On commutativity, total orders, and sorting

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

April 4, 2024

[1] University of Glasgow

[2] Università di Bologna and OLAS Team, INRIA

[a] supported by Marie Skłodowska-Curie award 101106046

Outline

Introduction

Background: Universal Algebra

Free (commutative) monoids

Sorting

Introduction

Consider a puzzle about sorting, inspired by Dijkstra's Dutch National Flag problem. Suppose there are balls of three colors, corresponding to the colors of the Dutch flag: red, white, and blue.

Given an unordered list of such balls, how many ways can you sort them into the Dutch flag?

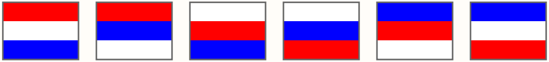
Obviously there is only one way, which is given by the order red $<$ white $<$ blue.

Introduction

What if we are avid enjoyers of vexillology who also want to consider other flags?

We might ask: how many ways can we sort our bag of balls?

We know that there are only $3! = 6$ permutations of $\{\text{red}, \text{white}, \text{blue}\}$, so there are only 6 possible orderings we can define. [1]



We claim because there are exactly 6 orderings, we can only define 6 *correct* sorting functions.

[1] I have no allegiance to any of the countries presented by the flags, hypothetical or otherwise – this is purely combinatorics!

Motivation

Sort functions are subset of functions from unordered lists to lists:

1. Formalize what $\mathsf{UnorderedList}(A)$ and $\mathsf{List}(A)$ are.
2. Nail down what the subset $\mathsf{Sort}(A)$ is.
3. Construct a full equivalence $\mathsf{Sort}(A) \simeq \mathsf{Ord}(A)$.

- We use category theory and type theory.
- Categorical language is used to describe the universal properties of free algebras.
- Type theory is used to construct free algebras.

The formalization is done in Cubical Agda.

Outline

Introduction

Background: Universal Algebra

Free (commutative) monoids

Sorting

A signature σ is:

- a set of function symbols op : hSet
- an arity function $ar : op \to$ hSet

This gives a signature endofunctor $F_\sigma(X) := \sum_{f:op} X^{ar(f)}$

A σ-structure is a F_σ-algebra:

- a carrier set X : *hSet*
- an interpretation function: $\alpha_X : F_\sigma(X) \to X$

A σ-algebra homomorphism $h : X \to Y$ is a function such that:

$$\begin{array}{ccc} F_\sigma(X) & \xrightarrow{\alpha_X} & X \\ {\scriptstyle F_\sigma(h)}\downarrow & & \downarrow{\scriptstyle h} \\ F_\sigma(Y) & \xrightarrow[\alpha_Y]{} & Y \end{array}$$

F_σ-algebras and their morphisms form a category σ-Alg.

Example: The signature σ_{Mon} has: op : hSet $= \{e, \bullet\}$ (or Fin_2 or $\mathbf{2}$), $ar : \sigma \to \text{hSet} = \{e \mapsto \mathbf{0}, , \bullet \mapsto \mathbf{2}\}$

Free Algebras

The free σ-algebra $\mathfrak{F}(X)$ on a carrier set X, if it exists, produces a left adjoint to the forgetful functor σ-Alg to hSet, given by:

- a type constructor F : hSet $\to$ hSet,
- a universal generators map $\eta_X : X \to F(X)$, such that
- for any σ-algebra $\mathfrak{Y}$, post-composition with η_X is an equivalence.

$$(\mathfrak{F}(X) \xrightarrow{f} \mathfrak{Y}) \quad \mapsto \quad (X \xrightarrow{\eta V} F(X) \xrightarrow{f} Y)$$

- The inverse of the equivalence is the extension operation $(\_)^\sharp : (X \to Y) \to (\mathfrak{F}(X) \to \mathfrak{Y})$.

We define the carrier set using an inductive type of trees $Tr_\sigma(V)$, generated by two constructors:

- leaf : $V \to Tr_\sigma(V)$, and
- node : $F_\sigma(Tr_\sigma(V)) \to Tr_\sigma(V)$.

