On commutativity, total orders, and sorting

Wind Wong [1] Vikraman Choudhury [2] Simon J. Gay [1]

[1]University of Glasgow

[2]Università 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
- 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 $A \to B$. A function f is said to be an equivalence if $\left(\sum_{g:B\to A}(f \circ g \sim \mathrm{id}_B)\right) \times \left(\sum_{g:B\to A}(g \circ f \sim \mathrm{id}_A)\right)$.

Univalence axiom

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

Higher Inductive Types

$$\mathrm{isContr}(A) :\equiv \sum_{(a:A)} \prod_{(x:A)} (a = x). \qquad \text{(e.g. } 1\text{)}$$

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

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

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

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

$$\vdots$$

Higher Inductive Types

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

_++_ : ∀ (xs ys : FMSet A) → FMSet A
[] ++ ys = ys
(x :: xs) ++ ys = x :: xs ++ ys
comm x y xs i ++ ys =
  (proof for x :: y :: (xs ++ ys) ≡ y :: x :: (xs ++ ys))
trunc xs zs p q i j ++ ys =
  (proof for cong (_++ ys) p ≡ cong (_++ ys) q)
```