The finite-multiset construction in HoTT

Vikraman Choudhury [1] Marcelo Fiore [2]

August 12, 2019

[1]Indiana University

[2]University of Cambridge

Outline

Free monoids

Free commutative monoids

Applications

Free symmetric monoidal categories

Free monoids

The forgetful functor from Mon to Set has a left adjoint.

$$\begin{tikzcd}
\mathsf{Mon} \arrow[d] \\
\mathsf{Set} \arrow[u, bend left, "\mathcal{L}"]
\end{tikzcd}$$

Free monoids

The forgetful functor from **Mon** to **Set** has a left adjoint.

$$\begin{tikzcd} \mathsf{Mon} \arrow[d] \\ \mathsf{Set} \arrow[u, bend left, "\mathcal{L}"] \end{tikzcd}$$

$$\mathcal{L}A = A^* = \text{finite strings with elements drawn from } A$$

Universal property

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\eta_A"'] & M(e, \otimes) \\ \mathcal{L}A \arrow[ur, dashed, "\exists!\, f^{\#}"'] & \end{tikzcd}$$

1

[1]HoTT book, lemma 6.11.5

Lists

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

Lists

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

_++_ : List A → List A → List A
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)
```

Lists

(**List** A,[],++) is a monoid

```
++-unitl : ∀ xs → [] ++ xs == xs

++-unitr : ∀ xs → xs ++ [] == xs

++-assoc : ∀ xs ys zs
  → xs ++ (ys ++ zs) == (xs ++ ys) ++ zs
```

Lists

Given a monoid `(M,e,⊗)` and `f : A → M`, we have

```
f♯ : List A → M
f♯ [] = e
f♯ (x :: xs) = f x ⊗ f♯ xs

f♯-++ : ∀ xs ys → f♯ (xs ++ ys) == f♯ xs ⊗ f♯ ys
```

For any monoid homomorphism `h : List A → M`,

```
f♯-unique : h == f♯
```

Outline

Free monoids

Free commutative monoids

Applications

Free symmetric monoidal categories

Free commutative monoids

The forgetful functor from **CMon** to **Set** also has a left adjoint.

$$\begin{tikzcd} \mathsf{CMon} \arrow[d] \\ \mathsf{Set} \arrow[u, bend left, "\mathcal{M}"] \end{tikzcd}$$

Free commutative monoids

The forgetful functor from **CMon** to **Set** also has a left adjoint.

$$\begin{tikzcd} \mathbf{CMon} \arrow[d] \\ \mathbf{Set} \arrow[u, bend left, "\mathcal{M}"] \end{tikzcd}$$

$\mathcal{M}\,A$ = finite multisets with elements drawn from A.

For example, the free commutative monoid on the set of prime numbers gives the natural numbers $\mathbb{N}$ with multiplication.

Universal property

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\eta_A"'] & M(e, \otimes) \\
\mathcal{M}\, A \arrow[ur, dashed, "\exists!\, f^{\#}"'] &
\end{tikzcd}
$$

Universal property

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\eta_A"'] & M(e, \otimes) \\ \mathcal{M}\,A \arrow[ur, dashed, "\exists!\, f^{\#}"'] & \end{tikzcd}$$

How do we define finite multisets in type theory?

Multiset/Bag

```
data Mset (A : Type) : Type where
  [] : Mset A
  _::_ : A → Mset A → Mset A
  swap : (x y : A) (xs : Mset A)
       →  x :: y :: xs == y :: x :: xs
  trunc : is-set (Mset A)
```

Multiset elimination

```
MsetElim : {B : Mset A → hSet}
  ([]* : B [])
  (_::*_ : (x : A) {xs : Mset A}
    → B xs → B (x :: xs))
  (swap* : (x y : A) {xs : Mset A} (b : B xs)
    → PathP (λ i → B (swap x y xs i))
        (x ::* (y ::* b)) (y ::* (x ::* b)))

MsetElimProp : {B : Mset A → hProp}
  ([]* : B [])
  (_::*_ : (x : A) {xs : Mset A}
    → B xs → B (x :: xs))
```

Multiset union