Expanding node: $(f : op) \times (\mathrm{ch} : ar(f) \to Tr_\sigma(V)) \to Tr_\sigma(V)$.

node is our algebra map $\alpha : F_\sigma(Tr_\sigma(V)) \to Tr_\sigma(V)$.

leaf is our generators map $\eta : V \to Tr_\sigma(V)$.

This gives a σ-algebra $\mathfrak{T}(V) = (\mathrm{Tr}_\sigma(V), \mathrm{node})$.

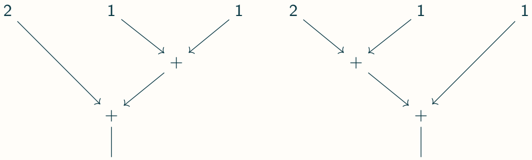
Theorem

$\mathfrak{T}(V)$ is the free σ-algebra on V.

$Tr_\sigma(V)$ can be represented by the W-type:

- the shape $S : \mathcal{U}$ given by $V + op_\sigma$,
- the family of positions $P : S \to \mathcal{U}$ given by $\{inl(v) \mapsto \bot, inr(v) \mapsto ar_\sigma\}$.

Trees for σ_{Mon} with the carrier set $\mathbb{N}$ would look like:



These trees should be equivalent by associativity since they are trees of a monoid...

Universal Algebra

So far there are no laws! How do we add laws?

Definition

An equational signature ε is given by:

- a set of equation symbols eq : hSet,
- an arity of free variables $fv : eq \to \mathrm{hSet}$

A system of equations (or equational theory T_ε) is a pair of natural transformations: $l, r : F_\varepsilon \Rightarrow \mathrm{Tr}_\sigma$.

$\mathfrak{X}$ satisfies T ($\mathfrak{X} \vDash T$) if for every assignment $\rho : V \to X$, $\rho^\sharp$ coequalizes l_V, r_V:

$$F_\varepsilon(V) \underset{r_V}{\overset{l_V}{\rightrightarrows}} \mathrm{Tr}_\sigma(V) \xrightarrow{\rho^\sharp} \mathfrak{X}$$

Universal Algebra

Example: $\mathbb{N}$ is a (lawful) monoid.

The equational signature σ_{Mon} has:

- the set of equation symbols $eq = \{\mathsf{unitl}, \mathsf{unitr}, \mathsf{assocr}\}$ (or Fin_3 or $\mathbf{3}$),
- the arity function $fv : eq \to \mathsf{hSet} = \{\mathsf{unitl} \mapsto \mathbf{1}, \mathsf{unitr} \mapsto \mathbf{1}, \mathsf{assocr} \mapsto \mathbf{3}\}$.

To show $(\mathbb{N}, 0, +) \vDash \mathsf{Mon}$:

$$\begin{aligned} \mathit{unitl} &: \forall(\rho : \mathbb{N}^{\mathsf{Fin}_1}).\, \rho(0) + 0 = \rho(0) \\ \mathit{unitr} &: \forall(\rho : \mathbb{N}^{\mathsf{Fin}_1}).\, 0 + \rho(0) = \rho(0) \\ \mathit{assocr} &: \forall(\rho : \mathbb{N}^{\mathsf{Fin}_3}).\, (\rho(0) + \rho(1)) + \rho(2) = \rho(0) + (\rho(1) + \rho(2)) \end{aligned}$$

The σ-algebras satisfying a theory T_ε form a subcategory (σ, ε)-Alg (or a variety of algebras).

Definition

A (σ, ε)-algebra $\mathfrak{F}(V)$ is free if post-composition with η_X is an equivalence: $(-) \circ \eta_X : (\sigma, \varepsilon)\text{-}\mathsf{Alg}(\mathfrak{F}(V), \mathfrak{X}) \xrightarrow{\sim} (V \to X)$.

Universal Algebra

In this talk, we only consider the construction of free objects for the special case of monoids and commutative monoids.

But can we construct any arbitrary free algebras?

We need choice to handle infinitary operations[2], and also avoid strict positivity checking.

