Ontological Logs, Higher Categories

and a path to programmatic representation

Noah Chrein

April 18, 2019

Part 1: Intuitions and Definitions

- Ologs
- $\mathfrak{C}at \implies (\mathfrak{C}at \downarrow \mathfrak{C}) \implies sm(\mathfrak{C})$
- Colimits / Coverings
- Anisotropy / monoidial \$
- $\mathfrak{G}$ / $\oplus$
- Site / Ontological Generation

Part 2: $sm : \mathfrak{Cat} \to \mathfrak{Cat}$, 2-categories

- $\mathfrak{Cat}$ as a category, sm, $Fun : \mathfrak{Cat} \to \mathfrak{Cat}$
- $\Delta : Id_{\mathfrak{Cat}} \to Fun$
- $colim : Fun(\mathfrak{C}) \to \mathfrak{C}$
- $\eta : Id \to \Delta \circ colim$ as initial in $\mathscr{L}_{fun,\mathfrak{C}}$

Part 3: Higher Categories: Simplicial sets, Python Ologs

- $\mathfrak{C}at$ as a 2-category
- Simplicial Sets
- Functorality as Naturality
- Pythonic Simplicial Sets
- Face and Degeneracy as ontological expansions

Ologs

Ontological Log

An **Ontological Log** is a labeled category.

- I.e. labeled objects and labeled morphisms
- An ontology represents concepts and their relations via category theory

Ontological Expansions

- An ontology itself is a category of concepts

Ontological Expansions

- An ontology itself is a category of concepts
- An ontology can also represent the subconcepts of a single concept.

Ontological Expansions

- An ontology itself is a category of concepts
- An ontology can also represent the subconcepts of a single concept.
- We want to obtain the ontologies describing single concepts.

Ontological Expansions

- An ontology itself is a category of concepts
- An ontology can also represent the subconcepts of a single concept.
- We want to obtain the ontologies describing single concepts.

This is formalized by an **ontological expansion**

Ontological Expansions

- The Naive guess is a functor $O : \mathfrak{C} \to \mathfrak{C}at$
  (Doesn't tell us how to relate subconcepts in O(c) to O(c'))

Ontological Expansions

- The Naive guess is a functor $O : \mathfrak{C} \to \mathfrak{C}at$
  (Doesn't tell us how to relate subconcepts in O(c) to O(c'))
- Another guess is that there is some "universe" category $\mathfrak{D}$ with $O : \mathfrak{C} \to (\mathfrak{C}at \downarrow \mathfrak{D})$
  (Gives relations between subconcepts, but naturality is too restrictive)

Ontological Expansions

- The Naive guess is a functor $O : \mathfrak{C} \to \mathfrak{C}at$
  (Doesn't tell us how to relate subconcepts in O(c) to O(c'))
- Another guess is that there is some "universe" category $\mathfrak{D}$
  with $O : \mathfrak{C} \to (\mathfrak{C}at \downarrow \mathfrak{D})$
  (Gives relations between subconcepts, but naturality is too restrictive)

removing the naturality conditions the morphisms of $(\mathfrak{C}at \downarrow \mathfrak{D})$ are collections of morphisms between images of small functors J,J':

Ontological Expansions

- The Naive guess is a functor $O : \mathfrak{C} \to \mathfrak{C}at$
  (Doesn't tell us how to relate subconcepts in O(c) to O(c'))
- Another guess is that there is some "universe" category $\mathfrak{D}$ with $O : \mathfrak{C} \to (\mathfrak{C}at \downarrow \mathfrak{D})$
  (Gives relations between subconcepts, but naturality is too restrictive)

removing the naturality conditions the morphisms of $(\mathfrak{C}at \downarrow \mathfrak{D})$ are collections of morphisms between images of small functors J,J':

Submorphism

A **submorphism** $\mathcal{F} : J \to J'$ is a set of maps between the images of J and J'

Ontological Expansions

This forms a category of small functors $J : S \to \mathfrak{D}$ and submorphisms, $sm(\mathfrak{D})$.

Ontological Expansions

This forms a category of small functors $J : S \to \mathfrak{D}$ and submorphisms, $sm(\mathfrak{D})$.
This construction is actually functorial $sm : \mathfrak{Cat} \to \mathfrak{Cat}$

Ontological Expansions

This forms a category of small functors $J : S \to \mathfrak{D}$ and submorphisms, $sm(\mathfrak{D})$.
This construction is actually functorial $sm : \mathfrak{Cat} \to \mathfrak{Cat}$

Ontological Expansion

an **Ontological Expansion** is a functor $O : \mathfrak{C} \to sm(\mathfrak{D})$

Colimits and Coverings

- For a cocomplete category $\mathfrak{C}$, the colimit forms a functor $colim : (\mathfrak{C}at \downarrow \mathfrak{C}) \to \mathfrak{C}$

Colimits and Coverings

- For a cocomplete category $\mathfrak{C}$, the colimit forms a functor $colim : (\mathfrak{C}at \downarrow \mathfrak{C}) \to \mathfrak{C}$
- The colimit also comes with a collection of "canonical morphisms" $\{\eta_s : J(s) \to colim(J)\}$

Colimits and Coverings

- For a cocomplete category $\mathfrak{C}$, the colimit forms a functor $colim : (\mathfrak{C}at \downarrow \mathfrak{C}) \to \mathfrak{C}$
- The colimit also comes with a collection of "canonical morphisms" $\{\eta_s : J(s) \to colim(J)\}$
- we want an analog for the colimit in $sm(\mathfrak{C})$, i.e. a functor $colim : sm(\mathfrak{C}) \to \mathfrak{C}$ with a collection $\{\eta_s : J(s) \to colim(J)\}$

Colimits and Coverings

- For a cocomplete category $\mathfrak{C}$, the colimit forms a functor $colim : (\mathfrak{Cat} \downarrow \mathfrak{C}) \to \mathfrak{C}$
- The colimit also comes with a collection of "canonical morphisms" $\{\eta_s : J(s) \to colim(J)\}$
- we want an analog for the colimit in $sm(\mathfrak{C})$, i.e. a functor $colim : sm(\mathfrak{C}) \to \mathfrak{C}$ with a collection $\{\eta_s : J(s) \to colim(J)\}$
- the **intuition** behind the functorality of colim, is that we should be able to deduce relations between concepts from the relations of their ontological expansion (i.e. colim(submorphism) = morphism)

