Weak Structures from Strict Ones

HoTTEST Conference 2020

Eric Finster

Joint work with Matthieu Sozeau
and Antoine Allioux

June 19, 2020

Higher Structures in HoTT

1. HoTT Imagined as a foundation for higher/proof relevant mathematics

Higher Structures in HoTT

1. HoTT Imagined as a foundation for higher/proof relevant mathematics
2. Problem: haven't been able to define interesting algebraic structures on non-truncated types

Higher Structures in HoTT

1. HoTT Imagined as a foundation for higher/proof relevant mathematics
2. Problem: haven't been able to define interesting algebraic structures on non-truncated types
3. Two main approaches proposed:

Higher Structures in HoTT

1. HoTT Imagined as a foundation for higher/proof relevant mathematics
2. Problem: haven't been able to define interesting algebraic structures on non-truncated types
3. Two main approaches proposed:
    1. Synthetic

Higher Structures in HoTT

1. HoTT Imagined as a foundation for higher/proof relevant mathematics
2. Problem: haven't been able to define interesting algebraic structures on non-truncated types
3. Two main approaches proposed:
    1. Synthetic
    2. Two-level

Higher Structures in HoTT

1. HoTT Imagined as a foundation for higher/proof relevant mathematics
2. Problem: haven't been able to define interesting algebraic structures on non-truncated types
3. Two main approaches proposed:
    1. Synthetic
    2. Two-level
4. Here I propose a third approach: adding a universe of "strict" structures.

Higher Structures in HoTT

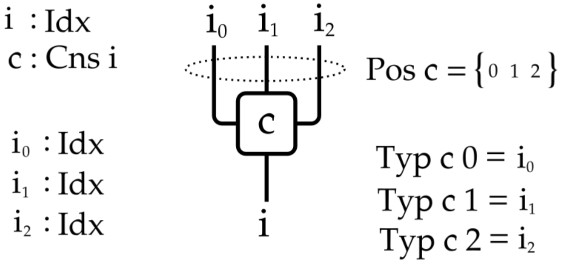
1. HoTT Imagined as a foundation for higher/proof relevant mathematics
2. Problem: haven't been able to define interesting algebraic structures on non-truncated types
3. Two main approaches proposed:
    1. Synthetic
    2. Two-level
4. Here I propose a third approach: adding a universe of "strict" structures.
5. `https://github.com/ericfinster/opetopic-types`

A Universe of Monads

Every monad (M : $\mathbb{M}$) has an underlying *polynomial*:

```
postulate

  𝕄 : Set

  Idx : 𝕄 → Set

  Cns : (M : 𝕄) (i : Idx M) → Set

  Pos : (M : 𝕄) {i : Idx M}
    → Cns M i → Set

  Typ : (M : 𝕄) {i : Idx M}
    → (c : Cns M i) (p : Pos M c)
    → Idx M
```

Elements of monads as constructors



$$\mathsf{Idx} \xleftarrow[\mathsf{Typ}]{} \sum_{i:\mathsf{Idx}} \sum_{c:\mathsf{Cns}} \mathsf{Pos}\, c \longrightarrow \sum_{i:\mathsf{Idx}} \mathsf{Cns} \longrightarrow \mathsf{Idx}$$

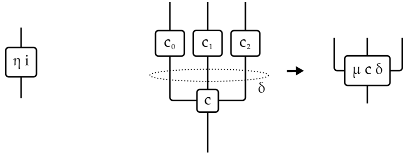
Monadic Structure

We then equip this data with some structure:

postulate

$\eta : (M : \mathbb{M})\ (i : \mathsf{Idx}\ M) \to \mathsf{Cns}\ M\ i$

$\mu : (M : \mathbb{M})\ \{i : \mathsf{Idx}\ M\}\ (c : \mathsf{Cns}\ M\ i)$
$\to (\delta : (p : \mathsf{Pos}\ M\ c) \to \mathsf{Cns}\ M\ (\mathsf{Typ}\ M\ c\ p))$
$\to \mathsf{Cns}\ M\ i$



