

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

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)

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

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

Higher Inductive Types

$$\text{isContr}(A) := \sum_{(a:A)} \prod_{(x:A)} (a = x). \quad (\text{e.g. } \mathbf{1})$$

$$\text{isProp}(A) := \prod_{(x,y:A)} (x = y). \quad (\text{e.g. } \mathbf{1}, \mathbf{0})$$

$$\text{isSet}(A) := \prod_{(x,y:A)} \text{isProp}(x = y). \quad (\text{e.g. } \mathbf{1}, \mathbf{0}, \mathbb{N}, \text{hProp})$$

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

$$\text{is2Groupoid}(A) := \prod_{(x,y:A)} \text{isGroupoid}(x = y). \quad (\text{e.g. } \text{hGroupoid})$$

\vdots

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)

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

[] **++** ys = ys

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

comm x y xs i **++** ys =

-- proof $x :: y :: (xs ++ ys) \equiv y :: x :: (xs ++ ys)$

trunc xs zs p q i j **++** ys =

-- proof $\text{cong } (_++ ys) p \equiv \text{cong } (_++ ys) q$

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