We didn't investigate further, but it should be possible to construct arbitrary free algebras in Cubical Agda.

[2] Blass, "Words, free algebras, and coequalizers".

Outline

Introduction

Background: Universal Algebra

Free (commutative) monoids

Sorting

Constructions of free (commutative) monoids

It's well known that Lists are free monoids[3]:

We can turn it into a free commutative monoid by either adding a path constructor[4] or by set quotients[5]:

Swapped cons lists

```
data SList (A : 𝒰) : 𝒰 where
  [] : SList A
  _::_ : A → SList A → SList A
  swap : ∀ x y xs → x :: y :: xs ≡ y :: x :: xs
  trunc : ∀ x y → (p q : x ≡ y) → p ≡ q
```

Cons lists upto permutation

$$\mathrm{PList}(A) = \mathrm{List}(A)/\mathrm{Perm}_{\approx}$$

[3] Dubuc, "Free monoids"; Kelly, "A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on".

[4] Choudhury and Fiore, "Free Commutative Monoids in Homotopy Type Theory".

[5] Joram and Veltri, "Constructive Final Semantics of Finite Bags".

Constructions of free (commutative) monoids

Another construction of free monoids is Array:

Array

$$\text{Array}(A) = (n : \mathbb{N}) \times (f : \text{Fin}_n \to A)$$

We can also turn it into a free commutative monoid by quotienting with symmetries[6]:

Bags

$$\text{Bag}(A) = \mathit{Array}(A) / \approx$$
$$(n, f) \approx (m, g) = \exists(\phi : \text{Fin}_n \xrightarrow{\sim} \text{Fin}_m).\ f = g \circ \phi$$

[6] Joram and Veltri, "Constructive Final Semantics of Finite Bags".

Constructions of free commutative monoids

Bags

$$\mathsf{Bag}(A) = \mathit{Array}(A)/ \approx$$

$$(n, f) \approx (m, g) = \exists(\phi : \mathsf{Fin_n} \xrightarrow{\sim} \mathsf{Fin_m}).\ f = g \circ \phi$$

Cons lists quotiented by permutations

$\mathsf{PList}(A) = \mathsf{List}(A)/\mathsf{Perm}_{\approx}$

A free monoid quotiented by a permutation relation must be a free commutative monoid.

From this, a relation $\approx$ is a correct permutation relation iff it:

- is reflexive, symmetric, transitive (equivalence),
- is a congruence wrt $\bullet$: $a \approx b \to c \approx d \to a \bullet c \approx b \bullet d$,
- is commutative: $a \bullet b \approx b \bullet a$, and
- respects $(-)^\sharp$: $\forall f.\, a \approx b \to f^\sharp(a) = f^\sharp(b)$.

Bags

$$\mathrm{Bag}(A) = \mathit{Array}(A)/\approx$$

$$(n, f) \approx (m, g) = \exists(\phi \colon \mathrm{Fin}_{n} \xrightarrow{\sim} \mathrm{Fin}_{m}).\ f = g \circ \phi$$

How to show $\approx$ respects commutativity: $a \bullet b \approx b \bullet a$?

Let $a = (n, f)$ and $b = (m, g)$, we need to compute an isomorphism $\phi \colon \mathrm{Fin}_{n+m} \xrightarrow{\sim} \mathrm{Fin}_{m+n}$, such that: $(f \oplus g) = (g \oplus f) \circ \phi$. Define,

$$\phi := \mathrm{Fin}_{n+m} \xrightarrow{\sim} \mathrm{Fin}_{n} + \mathrm{Fin}_{m} \xrightarrow{\mathrm{swap}_{+}} \mathrm{Fin}_{m} + \mathrm{Fin}_{n} \xrightarrow{\sim} \mathrm{Fin}_{m+n}$$

$$\{0, 1, \ldots, n-1, n, n+1, \ldots, n+m-1\}$$
$$\big\downarrow \phi$$
$$\{n, n+1 \ldots, n+m-1, 0, 1, \ldots, n-1\}$$

Bags

$$\mathrm{Bag}(A) = \mathit{Array}(A) / \approx$$
$$(n, f) \approx (m, g) = \exists(\phi : \mathrm{Fin}_n \xrightarrow{\sim} \mathrm{Fin}_m).\ f = g \circ \phi$$

How to show $\approx$ respects $(-)^\sharp$: $\forall f.\ a \approx b \to f^\sharp(a) = f^\sharp(b)$?

