Type Theory and the Opetopes

HDACT - Ljubljana

Eric Finster

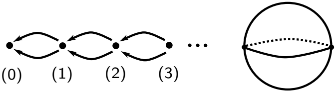
June 20, 2012

Outline

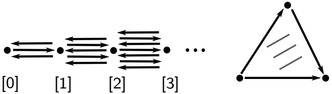
Shape Categories

Definitions of higher categories typically begin with the selection of a shape to represent higher dimensional cells:

- For example, there's the globe category $\mathbb{G}^{op}$:

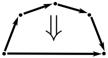


- We've got the simplicial category Δ^{op}:



- But there's also the category of opetopes $\mathcal{O}$:

???



The Idea of Opetopes

The two main priciples behind the definition of the opetopes are the following:

The Informal Version

1. Cells will be allowed to have many sources (input faces), but only a single target (output face)
2. Cells of dimension $n+1$ should be in bijection with *pasting diagrams* in dimension n, that is, all possible ways of attaching cells by gluing compatible sources and targets

We think of the process of turning a given pasting diagram into a cell as *extruding it* into the next dimension up.

Low Dimensions

- In dimension 0, we have a point. It has no source and no target.

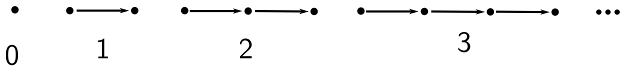
$$\bullet$$

- The only way to arrange a family of points, gluing sources to targets is to simply have a single point. Points do not cohere in any meaningful way.
- Extending our unique 0-dimensional pasting diagram gives us the unique 1-dimensional cell, the arrow.

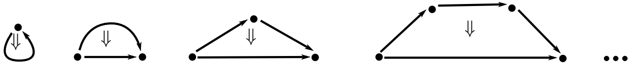
$$\bullet \qquad\qquad \bullet \longrightarrow \bullet$$

Low Dimensions (cont'd)

- Now in dimension 1, we have the arrow: it has a single source and a single target.
- What are all the ways of coherently gluing sources to targets in a collection of arrows?
- There are an $\mathbb{N}$'s worth:

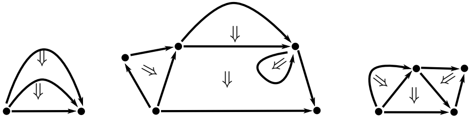


- Now we extrude each pasting diagram into the next dimension, and give it an "appropriate" target. In the case at hand, we have only one choice: the arrow.
- So our two cells look like this:

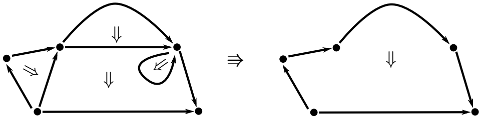


Low Dimensions (cont'd)

- Here are some 2-dimensional pasting diagrams:

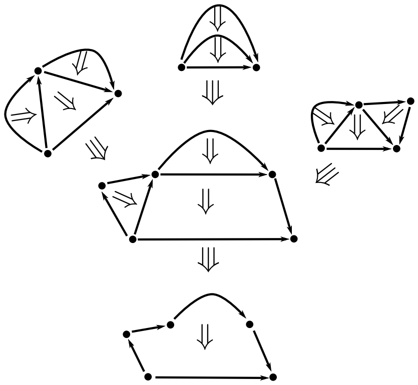


- And and example 3-dimensional cell:



Low Dimensions (cont'd)

And finally a 3-pasting diagram:



Opetopes from Polynomial Funtors

1. How can we make this intuitive definition precise?
2. One of the simplest ways to do this (due to Kock, Joyal, Batanin and Mascari) is to realize these shapes as a canonical sequence *polynomial functors*
3. These have different names in the computer science community: inductive families, indexed containers, indexed W-types, . . .