```
_∪_ : Mset A → Mset A → Mset A
[] ∪ ys = ys
(x :: xs) ∪ ys = x :: (xs ∪ ys)
(swap x y xs i) ∪ ys = swap x y (xs ∪ ys) i
(trunc xs zs p q i j) ∪ ys =
  trunc (xs ∪ ys) (zs ∪ ys)
    (λ i → p i ∪ ys) (λ i → q i ∪ ys) i j
```

Multiset union

(`Mset A,[],⋃`) is a monoid

```
⋃-assoc : ∀ xs ys zs
  → xs ⋃ (ys ⋃ zs) == (xs ⋃ ys) ⋃ zs

⋃-unitl : ∀ xs → [] ⋃ xs == xs

⋃-unitr : ∀ xs → xs ⋃ [] == xs
```

Commutativity of union

Canonical form for `x :: xs`

```
::-⋃ : ∀ x xs → x :: xs == xs ⋃ [ x ]
::-⋃ x [] i = [ x ]
::-⋃ x (y :: xs) i =
```

$$
\begin{tikzcd}
\texttt{x :: y :: xs} \arrow[r, dashed] & \texttt{y :: (xs} \bigcup \texttt{[x])} \\
\texttt{x :: y :: xs} \arrow[u, "\texttt{x :: y :: xs}"] \arrow[r, "\texttt{swap x y xs i}"'] & \texttt{y :: x :: xs} \arrow[u, "\texttt{y :: (::-}\bigcup\texttt{ x xs j)}"']
\end{tikzcd}
$$

Commutativity of union

```
⋃-comm : ∀ xs ys → xs ⋃ ys == ys ⋃ xs
⋃-comm [] ys i = ⋃-unitr ys (~ i)
⋃-comm (x :: xs) ys i =
```

$$\begin{tikzcd}
\texttt{x :: (xs ⋃ ys)} \arrow[r, dashed] & \texttt{ys ⋃ (x :: xs)} & \texttt{ys ⋃ (x :: xs)} \arrow[l, "\texttt{ys ⋃ (x :: xs)}"'] \\
\texttt{x :: (ys ⋃ xs)} \arrow[u, "\texttt{x :: ⋃-comm xs ys (\~{} j)}"] \arrow[r, "\texttt{::-⋃ x (ys ⋃ xs) i}"'] & \texttt{(ys ⋃ xs) ⋃ [x]} \arrow[u, dashed] & \texttt{ys ⋃ (xs ⋃ [x])} \arrow[u, "\texttt{ys ⋃ (::-⋃ x xs (\~{} j))}"'] \arrow[l, "\texttt{assoc-⋃ ys xs [x] k}"]
\end{tikzcd}$$

Multiset

Given a commutative monoid (`M,e,⊗`) and `f : A → M`, we have

```
f♯ : Mset A → M

f♯-[] : f♯ [] == e
f♯-⋃ : ∀ xs ys → f♯ (xs ⋃ ys) == f♯ xs ⊗ f♯ ys
```

For any commutative monoid homomorphism `h : List A → M`,

```
f♯-unique : h == f♯
```

Path space

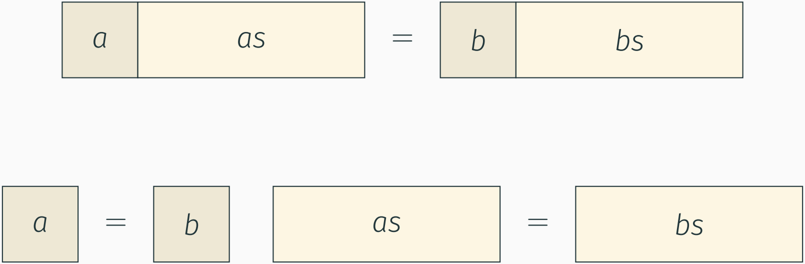
Can we characterise the path space of **Mset** A?

