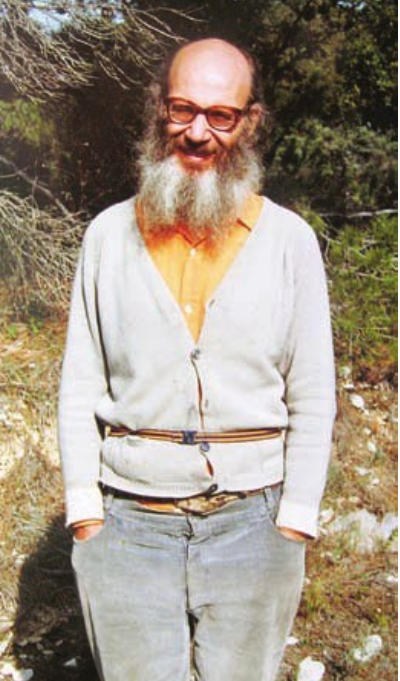
Understanding free ∞-categories

Jamie Vicary
University of Cambridge

(with Christopher Dean, Eric Finster, Ioannis Markakis and David Reutter)

Topos Institute Colloquium
16 September 2021



Alexander Grothendieck (1928–2014)

Creator of modern algebraic geometry, Fields medal 1966.

Obituary in Nature:

His unique skill was to eliminate all unnecessary hypotheses and burrow into an area so deeply that its inner patterns revealed themselves ... he was considered by many the greatest mathematician of the 20th century.

He felt his insights were not always well-received, writing:

A wind of disrepute for any foundational matters is blowing nowadays.

Foundations

Grothendieck was obsessed with finding an axiomatic system for 'well-behaved' topological spaces, avoiding paradoxes like Banach-Tarski.

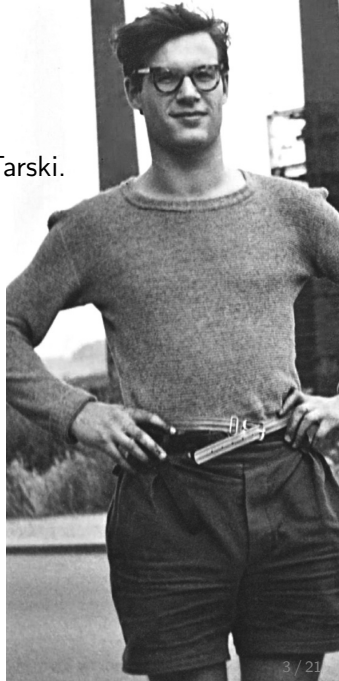
One thing which strikes me ... is the absence of proper foundations for topology itself!

He wrote a famous letter in 1983 to the mathematician Daniel Quillen, where he sketched some possible solutions, but concluded:

One seems caught in an infinite chain of ever messier structures ... one is going to get hopelessly lost, unless one discovers some simple guiding principle.

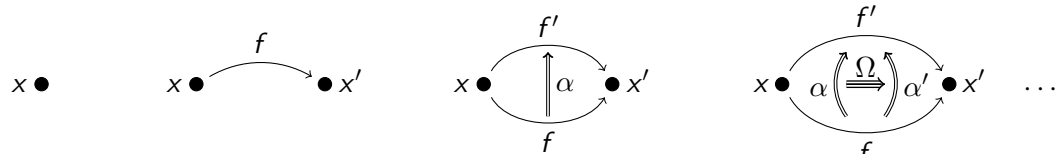
The next day he solved the problem, and wrote in another letter:

I went on pondering ... motivation does furnish a simple guiding principle in order not to get lost in the messiness of higher structures.



Paths as types

Grothendieck's idea begins with n-dimensional balls:



$D_0 := x : \star$

$D_1 := x : \star,\ x' : \star,\ f : x \to y$

$D_2 := x : \star,\ x' : \star,\ f : x \to x',\ f' : x \to x',\ \alpha : f \to x'$

$D_3 := x : \star,\ x' : \star,\ f : x \to x',\ f' : x \to x',\ \alpha : f \to f',\ \alpha' : f \to f',\ \Omega : \alpha \to \alpha'$

The type $\star$ means *point*. An arrow type $S \to T$ means *path*.

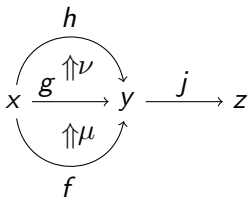
We represent these balls by giving lists of the points and paths in their neighbourhood.