Definitional Laws

```
μ-unit-right : (M : 𝕄) (i : Idx M) (c : Cns M i)
  → μ M c (λ p → η M (Typ M c p)) ↦ c
{-# REWRITE μ-unit-right #-}

μ-unit-left : (M : 𝕄) (i : Idx M)
  → (δ : (p : Pos M (η M i)) → Cns M i)
  → μ M (η M i) δ ↦ δ (η-pos M i)
{-# REWRITE μ-unit-left #-}

μ-assoc : (M : 𝕄) {i : Idx M} (c : Cns M i)
  → (δ : (p : Pos M c) → Cns M (Typ M c p))
  → (ε : (p : Pos M (μ M c δ)) → Cns M (Typ M (μ M c δ) p))
  → μ M (μ M c δ) ε ↦
      μ M c (λ p → μ M (δ p) (λ q → ε (μ-pos M c δ p q)))
{-# REWRITE μ-assoc #-}
```

We now begin populating our universe:

```
postulate
  IdMnd : 𝕄

Idxᵢ = ⊤ᵢ
Cnsᵢ _ = ⊤ᵢ
Posᵢ _ = ⊤ᵢ
Typᵢ _ _ = ttᵢ

ηᵢ _ = ttᵢ
μᵢ _ δ = δ ttᵢ
```

$$\top_i \longleftarrow \top_i \longrightarrow \top_i \longrightarrow \top_i$$

The Reindexed Monad

We can "reindex" a monad to get a new monad:

```
postulate
  Pb : (M : 𝕄) (X : Idx M → Set) → 𝕄

Idxₚ M X = Σ (Idx M) X

Cnsₚ M X (i , x) =
  Σ (Cns M i) (λ c → (p : Pos M c) → X (Typ M c p))

Posₚ M X (c , ν) = Pos M c

Typₚ M X {i = i , x} (c , ν) p = Typ M c p , ν p
```

The Reindexed Monad (Cont'd)

```
ηₚ M X (i , x) = η M i , λ _ → x

μₚ M X {i = i , x} (c , ν) κ =
  let κ' p = fst (κ p)
      ν' p = snd (κ (μ-pos-fst M c κ' p))
                 (μ-pos-snd M c κ' p)
  in μ M c κ' , ν'
```

$$
\begin{array}{ccccccc}
X & \longleftarrow & P & \longrightarrow & MX \times_{\mathsf{Idx}} X & \longrightarrow & X \\
\downarrow & & \downarrow \lrcorner & & \downarrow & & \downarrow \\
\mathsf{Idx} & \longleftarrow & \mathsf{Pos} & \longrightarrow & \mathsf{Cns} & \longrightarrow & \mathsf{Idx}
\end{array}
$$

The Baez-Dolan Slice Construction

```
Idxₛ M = Σ (Idx M) (Cns M)

data Cnsₛ M where

  lf : (i : Idx M) → Cnsₛ M (i , η M i)

  nd : {i : Idx M} (c : Cns M i)
    → (δ : (p : Pos M c) → Cns M (Typ M c p))
    → (ε : (p : Pos M c) → Cnsₛ M (Typ M c p , δ p))
    → Cnsₛ M (i , μ M c δ)

Posₛ M (lf τ) = ⊥
Posₛ M (nd c δ ε) = ⊤ ⊔ (Σ (Pos M c) (λ p → Posₛ M (ε p)))

Typₛ M (nd {i} c δ ε) (inl unit) = i , c
Typₛ M (nd c δ ε) (inr (p , q)) = Typₛ M (ε p) q
```

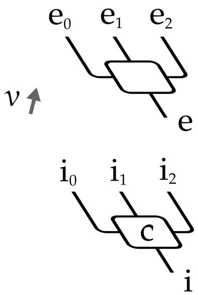
This setup is already sufficient to give a coinductive definition of opetopic type:

```
record OpetopicType (M : 𝕄) : Set₁ where
  coinductive
  field

    El : Idx M → Set
    Fill : OpetopicType (Slice (Pb M El))
```