We can prove this by showing $f^\sharp$ is invariant under permutation: for all $\phi : \mathrm{Fin}_n \xrightarrow{\sim} \mathrm{Fin}_n$, $f^\sharp(n, i) = f^\sharp(n, i \circ \phi)$.

Constructions of free commutative monoids

W.T.S. for all $\phi\colon \mathrm{Fin}_n \xrightarrow{\sim} \mathrm{Fin}_n$ $f^\sharp(n, i) = f^\sharp(n, i \circ \phi)$.

- The image of $f^\sharp$ is a commutative monoid, so permuting the array's elements should not affect anything
- But how do we actually prove this?
- If $\phi(0) = 0$, we can prove this by induction:

Theorem

Given $\tau\colon \mathrm{Fin}_{S(n)} \xrightarrow{\sim} \mathrm{Fin}_{S(n)}$ where $\tau(0) = 0$, there is a $\psi : \mathrm{Fin}_n \xrightarrow{\sim} \mathrm{Fin}_n$ such that $\tau \circ S = S \circ \psi$.

$$\begin{array}{c} \{0, 1, 2, 3, \ldots\} \\ \big\downarrow \tau \\ \{0, x, y, z \ldots\} \end{array} \qquad \begin{array}{c} \{0, 1, 2, \ldots\} \\ \big\downarrow \psi \\ \{x - 1, y - 1, z - 1 \ldots\} \end{array}$$

This is a special case of `punchIn` and `punchOut`, where $k = 0$.

W.T.S. for all $\phi\colon \mathrm{Fin}_{\mathrm{n}} \xrightarrow{\sim} \mathrm{Fin}_{\mathrm{n}}.\ f^\sharp(n, i) = f^\sharp(n, i \circ \phi)$.

Theorem

Given $\phi\colon \mathrm{Fin}_{\mathrm{S(n)}} \xrightarrow{\sim} \mathrm{Fin}_{\mathrm{S(n)}}$, there is a $\tau\colon \mathrm{Fin}_{\mathrm{S(n)}} \xrightarrow{\sim} \mathrm{Fin}_{\mathrm{S(n)}}$ such that $\tau(0) = 0$, and $f^\sharp(S(n), i \circ \phi) = f^\sharp(S(n), i \circ \tau)$.

Let k be $\phi^{-1}(0)$:

$$\begin{array}{c} \{0, 1, 2, \ldots, k, k+1, k+2, \ldots\} \\ \downarrow \phi \\ \{x, y, z, \ldots, 0, u, v, \ldots\} \end{array}$$

$$\begin{array}{c} \{0, 1, 2, \ldots, k, k+1, k+2, \ldots\} \\ \downarrow \tau \\ \{0, u, v, \ldots, x, y, z, \ldots\} \end{array}$$

Arrays quotiented by symmetries

W.T.S. for all $\phi \colon \mathsf{Fin}_n \xrightarrow{\sim} \mathsf{Fin}_n.\ f^\sharp(n, i) = f^\sharp(n, i \circ \phi)$.

Theorem

For all $\phi \colon \mathsf{Fin}_n \xrightarrow{\sim} \mathsf{Fin}_n.\ f^\sharp(n, i) = f^\sharp(n, i \circ \phi)$.

$$
\begin{aligned}
& f^\sharp(S(n), i \circ \phi) \\
& = f^\sharp(S(n), i \circ \tau) \\
& = f(i(0)) \bullet f^\sharp(n, i \circ \psi) \\
& = f(i(0)) \bullet f^\sharp(n, i) && \text{(induction)} \\
& = f^\sharp(S(n), i)
\end{aligned}
$$

Bag satisfies the universal property of free commutative monoids!