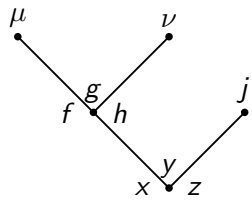
These lists are examples of *computads*.

Gluing balls with trees

Grothendieck suggested "gluing" these balls together, forming geometric *pasting schemes*.

They are represented by *trees* (or *Batanin trees*), and we can also encode them as *lists*.





$x : \star,\ y : \star,\ z : \star,$
$f : x \to y,\ g : x \to y,$
$h : x \to y,\ j : y \to z,$
$\mu : f \to g,\ \nu : g \to h$

We can also define the *boundaries* $\partial^{\pm}$ of a pasting scheme. In this case:

$$\partial^{+} = x \xrightarrow{h} y \xrightarrow{j} z \qquad \partial^{-} = x \xrightarrow{f} y \xrightarrow{j} z$$

Grothendieck wanted a scheme for describing *composites* of these balls.

The boundary principle

Here are Grothendieck's main ideas:

- Given a $\partial^-(\Gamma)$-composite u, and $\partial^+(\Gamma)$-composite v, we get a Γ-composite $\mathrm{coh}(u, v)$.
- Given Γ-composites u, v, we get a Γ-composite $\mathrm{coh}(u, v)$.

$$\partial^+(\Gamma) \qquad x \xrightarrow{h} y \xrightarrow{j} z \qquad h \circ (j \circ \mathrm{id}(z))$$

$$\Gamma \qquad x \underset{f}{\overset{h}{\xrightarrow[\Uparrow\mu]{g\ \Uparrow\nu}}} y \xrightarrow{j} z \qquad \mathrm{coh}(f \circ j, h \circ (j \circ \mathrm{id}(z)))$$

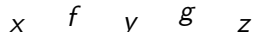
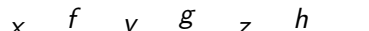
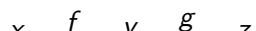
$$\partial^-(\Gamma) \qquad x \xrightarrow{f} y \xrightarrow{j} z \qquad f \circ j$$

There are *side conditions*:

- the given composites must be in the same hom-set (i.e. have the same type)
- the given composites must be "entire", using all the variables of their domains.

These are the *operations* of the theory. The *composites* are obtained by substitution.

Some examples

$\Gamma =$ 	$\mathsf{coh}(x, z)$	"$f \bullet g$"
$\Gamma =$ 	$\mathsf{coh}(\Gamma, x, x)$	"$\mathsf{id}(x)$"
$\Gamma =$ 	$\mathsf{coh}(\Gamma, \mathsf{id}(x) \bullet f, f)$	"λ_f"
$\Gamma =$ 	$\mathsf{coh}(\Gamma, (f \bullet g) \bullet h, f \bullet (g \bullet h))$	"$\alpha_{f,g,h}$"
~~$\Gamma =$~~ 	~~$\mathsf{coh}(\Gamma, y, z)$~~	~~"f^{-1}"~~
~~$\Gamma =$~~ 	~~$\mathsf{coh}(\Gamma, f, f)$~~	

Grothendieck's original letter was ambiguous on how new terms are built from old ones.

The CS perspective gives a clear answer: *by induction.*

Induction, recursion, induction-recursion

Induction allows us to build sets from valid construction rules.

Here is a simple example of two construction rules:	$\star \in S$ Given $p \in S$, then $f(p) \in S$
Then S looks like this:	$S = \{\star, f(\star), f(f(\star)), f(f(f(\star))), \ldots\}$

Recursion is the principle that lets us define functions *out* of an inductive set.

For example, we can define a function $J : S \to \mathbb{N}$ like this:	$J(\star) = 0$ Given $p \in S$, then $J(f(p)) = J(p) + 1$
Then J acts as follows:	$J(\star) = 0, J(f(\star)) = 1, J(f(f(\star)) = 2, \ldots$

Inductive sets are "completely free", and easy for computers to build and work with.

Induction-recursion

Mutual induction allows us to build multiple such sets simultaneously.

For example, consider the following rules:

$\star \in S$
Given $p \in S$, then $f(p) \in S$
Given $p \in S$, then $g(p) \in T$
Given $q \in T$, then $h(q) \in S$

Then $S = \{\star, f(\star), h(g(\star)), f(f(\star)), h(g(f(\star))), \ldots\}$ and $T = \{g(\star), g(f(\star)), g(h(g(\star))), \ldots\}$

Induction-recursion lets us define the inductive sets *and* recursive functions simultaneously.