Polynomial Functors

Definition

A *polynomial* P is a diagram of sets

$$\begin{tikzcd} & E \arrow[dl, "t"'] \arrow[r, "p"] & B \arrow[dr, "r"] & \\ I & & & I \end{tikzcd}$$

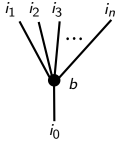
Any polynomial dertermines a functor $[\![P]\!] : \mathcal{S}et/I \to \mathcal{S}et/I$ (its *extension*) defined for an I-Set $X \to I$ by the formula:

$$[\![P]\!](X) = \sum_{b \in B} \prod_{p \in E_b} X_{t(p)}$$

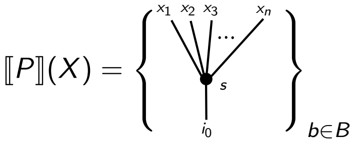
(Lower subscripts indicate the fibers of appropriate maps.)

Graphical Interpretation

- It's useful to represent the elements $b \in B$ as *corollas*



- We can then picture the set $[\![P]\!](X)$ as the collection of such corollas labelled with elements from X of the correct type:

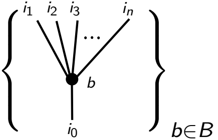


That is, $t(x_k) = i_k$.

Useful Special Cases

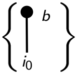
- Write 1_I for the terminal object of $\mathcal{S}et/I$. Then it is easily seen that $[\![P]\!](1_I) = B$. Graphically:

$$[\![P]\!](1_I) =$$



- For the initial object, we have

$$[\![P]\!](\emptyset) =$$

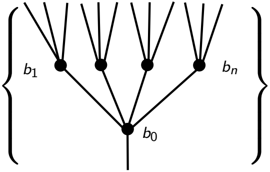


i.e., the set of constructors with no places.

Useful Special Cases (cont'd)

- By iterating the functor, we generate trees: for example, $[\![P]\!]^2(1_I) = [\![P]\!](B)$ is the set of two leveled trees

$$[\![P]\!]^2(1_I) =$$

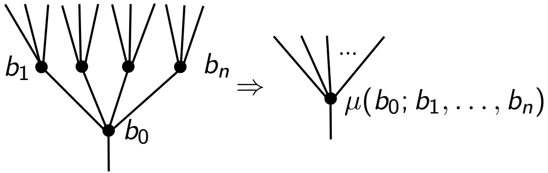


Monads

- When is the extension of an indexed container a monad?
- In particular, we would need to have a map

$$\mu_1 : [\![P]\!]^2(1_I) \to [\![P]\!](1_I) = B$$

- We can view this as a way to compose two-leveled trees:



- We say the monad is *cartesian* if the places of the multiplied constructor are in bijection with the leaves of the two-level tree (and their types match)

The Free Monad

- We can freely generate a monad from any polynomial, and moreover, this functor is again the extension of a polynomial
- Write

$$tr(P) = \bigcup_{n \to \infty} (I + [\![P]\!])^n(\emptyset)$$

- The elemenets are the finite tree's built from constructors in P (plus some units)

The Free Monad (cont'd)

- For a tree $t \in tr(P)$ write $L(t)$ for its set of leaves
- Then the free monad on $[\![P]\!]$ is given by the polynomial

$$\begin{tikzcd} \sum_{t \in tr(P)} L(t) \arrow[r, "\pi"] \arrow[d] & tr(P) \arrow[d] \\ I & I \end{tikzcd}$$

- The multiplication in this monad is given by simply grafting trees together at their leaves

The Slice Construction

- Observe that when P is a (cartesian) monad, we have a map

$$\mu^\infty : tr(P) \to B$$

which "collapses" each tree to a corolla

- Write $N(t)$ for the set of internal nodes of a tree $t \in tr(P)$
- The *slice construction* P^+ on P is the polynomial