A Bit of Intuition

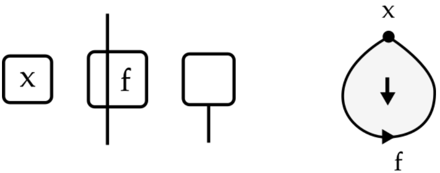
El' : Idx M → Set
Fill' : Idx (Slice (Pb M El')) → Set

i : Idx M
e : El' i
c : Cns M i
ν : (p : Pos M c) → El' (Typ M c p)



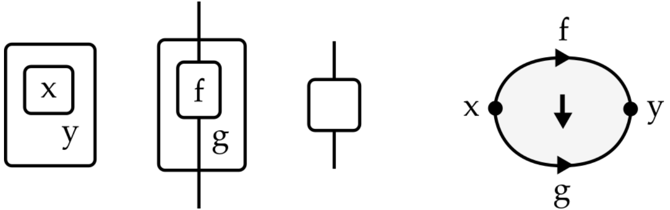
Opetopic Types: Examples

```
module _ (X : OpetopicType IdMnd) where

  Obj : Set
  Obj = El X tt_i

  Arrow : (x y : Obj) → Set
  Arrow x y = El (Fill X) ((tt_i , y) , (tt_i , cst x))

  NullHomotopy : {x : Obj} (f : Arrow x x) → Set
  NullHomotopy {x} f = El (Fill (Fill X))
    ((((tt_i , x) , (tt_i , cst x)) , f) , lf (tt_i , x) , ⊥-elim)
```



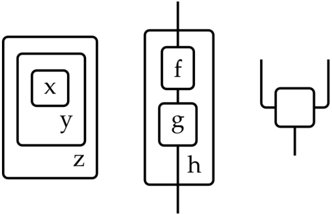
Opetopic Types: Examples (Cont'd)

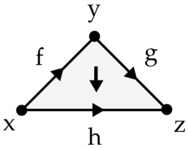
```
Disc : {x y : Obj} (f : Arrow x y) (g : Arrow x y)
  → Set
Disc {x} {y} f g = El (Fill (Fill X))
  ((((ttᵢ , y) , (ttᵢ , cst x)) , g) ,
    (nd (ttᵢ , cst x) (cst (ttᵢ , (cst x))) (cst (lf (ttᵢ , x)))) ,
      (λ { true → f }))
```



Opetopic Types: Examples (Cont'd)

```
Simplex : {x y z : Obj}
  → (f : Arrow x y) (g : Arrow y z)
  → (h : Arrow x z) → Set
Simplex {x} {y} {z} f g h = El (Fill (Fill X))
  ((((ttᵢ , z) , (ttᵢ , cst x)) , h) ,
    (nd (ttᵢ , cst y) (cst (ttᵢ , cst x)) (cst
      (nd (ttᵢ , (cst x)) (cst (ttᵢ , cst x)) (cst (lf (ttᵢ , x)))))) ,
    (λ { true → g ;
         (inr (ttᵢ , true)) → f }))
```



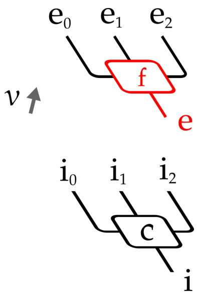


Fibrant Opetopic Types

```
unique-action : (M : 𝕄) (El : Idx M → Set)
  → (Fill : Idx (Slice (Pb M El)) → Set)
  → Set
unique-action M El Fill = (i : Idx M) (c : Cns M i)
  → (ν : (p : Pos M c) → El (Typ M c p))
  → is-contr (Σ (El i) (λ a → Fill ((i , a) , c , ν)))
```

```
record is-fibrant {M : 𝕄} (X : OpetopicType M) : Set where
  coinductive
  field
    ob-fibrant : unique-action M (El X) (El (Fill X))
    hom-fibrant : is-fibrant (Fill X)
```

A fibrant opetopic type should be thought of as a *weak algebra* over M.