Colimits and Coverings

- For a cocomplete category $\mathfrak{C}$, the colimit forms a functor $colim : (\mathfrak{C}at \downarrow \mathfrak{C}) \to \mathfrak{C}$
- The colimit also comes with a collection of "canonical morphisms" $\{\eta_s : J(s) \to colim(J)\}$
- we want an analog for the colimit in $sm(\mathfrak{C})$, i.e. a functor $colim : sm(\mathfrak{C}) \to \mathfrak{C}$ with a collection $\{\eta_s : J(s) \to colim(J)\}$
- the **intuition** behind the functorality of colim, is that we should be able to deduce relations between concepts from the relations of their ontological expansion (i.e. colim(submorphism) $=$ morphism)

in this case, for an ontological expansion $O : \mathfrak{C} \to sm(\mathfrak{D})$ we then have a covering $\{\eta_s : O(c)(s) \to colim(O(c))\}$

Colimits and Coverings, First OG assumption

If we make the further assumption that:

$$O : \mathfrak{C} \to sm(\mathfrak{C}) \text{ and } colim(O(c)) = c$$

Colimits and Coverings, First OG assumption

If we make the further assumption that:

$$O : \mathfrak{C} \to sm(\mathfrak{C}) \text{ and } colim(O(c)) = c$$

we get a covering $\{O(c)(s) \to c\}$ of c by its expansion

Colimits and Coverings, First OG assumption

If we make the further assumption that:

$$O : \mathfrak{C} \to sm(\mathfrak{C}) \text{ and } colim(O(c)) = c$$

we get a covering $\{O(c)(s) \to c\}$ of c by its expansion

- The **intuition** behind these assertions is that an ontological expansion O(c) actually describes c

Colimits and Coverings, First OG assumption

If we make the further assumption that:

$$O : \mathfrak{C} \to sm(\mathfrak{C}) \text{ and } colim(O(c)) = c$$

we get a covering $\{O(c)(s) \to c\}$ of c by its expansion

- The **intuition** behind these assertions is that an ontological expansion O(c) actually describes c

This assertion is the first condition of an **Ontological Generator**

Anisotropy and $

- Ontological Expansion tells us how to construct ontologies that describe a given concept

Anisotropy and $

- Ontological Expansion tells us how to construct ontologies that describe a given concept
- In general, there is more than one ontological expansion

Anisotropy and \$

- Ontological Expansion tells us how to construct ontologies that describe a given concept
- In general, there is more than one ontological expansion

For example, we can describe a cat by
O(cat) = macroscopic body parts, or
O'(cat) = cellular anatomy

Anisotropy and $

- Ontological Expansion tells us how to construct ontologies that describe a given concept
- In general, there is more than one ontological expansion

For example, we can describe a cat by
O(cat) $=$ macroscopic body parts, or
O'(cat) $=$ cellular anatomy

Goal

To organize different expansions and how they relate

Anisotropy and $

- The idea is that these ontological expansion functors are actually part of a parameterized functor OG_s, which we will call an **ontological generator**

Anisotropy and \$

- The idea is that these ontological expansion functors are actually part of a parameterized functor OG_s, which we will call an **ontological generator**

The **intuition** is that different ontological expansions all describe the objects they expand, but may be more costly
(cellular anatomy is much more data than macroscopic body parts)

Anisotropy and \$

- The idea is that these ontological expansion functors are actually part of a parameterized functor OG_s, which we will call an **ontological generator**

The **intuition** is that different ontological expansions all describe the objects they expand, but may be more costly
(cellular anatomy is much more data than macroscopic body parts)

- We also want to relate two ontological expansions

So our proto-definition of an ontological generator is now a functor

$$OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$$

Operations on Ontological Expansions: ໑

- recall that $sm : \mathfrak{C}at \to \mathfrak{C}at$ is a functor
- given $O : \mathfrak{C} \to sm(\mathfrak{C})$, $sm(O) : sm(\mathfrak{C}) \to sm(sm(\mathfrak{C}))$

Operations on Ontological Expansions: 9

- recall that $sm : \mathfrak{Cat} \to \mathfrak{Cat}$ is a functor
- given $O : \mathfrak{C} \to sm(\mathfrak{C})$, $sm(O) : sm(\mathfrak{C}) \to sm(sm(\mathfrak{C}))$
- also $colim : sm(sm(\mathfrak{C})) \to sm(\mathfrak{C})$

Operations on Ontological Expansions: ໑

- recall that $sm : \mathfrak{C}at \to \mathfrak{C}at$ is a functor
- given $O : \mathfrak{C} \to sm(\mathfrak{C})$, $sm(O) : sm(\mathfrak{C}) \to sm(sm(\mathfrak{C}))$
- also $colim : sm(sm(\mathfrak{C})) \to sm(\mathfrak{C})$

given two ontological expansions $O, O' : \mathfrak{C} \to sm(\mathfrak{C})$ we can create a composable chain:

$$\mathfrak{C} \xrightarrow{O} sm(\mathfrak{C}) \xrightarrow{sm(O')} sm(sm(\mathfrak{C})) \xrightarrow{colim} sm(\mathfrak{C})$$

Operations on Ontological Expansions: ⅁

- recall that $sm : \mathfrak{Cat} \to \mathfrak{Cat}$ is a functor
- given $O : \mathfrak{C} \to sm(\mathfrak{C})$, $sm(O) : sm(\mathfrak{C}) \to sm(sm(\mathfrak{C}))$
- also $colim : sm(sm(\mathfrak{C})) \to sm(\mathfrak{C})$

given two ontological expansions $O, O' : \mathfrak{C} \to sm(\mathfrak{C})$ we can create a composable chain:

$$\mathfrak{C} \xrightarrow{O} sm(\mathfrak{C}) \xrightarrow{sm(O')} sm(sm(\mathfrak{C})) \xrightarrow{colim} sm(\mathfrak{C})$$

This composition yeilds the **spiral product**

$$O' \mathbin{⅁} O = colim \circ sm(O') \circ O$$

Operations on Ontological Expansions: ୭

$$O' \text{୭} O = colim \circ sm(O') \circ O$$

The **intuition** behind the spiral product is that we are aggregating two ontological expansions into one big one:

- First expand an object c to an ontology O(c)

Operations on Ontological Expansions: ໑

$$O' \text{໑} O = colim \circ sm(O') \circ O$$

The **intuition** behind the spiral product is that we are aggregating two ontological expansions into one big one:

- First expand an object c to an ontology O(c)
- Then expand the objects c' of O(c) to an ontologies O'(c')

Operations on Ontological Expansions: ໑

$$O' \,໑\, O = colim \circ sm(O') \circ O$$

The **intuition** behind the spiral product is that we are aggregating two ontological expansions into one big one:

- First expand an object c to an ontology O(c)
- Then expand the objects c' of O(c) to an ontologies O'(c')
- Finally collect ontologies into one large on that "looks like"

$$\bigcup_{c' \in O(c)} O'(c')$$

(careful, in general the spiral product isn't just an ordinary union)

Monoidial Category $, Second OG assumption

consider a (proto-)ontological generator $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$

Monoidial Category \$, Second OG assumption

consider a (proto-)ontological generator $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$
What we now want is a (non-abelian) monoidial product $s \otimes s'$ on \$ that helps us organize subsequent spiral products.

