# Something about commutativity and well-orders

Anonymous

January 19, 2024

In this talk, we study free monoids, free commutative monoids, and their connections with sorting and well-orders, using univalent type theory, implemented in Cubical Agda.

In the first part of the talk, we present a general framework for doing universal algebra, and the definition of free algebras and their universal property. A signature $\sigma$ is given by a type of operations $\mathsf{op}\colon \mathcal{U}$ with an arity function $\mathsf{ar}\colon \mathsf{op} \to \mathcal{U}$. This gives a signature endofunctor $\Sigma_\sigma(X) :\equiv \sum_{f:\mathsf{op}} \mathsf{ar}(f) \to X$ on $\mathcal{U}$. A $\sigma$-structure is a $\Sigma_\sigma$-algebra, and a morphism of $\sigma$-structures is a $\Sigma_\sigma$-algebra morphism, which produces the category of $\sigma$-algebras $\sigma\text{−}\mathsf{Alg}$.

There is a forgetful functor from $\sigma\text{−}\mathsf{Alg}$ to $\mathsf{hSet}$, which admits a left adjoint, giving the free $\sigma$-algebra construction on a carrier $\mathsf{hSet}$ $V$. As is standard, this construction is given by a W-type $\mathsf{Tree}_\sigma(V)$ where the leaves are... and the sups are..., or an inductive type generated by two constructurs, $\mathsf{leaf}\colon V \to \mathsf{Tree}_\sigma(V)$ and $\mathsf{node}\colon \Sigma_\sigma(\mathsf{Tree}_\sigma(V)) \to \mathsf{Tree}_\sigma(V)$. $\mathsf{Tree}_\sigma(V)$ is canonically a $\sigma$-algebra $\mathfrak{T}(V) = (\mathsf{Tree}_\sigma(V), \mathsf{node})$, and the universal map $\eta_V : V \to \mathsf{Tree}_\sigma(V)$ is given by $\mathsf{leaf}$. The universal property states that, given any $\sigma$-structure $\mathfrak{X}$, there is a canonical equivalence between $\sigma\text{−}\mathsf{Alg}(\mathfrak{T}(V), \mathfrak{X})$ and maps $V \to X$, given by post-composition with $\eta_V$, that is, $\sigma\text{−}\mathsf{Alg}(\mathfrak{T}(V), \mathfrak{X}) \xrightarrow[\sim]{(-)\circ\eta_V} (V \to X)$. The inverse to this map is the extension operation $(-)^\sharp$, which extends a map $f\colon V \to X$ to a homomorphism $f^\sharp\colon \mathfrak{T}(V) \to \mathfrak{X}$.

A signature of equations $\epsilon$ over $\sigma$ is given by a type of equations $\mathsf{eq}\colon \mathcal{U}$ with an arity of free variables $\mathsf{fv}\colon \mathsf{eq} \to \mathcal{U}$. A system of equations (or a theory $T$) over $(\sigma, \epsilon)$ is given by $\prod_{e:\mathsf{eq}} \mathsf{Tree}_\sigma(\mathsf{fv}(e)) \times \mathsf{Tree}_\sigma(\mathsf{fv}(e))$, that is, a pair of trees for each equation. A $\sigma$-structure $\mathfrak{X}$ entails $T$, $\mathfrak{X} \vDash T$, if, for each $e\colon \mathsf{eq}$ and $\rho\colon \mathsf{fv}(e) \to X$, $\rho^\sharp(\pi_1(T(e))) = \rho^\sharp(\pi_2(T(e)))$.