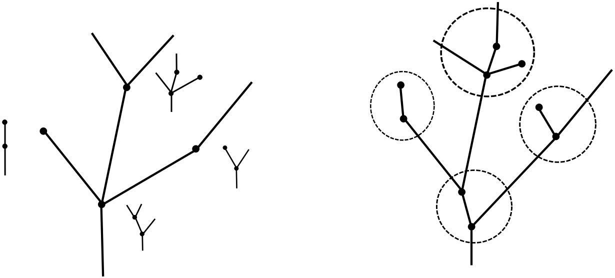
$$\begin{tikzcd}
& \sum_{t \in tr(P)} N(t) \arrow[dl] \arrow[r, "\pi"] & tr(P) \arrow[dr, "\mu^\infty"] & \\
B & & & B
\end{tikzcd}$$

Multiplication in the Slice Construction

Theorem

The slice construction P^+ is again a (cartesian) monad

Multiplication is given by *substitution* of trees.



Definition of the Opetopes

One useful monad is the identity functor on $\mathcal{S}et$, represented by the trivial polynomial:

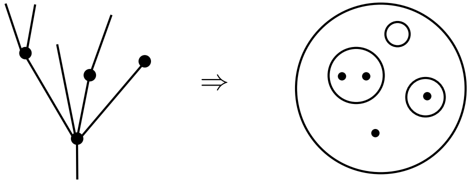
$$\begin{tikzcd}
& * \arrow[r] \arrow[dl] & * = \mathcal{O}(1) \arrow[dr, "\mu^\infty"] & \\
* & & & * = \mathcal{O}(0)
\end{tikzcd}$$

Definition

The set $\mathcal{O}(n)$ of n-dimensional opetopes is the indexing set of the n-th slice of the identity functor on $\mathcal{S}et$.

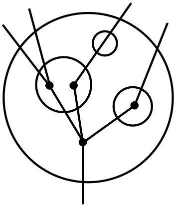
Notation for the Opetopes

- Our picture of tree substitution above leads naturally to the following graphical notation for depicting opetopes in all dimensions
- A *nesting* is a configuration of non-intersecting cicles and dots in the plane which corresponds to a tree



Notation (cont'd)

- A *constellation* is a nesting and a tree *superimposed* so that the nodes of the tree are the dots in the nesting



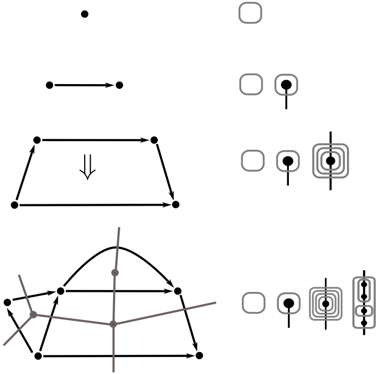
- These are subject to two rules
  1. There must be an outer circle containing all other dots and circles, except possibly if the tree contains exactly one node
  2. Every circle must cut a subtree (no "hanging" circles)

Notation (cont'd)

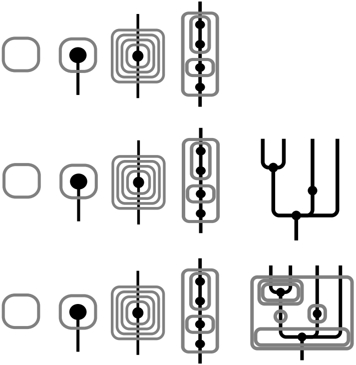
- An opetope can now be represented by a sequence of such constellations, with the dimension given by the number of terms in the sequence
- This is subject to an initial condition and a simple rule for moving to higher dimensions
- You can play with this notation in a graphical editor here:

  `http://sma.epfl.ch/~finster/opetope/opetope.html`

Notational Example

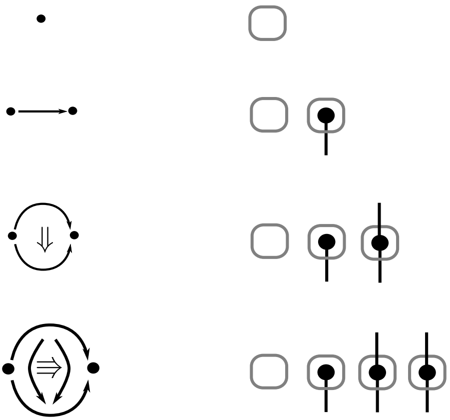


Notation (cont'd)



Opetopes and Globs

Globular shapes are a special case of opetopes:



Implemetation

- Opetopes can be represented by the following inductive type:

```
data MTree (A : Set) : ℕ → Set where
  obj : MTree A 0
  drop : {n : ℕ} → MTree ⊤ n → MTree A (n + 2)
  node : {n : ℕ} → A → MTree (MTree A (n + 1)) n
         → MTree A (n + 1)
```

- Elements of this type are “possible ill-typed A-labelled pasting diagrams”
- It is not hard to implement a “type-checker”

The Derivative

For implementing type-checking, the following "higher-dimensional zipper" is extremely useful:

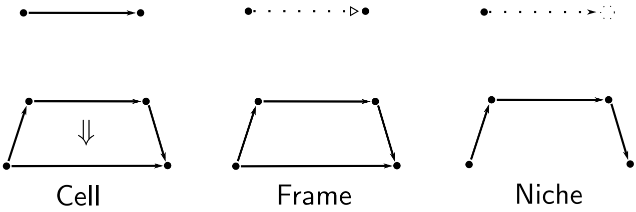
```
data Deriv (A : Set) : ℕ → Set where
  ∂ : {n : ℕ} → MTree (MTree A (n + 1)) n
                → Zipper A (n + 1) → Deriv A (n + 1)

data Zipper (A : Set) : ℕ → Set where
 Nil : {n : ℕ} → Zipper A (n + 1)
 Cons : {n : ℕ} → A → Deriv (MTree A (n + 1)) n
         → Zipper A (n + 1) → Zipper A (n + 1)

Context : Set → ℕ → Set
Context A n = Tree A n × Zipper A n
```

Cells, Frames and Niches

- When working with simplicial sets, we have three canonical families
  1. Simplices: Δ^n
  2. Boundaries: $\partial\Delta^n$
  3. Horns: Λ^n_k
- Opetopic sets have similar notations:



Opetopic “Identity” Types

- Consider the formation rule for identity types:

$$\frac{\Gamma \vdash A : Type}{\Gamma, x : A, y : A \vdash Id_A(x, y) : Type}$$

- Iteration gives a derived rule:

$$\frac{\Gamma \vdash A : Type}{\Gamma, x : A, y : A, f : Id_A(x, y), g : Id_A(x, y) \vdash Id_{Id_A(x,y)}(f, g) : Type}$$

- In each case, the data required in the context is exactly corresponds to a *frame* for a globular opetope.

Opetopic "Identity" Types (cont'd)

- Let π denote an abitrary opetope
- Write $\Gamma, [\![F : A]\!]_\pi \vdash \cdots$ as shorthand for the assumption of a variable for every face of the frame associated to π
- Example: for π the 2-frame below

$$\begin{tikzcd} y \arrow[r, "g"] & z \arrow[d, "h"] \\ x \arrow[u, "f"] \arrow[r, "k"'] & w \end{tikzcd}$$

  we would have

$$\Gamma, x : A, y : A, z : A, w : A, f : Id_A(x, y), \cdots \vdash \cdots$$

- Similarly, $\Gamma, [N : A]_\pi \vdash \cdots$ means enough variables for the faces of the niche associated to π