For example, consider these rules, where again $J : S \to \mathbb{N}$:

$\star \in S$
Given $p \in S$, then $f(p) \in S$
Given $p \in S$ with $J(p) = 0$, then $g(p) \in T$
Given $q \in T$, then $h(q) \in S$
$J(\star) = 0$, $J(f(p)) = J(p) + 1$, $J(h(q)) = 0$

Then we have $S = \{\star, f(\star), h(g(\star)), \ldots\}$ and $T = \{g(\star), g(h(g(\star))), \ldots\}$

This might seem strange, because we're defining J and S in terms of *each other*. But it's OK.

If everything is a set, induction-recursion has the *same power* as induction. *Hancock, McBride, Ghani, Malatesta, Altenkirch (2013)*

Now we're going to play...

FREE STRUCTURES BY INDUCTION

For a chosen algebraic structure, can we build free examples inductively, with structural functions given by recursion?

Monoids. Consider the free monoid over a set $S = \{A, B, C\}$. Its underlying set is given by:

- binary bracketed strings over S, e.g. $(C(A(BB))1))B$,
- *quotiented* by $(XY)Z = X(YZ)$ and $X1 = X = 1X$.

Can we build this quotient inductively? **WIN**, just finite lists in $\{A, B, C\}$.
And monoid multiplication is then a recursive function.

Groups. Consider the free group over $S = \{A, B, C\}$. Its underlying set is given by:

- binary bracketed strings over S, with inverses, e.g. $(C(A^{-1}(BB^{-1}))1))B$,
- *quotiented* by $(XY)Z = X(YZ)$ and $X1 = X = 1X$ and $XX^{-1} = 1 = X^{-1}X$

Can we build this quotient inductively? Lists don't work, as $ABB^{-1}C \neq AC$ as lists.
We need "lists without adjacent inverse pairs". We can build this via *induction-recursion*.
Via HMGMA, we can do this just by induction, so we **WIN**.

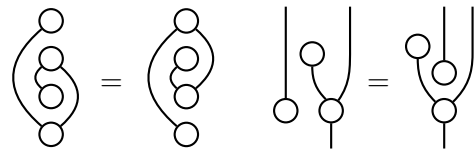
Now we're going to play...

FREE STRUCTURES BY INDUCTION

Monoidal categories. Consider the free monoidal category on some given objects and morphisms.

Its set of objects is just bracketed strings in the generating objects, which is inductive.

But its set of morphisms is the set of string diagrams, quotiented by *homotopy*:



Alternatively, the morphisms are expressions in the formal syntax of monoidal categories, quotiented by the triangle and pentagon equations.

There seems to be no way to inductively construct the quotient space, so we **LOSE**.

Strict monoidal categories. We ... **LOSE**.

Strict ∞-categories. We ... **LOSE**.

Weak ∞-categories. We ... **WIN**! (That's the main point of this talk.)

Side question. How can we determine this for some general structure?

Computads

An *n-computad* is the generating data for a free n-category. Let's see some examples.

$n = 0$. A *0-computad* is a set.

$n = 1$. A *1-computad* is just a directed graph:

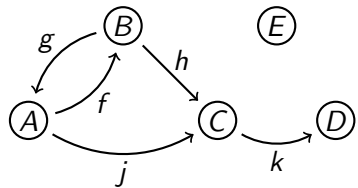
- an underlying 0-computad V (vertices)
- a set E of 1-generators (edges)
- functions $s, t : E \to V$ (source and target)

Given a 1-computad D, we write $\mathrm{Cell}_1(D)$ for its 1-cells, e.g. $(f \circ g) \circ h \in \mathrm{Cell}_1(D)$.

$n = 2$. A *2-computad* is the generating data for a bicategory:

- an underlying 1-computad D (directed graph)
- a set G of 2-generators (2-cells)
- functions $s, t : G \to \mathrm{Cell}_1(D)$ satisfying globularity

We generalize this to all $n \in \mathbb{N}$.



$\mu : j \Rightarrow (f \circ g) \circ f \circ h$

$\nu : \mathrm{id}(E) \Rightarrow \mathrm{id}(E)$

Free ∞-categories inductively

We will build the following structures by *indexed mutual induction-recursion*, for all $n \in \mathbb{N}$:

Categories

- Comp_n of n-computads and n-homomorphisms.

Functors

- $u_n : \mathrm{Comp}_n \to \mathrm{Comp}_{n-1}$, giving the underlying $(n-1)$-computad.
- $\mathrm{Cell}_n : \mathrm{Comp}_n \to \mathrm{Set}$, giving the set of n-cells.
- $\mathrm{Type}_n : \mathrm{Comp}_n \to \mathrm{Set}$, giving the set of n-types.