We present a general framework for doing universal algebra and the construction of free algebra in Cubical Agda. We first define the signature $\sigma$ of an algebra by $op : \mathcal{U}$ and $arity : op \to \mathcal{U}$, where $op$ is the set of operation symbols and $arity$ determines the arity of the operation by the cardinality of the type it maps the operation to. By allowing the arity of the operation to be the cardinality of an arbitary type we can express infinitary operations within the framework. We define $sig\ \sigma\ X : \Sigma(s : \sigma\ .op).\ (\sigma\ .arity\ s \to X)$ to represent the symbol of an operation and a lookup function for the arguments of the operation. We then define a $\sigma$-structure to be $car : \mathcal{U}$ and $alg : sig\ \sigma\ car \to car$, where $car$ is the carrier of the algebra and $alg$ evaluates the algebra given an operation and its arguments. We also define an ADT for $\sigma$-expression trees on carrier $X$, $Tree\ \sigma\ X$, with the constructors $leaf : X \to Tree\ X$ and $node : sig\ \sigma\ (Tree\ X) \to Tree\ X$, which we will use to represent the left hand side and right hand side of equations. We define the signature of equations by $name : \mathcal{U}$ and $free : name \to \mathcal{U}$, where $name$ is a set of names used to identity the axiom equation of an algebra, and $free$ determines the number of free variables in the equation by the cardinality of the type it maps the name to, again allowing for infinitary equations. Finally, given a equation signature $\tau$ for a $\sigma$-algebra, we define a system of equations as $(e : \tau\ .name) \to Tree\ (\tau\ .free\ e) \times Tree\ (\tau\ .free\ e)$, using $Tree$ to express the right hand side and left hand side of equations. We can show that a construction satisfies an algebraic structure by showing that an evaluated expression tree would indeed satisfy the system of equations, for example, $(\mathbb{N}, 0, +)$ satisfies the unit and associativity laws of a monoid. We can then formalize the notion of homomorphisms on $\sigma$-algebras by defining it as functions which satisfies $\sigma(h(x), h(y), h(z), ...) = h(\sigma(x, y, z, ...))$ for a homomorphism $h$. Finally, we formalize the definition of a free algebra by its universal property, namely: for a free algebra $F$ on $A$ and an algebra $\mathfrak{B}$, a set function $g : A \to \mathfrak{B}$, there exists a unique homomorphism $f : F(A) \to \mathfrak{B}$ such that $g = f \circ i$, where $i : A \to F(A)$ is the canonical injection morphism. We do so by requiring a $\sigma$-structure to satisfy a property for any $\sigma$-structure $\mathfrak{Y}$ $isFree : isEquiv\ (\lambda f.\ f \circ i)$ where $f$ is a homomorphism to $\mathfrak{Y}$ and $i$ is the canonical injection. We then define the $\# : (A \to \mathfrak{B}) \to (F(A) \to \mathfrak{B})$ operation, which lifts a set function $A \to \mathfrak{B}$ to a homomorphism $F(A) \to \mathfrak{B}$ using the universal property.

Using the framework we present different constructions of free monoids, such as list $List$, free monoid as a HIT $FreeMon$, and as an index function $\Sigma(n : \mathbb{N}).Fin\ n \to X$ which we call $Array$. We then extend our work by constructing free commutative monoids, such as swapped-list $SList$ and free monoid quotiented by a permutation relation $QFreeMon$, for example $List$ quotiented by permutation ($PList$) and $Array$ quotiented by isomorphism on index $\Sigma(\sigma : Fin\ n \simeq Fin\ m).\ v = w \circ \sigma$ ($Bag$). We prove these constructions are indeed free algebras by proving their universal property directly, and we can derive canonical maps between different constructions directly using the universal property.

Using the same framework, we can axiomatize the notion of a sort function.

**Definition 1.** *Given a section $s : SList\ X \to List\ X$ to the canonical map $List\ X \to SList\ X$, $xs$ is said to be sorted if $xs$ is in the image of $s$.*

We define the proposition *is-sorted* : $List\ X \to \mathcal{U}$ to be $\lambda xs.\ \exists(ys : SList\ X).\ s(ys) = xs$. We also use the universal property of free monoid to define membership proofs for $List\ X$ and $SList\ X$. We do so by noting propositions form a commutative monoid under $\vee$, which allow us to define membership proof for an element $x$ using the extension operation $(-)^\sharp$ by lifting the function $\lambda y.\ x = y$ from $X \to Prop$ to $List\ X \to Prop$ and $SList\ X \to Prop$ respectively.

**Proposition 1.** *A section $s : SList X \to List X$ to the canonical map from the free monoid to the free commutative monoid on set $X$ implies a total order on set $X$ iff $\forall x\ y\ xs.$ is-sorted$(x :: xs) \to y \in x :: xs \to$ is-sorted$([x, y])$.*

It is well known that a total order on set $X$ would imply a sort function on $List\ X$. It should follow that a sort function on set $X$ would imply a total order on $X$. We can formalize the notion of a sort function as a section to the canonical map from *List* to *SList*, which can be thought of as a function which picks a canonical representation from an unordered list, thereby sorting the list in the process. However, we cannot prove transitivity purely from $s$ being a section, one example being a function $s : SList\ \mathbb{N} \to List\ \mathbb{N}$ which sorts ascendingly given an odd-lengthed *SList* and descendingly given an even-lengthed *SList*. Hence, a stronger assumption is needed to fully construct a total order.

**Definition 2.** *Given a section $s : SList\ X \to List\ X$ to the canonical map $List\ X \to SList\ X$, we define a relation $\leq$: if $x$ is the head of $s(\{x, y\})$, $x \leq y$.*

We note that $x \leq y$ iff *is-sorted*$([x, y])$.