Outline

Introduction

Background: Universal Algebra

Free (commutative) monoids

Sorting

Any presentation of free monoids or free commutative monoids has a:

- `length : F(A) → Nat` function, given by $(\lambda x.\, 1)^\sharp$
- a membership predicate: `_∈_ : A → F(A) → hProp`.
  Assuming A is a set, and $x : A$, we define $よ_A(y) = x = y : A \to \text{hProp}$.
  $x \in -$ is given by ${よ_A}^\sharp$!

`List` vs `SList`

Consider the `head : List A → A` function.

Can we define `head` for both `List`s and `SList`s?

We consider by cases on the length of the `List`/`SList`.

- For empty (s)lists, `head` doesn't exist (e.g. consider $A = \mathbf{0}$).
- For singleton (s)lists, `head` is an equivalence (injectivity of η).
- For lists of length ≥ 2, we can just take the first element.
  For slists of length ≥ 2, by swap:

$$\begin{aligned} \{x,y\} &= \{y,x\} \\ \mathtt{head}(\{x,y\}) &= \mathtt{head}(\{y,x\}) \end{aligned}$$

Which one do we pick? Commutativity enforce unorderedness!

Let $\mathcal{L}(A)$ be the free monoid, and $\mathcal{M}(A)$ the free commutative monoid on A.

$$\mathcal{L}(A) \underset{s}{\overset{q}{\rightleftarrows}} \mathcal{M}(A)$$

q is the canonical map (surjection) from $\mathcal{L}(A)$ to $\mathcal{M}(A)$ (given by extending $\eta_A^{\mathcal{M}}$).

Question

Without choice axioms, constructively, does q have a section?

To give a section is to turn an unordered list into an ordered list. How should s order the elements? By sorting! (which requires a total order on A...)

We will show that sorting can be axiomatized from this point of view.

Sorting

Informally, we prove:

1. if A has a decidable total order, there is a well-behaved section.
2. if there is a well-behaved section, A is totally ordered.

This well-behaved section gives a correct sort function!

Axioms of total order:

- reflexivity: $x \leq x$
- transitivity: if $x \leq y$ and $y \leq z$, then $x \leq z$
- antisymmetry: if $x \leq y$ and $y \leq x$, then $x = y$
- totality: forall x and y, we have merely either $x \leq y$ or $y \leq x$

Proposition

Assume there is a decidable total order on A. There is a sort function $s \colon \mathcal{M}(A) \to \mathcal{L}(A)$ which constructs a section to $q \colon \mathcal{L}(A) \twoheadrightarrow \mathcal{M}(A)$.

We can construct a section s by any sorting algorithm, we chose insertion sort.

Sorting

To go the other way, given a section s, we can construct a relation that satisfies reflexivity, antisymmetry, and totality!

Definition

Given a section s, define:

$$\text{least}(xs) \coloneqq \text{head}(s(xs))$$
$$x \preceq y \coloneqq \text{least}(\{x, y\}) = x$$

We prove:

- reflexivity: $x \preceq x$:
  $\text{least}(\{x, x\}))$ must be x.
- antisymmetry: if $x \preceq y$ and $y \preceq x$, then $x = y$:
  $x = \text{least}(\{x, y\}) = y$
- totality: for all x and y, either $x \preceq y$ or $y \preceq x$:
  $\text{least}(\{x, y\})$ is merely either x or y.

But what about transitivity?

Sorting

Consider this section $s : \mathit{SList}(\mathbb{N}) \to \mathit{List}(\mathbb{N})$:

$$
\begin{aligned}
s(xs) &= \begin{cases} \mathrm{sort}(xs) & \text{if } \mathrm{length}(xs) \text{ is odd} \\ \mathrm{reverse}(\mathrm{sort}(xs)) & \text{otherwise} \end{cases} \\
s(\{2,3,1,4\}) &= [4,3,2,1] \\
s(\{2,3,1\}) &= [1,2,3]
\end{aligned}
$$

s doesn't sort and violates transitivity!

A correct sort function needs more constraints ...