Natural transformations

- $\mathrm{ty}_n : \mathrm{Cell}_n \to \mathrm{Type}_{n-1} \circ u_n$, giving for every n-cell its underlying $(n-1)$-type.

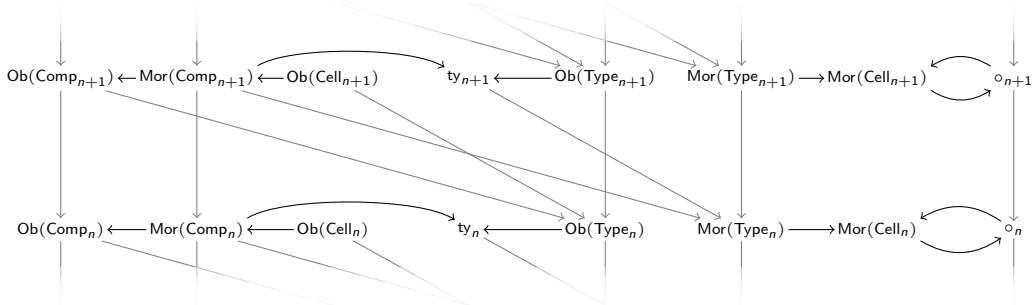
Additional data

- For every tree B, an n-computad t_n^B.
- For every tree B, a distinguished subset of $\mathrm{Type}_n(\mathrm{t}_n^B)$ called the *full types*.

Here all the sets are indexed inductive-recursive, and all the functions are recursive.

Structure of the definition

These sets and functions have a complex dependency pattern that repeats at each level:



DANGER, cyclic dependency between the recursive functions $\mathrm{Mor}(\mathrm{Cell}_{n+1})$ and $\circ_{n+1}$.

So a careful check is required that everything is well-founded.

The definition

Let's look at some parts of the definition in more detail.

Objects. The objects of Comp_{n+1} are triples (C, V, ϕ), called $(n+1)$-*computads*, where:

- $C \in \mathrm{Ob}(\mathrm{Comp}_n)$
- V is a set of *variables*
- $\phi : V \to \mathrm{Type}_n(C)$ is a function assigning to each variable a *type* of C

Morphisms. A morphism of Comp_{n+1} is a pair $(\sigma_0, \sigma_V) : (C, V_C, \phi_C) \to (D, V_D, \phi_D)$ where:

- $\sigma_0 \in \mathrm{Mor}(\mathrm{Comp}_n(C, D))$
- $\sigma_V : V_C \to \mathrm{Cell}_{n+1}(D, V_D, \phi_D)$ such that for all $v \in V_C$ we have

$$\mathrm{ty}_{n+1}(D)(\sigma_V(v)) = \mathrm{Type}_n(\sigma_0)(\phi_C(v))$$

This just says "if $g : a \to b$, then we must have $\sigma(g) : \sigma(u) \to \sigma(v)$".

The definition

Elements of $\mathsf{Cell}_n(C, V, \phi)$ are defined inductively following Grothendieck's scheme.

Cells. For an n-computad (C, V_C, ϕ_C), its set of n-cells $\mathsf{Cell}_n(C, V_C, \phi_C)$ is inductively generated as follows:

- for every element $v \in V_C$, we have an element $\mathsf{var}(v) \in \mathsf{Cell}_n(C, V_C, \phi_C)$.
- for every triple (B, A, σ) where
  - B is a tree with $\dim(B) \leq n$
  - A is a full type in $\mathsf{Type}_{n-1}(\mathsf{t}^B_{n-1})$
  - $\sigma = (\sigma_0, \sigma_v)$ is an n-computad morphism $\mathsf{t}^B_n \to (C, V_C, \phi_C)$.

  we have an element $\mathsf{coh}(B, A, \sigma) \in \mathsf{Cell}_n(C, V_C, \phi_C)$.