**Lemma 1.** *Given a section $s : SList\ X \to List\ X$ to the canonical map $List\ X \to SList\ X$, $s(\{x, y\})$ must either be $[x, y]$ or $[y, x]$.*

We note that the canonical map $q : List\ X \to SList\ X$ preserves length and preserves membership, $x \in xs$ iff $x \in q(xs)$. Since $q(s(\{x, y\})) = \{x, y\}$ by definition, $s(\{x, y\})$ must therefore have length 2, and $x, y \in s(\{x, y\})$. Let $q(\{x, y\})$ be $[u, v]$, we perform a proof by cases. For case $x = u$, $y = v$ and $x = v$, $y = u$ the proof is trivial. For case $x = u$, $y = u$ and $x = v$, $y = v$, we obtain a proof $x = y$. Since $s(\{x, x\}) = [u, v]$ by assumption, and $q([u, v]) = \{x, x\}$ by definition of $s$, $u, v \in \{x, x\}$, therefore $u, v$ must equal to $x$. Since $u = v = x = y$, and $s(\{x, y\}) = [u, v]$, $s(\{x, y\}) = [x, y]$.

With this lemma we can then prove $\leq$ does indeed satisfy all axioms of total order.

**Proposition 2.** $\leq$ *is reflexive.*

By Lemma 1, $s(\{x, x\})$ must either be $[x, x]$ or $[x, x]$. Either case we obtain a proof that $s(\{x, x\}) = [x, x]$. Since $x$ is the head of $[x, x]$, $x \leq x$.

**Proposition 3.** $\leq$ *is antisymmetric.*

Given $x \leq y$ and $y \leq x$, we want to show $x = y$. By Lemma 1, $s(\{x, y\})$ must either be $[x, y]$ or $[y, x]$. In the case $s(\{x, y\}) = [x, y]$, since $y \leq x$, $y$ is the head of $[x, y]$, and we obtain a proof $y = x$. In the case $s(\{x, y\}) = [y, x]$, we invert $x$ and $y$ in the previous proof and obtain $x = y$. Either case we obtain $x = y$.

**Proposition 4.** $\leq$ *is total.*

We want to show for any $x$ and $y$, either $x \leq y$ or $y \leq x$. By Lemma 1, $s(\{x, y\})$ must either be $[x, y]$ or $[y, x]$, therefore either $x$ or $y$ is the head of $s(\{x, y\})$. We obtain either $x \leq y$ or $y \leq x$.

**Proposition 5.** $\leq$ *is transitive.*

Given $x \leq y$ and $y \leq z$, we want to show $x \leq z$. We first note that the head of $s(\{x, y, z\})$ must either be $x$, $y$, or $z$. For case $x$, by assumption $[x, z]$ is sorted, therefore $x \leq z$. For case $y$, by assumption $[y, x]$ is sorted, therefore $y \leq x$, and since $x \leq y$ by assumption, by antisymmetry $x = y$, and by assumption $y \leq z$, therefore $x \leq z$. For case $z$, by assumption $[z, y]$ is sorted, therefore $z \leq y$, and since $y \leq z$ by assumption, by antisymmetry $y = z$, and by assumption $x \leq y$, therefore $x \leq z$.

If we assume $X$ to have decidable equality, we can construct a linear order from the total order. With more constraint it should be possible to prove that the constructed linear order would be a well order, and therefore imply $X$ is a choice set. We can then show that by assuming every set $X$ is decidable and has a section to the canonical map from the free monoid to the free commutative monoid, we would be able to derive the axiom of choice.

**Conjecture 1.** *SList and Bag are free symmetric monoidal groupoid*

Previous works already established *SList* and *Bag* are free commutative monoids when 0-truncated, and it is folklore that when 1-truncated they should be free symmetric monoidal groupoid, although this has not yet been shown internally in HoTT.

In the case of *SList*, higher path constructors are needed. To establish the coherence of *swap* we need a higher *hexagon* path constructor. However to define the *hexagon* constructor it would involve compositions of *swap* which leads to a regularity problem. To avoid this we need to split the *hexagon* equations as below:

```
hexagon- : (a b c : A) (cs : SList A)
         -> a :: b :: c :: cs = c :: b :: a :: cs
hexagon↑ : (a b c : A) (cs : SList A)
         -> Square (\i -> b :: swap a c cs i) (hexagon- a b c cs)
                   (swap b a (c :: cs)) (swap b c (a :: cs))
hexagon↓ : (a b c : A) (cs : SList A)
         -> Square (hexagon- a b c cs) (swap a c (b :: cs))
                   (\i -> a :: swap b c cs i) (\i -> c :: swap b a cs i)
```

In the case of *Bag*, we first need to show that *Array* itself is a free monoidal groupoid. We then need to show a free monoidal groupoid quotiented by a permutation relation would be a free symmetric monoidal groupoid, and show that by quotienting *Array* with an isomorphism on the index, we would indeed get a free symmetric monoidal groupoid.