Correctness of Sorting

Given a section s:

is-sorted

A list xs is sorted if $\exists ys.\, s(ys) = xs$.

is-head-least

s satisfies *is-head-least* if
$\forall x\, xs.\, \text{is-sorted}(x :: xs) \land y \in (x :: xs) \to \text{is-sorted}([x, y])$.

Lemma

is-head-least is equivalent to transitivity of $\preceq$.

Corollary

If s satisfies *is-head-list*, then $\preceq$ is a total order on A.

Next step: we want to upgrade this proof to an equivalence between total orders on A, and well-behaved sections s.

Given a decidable total order $\leq$, we use it to construct a sort function (e.g. insertion sort). Insertion sort satisfies *is-head-least*, and we use it to construct a total order $\preceq$.

Question

- Is $\preceq = \leq$?
- If we use s to construct $\preceq$, can we reconstruct s from $\preceq$?

As it turns out, *is-head-least* is not enough to axiomatize sorting functions!

Axiomatics of Sorting

If we use s to construct $\preceq$, can we reconstruct s from $\preceq$?

Let sort be insertion sort by $\preceq$. Consider this section $s : \mathit{SList}(\mathbb{N}) \to \mathit{List}(\mathbb{N})$:

$$
\begin{aligned}
s(xs) &= \mathrm{least}(xs) :: \mathrm{reverse}(\mathrm{tail}(\mathrm{sort}(xs))) \\
s(\{2,3,1,4\}) &= [1,4,3,2] \\
s(\{2,3,1\}) &= [1,3,2]
\end{aligned}
$$

s is not the same as insertion sort, but both give us the same $\preceq$!

We need another constraint:

is-tail-sort

A section s satisfies *is-tail-sort* if:
$\forall x\, xs.\ \mathrm{is\text{-}sorted}(x :: xs) \to \mathrm{is\text{-}sorted}(xs)$.

Our final theorem:

Definition

- $\mathsf{DecTotOrd}(A)$ = decidable total orders on A
- $\mathsf{Sort}(A)$ = sections $s\colon \mathcal{M}(A) \to \mathcal{L}(A)$ to q, satisfying is-head-least and is-tail-sort, where A has decidable equality

Theorem

$\mathsf{o2s}\colon \mathsf{DecTotOrd}(A) \to \mathsf{Sort}(A)$ is an equivalence.

There is a decidable total order on A iff A has decidable equality and a section satisfying is-head-least and is-tail-sort!

Main Result

Theorem

o2s: DecTotOrd(A) $\to$ Sort(A) is an equivalence.

Given DecTotOrd(A):

- We can construct a section $s \colon \mathcal{M}(A) \to \mathcal{L}(A)$ with insertion sort, which satisfies is-head-least and is-tail-sort
- We can show A has decidable equality by determining if $x \leq y$ and $y \leq x$, antisymmetry gives us $x = y$ if $x \leq y$ and $y \leq x$

Given Sort(A):

- We can construct a total order $x \preceq y := \mathit{least}(\{x, y\}) = x$ as shown previously
- Because A has decidable equality, we can determine $\mathit{least}(\{x, y\}) = x$, so $\preceq$ is decidable

Main Result

$$\mathrm{DecTotOrd}(A) \xrightarrow{o2s} \mathrm{Sort}(A) \xrightarrow{o2s^{-1}} \mathrm{DecTotOrd}(A)$$

- $\mathrm{least}(\{x, y\}) = x$ iff $x \leq y$

$$\mathrm{Sort}(A) \xrightarrow{o2s^{-1}} \mathrm{DecTotOrd}(A) \xrightarrow{o2s} \mathrm{Sort}(A)$$

- Given a section s that satisfies is-head-least and is-tail-sort, s is equal to insertion sort with the order $\preceq$ generated by s.
- is-head-least lets us create the total order $\preceq$.

Definition

We define a witness for sorted lists:

```
data Sorted (≤ : A → A → 𝓤) : List A → 𝓤 where
  sorted-[] : Sorted []
  sorted-one : ∀ x → Sorted [ x ]
  sorted-:: : ∀ x y zs → x ≤ y → Sorted (y :: zs)
            → Sorted (x :: y :: zs)
```

- is-tail-sort lets us inductively prove $\forall xs.\ \mathrm{Sorted}_{\preceq}(s(xs))$
- Both s and insertion sort produce lists sorted by $\preceq$, and they're the same!