Opetopic Formation and Introduction

$$\frac{\Gamma \vdash A : \mathit{Type}}{\Gamma, [\![F : A]\!]_\pi \vdash \mathit{Fill}(F) : \mathit{Type}} \ \mathcal{O}\text{-Formation}$$

$$\frac{\Gamma \vdash [N : A]_\pi}{\Gamma \vdash \mathit{comp}(N) : \mathit{Fill}(N|_{\tau(\pi)})} \ \mathcal{O}\text{-composition}$$

$$\frac{\Gamma \vdash [N : A]_\pi}{\Gamma \vdash \mathit{refl}(N) : \mathit{Fill}(N \triangleright \mathit{comp}(N))} \ \mathcal{O}\text{-reflection}$$

Introduction Examples

$$
\begin{tikzcd}
x \arrow[d, "refl(x)"] \\
comp(x)
\end{tikzcd}
\qquad
\begin{tikzcd}
y \arrow[r, "g"] & z \arrow[d, "h"] \\
x \arrow[u, "f"] \arrow[r, "comp(N)"'] \arrow[ur, phantom, "refl(N)"] & w
\end{tikzcd}
$$

- When π is a glob, it contains a unique top dimensional source face, say x, and a new reduction rule says that $comp(x) \to x$ in this case
- This corresponds to the slogan "a nullary composition is an isomorphism"

A Generalized J-Rule

- The J-Rule

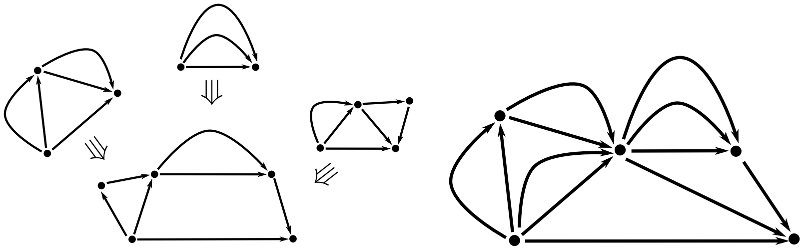
$$\frac{\begin{array}{c}\Gamma, x : A, y : A, f : Id_A(x,y) \vdash P(x,y,f) : Type \\ \Gamma, x : A \vdash p(x) : P(x,x,refl(x)) \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash g : Fill(G)\end{array}}{\Gamma \vdash J(a,b,g) : P(a,b,g)}$$

- An Opetopic J-Rule:

$$\frac{\begin{array}{c}\Gamma, [\![F : A]\!]_\pi, \alpha : Fill(F) \vdash P(F,\alpha) : Type \\ \Gamma, [N : A]_\pi \vdash p(N) : P(N \triangleright comp(N), refl(N)) \\ \Gamma \vdash [\![G : A]\!]_\pi \qquad \Gamma \vdash \beta : Fill(G)\end{array}}{\Gamma \vdash J(G,\beta) : P(G,\beta)}$$

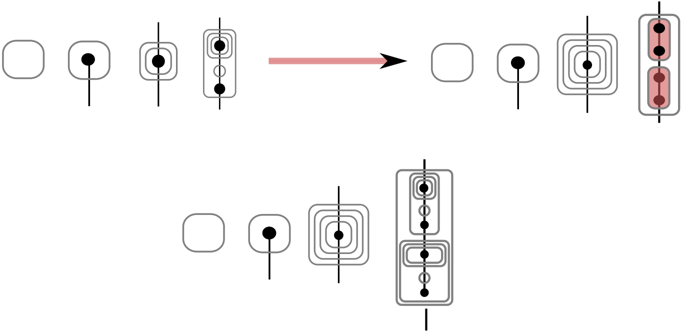
The Opetopes as a Substitution Calculus

- The opetopes come equipped with a natural substitution operation arising from the fact that they are constructors in a polynomial monad



Substitution (cont'd)

- By introducing binding, we can build a rewrite system reminiscent of λ-calculus:



Opetopic Type Theory

The opetopes provide a natural framework for organizing higher dimensional type-theoretic concepts geometrically:

Dimension	Terms
0	Contexts
1	Types
2	Proofs
3	Proofs w/ Metavariables
...	...