Monoidial Category \$, Second OG assumption

consider a (proto-)ontological generator $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$
What we now want is a (non-abelian) monoidial product $s \otimes s'$ on \$ that helps us organize subsequent spiral products.
i.e. $\otimes$ satisfies the equation

$$OG(s' \otimes s) = OG(s) @ OG(s')$$

Monoidial Category \$, Second OG assumption

consider a (proto-)ontological generator $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$
What we now want is a (non-abelian) monoidial product $s \otimes s'$ on \$ that helps us organize subsequent spiral products.
i.e. $\otimes$ satisfies the equation

$$OG(s' \otimes s) = OG(s) \circlearrowleft OG(s')$$

In this case:
a) OG is closed under the spiral product

Monoidial Category \$, Second OG assumption

consider a (proto-)ontological generator $OG : \$ \rightarrow sm(\mathfrak{C})^{\mathfrak{C}}$
What we now want is a (non-abelian) monoidial product $s \otimes s'$ on \$ that helps us organize subsequent spiral products.
i.e. $\otimes$ satisfies the equation

$$OG(s' \otimes s) = OG(s) \mathbin{\text{๑}} OG(s')$$

In this case:
a) OG is closed under the spiral product
b) \$ tells us how to compose ontological expansions in OG

Monoidial Category \$, Second OG assumption

consider a (proto-)ontological generator $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$
What we now want is a (non-abelian) monoidial product $s \otimes s'$ on \$ that helps us organize subsequent spiral products.
i.e. $\otimes$ satisfies the equation

$$OG(s' \otimes s) = OG(s) \mathfrak{S} OG(s')$$

In this case:
a) OG is closed under the spiral product
b) \$ tells us how to compose ontological expansions in OG

Site, Final OG assumption

Consider an $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ satisfying $\mathrm{colim}(\mathrm{OG}_s(c)) = c$

Site, Final OG assumption

Consider an $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ satisfying $\mathrm{colim}(\mathrm{OG}_s(c)) = c$
Let $Cov = \{OG_s(c) \to c\}_{s \in \$, c \in \mathfrak{C}}$

Site, Final OG assumption

Consider an $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ satisfying $\mathrm{colim}(\mathrm{OG}_s(c)) = c$
Let $Cov = \{OG_s(c) \to c\}_{s \in \$, c \in \mathfrak{C}}$
The final assumption is that Cov makes $\mathfrak{C}$ into a **site**

Site, Final OG assumption

Consider an $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ satisfying $\mathrm{colim}(\mathrm{OG}_s(c)) = c$
Let $Cov = \{OG_s(c) \to c\}_{s \in \$, c \in \mathfrak{C}}$
The final assumption is that Cov makes $\mathfrak{C}$ into a **site**

intuition

We want to actually measure data about the objects of $\mathfrak{C}$. The site assumption allows us to formalize data in terms of sheaves $\mathscr{F} : \mathfrak{C} \to \mathfrak{D}$.

Site, Final OG assumption

Consider an $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ satisfying $\mathrm{colim}(\mathrm{OG}_s(c)) = c$
Let $Cov = \{OG_s(c) \to c\}_{s \in \$, c \in \mathfrak{C}}$
The final assumption is that Cov makes $\mathfrak{C}$ into a **site**

intuition

We want to actually measure data about the objects of $\mathfrak{C}$. The site assumption allows us to formalize data in terms of sheaves $\mathscr{F} : \mathfrak{C} \to \mathfrak{D}$.

Moreover given an ontological expansion $OG_s(c)$, we can use the sheaf condition to "glue together" data from $c' \in OG_s(c)$ to c-data

Definition: Ontological Generators

Let $\mathfrak{C}$ be a small category whose $sm(\mathfrak{C})$ admits a colimit, and $\mathfrak{S}$ be a small monoidial category with a terminal object.

Definition: Ontological Generators

Let $\mathfrak{C}$ be a small category whose $sm(\mathfrak{C})$ admits a colimit, and $\$$ be a small monoidial category with a terminal object.

Ontological Generator

An **ontological generator** is a functor $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ such that:

Definition: Ontological Generators

Let $\mathfrak{C}$ be a small category whose $sm(\mathfrak{C})$ admits a colimit, and $\$$ be a small monoidial category with a terminal object.

Ontological Generator

An **ontological generator** is a functor $OG : \$ \rightarrow sm(\mathfrak{C})^{\mathfrak{C}}$ such that:
1)$colim \circ OG(s) = Id_{\mathfrak{C}}, \forall s \in \$$

Definition: Ontological Generators

Let $\mathfrak{C}$ be a small category whose $sm(\mathfrak{C})$ admits a colimit, and $\$$ be a small monoidial category with a terminal object.

Ontological Generator

An **ontological generator** is a functor $OG : \$ \rightarrow sm(\mathfrak{C})^{\mathfrak{C}}$ such that:
1)$colim \circ OG(s) = Id_{\mathfrak{C}}, \forall s \in \$$
2)$OG(s' \otimes s) = OG(s') \circledcirc OG(s)$

Definition: Ontological Generators

Let $\mathfrak{C}$ be a small category whose $sm(\mathfrak{C})$ admits a colimit, and $\$$ be a small monoidial category with a terminal object.

Ontological Generator

An **ontological generator** is a functor $OG : \$ \to sm(\mathfrak{C})^{\mathfrak{C}}$ such that:
1)$colim \circ OG(s) = Id_{\mathfrak{C}}, \forall s \in \$$
2)$OG(s' \otimes s) = OG(s') \wp OG(s)$
3)$Cov = \{OG(s)(c) \to c\}_{s \in \$, c \in \mathfrak{C}}$ makes $\mathfrak{C}$ into a site

Part 2: $sm : \mathfrak{C}at \to \mathfrak{C}at$ and colim

The point of this section is to recast the colimit as an initial object in some category

Part 2: $sm : \mathfrak{Cat} \to \mathfrak{Cat}$ and colim

The point of this section is to recast the colimit as an initial object in some category

- $\mathfrak{Cat}$ as a category, $sm, fun : \mathfrak{Cat} \to \mathfrak{Cat}$
- $\Delta : Id_{\mathfrak{Cat}} \to fun$
- $colim : fun(\mathfrak{C}) \to \mathfrak{C}$
- $\eta : Id \to \Delta \circ colim$ is initial in $\mathscr{L}_{fun,\mathfrak{C}}$

Part 2: $sm : \mathfrak{Cat} \to \mathfrak{Cat}$ and colim

The point of this section is to recast the colimit as an initial object in some category

