

On commutativity, total orders, and sorting

Wind Wong¹ Vikraman Choudhury² Simon J. Gay¹

¹University of Glasgow

²Università di Bologna and OLAS Team, INRIA

February 14, 2024

Motivation

- ▶ The goal is to study free monoids and free commutative monoids.
- ▶ We created a framework to formalize different algebraic structures, free algebras and their universal properties.
- ▶ Univalent type theory gives us higher inductive types, which allows us to reason with commutativity and equations of algebras. (No setoid hell!)
- ▶ Using the framework, we study the relationship between sorting and total orders.

Homotopy Type Theory

Homotopy Type Theory extends intensional MLTT and allows us to reason with equivalences more powerfully.

- ▶ Function extensionality ($\forall x. f(x) = g(x) \rightarrow f = g$)
- ▶ Quotient types (via higher inductive types)
- ▶ Mere propositions
- ▶ Equalities between types (via univalence)

Homotopy Type Theory

Function extensionality ($\forall x. f(x) = g(x) \rightarrow f = g$)

- ▶ MLTT by itself does not have function extensionality
- ▶ It has to be added as an axiom (we lose canonicity!)
- ▶ funExt can be derived as a theorem in HoTT

Homotopy Type Theory

Quotient types

- ▶ In MLTT we can only emulate quotient types with setoids
- ▶ We need to prove functions are setoid homomorphisms when defining a function
- ▶ A lot of proof obligation
- ▶ HoTT lets us define quotient types directly with HITs (no more setoid hell!)

Homotopy Type Theory

Mere propositions

- ▶ In MLTT we don't have a distinction between sets and propositions (both are types)
- ▶ We might end up needing a stronger theorem to prove a proposition
- ▶ E.g. existential proofs are done with Σ -types, requiring us to construct the element
- ▶ HoTT allows us to have types that are "mere propositions"
- ▶ E.g. existential proofs can be done with propositionally truncated Σ -types (mere existence)
- ▶ We can use mere existential proofs to prove other propositions, even if we don't have the specific element

Homotopy Type Theory

Equalities between types

- ▶ In MLTT we don't have equalities between types
- ▶ HoTT gives us equalities between types by the univalence axiom
- ▶ E.g. given $A, B : \mathcal{U}, P : \mathcal{U} \rightarrow \mathcal{U}, A = B$, we can get $P(B)$ from $P(A)$ by substitution

Definition

Given types A and B , A is equivalent to B ($A \simeq B$) if there exists an equivalence $A \rightarrow B$. A function f is said to be an equivalence if $\left(\sum_{g:B \rightarrow A} (f \circ g \sim \text{id}_B) \right) \times \left(\sum_{g:B \rightarrow A} (g \circ f \sim \text{id}_A) \right)$.

Univalence axiom

$$(A = B) \simeq (A \simeq B)$$

Homotopy Levels

$$\text{isContr}(A) := \sum_{(a:A)} \prod_{(x:A)} (a = x).$$

$$\text{isProp}(A) := \prod_{(x,y:A)} (x = y).$$

$$\text{isSet}(A) := \prod_{(x,y:A)} \text{isProp}(x = y).$$

$$\text{isGroupoid}(A) := \prod_{(x,y:A)} \text{isSet}(x = y).$$

$$\begin{aligned} \text{is2Groupoid}(A) &:= \prod_{(x,y:A)} \text{isGroupoid}(x = y). \\ &\vdots \end{aligned}$$

Homotopy Levels

H-Level 0: Contractible types

$$\text{isContr}(A) := \sum_{(a:A)} \prod_{(x:A)} (a = x).$$

There exists a center of contraction such that all elements of A equals to the center of contraction.

Examples:

- ▶ **1** (Unit type)

All contractible types are equivalent to **1**!

Homotopy Levels

H-Level 1: Mere propositions

$$\text{isProp}(A) := \prod_{(x,y:A)} (x = y).$$

All elements of A are equal to each other.

Examples:

- ▶ **1** (Unit type)
- ▶ **0** (Void type)

All h-propositions are equivalent to either **1** or **0**.

A type that represents a proposition should be an `hProp`.

E.g. mere existential proofs can be done with a truncated Σ -type.
(Σ -type wrapped in a HIT that makes all its elements equal.)