```
code : Mset A → Mset A → hProp
code [] [] = ⊤
...
code (a :: as) (b :: bs) =
  (a == b) ∧ code as bs ...
```

Path space

$$\boxed{a}\boxed{\quad as \quad} = \boxed{b}\boxed{\quad bs \quad}$$

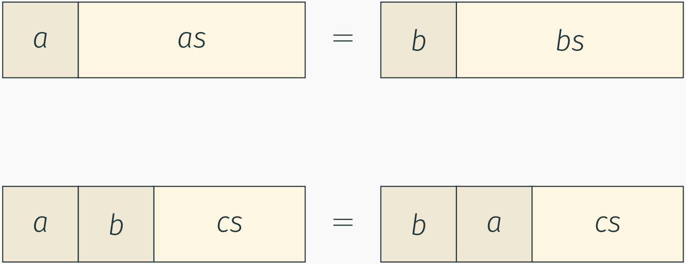
Path space



Path space

$$\boxed{a}\boxed{\quad as \quad} = \boxed{b}\boxed{\quad bs \quad}$$

Path space



Path space

```
code : Mset A → Mset A → hProp
code [] [] = ⊤
...
code (a :: as) (b :: bs) =
   (a == b) ∧ code as bs
 ∨ ∃ cs. code as (b :: cs) ∧ code bs (a :: cs)
```

Multiset

```
commrel : (a b c : A) (as bs cs : Mset A)
        → (p : as == b :: cs)
        → (q : a :: cs == bs)
        →  a :: as == b :: bs ²
```

```
swap x y xs =
  comm x y (y :: xs) (x :: xs) xs refl refl
```

[2] Marcelo Fiore. “An axiomatics and a combinatorial model of creation/annihilation operators”. In: *arXiv preprint arXiv:1506.06402* (2015).

Multiset

```
data Mset (A : Type) : Type where
  [] : Mset A
  _::_ : A → Mset A → Mset A
  commrel : (a b c : A) (as bs cs : Mset A)
          → (p : as == b :: cs)
          → (q : a :: cs == bs)
          →  a :: as == b :: bs
  trunc : is-set (Mset A)
```

Multiset

```
data Mset (A : Type) : Type where
  [] : Mset A
  _::_ : A → Mset A → Mset A
  commrel : (a b c : A) (as bs cs : Mset A)
          → (p : as == b :: cs)
          → (q : a :: cs == bs)
          →  a :: as == b :: bs
  trunc : is-set (Mset A)
```

This also satisfies the same universal property!

Outline

Free monoids

Free commutative monoids

Applications

Free symmetric monoidal categories

Strong symmetric monoidal functor

$$\mathcal{M}\,(A+B) \simeq \mathcal{M}\,A \times \mathcal{M}\,B$$

$$h : A + B \to \mathcal{M}\,A \times \mathcal{M}\,B$$
$$h(inl(a)) = ([a], [\;\;])$$
$$h(inr(b)) = ([\;\;], [b])$$

$$f : \mathcal{M}\,(A+B) \to \mathcal{M}\,A \times \mathcal{M}\,B$$
$$f = h^{\#}$$

$$g : \mathcal{M}\,A \times \mathcal{M}\,B \xrightarrow{\mathcal{M}\,(inl) \times \mathcal{M}\,(inr)} \mathcal{M}\,(A+B) \times \mathcal{M}\,(A+B) \xrightarrow{\cup} \mathcal{M}\,(A+B)$$

Monad on hSet

$$\begin{tikzcd} hSet \arrow["\mathcal{M}"', loop above] \end{tikzcd} \qquad \begin{tikzcd} hSet \arrow["\mathcal{P}=hProp^{(-)}"', loop above] \end{tikzcd}$$

$$\eta_A : A \to \mathcal{M}\,A \qquad \eta_A : A \to \mathcal{P}\,A$$

$$\eta_A(a) \coloneqq [a] \qquad \eta_A(a) \coloneqq \lambda x.a = x$$

$$\mu_A : \mathcal{M}^2\,A \to \mathcal{M}\,A \qquad \mu_A : \mathcal{P}^2\,A \to \mathcal{P}\,A$$

$$\mu_A \coloneqq id^{\#} \qquad \mu_A(f) \coloneqq \lambda x.\exists y.f(y)(x)$$

$\mathcal{M}$ *Rel*

$$f : A \twoheadrightarrow B \coloneqq \mathcal{M}\, A \times B \to hProp$$

$$\hat{f} : B \to (\mathcal{M}\, A \to hProp)$$

$$\hat{f}(b)(\alpha) \coloneqq f(\alpha, b)$$

$$\hat{f}^{\#} : \mathcal{M}\, B \to (\mathcal{M}\, A \to hProp)$$

$$id_A : A \twoheadrightarrow A$$

$$id_A(\alpha, a) \coloneqq \alpha = [a]$$

$$f : A \twoheadrightarrow B,\ g : B \twoheadrightarrow C$$

$$g \circ f(\alpha, c) \coloneqq \exists \beta . \hat{f}^{\#}(\beta)(\alpha) \wedge g(\beta, c)$$

$$A \times B \coloneqq A + B$$

$$A \Rightarrow B \coloneqq \mathcal{M}\, A \times B$$

---

[2] $(M, e, \cdot)$ acts on *hProp*

$$\hat{e} = \lambda x . x = e$$

$$p \,\hat{\cdot}\, q = \lambda x . \exists x_1 x_2 . p(x_1) \wedge p(x_2) \wedge x = x_1 \cdot x_2$$

Monoidal structure

Given $f, g : A \relbar\joinrel\mapstochar\joinrel\rightarrow B$,

Addition

$$(f + g)(\alpha, b) \coloneqq f(\alpha, b) \vee g(\alpha, b)$$

Multiplication

$$(f \cdot g)(\alpha, b) \coloneqq f(\alpha, b) \mathbin{\hat{\bullet}} g(\alpha, b)$$

Differentiation

$$\partial f : A \twoheadrightarrow A \times B$$

$$\partial f(\alpha, (a, b)) \coloneqq f(\alpha \cup [a], b)$$

Leibniz's Rule

$$\partial(f \cdot g) = \partial f \cdot g + \partial g \cdot f$$

2

$$\alpha \cup [a] = \alpha_1 \cup \alpha_2 \simeq$$
$$\exists \alpha_0.(\alpha = \alpha_0 \cup \alpha_2) \wedge (\alpha_0 \cup [a] = \alpha_1)$$
$$\vee(\alpha = \alpha_1 \cup \alpha_0) \wedge (\alpha_0 \cup [a] = \alpha_2)$$

Outline

Free monoids

Free commutative monoids

Applications

Free symmetric monoidal categories

Free symmetric monoidal completion (Work in Progress)