Some Higher Structures

We can now define

```
∞-Groupoid : Set₁
∞-Groupoid = Σ (OpetopicType IdMnd) is-fibrant

∞-Category : Set₁
∞-Category = Σ (OpetopicType IdMnd)
  (λ X → is-fibrant (Fill X))

A∞-Space : Set₁
A∞-Space = Σ (OpetopicType (Slice IdMnd)) is-fibrant

∞-PlanarOperad : Set₁
∞-PlanarOperad = Σ (OpetopicType (Slice (Slice IdMnd)))
  is-fibrant
```

Globular Types

Recall the definition of globular types:

```
record GType : Set₁ where
  coinductive
  field
    Ob : Set
    Hom : (x y : Ob) → GType
```

Every type determines a globular type:

```
IdG : (X : Set) → GType
Ob (IdG X) = X
Hom (IdG X) x y = IdG (x == y)
```

Globularity Cont'd

We can introduce globular equivalences:

```
record _≃g_ (X Y : GType) : Set where
  coinductive
  field
    ObEqv : Ob X ≃ Ob Y
    HomEqv : (x y : Ob X)
      → (Hom X x y) ≃g
        (Hom Y (–> ObEqv x) (–> ObEqv y))
```

Opetopic To Globular

Every opetopic type determines a globular one:

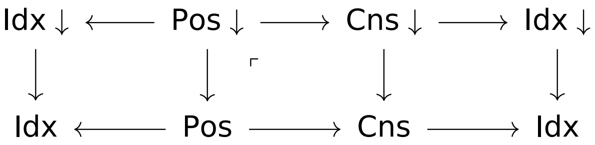
```
OpToGlob : (M : 𝕄) (X : OpetopicType M)
  → Idx M → GType
Ob (OpToGlob M X i) = El X i
Hom (OpToGlob M X i) x y =
  OpToGlob (Slice (Pb M (El X))) (Fill X)
                        ((i , y) , (η M i , λ _ → x))
```

Theorem

```
thm : (M : 𝕄) (X : OpetopicType M)
  → (i : Idx M) (is-fib : is-fibrant X)
  → OpToGlob M X i ≃g IdG (El X i)
```

Dependent Monads

```
postulate
  𝕄↓ : (M : 𝕄) → Set
  Idx↓ : {M : 𝕄} (M↓ : 𝕄↓ M) → Idx M → Set
  Cns↓ : {M : 𝕄} (M↓ : 𝕄↓ M)
    → {i : Idx M} (i↓ : Idx↓ M↓ i)
    → Cns M i → Set
  Typ↓ : {M : 𝕄} (M↓ : 𝕄↓ M)
    → {i : Idx M} {c : Cns M i}
    → {i↓ : Idx↓ M↓ i} (c↓ : Cns↓ M↓ i↓ c)
    → (p : Pos M c) → Idx↓ M↓ (Typ M c p)
```

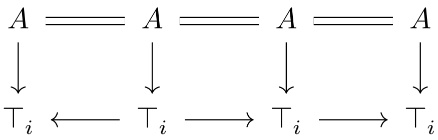


$$
\begin{aligned}
&\eta\!\downarrow\ :\ \{M : \mathbb{M}\}\ (M\!\downarrow\ :\ \mathbb{M}\!\downarrow\ M)\\
&\quad \to \{i : \mathsf{Idx}\ M\}\ (i\!\downarrow\ :\ \mathsf{Idx}\!\downarrow\ M\!\downarrow\ i)\\
&\quad \to \mathsf{Cns}\!\downarrow\ M\!\downarrow\ i\!\downarrow\ (\eta\ M\ i)
\end{aligned}
$$

$$
\begin{aligned}
&\mu\!\downarrow\ :\ \{M : \mathbb{M}\}\ (M\!\downarrow\ :\ \mathbb{M}\!\downarrow\ M)\\
&\quad \to \{i : \mathsf{Idx}\ M\}\ \{c : \mathsf{Cns}\ M\ i\}\\
&\quad \to \{\delta : (p : \mathsf{Pos}\ M\ c) \to \mathsf{Cns}\ M\ (\mathsf{Typ}\ M\ c\ p)\}\\
&\quad \to \{i\!\downarrow\ :\ \mathsf{Idx}\!\downarrow\ M\!\downarrow\ i\}\ (c\!\downarrow\ :\ \mathsf{Cns}\!\downarrow\ M\!\downarrow\ i\!\downarrow\ c)\\
&\quad \to (\delta\!\downarrow\ :\ (p : \mathsf{Pos}\ M\ c) \to \mathsf{Cns}\!\downarrow\ M\!\downarrow\ (\mathsf{Typ}\!\downarrow\ M\!\downarrow\ c\!\downarrow\ p)\ (\delta\ p))\\
&\quad \to \mathsf{Cns}\!\downarrow\ M\!\downarrow\ i\!\downarrow\ (\mu\ M\ c\ \delta)
\end{aligned}
$$