- $\mathfrak{Cat}$ as a category, $sm, fun : \mathfrak{Cat} \to \mathfrak{Cat}$
- $\Delta : Id_{\mathfrak{Cat}} \to fun$
- $colim : fun(\mathfrak{C}) \to \mathfrak{C}$
- $\eta : Id \to \Delta \circ colim$ is initial in $\mathscr{L}_{fun,\mathfrak{C}}$
- Conjecture: $\eta : Id \to \Delta \circ colim$ is initial in $\mathscr{L}_{sm,\mathfrak{C}}$

$(\mathfrak{C}at \downarrow \mathfrak{C})$

- The category of small categories is actually a category itself

$(\mathfrak{C}at \downarrow \mathfrak{C})$

- The category of small categories is actually a category itself
- The objects are small categories $\mathfrak{C}, \mathfrak{D}$
- The morphisms are functors $F : \mathfrak{C} \to \mathfrak{D}$

$(\mathfrak{Cat} \downarrow \mathfrak{C})$

- The category of small categories is actually a category itself
- The objects are small categories $\mathfrak{C}, \mathfrak{D}$
- The morphisms are functors $F : \mathfrak{C} \to \mathfrak{D}$

From a small category $\mathfrak{C}$ we can form the overcategory $(\mathfrak{C}at \downarrow \mathfrak{C})$:

$(\mathfrak{Cat} \downarrow \mathfrak{C})$

- The category of small categories is actually a category itself
- The objects are small categories $\mathfrak{C}, \mathfrak{D}$
- The morphisms are functors $F : \mathfrak{C} \to \mathfrak{D}$

From a small category $\mathfrak{C}$ we can form the overcategory $(\mathfrak{Cat} \downarrow \mathfrak{C})$:

- objects are functors $J : S \to \mathfrak{C}$
- morphisms are pairs $(P : S \to S', \phi : J \to J' \circ P)$

$(\mathfrak{C}at \downarrow \mathfrak{C})$

- The category of small categories is actually a category itself
- The objects are small categories $\mathfrak{C}, \mathfrak{D}$
- The morphisms are functors $F : \mathfrak{C} \to \mathfrak{D}$

From a small category $\mathfrak{C}$ we can form the overcategory $(\mathfrak{C}at \downarrow \mathfrak{C})$:

- objects are functors $J : S \to \mathfrak{C}$
- morphisms are pairs $(P : S \to S', \phi : J \to J' \circ P)$

P is a functor and ϕ a natural transformation as in the diagram:

$$\begin{tikzcd}
S \arrow[rr, "P"] \arrow[dr, "J"'] \arrow[rr, phantom, shift right=4ex, "\overset{\phi}{\Rightarrow}"] & & S' \arrow[dl, "J'"] \\
& \mathfrak{C} &
\end{tikzcd}$$

$fun : \mathfrak{Cat} \to \mathfrak{Cat}$

Lets consider the functor $fun : \mathfrak{Cat} \to \mathfrak{Cat}$

$fun : \mathfrak{Cat} \to \mathfrak{Cat}$

Lets consider the functor $fun : \mathfrak{Cat} \to \mathfrak{Cat}$

- $fun(\mathfrak{C}) = (\mathfrak{Cat} \downarrow \mathfrak{C})$
- $fun(F : \mathfrak{C} \to \mathfrak{D}) : (\mathfrak{Cat} \downarrow \mathfrak{C}) \to (\mathfrak{Cat} \downarrow \mathfrak{D})$

$\mathit{fun} : \mathfrak{Cat} \to \mathfrak{Cat}$

Lets consider the functor $\mathit{fun} : \mathfrak{Cat} \to \mathfrak{Cat}$

- $\mathit{fun}(\mathfrak{C}) = (\mathfrak{Cat} \downarrow \mathfrak{C})$
- $\mathit{fun}(F : \mathfrak{C} \to \mathfrak{D}) : (\mathfrak{Cat} \downarrow \mathfrak{C}) \to (\mathfrak{Cat} \downarrow \mathfrak{D})$

That is, $\mathit{fun}(F)$ is a functor:

- for $J : S \to \mathfrak{C}$, $\mathit{fun}(F)(J) = F \circ J : S \to \mathfrak{D}$
- for $(P, \phi : J \to J' \circ P)$, $\mathit{fun}(F)(P, \phi) = (P, F(\phi))$

$\mathit{fun} : \mathfrak{Cat} \to \mathfrak{Cat}$

Lets consider the functor $\mathit{fun} : \mathfrak{Cat} \to \mathfrak{Cat}$

- $\mathit{fun}(\mathfrak{C}) = (\mathfrak{Cat} \downarrow \mathfrak{C})$
- $\mathit{fun}(F : \mathfrak{C} \to \mathfrak{D}) : (\mathfrak{Cat} \downarrow \mathfrak{C}) \to (\mathfrak{Cat} \downarrow \mathfrak{D})$

That is, $\mathit{fun}(F)$ is a functor:

- for $J : S \to \mathfrak{C}$, $\mathit{fun}(F)(J) = F \circ J : S \to \mathfrak{D}$
- for $(P, \phi : J \to J' \circ P)$, $\mathit{fun}(F)(P, \phi) = (P, F(\phi))$

$$
\mathrm{fun(F)}\left(
\begin{tikzcd}
S \arrow[rr, "P"] \arrow[dr, "J"'] \arrow[rr, phantom, shift right=4, "\overset{\phi}{\Rightarrow}"] & & S' \arrow[dl, "J'"] \\
& \mathfrak{C} &
\end{tikzcd}
\right) =
\begin{tikzcd}
S \arrow[rr, "P"] \arrow[dr, "F\circ J"'] \arrow[rr, phantom, shift right=4, "\overset{F(\phi)}{\Rightarrow}"] & & S' \arrow[dl, "F\circ J'"] \\
& \mathfrak{D} &
\end{tikzcd}
$$

$\Delta : Id \to Fun$

- Let $\mathfrak{C}$ be a small category and $*$ the terminal (one-point) category

$\Delta : \mathit{Id} \to \mathit{Fun}$

- Let $\mathfrak{C}$ be a small category and $*$ the terminal (one-point) category
- for $X \in \mathfrak{C}$, define the small functor $\Delta_{\mathfrak{C}}(X) : * \to \mathfrak{C}$ by

$$\Delta_{\mathfrak{C}}(X)(*) = X$$

$\Delta : Id \to Fun$

- Let $\mathfrak{C}$ be a small category and $*$ the terminal (one-point) category
- for $X \in \mathfrak{C}$, define the small functor $\Delta_{\mathfrak{C}}(X) : * \to \mathfrak{C}$ by

$$\Delta_{\mathfrak{C}}(X)(*) = X$$

- this is actually a functor $\Delta_{\mathfrak{C}} : \mathfrak{C} \to fun(\mathfrak{C})$

$\Delta : Id \to Fun$

- Let $\mathfrak{C}$ be a small category and $*$ the terminal (one-point) category
- for $X \in \mathfrak{C}$, define the small functor $\Delta_{\mathfrak{C}}(X) : * \to \mathfrak{C}$ by

$$\Delta_{\mathfrak{C}}(X)(*) = X$$

- this is actually a functor $\Delta_{\mathfrak{C}} : \mathfrak{C} \to fun(\mathfrak{C})$

Consider the identity functor $Id_{\mathfrak{C}at}$. Both fun and Id are endofunctors of the category $\mathfrak{C}at$

- Δ is a natural transformation $\Delta : Id_{\mathfrak{C}at} \to fun$.