This matches the structure of operations in Grothendieck's original theory that we saw earlier:

$\partial^+(\Gamma)$ $\quad x \xrightarrow{h} y \xrightarrow{j} z \quad h \circ (j \circ \mathsf{id}(z))$

Γ $\quad x \xrightarrow[f]{h \;\Downarrow g\nu\mu} y \xrightarrow{j} z$

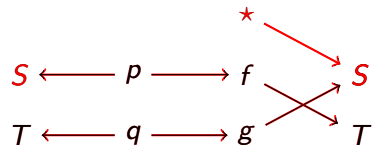
$\partial^-(\Gamma)$ $\quad x \xrightarrow{f} y \xrightarrow{j} z \quad f \circ j$

Polynomial functors

Recall our simple mutually inductive definition from earlier:

$$\star \in S \star \in S \quad p \in S \Rightarrow f(p) \in T \quad q \in T \Rightarrow g(q) \in S q \in T \Rightarrow g(q) \in S p \in S \Rightarrow f(p) \in T \quad q$$

We can encode this in a nice way, as follows:



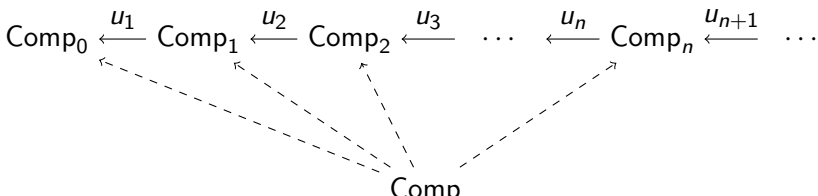
Such a triple of functions is called a *polynomial endofunctor*:

$$A \xleftarrow{f} B \xrightarrow{g} C \xrightarrow{h} A$$

Question. Our definition is "large indexed inductive-recursive".
Can we remove the induction-recursion and express it as a polynomial functor?

Computads

Definition. The category Comp of computads is the limit in Cat of the following diagram:



A *computad* therefore comprises a choice for each n of an n-computad C_n, with the property that $C_{n+1} = (C_n, V_n^C, \phi_n^C)$ for some variable sets V_n^C and typing functions ϕ_n^C.

A computad may have nontrivial structure in every dimension.

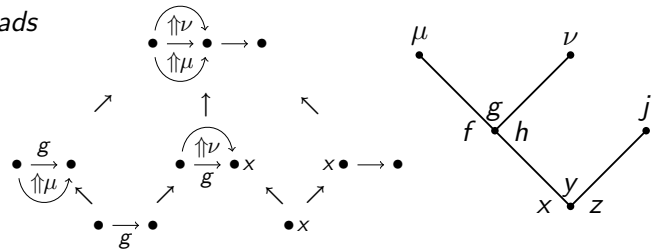
General ∞-categories

Definition. The full subcategory $\mathrm{Comp}_t \subset \mathrm{Comp}$ has tree computads as objects.

Theorem. *In* Comp_t*, tree computads are colimits of the leaf disks ("globular sums").*

Definition. An ∞-*category* is a presheaf $(\mathrm{Comp}_t)^{\mathrm{op}} \to \mathrm{Set}$ preserving globular sums.

This gives a category $\infty\mathrm{Cat}$.



Known to agree with the definition of *contractible* ∞-*category* (Grothendieck, Maltsiniotis, Batanin, Leinster, Brunerie) via recent work of Ara, Bourke and Benjamin.

This is a lightweight approach:

- no globular extension technology (Grothendieck/Maltsiniotis)
- no globular operad technology (Batanin/Leinster)

Equivalence

We saw earlier that we can describe the elements of a free monoid inductively.

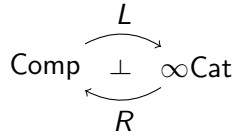
. . . but not every monoid is free, so this is of limited usefulness!

We've also seen that the cells of a free ∞-category can be described inductively. . .

. . . the difference is that *every* ∞-category is equivalent to a free one.

So in some sense, Comp represents the *entire theory* of ∞-categories.

There is an adjunction as follows, where
L constructs the free ∞-category on a computad, and
R constructs the associated computad of the ∞-category:



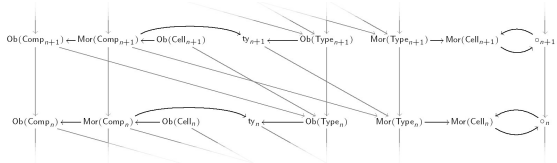
For an ∞-category C, the computad $R(C)$ has all the cells of C as generators.

The ∞-category $L(R(C))$ is a *cofibrant replacement* of C, with some nicer properties.

Richard Garner has a great paper on n-computads, and our explicit construction fits in perfectly with his theory (arXiv:0810.4450).

Summary

Free ∞-categories can be constructed via *indexed induction-recursion*.



Since every ∞-category is equivalent to a free one, these computads give a *universal model* for ∞-categories.

They are more-or-less a direct encoding of Grothendieck's original idea.

Further questions:

- What free algebraic structures admit inductive presentations?
- Can the induction-recursion be removed in this case, giving a polynomial functor?
- What about other models (e.g. cubical)? (Work in progress.)

Thanks for listening!