Main Result

Question

What is a correct sorting algorithm?

Answer

A sort function is a section $s \colon \mathcal{M}(A) \to \mathcal{L}(A)$ to the canonical map $q \colon \mathcal{L}(A) \twoheadrightarrow \mathcal{M}(A)$, satisfying:

- *is-head-least*:
  $\forall x\, xs.\, \text{is-sorted}(x :: xs) \wedge y \in (x :: xs) \to \text{is-sorted}([x, y])$,
- *is-tail-sort*: $\forall x\, xs.\, \text{is-sorted}(x :: xs) \to \text{is-sorted}(xs)$.

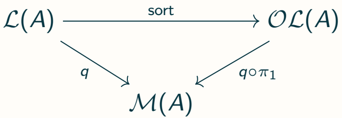
where xs *is-sorted* if it is in the truncated fiber of s.

Remarks

- Other specifications of sorting (in Coq, or the VFA livre) are given in terms of `sort : List Nat → List Nat`.
- These are special cases of our axiomatic understanding of sorting!

As a sanity check for our axioms, we can see how `Sorted` from VFA relates to our axioms.

Let $\mathcal{OL}(A) = \Sigma_{xs:\mathcal{L}(A)} Sorted_{\leq}(xs)$:



We set *sort* to $(s \circ q, p \circ q)$, where p is the proof $\forall xs.\ \mathrm{Sorted}_{\preceq}(s(xs))$

Conclusion

We developed new axiomitizations for sort functions by showing the correspondence between:

- sort functions
- well behaved sections
- decidable total orders

Future works:

- Are all sections defined in terms of well-behaved sections?
  - Does the existence of a section $\mathcal{M}(A) \to \mathcal{L}(A)$ imply a total order on A?
- Generalize the universal algebra framework from sets to groupoids.
  - How to define system of coherences?

Thank you!

References i

Blass, Andreas. **"Words, free algebras, and coequalizers".** en. In: *Fundamenta Mathematicae* 117.2 (1983), pp. 117–160. ISSN: 0016-2736, 1730-6329. DOI: 10.4064/fm-117-2-117-160. URL: http://www.impan.pl/get/doi/10.4064/fm-117-2-117-160 (visited on 02/25/2024).

Choudhury, Vikraman and Marcelo Fiore. **"Free Commutative Monoids in Homotopy Type Theory".** en. In: *Electronic Notes in Theoretical Informatics and Computer Science* Volume 1 - Proceedings of... (Feb. 2023), p. 10492. DOI: 10.46298/entics.10492. URL: https://entics.episciences.org/10492 (visited on 02/27/2023).

Dubuc, Eduardo J. **"Free monoids".** In: *Journal of Algebra* 29.2 (May 1974), pp. 208–228. ISSN: 0021-8693. DOI: 10.1016/0021-8693(74)90095-7. URL: https://www.sciencedirect.com/science/article/pii/0021869374900957 (visited on 02/14/2024).

Joram, Philipp and Niccolò Veltri. **"Constructive Final Semantics of Finite Bags".** en. In: *DROPS-IDN/v2/document/10.4230/LIPIcs.ITP.2023.20.* Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.ITP.2023.20. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2023.20 (visited on 02/14/2024).

Kelly, G. M. **"A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on".** en. In: *Bulletin of the Australian Mathematical Society* 22.1 (Aug. 1980). Publisher: Cambridge University Press, pp. 1–83. ISSN: 1755-1633, 0004-9727. DOI: 10.1017/S0004972700006353. URL: https://www.cambridge.org/core/journals/bulletin-of-the-australian-mathematical-society/article/unified-treatment-of-transfinite-constructions-for-free-algebras-free-monoids-colimits-associated-sheaves-and-so-on/FE2E25E4959E4D8B4DE721718E7F55EE (visited on 02/14/2024).