Type Theory and the Opetopes

HDACT - Ljubljana

Eric Finster

June 20, 2012
Outline

1. What are the Opetopes?
2. Formalizing the Definition
3. Notation and Implementation
4. The Opetopes and Type Theory
Definitions of higher categories typically begin with the selection of a shape to represent higher dimensional cells:

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

- We’ve got the simplicial category $\Delta^{op}$:

- But there’s also the category of opetopes $O$:
The two main principles 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.

- 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.
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:

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & \cdots \\
\end{array}
\]

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:
Here are some 2-dimensional pasting diagrams:

And an example 3-dimensional cell:
And finally a 3-pasting diagram:
How can we make this intuitive definition precise?

One of the simplest ways to do this (due to Kock, Joyal, Batanin and Mascari) is to realize these shapes as a canonical sequence of polynomial functors.

These have different names in the computer science community: inductive families, indexed containers, indexed $W$-types, . . .
**Definition**

A *polynomial* $P$ is a diagram of sets

\[
\begin{array}{ccc}
E & \xrightarrow{p} & B \\
\downarrow t & & \downarrow r \\
I & & I
\end{array}
\]

Any polynomial determines a functor $\llbracket P \rrbracket : \text{Set}/I \to \text{Set}/I$ (its *extension*) defined for an $I$-Set $X \to I$ by the formula:

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

(Lower subscripts indicate the fibers of appropriate maps.)
It’s useful to represent the elements $b \in B$ as corollas

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

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

$\llbracket P \rrbracket(X) = \begin{cases} \{ \cdots \} \\ b \in B \end{cases}$
Useful Special Cases

Write $1_I$ for the terminal object of $Set/I$. Then it is easily seen that $\llbracket P \rrbracket (1_I) = B$. Graphically:

For the initial object, we have

\[ \llbracket P \rrbracket (\emptyset) = \left\{ b \right\} \]

i.e., the set of constructors with no places.
By iterating the functor, we generate trees: for example, $\llbracket P \rrbracket^2(1_I) = \llbracket P \rrbracket(B)$ is the set of two leveled trees:

\[
\llbracket P \rrbracket^2(1_I) = \{ b_0, b_1, \ldots, b_n \}
\]
Monads

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

\[ \mu_1 : \left[ P \right]^2(1_I) \rightarrow \left[ P \right](1_I) = B \]

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

\[
\begin{array}{c}
\mu(b_0; b_1, \ldots, b_n) \\
\end{array}
\]

- 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 elements 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

\[
\sum_{t \in tr(P)} L(t) \xrightarrow{\pi} tr(P)
\]

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

$$\sum_{t \in tr(P)} N(t) \xrightarrow{\pi} tr(P)$$
Theorem

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

Multiplication is given by *substitution* of trees.
One useful monad is the identity functor on $Set$, represented by the trivial polynomial:

$$
\ast \longrightarrow \ast = \mathcal{O}(1)
$$

$$
\ast \leftarrow \ast = \mathcal{O}(0)
$$

**Definition**

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

Eric Finster

Type Theory and 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 circles and dots in the plane which corresponds to a tree.
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).
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

\begin{itemize}
  \item \begin{tikzpicture}
    \draw[fill=black] (0,0) circle (0.1); 
    \draw[fill=white] (1,0) circle (0.1); 
    \draw[->] (0,0) -- (1,0); 
  \end{tikzpicture}
  \item \begin{tikzpicture}
    \draw[fill=black] (0,0) circle (0.1); 
    \draw[fill=white] (1,0) circle (0.1); 
    \draw[fill=white] (2,0) circle (0.1); 
    \draw[fill=white] (3,0) circle (0.1); 
    \draw[->] (0,0) -- (1,0); 
    \draw[->] (1,0) -- (2,0); 
    \draw[->] (2,0) -- (3,0); 
  \end{tikzpicture}
  \item \begin{tikzpicture}
    \draw[fill=black] (0,0) circle (0.1); 
    \draw[fill=white] (1,0) circle (0.1); 
    \draw[fill=white] (1.5,0) circle (0.1); 
    \draw[fill=white] (2,0) circle (0.1); 
    \draw[->] (0,0) -- (1,0); 
    \draw[->] (1,0) -- (1.5,0); 
    \draw[->] (1.5,0) -- (2,0); 
    \draw[<->] (1,0) -- (1.5,0); 
  \end{tikzpicture}
  \item \begin{tikzpicture}
    \draw[fill=black] (0,0) circle (0.1); 
    \draw[fill=white] (1,0) circle (0.1); 
    \draw[fill=white] (2,0) circle (0.1); 
    \draw[fill=white] (3,0) circle (0.1); 
    \draw[->] (0,0) -- (1,0); 
    \draw[->] (1,0) -- (2,0); 
    \draw[->] (2,0) -- (3,0); 
    \draw[<->] (1,0) -- (2,0); 
  \end{tikzpicture}
\end{itemize}
Notation (cont’d)
Globular shapes are a special case of opetopes:
Opetopes can be represented by the following inductive type:

```haskell
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”
For implementing type-checking, the following “higher-dimensional zipper” is extremely useful:

```haskell
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: \( \Delta^n \)
  2. Boundaries: \( \partial \Delta^n \)
  3. Horns: \( \Lambda^n_k \)

- Opetopic sets have similar notations:
  
  ![Cell](cell.png)  ![Frame](frame.png)  ![Niche](niche.png)

  - Cell
  - Frame
  - Niche
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.
Let $\pi$ denote an arbitrary opetope.

Write $\Gamma, [F : A]_\pi \vdash \cdots$ as shorthand for the assumption of a variable for every face of the frame associated to $\pi$.

Example: for $\pi$ the 2-frame below

```
\begin{array}{ccccc}
  & y & g & z \\
  f & & & \\
  x & k & & w \\
\end{array}
```

we would have

$$\Gamma, x : A, y : A, z : A, w : A, f : \text{Id}_A(x, y), \cdots \vdash \cdots$$

Similarly, $\Gamma, [N : A]_\pi \vdash \cdots$ means enough variables for the faces of the niche associated to $\pi$. 
Opetopic Formation and Introduction

\[ \Gamma \vdash A : Type \]
\[ \Gamma, [F : A]_\pi \vdash Fill(F) : Type \]  \hspace{1cm} \text{\(\mathcal{O}\)-Formation}

\[ \Gamma \vdash [N : A]_\pi \]
\[ \Gamma \vdash \text{comp}(N) : Fill(N|_{\tau(\pi)}) \]  \hspace{1cm} \text{\(\mathcal{O}\)-composition}

\[ \Gamma \vdash [N : A]_\pi \]
\[ \Gamma \vdash \text{refl}(N) : Fill(N \triangleright \text{comp}(N)) \]  \hspace{1cm} \text{\(\mathcal{O}\)-reflection}
When $\pi$ is a glob, it contains a unique top dimensional source face, say $x$, and a new reduction rule says that $\text{comp}(x) \rightarrow x$ in this case.

This corresponds to the slogan “a nullary composition is an isomorphism”
A Generalized J-Rule

- The J-Rule
  \[
  \Gamma, x : A, y : A, f : \text{Id}_A(x, y) \vdash P(x, y, f) : \text{Type} \\
  \Gamma, x : A \vdash p(x) : P(x, x, \text{refl}(x)) \\
  \Gamma \vdash a : A \quad \Gamma \vdash b : A \quad \Gamma \vdash g : \text{Fill}(G) \\
  \Gamma \vdash J(a, b, g) : P(a, b, g)
  \]

- An Opetopic J-Rule:
  \[
  \Gamma, \llbracket F : A \rrbracket_\pi, \alpha : \text{Fill}(F) \vdash P(F, \alpha) : \text{Type} \\
  \Gamma, \llbracket N : A \rrbracket_\pi \vdash p(N) : P(N \triangleright \text{comp}(N), \text{refl}(N)) \\
  \Gamma \vdash \llbracket G : A \rrbracket_\pi \quad \Gamma \vdash \beta : \text{Fill}(G) \\
  \Gamma \vdash J(G, \beta) : P(G, \beta)
  \]
The opetopes come equipped with a natural substitution operation arising from the fact that they are constructors in a polynomial monad.
By introducing binding, we can build a rewrite system reminiscent of λ-calculus:
The opetopes provide a natural framework for organizing higher dimensional type-theoretic concepts geometrically:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Contexts</td>
</tr>
<tr>
<td>1</td>
<td>Types</td>
</tr>
<tr>
<td>2</td>
<td>Proofs</td>
</tr>
<tr>
<td>3</td>
<td>Proofs w/ Metavariables</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>