Colimit as a natural transformation

- Let $L : fun(\mathfrak{C}) \to \mathfrak{C}$ a functor

Colimit as a natural transformation

- Let $L : fun(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$

Colimit as a natural transformation

- Let $L : \mathit{fun}(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$
- therefore $\Delta(L(J)) : * \to \mathfrak{C} \in \mathit{fun}(\mathfrak{C})$

Colimit as a natural transformation

- Let $L : \mathit{fun}(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$
- therefore $\Delta(L(J)) : * \to \mathfrak{C} \in \mathit{fun}(\mathfrak{C})$
- hence $\Delta \circ L$ is an endofunctor of $\mathit{fun}(\mathfrak{C})$

Colimit as a natural transformation

- Let $L : fun(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$
- therefore $\Delta(L(J)) : * \to \mathfrak{C} \in fun(\mathfrak{C})$
- hence $\Delta \circ L$ is an endofunctor of $fun(\mathfrak{C})$
- consider a natural transformation $l : Id_{fun(\mathfrak{C})} \to \Delta \circ L$

Colimit as a natural transformation

- Let $L : \mathit{fun}(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$
- therefore $\Delta(L(J)) : * \to \mathfrak{C} \in \mathit{fun}(\mathfrak{C})$
- hence $\Delta \circ L$ is an endofunctor of $\mathit{fun}(\mathfrak{C})$
- consider a natural transformation $l : Id_{\mathit{fun}(\mathfrak{C})} \to \Delta \circ L$

$l_J : J \to \Delta_{\mathfrak{C}} \circ L(J)$ is then a morphism in $\mathit{fun}(\mathfrak{C})$
i.e. a collection of maps $l_{J,s} : J(s) \to L(J)$

Colimit as a natural transformation

- Let $L : fun(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$
- therefore $\Delta(L(J)) : * \to \mathfrak{C} \in fun(\mathfrak{C})$
- hence $\Delta \circ L$ is an endofunctor of $fun(\mathfrak{C})$
- consider a natural transformation $l : Id_{fun(\mathfrak{C})} \to \Delta \circ L$

$l_J : J \to \Delta_{\mathfrak{C}} \circ L(J)$ is then a morphism in $fun(\mathfrak{C})$
i.e. a collection of maps $l_{J,s} : J(s) \to L(J)$

If $\mathfrak{C}$ is cocomplete, colimit becomes a functor $colim : fun(\mathfrak{C}) \to \mathfrak{C}$

Colimit as a natural transformation

- Let $L : fun(\mathfrak{C}) \to \mathfrak{C}$ a functor
- for $J : S \to \mathfrak{C}$, $L(J) \in \mathfrak{C}$
- therefore $\Delta(L(J)) : * \to \mathfrak{C} \in fun(\mathfrak{C})$
- hence $\Delta \circ L$ is an endofunctor of $fun(\mathfrak{C})$
- consider a natural transformation $l : Id_{fun(\mathfrak{C})} \to \Delta \circ L$

$l_J : J \to \Delta_{\mathfrak{C}} \circ L(J)$ is then a morphism in $fun(\mathfrak{C})$
i.e. a collection of maps $l_{J,s} : J(s) \to L(J)$

If $\mathfrak{C}$ is cocomplete, colimit becomes a functor $colim : fun(\mathfrak{C}) \to \mathfrak{C}$
The canonical morphisms give us a natural $\eta : Id \to \Delta \circ colim$

$\mathscr{L}_{fun,\mathfrak{C}}$

let $\mathscr{L}_{fun,\mathfrak{C}}$ be the category such that:

- objects are pairs $(L, l : Id_{fun(\mathfrak{C})} \to \Delta_C \circ L)$
- morphisms are natural transformations $\phi : L \to L'$ such that:

$\mathscr{L}_{fun,\mathfrak{C}}$

let $\mathscr{L}_{fun,\mathfrak{C}}$ be the category such that:

- objects are pairs $(L, l : Id_{fun(\mathfrak{C})} \to \Delta_C \circ L)$
- morphisms are natural transformations $\phi : L \to L'$ such that:

$$\begin{tikzcd} & Id_{fun(\mathfrak{C})} \arrow[dl, "l"'] \arrow[dr, "l'"] & \\ \Delta(L) \arrow[rr, "\Delta(\phi)"] & & \Delta(L') \end{tikzcd}$$

colim is initial in $\mathscr{L}_{fun,\mathfrak{C}}$

Let's step back: for a small functor $J : S \to \mathfrak{C}$,
$\phi_J : L(J) \to L'(J)$ is just a single morphism

colim is initial in $\mathscr{L}_{fun,\mathfrak{C}}$

Let's step back: for a small functor $J : S \to \mathfrak{C}$,
$\phi_J : L(J) \to L'(J)$ is just a single morphism
the above diagram becomes

$$\begin{tikzcd}
 & J \arrow[dl, "l_J"'] \arrow[dr, "l'_J"] & \\
\Delta(L(J)) \arrow[rr, "\phi_J"] & & \Delta(L'(J))
\end{tikzcd}$$

colim is initial in $\mathscr{L}_{fun,\mathfrak{C}}$

Let's step back: for a small functor $J : S \to \mathfrak{C}$,
$\phi_J : L(J) \to L'(J)$ is just a single morphism
the above diagram becomes

$$\begin{tikzcd} & J \arrow[dl, "l_J"'] \arrow[dr, "l'_J"] & \\ \Delta(L(J)) \arrow[rr, "\phi_J"] & & \Delta(L'(J)) \end{tikzcd}$$

that is a map

$\phi_J : L(J) \to L'(J)$ such that for all $s \in S$

$$\begin{tikzcd} & J(s) \arrow[dl, "l_{J,s}"'] \arrow[dr, "l'_{J,s}"] & \\ L(J) \arrow[rr, "\phi_J"] & & L'(J) \end{tikzcd}$$

colim is initial in $\mathscr{L}_{fun,\mathfrak{C}}$

If $\mathfrak{C}$ is cocomplete, the universal property of the colimit is exactly the statement that, for all L,J, there is a unique $\phi_J : colim(J) \to L(J)$ such that

$$\begin{tikzcd} & J(s) \arrow[dl, "\eta_{J,s}"'] \arrow[dr, "l_{J,s}"] & \\ colim(J) \arrow[rr, "\Delta(\phi)"] & & L(J) \end{tikzcd}$$

that is, colim is initial in the category $\mathscr{L}_{fun,\mathfrak{C}}$

Want: initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

- Analogously, define $\mathscr{L}_{sm,\mathfrak{C}}$ for the endofunctor $sm : \mathfrak{C}at \to \mathfrak{C}at$