Homotopy Levels

H-Level 2: Sets

$$\text{isSet}(A) := \prod_{(x,y:A)} \text{isProp}(x = y).$$

The identity type of A is a proposition. All elements of $x = x$ are equal.

Examples:

- ▶ $\mathbf{1}, \mathbf{0}$
- ▶ \mathbb{N}
- ▶ $\text{hProp}(\sum_{A:\mathcal{U}} \text{isProp}(A))$

For the scope of the project, this is primarily what we are working with!

Homotopy Levels

H-Level 3: Groupoids

$$\text{isGroupoid}(A) := \prod_{(x,y:A)} \text{isSet}(x = y).$$

The identity type of A is a set.

Examples:

- ▶ Any type that satisfies $\text{isSet} \dots$
- ▶ $\text{hSet} (\sum_{A:\mathcal{U}} \text{isSet}(A))$

Consider $\mathbb{B} : \text{hSet}$, univalence gives us two $\mathbb{B} = \mathbb{B}$ generated by id and not .

These two equalities are not equal!

Homotopy Levels

$$\text{isContr}(A) := \sum_{(a:A)} \prod_{(x:A)} (a = x).$$

$$\text{isProp}(A) := \prod_{(x,y:A)} (x = y).$$

$$\text{isSet}(A) := \prod_{(x,y:A)} \text{isProp}(x = y).$$

$$\text{isGroupoid}(A) := \prod_{(x,y:A)} \text{isSet}(x = y).$$

$$\begin{aligned} \text{is2Groupoid}(A) &:= \prod_{(x,y:A)} \text{isGroupoid}(x = y). \\ &\vdots \end{aligned}$$

Higher Inductive Types

```
data List (A : Type) : Type where
  [] : List A
  _::_ : A → List A → List A
```

-- Swap list as HIT

```
data FMSet (A : Type) : Type where
  [] : FMSet A
  _::_ : (x : A) → (xs : FMSet A) → FMSet A
  comm : ∀ x y xs → x :: y :: xs ≡ y :: x :: xs
  trunc : isSet (FMSet A)
```

Higher Inductive Types

-- Swap list as HIT

data **FMSet** (A : Type) : Type **where**

[] : **FMSet** A

:: : (x : A) → (xs : **FMSet** A) → **FMSet** A

comm : $\forall x\ y\ xs \rightarrow x :: y :: xs \equiv y :: x :: xs$

trunc : **isSet** (**FMSet** A)

-- alternatively

-- trunc : (x y : FMSet A) → (p q : x ≡ y) → p ≡ q

+ : $\forall (xs\ ys : \text{FMSet } A) \rightarrow \text{FMSet } A$

[] ++ ys = ys

(x :: xs) ++ ys = x :: xs ++ ys

comm x y xs i ++ ys =

-- proof x :: y :: (xs ++ ys) ≡ y :: x :: (xs ++ ys)

trunc xs zs p q i j ++ ys =

-- proof cong (_++ ys) p ≡ cong (_++ ys) q

Higher Inductive Types

-- Set quotient

data $_/_$ (A : Type) (R : A \rightarrow A \rightarrow Type) : Type **where**

[_] : (a : A) \rightarrow A / R

eq/ : (a b : A) \rightarrow (r : R a b) \rightarrow [a] \equiv [b]

squash/ : (x y : A / R) \rightarrow (p q : x \equiv y) \rightarrow p \equiv q

data Perm {A : Type} : List A \rightarrow List A \rightarrow Type **where**

perm-refl : \forall {xs} \rightarrow Perm xs xs

perm-swap : \forall {x y xs ys zs}

\rightarrow Perm (xs ++ x :: y :: ys) zs

\rightarrow Perm (xs ++ y :: x :: ys) zs

-- Swap list as quotient

FMSet : Type \rightarrow Type

FMSet A = List A / Perm

Cubical Type Theory

- ▶ The project is done in Cubical Agda, an implementation of Cubical Type Theory
- ▶ Cubical Type Theory is a variant of HoTT that preserves computational content for proofs
- ▶ Univalence is not postulated and can be computationally derived
- ▶ Axioms are designed to preserve canonicity, using univalence won't destroy computational content of a proof