrow A\) as a fibration.

A path \(p : a =_A a'\) in the base space acts on fibres:

\[
\text{transport}_{A,B}(a, a', p) : B(a) \rightarrow B(a')
\]
Univalence
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

\[
\text{funext} : f \sim g \to f = g.
\]
The analogous question for types

Given $A, B : U$, what should $A =_U B$ be?

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

**Definition**

Univalence is the *axiom* that $\text{idtoeqv}$ is an equivalence. We name its quasi-inverse

$$\text{ua} : (A \simeq B) \to (A =_U B).$$
Some repercussions

**Facts**

- Univalence implies function extensionality
- Consistent to assume (Voevodsky’s model in sSET)
- Consequently, for any proposition $P : U \rightarrow U$ and witness $A \simeq B$, one cannot show $P(A)$ but not $P(B)$.

**METATHEORY**

- 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

\[ \text{Magma} :\equiv \sum (A : \text{Set}), A \to (A \to A) \]

**Audience Participation**

Given \((A, \ast), (B, \bullet) : \text{Magma}\), what does equality mean?

\[ (f : A \simeq B, \ t : \prod (a, b : A), f(a \ast b) =_B (fa) \bullet (fb)) \]

\[ (f : A \simeq B, \ s : (f \ast =_{B \to (B \to B)} \bullet (f \times f))) \]

\[ (q : A =_U B, \ r : \text{transport}(q, \ast) =_{B \to (B \to B)} \bullet) \]

\[ p : (A, \ast) =_{\text{Magma}} (B, \bullet) \]
This work is licensed under a Creative Commons “Attribution-ShareAlike 4.0 International” license.