Want: initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

- Analogously, define $\mathscr{L}_{sm,\mathfrak{C}}$ for the endofunctor $sm : \mathfrak{C}at \to \mathfrak{C}at$
- the colimit we are looking for should be the initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

Want: initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

- Analogously, define $\mathscr{L}_{sm,\mathfrak{C}}$ for the endofunctor $sm : \mathfrak{C}at \to \mathfrak{C}at$
- the colimit we are looking for should be the initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

Consider the faithful functor $i : fun(\mathfrak{C}) \to sm(\mathfrak{C})$:
$i(J) = J$, $i(P, \phi) = \phi$

Want: initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

- Analogously, define $\mathscr{L}_{sm,\mathfrak{C}}$ for the endofunctor $sm : \mathfrak{C}at \to \mathfrak{C}at$
- the colimit we are looking for should be the initial object in $\mathscr{L}_{sm,\mathfrak{C}}$

Consider the faithful functor $i : fun(\mathfrak{C}) \to sm(\mathfrak{C})$:
$i(J) = J$, $i(P, \phi) = \phi$
that is, i forgets that ϕ is a natural transformation, and instead regards it just as a collection of maps $\phi_s : J(s) \to J'(P(s))$, i.e. a submorphism.

Conjecture: i^* is faithfull and essentially surjective

Lift i to a functor $i^* : \mathfrak{C}^{sm(\mathfrak{C})} \to \mathfrak{C}^{fun(\mathfrak{C})}$:

$$i^*(L)(J) = L \circ i(J)$$

Conjecture: i^* is faithfull and essentially surjective

Lift i to a functor $i^* : \mathfrak{C}^{sm(\mathfrak{C})} \to \mathfrak{C}^{fun(\mathfrak{C})}$:

$$i^*(L)(J) = L \circ i(J)$$

and further to a functor $i^* : \mathscr{L}_{sm,\mathfrak{C}} \to \mathscr{L}_{fun,\mathfrak{C}}$

$$i^*(\ell : Id_{sm(\mathfrak{C})} \to \Delta_C \circ L) = \bar{\ell} : Id_{sm(\mathfrak{C})} \circ i \to \Delta_C \circ L \circ i$$
$$= \bar{\ell} : Id_{fun(\mathfrak{C})} \to \Delta_C \circ L$$

Conjecture: i^* is faithfull and essentially surjective

Lift i to a functor $i^* : \mathfrak{C}^{sm(\mathfrak{C})} \to \mathfrak{C}^{fun(\mathfrak{C})}$:

$$i^*(L)(J) = L \circ i(J)$$

and further to a functor $i^* : \mathscr{L}_{sm,\mathfrak{C}} \to \mathscr{L}_{fun,\mathfrak{C}}$

$$i^*(\ell : Id_{sm(\mathfrak{C})} \to \Delta_C \circ L) = \overline{\ell} : Id_{sm(\mathfrak{C})} \circ i \to \Delta_C \circ L \circ i$$
$$= \overline{\ell} : Id_{fun(\mathfrak{C})} \to \Delta_C \circ L$$

Conjecture:

i^* is faithful and essentially surjective.

Conjecture: i^* is faithfull and essentially surjective

Lift i to a functor $i^* : \mathfrak{C}^{sm(\mathfrak{C})} \to \mathfrak{C}^{fun(\mathfrak{C})}$:

$$i^*(L)(J) = L \circ i(J)$$

and further to a functor $i^* : \mathscr{L}_{sm,\mathfrak{C}} \to \mathscr{L}_{fun,\mathfrak{C}}$

$$i^*(\ell : Id_{sm(\mathfrak{C})} \to \Delta_C \circ L) = \bar{\ell} : Id_{sm(\mathfrak{C})} \circ i \to \Delta_C \circ L \circ i$$
$$= \bar{\ell} : Id_{fun(\mathfrak{C})} \to \Delta_C \circ L$$

Conjecture:

i^* is faithful and essentially surjective.

if this is true then $\mathscr{L}_{sm,\mathfrak{C}}$ has an initial object, which will be the colimit $sm(\mathfrak{C}) \to \mathfrak{C}$ we are looking for, but this is still left to prove.

Part 3: Towards higher ontologies; programmatic representations

- $\mathfrak{C}at$ and the 2-categorical secret
- simplicial sets
- higher ontologies
- faces and degeneracies as ontological expansions
- programming higher ontologies

$\mathfrak{C}at$ as a 2-category

- The category of small categories is not just a category, but a 2-category

$\mathfrak{C}at$ as a 2-category

- The category of small categories is not just a category, but a 2-category

that is

$\mathfrak{C}at$ as a 2-category

- The category of small categories is not just a category, but a 2-category

that is

objects are small categories $\mathfrak{C}, \mathfrak{D}$

$\mathfrak{C}at$ as a 2-category

- The category of small categories is not just a category, but a 2-category

that is
objects are small categories $\mathfrak{C}, \mathfrak{D}$
morphisms are functors $F : \mathfrak{C} \to \mathfrak{D}$

$\mathfrak{Cat}$ as a 2-category

- The category of small categories is not just a category, but a 2-category

that is
objects are small categories $\mathfrak{C}, \mathfrak{D}$
morphisms are functors $F : \mathfrak{C} \to \mathfrak{D}$

2-morphisms are commutative diagrams:
$$\begin{tikzcd} \mathfrak{C} \arrow[r, "P"] \arrow[d, "F"'] & \mathfrak{D} \arrow[d, "F'"] \\ \mathfrak{C} \arrow[r, "Q"'] \arrow[ur, Rightarrow, "\eta"] & \mathfrak{D}' \end{tikzcd}$$

where $\eta : Q \circ F \to F' \circ P$ is a natural transformation

Simplecies in $\mathfrak{C}at$

The diagram
$$\begin{tikzcd} \mathfrak{C} \arrow[r, "P"] \arrow[d, "F"] & \mathfrak{D} \arrow[d, "F'"] \\ \mathfrak{C}' \arrow[r, "Q"] \arrow[ur, Rightarrow, "\eta"] & \mathfrak{D}' \end{tikzcd}$$