Example: Relative Identity

```
Idx↓ᵢ _ = A
Cns↓ᵢ a _ = ⊤ᵢ
Typ↓ᵢ {a = a} _ _ = a

η↓ᵢ a = ttᵢ
μ↓ᵢ {δ = δ} {a = a} d δ↓ = δ↓ ttᵢ
```



Fibered Constructors

postulate

$$
\begin{aligned}
&\mathsf{Pb}{\downarrow} : \{M : \mathbb{M}\}\ (M{\downarrow} : \mathbb{M}{\downarrow}\ M) \\
&\quad \to (X : \mathsf{Idx}\ M \to \mathsf{Set}) \\
&\quad \to (Y : (i : \mathsf{Idx}\ M) \to \mathsf{Idx}{\downarrow}\ M{\downarrow}\ i \to X\ i \to \mathsf{Set}) \\
&\quad \to \mathbb{M}{\downarrow}\ (\mathsf{Pb}\ M\ X)
\end{aligned}
$$

$$
\mathsf{Slice}{\downarrow} : \{M : \mathbb{M}\}\ (M{\downarrow} : \mathbb{M}{\downarrow}\ M) \to \mathbb{M}{\downarrow}\ (\mathsf{Slice}\ M)
$$

↓-to-OpType : (M : 𝕄) (M↓ : 𝕄↓ M)
  → OpetopicType M
El (↓-to-OpType M M↓) = Idx↓ M↓
Fill (↓-to-OpType M M↓) =
  ↓-to-OpType (Slice (Pb M (Idx↓ M↓)))
    (Slice↓ (Pb↓ M↓ (Idx↓ M↓) (λ i j k → j == k)))

In particular, we have:

TypeToOpType : (A : Set) → OpetopicType IdMnd
TypeToOpType A = ↓-to-OpType IdMnd (IdMnd↓ A)

```
record alg-comp (M : 𝕄) (M↓ : 𝕄↓ M)
  (i : Idx M) (c : Cns M i)
  (ν : (p : Pos M c) → Idx↓ M↓ (Typ M c p)) : Set where
  constructor ⟦_|_|_⟧
  field
    idx : Idx↓ M↓ i
    cns : Cns↓ M↓ idx c
    typ : Typ↓ M↓ cns == ν

is-algebraic : (M : 𝕄) (M↓ : 𝕄↓ M) → Set
is-algebraic M M↓ = (i : Idx M) (c : Cns M i)
  → (ν : (p : Pos M c) → Idx↓ M↓ (Typ M c p))
  → is-contr (alg-comp M M↓ i c ν)
```

The Slice is Always Algebraic

```
module _ (M : 𝕄) (M↓ : 𝕄↓ M) where

  Slc : 𝕄
  Slc = Slice (Pb M (Idx↓ M↓))

  Slc↓ : 𝕄↓ Slc
  Slc↓ = Slice↓ (Pb↓ M↓ (Idx↓ M↓) (λ i j k → j == k))
```

Theorem

slc-algebraic : is-algebraic Slc Slc↓

Types give ∞-groupoids

Corollary

We have a map:

$$\textit{Type-to-}\infty\textit{-Groupoid} : \textit{Set} \rightarrow \infty\textit{-Groupoid}$$

Furthermore:

$$\textit{globes-are-paths} : (A : \textit{Set}) \rightarrow \textit{OpToGlob IdMnd (TypeToOpType } A) \; tt_i \simeq g \; \textit{IdG } A$$

Proof.

The identity monad is easily checked to be algebraic. An algebraic extension implies that the indices over have a unique action, i.e. it is fibrant at the first level. Since the slice is *always* algebraic, we can now iterate at will. □

Conjecture: This map is an equivalence.