```
data SMC (A : Type) : Type where
  [] : SMC A
  _::_ : A → SMC A → SMC A
  swap : (x y : A) (xs : SMC A)
      →  x :: y :: xs == y :: x :: xs
  ...
  trunc : is-gpd (SMC A)
```

Free symmetric monoidal completion (Work in Progress)

```
data SMC (A : Type) : Type where
  [] : SMC A
  _::_ : A → SMC A → SMC A
  swap : (x y : A) (xs : SMC A)
      →  x :: y :: xs == y :: x :: xs
  ...
  trunc : is-gpd (SMC A)
```

$$
\begin{tikzcd}
\texttt{y :: x :: z :: xs} \arrow[r, "\texttt{swap y x (z :: xs) i}"] \arrow[d, equal] & \texttt{x :: y :: z :: xs} \arrow[r, "\texttt{x :: (swap y z xs) i}"] & \texttt{x :: z :: y :: xs} \arrow[r, "\texttt{swap x z (:: y xs) i}"] & \texttt{z :: x :: y :: xs} \arrow[d, equal] \\
\texttt{y :: x :: z :: xs} \arrow[r, "\texttt{y :: (swap x z xs i)}"'] & \texttt{y :: z :: x :: xs} \arrow[r, "\texttt{swap y z (x :: xs) i}"'] & \texttt{z :: y :: x :: xs} \arrow[r, "\texttt{z :: (swap y x xs) i}"'] & \texttt{z :: x :: y :: xs}
\end{tikzcd}
$$

Other applications

- Differential calculus of generalised species[3]
- $SMC(1) \simeq \sum_{n:\mathbb{N}} \sum_{X:U} \|X = Fin(n)\|$ gives a denotational semantics for reversible languages[4]

[3] M. Fiore et al. "The cartesian closed bicategory of generalised species of structures". In: *Journal of the London Mathematical Society* 77.1 (2008), pp. 203–220.

[4] Jacques Carette et al. "From Reversible Programs to Univalent Universes and Back". In: *Electr. Notes Theor. Comput. Sci.* 336 (2018), pp. 5–25.