Can actually be described entirely by a "2-simplex":
$$\begin{tikzcd} \mathfrak{C} \arrow[d, "F"'] \arrow[dr, "F' \circ P"] & \\ \mathfrak{C}' \arrow[r, "Q"'] \arrow[ur, phantom, "\eta \Rightarrow"] & \mathfrak{D}' \end{tikzcd}$$

Simplecies in $\mathfrak{C}at$

Following this line of reasoning, we can recast:

- categories $\mathfrak{C}$ as 0-simplecies

Simplecies in $\mathfrak{C}at$

Following this line of reasoning, we can recast:

- categories $\mathfrak{C}$ as 0-simplecies
- functors $F : \mathfrak{C} \to \mathfrak{D}$ as 1-simplecies

Simplecies in $\mathfrak{C}at$

Following this line of reasoning, we can recast:

- categories $\mathfrak{C}$ as 0-simplecies
- functors $F : \mathfrak{C} \to \mathfrak{D}$ as 1-simplecies
- commutative triangles up to natural transformation as 2-simplecies

Simplecies in $\mathfrak{Cat}$

Following this line of reasoning, we can recast:

- categories $\mathfrak{C}$ as 0-simplecies
- functors $F : \mathfrak{C} \to \mathfrak{D}$ as 1-simplecies
- commutative triangles up to natural transformation as 2-simplecies

Just as two 0-simplecies(categories) can have multiple 1-simplecies(functors) between them
3 functors (1-simplecies) can have multiple natural transformations (2-simplecies) between them

$$
\begin{tikzcd}
\mathfrak{C} \arrow[d, "F"'] \arrow[dr, "H"] & \\
\mathfrak{C}' \arrow[r, "G"'] \arrow[ur, phantom, "\overset{\eta}{\Nearrow}"] & \mathfrak{D}'
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{C} \arrow[d, "F"'] \arrow[dr, "H"] & \\
\mathfrak{C}' \arrow[r, "G"'] \arrow[ur, phantom, "\overset{\mu}{\Nearrow}"] & \mathfrak{D}'
\end{tikzcd}
$$

face and degeneracy

given a 2-simplex $\sigma^2 =$
$$\begin{tikzcd} \mathfrak{C} \arrow[d, "F"'] \arrow[dr, "H"] & \\ \mathfrak{C}' \arrow[r, "G"'] \arrow[ur, phantom, "\overset{\eta}{\Nearrow}"] & \mathfrak{D}' \end{tikzcd}$$

face and degeneracy

given a 2-simplex $\sigma^2 =$

$$\begin{tikzcd} \mathfrak{C} \arrow[d, "F"'] \arrow[dr, "H"] & \\ \mathfrak{C}' \arrow[r, "G"'] \arrow[ur, phantom, "\stackrel{\eta}{\Nearrow}"] & \mathfrak{D}' \end{tikzcd}$$

we can extract 3 bits of information, namely the faces:
$face_0(\sigma^2) = H$, $face_1(\sigma^2) = F$, $face_2(\sigma^2) = G$

face and degeneracy

given a 2-simplex $\sigma^2 =$
$$\begin{tikzcd} \mathfrak{C} \arrow[d, "F"'] \arrow[dr, "H"] \arrow[d, phantom, shift left=4ex, "\overset{\eta}{\nearrow}"] & \\ \mathfrak{C}' \arrow[r, "G"'] & \mathfrak{D}' \end{tikzcd}$$

we can extract 3 bits of information, namely the faces:
$face_0(\sigma^2) = H$, $face_1(\sigma^2) = F$, $face_2(\sigma^2) = G$

On the other hand, given a 1-simplex: $\sigma^1 = F : \mathfrak{C} \to \mathfrak{D}$, we can get two "degenerate" 2-simplecies:

face and degeneracy

given a 2-simplex $\sigma^2 =$
$$\begin{tikzcd} \mathfrak{C} \arrow[d, "F"'] \arrow[dr, "H"] & \\ \mathfrak{C}' \arrow[r, "G"'] \arrow[ur, phantom, "\Nearrow\eta"] & \mathfrak{D}' \end{tikzcd}$$

we can extract 3 bits of information, namely the faces: $face_0(\sigma^2) = H$, $face_1(\sigma^2) = F$, $face_2(\sigma^2) = G$

On the other hand, given a 1-simplex: $\sigma^1 = F : \mathfrak{C} \to \mathfrak{D}$, we can get two "degenerate" 2-simplecies:

$$d_0(\sigma^1) = \begin{tikzcd} \mathfrak{C} \arrow[d, "Id_{\mathfrak{C}}"'] \arrow[dr, "F"] & \\ \mathfrak{C} \arrow[r, "F"'] \arrow[ur, phantom, "\Nearrow Id"] & \mathfrak{D} \end{tikzcd} \quad \text{and } d_1(\sigma^1) = \begin{tikzcd} \mathfrak{C} \arrow[d, "F"'] \arrow[dr, "F"] & \\ \mathfrak{D} \arrow[r, "Id_{\mathfrak{D}}"'] \arrow[ur, phantom, "\Nearrow Id"] & \mathfrak{D} \end{tikzcd}$$

face and degeneracy

if Σ_n is the set of n-simplecies, the faces and degeneracies are, in general, functions:

$$\Sigma_0 \underset{\{f_i^1\}_{i=0,1}}{\overset{\{d_0^0\}}{\rightleftarrows}} \Sigma_1 \underset{\{f_i^2\}_{i=0,1,2}}{\overset{\{d_i^1\}_{i=1,2}}{\rightleftarrows}} \Sigma_2$$

face and degeneracy

if Σ_n is the set of n-simplecies, the faces and degeneracies are, in general, functions:

$$\Sigma_0 \underset{\{f_i^1\}_{i=0,1}}{\overset{\{d_0^0\}}{\rightleftarrows}} \Sigma_1 \underset{\{f_i^2\}_{i=0,1,2}}{\overset{\{d_i^1\}_{i=1,2}}{\rightleftarrows}} \Sigma_2$$

of course, in more general situations, this chain continues as:

$$\Sigma_{n-1} \underset{\{f_i^n\}_{0\leq i\leq n}}{\longleftarrow} \Sigma_n \overset{\{d_i^n\}_{0\leq i\leq n}}{\longrightarrow} \Sigma_{n+1}$$

Simplicial Sets

Consider the category fin:

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$
- morphisms are increasing functions

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$
- morphisms are increasing functions

There are two special types of increasing functions which will represent primordial face and degeneracy maps:

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$
- morphisms are increasing functions

There are two special types of increasing functions which will represent primordial face and degeneracy maps:
$f_i^n : [n] \to [n+1]$, $f_i^n(0, ..., i-1, i, n) = (0, ..., i-1,, i, ..., n)$

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$
- morphisms are increasing functions

There are two special types of increasing functions which will represent primordial face and degeneracy maps:
$f_i^n : [n] \to [n+1]$, $f_i^n(0, ..., i-1, i, n) = (0, ..., i-1,, i, ..., n)$
$d_i^n : [n] \to [n-1]$, $d_i^n(0, ..., i, i+1, ...n) = (0, ..., (i, i+1), ..., n)$

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$
- morphisms are increasing functions

There are two special types of increasing functions which will represent primordial face and degeneracy maps:
$f_i^n : [n] \to [n+1]$, $f_i^n(0, ..., i-1, i, n) = (0, ..., i-1,, i, ..., n)$
$d_i^n : [n] \to [n-1]$, $d_i^n(0, ..., i, i+1, ...n) = (0, ..., (i, i+1), ..., n)$

Simplicial Set

A Simplicial Set is then a contravariant functor $\Sigma : fin \to \mathfrak{S}et$

Simplicial Sets

Consider the category fin:

- objects are the finite ordered sets $[n] = (0, ..., n)$
- morphisms are increasing functions

There are two special types of increasing functions which will represent primordial face and degeneracy maps:
$f_i^n : [n] \to [n+1],\ f_i^n(0, ..., i-1, i, n) = (0, ..., i-1,, i, ..., n)$
$d_i^n : [n] \to [n-1],\ d_i^n(0, ..., i, i+1, ...n) = (0, ..., (i, i+1), ..., n)$

Simplicial Set

A Simplicial Set is then a contravariant functor $\Sigma : fin \to \mathfrak{S}et$

A morphism of simplicial sets, is then just a natural transformation $\eta : \Sigma \to \Sigma'$

Carrying Intuition from 1-categories

To consider a (small) 1-category as a Simplicial Set we need the "nerve" functor, but let's just consider this intuitively

Carrying Intuition from 1-categories

To consider a (small) 1-category as a Simplicial Set we need the "nerve" functor, but let's just consider this intuitively

a 1-category has:

Carrying Intuition from 1-categories

To consider a (small) 1-category as a Simplicial Set we need the "nerve" functor, but let's just consider this intuitively

a 1-category has:

- 0-simplecies objects

Carrying Intuition from 1-categories

To consider a (small) 1-category as a Simplicial Set we need the "nerve" functor, but let's just consider this intuitively

a 1-category has:

- 0-simplecies objects
- 1-simplecies morphisms

Carrying Intuition from 1-categories

To consider a (small) 1-category as a Simplicial Set we need the "nerve" functor, but let's just consider this intuitively

a 1-category has:

- 0-simplecies objects
- 1-simplecies morphisms
- 2-simplies commutative triangles (on the nose)

Carrying Intuition from 1-categories

A natural transformation between 1-categories $F : \mathfrak{C} \to \mathfrak{D}$ (considered as simplicial sets) will then yeild the following conditions:

Carrying Intuition from 1-categories

A natural transformation between 1-categories $F : \mathfrak{C} \to \mathfrak{D}$ (considered as simplicial sets) will then yeild the following conditions:

naturality with respect to face maps gives:

$$F_2\left(\begin{tikzcd} a \arrow[d, "f"'] \arrow[dr, "h"] & \\ b \arrow[r, "g"'] & c \end{tikzcd} \right) = \begin{tikzcd} F_0(a) \arrow[d, "F_1(f)"'] \arrow[dr, "F_1(h)"] & \\ F_0(b) \arrow[r, "F_1(g)"'] & F_0(c) \end{tikzcd}$$

Carrying Intuition from 1-categories

A natural transformation between 1-categories $F : \mathfrak{C} \to \mathfrak{D}$ (considered as simplicial sets) will then yeild the following conditions:

naturality with respect to face maps gives:

$$F_2\left(\begin{tikzcd} a \arrow[d, "f"'] \arrow[dr, "h"] & \\ b \arrow[r, "g"'] & c \end{tikzcd}\right) = \begin{tikzcd} F_0(a) \arrow[d, "F_1(f)"'] \arrow[dr, "F_1(h)"] & \\ F_0(b) \arrow[r, "F_1(g)"'] & F_0(c) \end{tikzcd}$$

that is: $F(g \circ f) = F(g) \circ F(g)$

Carrying Intuition from 1-categories

A natural transformation between 1-categories $F : \mathfrak{C} \to \mathfrak{D}$ (considered as simplicial sets) will then yeild the following conditions:

naturality with respect to face maps gives:

$$F_2\left(\begin{tikzcd} a \arrow[d, "f"'] \arrow[dr, "h"] & \\ b \arrow[r, "g"'] & c \end{tikzcd} \right) = \begin{tikzcd} F_0(a) \arrow[d, "F_1(f)"'] \arrow[dr, "F_1(h)"] & \\ F_0(b) \arrow[r, "F_1(g)"'] & F_0(c) \end{tikzcd}$$

that is: $F(g \circ f) = F(g) \circ F(g)$

naturality with respect to degeneracy gives $F(Id_a) = Id_{F(a)}$

Carrying Intuition from 1-categories

A natural transformation between 1-categories $F : \mathfrak{C} \to \mathfrak{D}$ (considered as simplicial sets) will then yeild the following conditions:
naturality with respect to face maps gives:

$$F_2\left(\begin{tikzcd} a \arrow[d, "f"'] \arrow[dr, "h"] & \\ b \arrow[r, "g"'] & c \end{tikzcd} \right) = \begin{tikzcd} F_0(a) \arrow[d, "F_1(f)"'] \arrow[dr, "F_1(h)"] & \\ F_0(b) \arrow[r, "F_1(g)"'] & F_0(c) \end{tikzcd}$$

that is: $F(g \circ f) = F(g) \circ F(g)$

naturality with respect to degeneracy gives $F(Id_a) = Id_{F(a)}$
i.e. a natural transformation between 1-categories (as simplicial sets) is actually just a functor.

Benefits of simplicial sets: Higher Ontologies

There are two benefits to simplicial sets, the first is a path to defining Higher Ontologies:

Benefits of simplicial sets: Higher Ontologies

There are two benefits to simplicial sets, the first is a path to defining Higher Ontologies:

- Ontologies represent concepts and relations between them

Benefits of simplicial sets: Higher Ontologies

There are two benefits to simplicial sets, the first is a path to defining Higher Ontologies:

- Ontologies represent concepts and relations between them
- Simplicial Sets allow us to stage morphisms as concepts and use 2-simplecies to find relations between morphisms, (or relations between relations).

Benefits of simplicial sets: Higher Ontologies

There are two benefits to simplicial sets, the first is a path to defining Higher Ontologies:

- Ontologies represent concepts and relations between them
- Simplicial Sets allow us to stage morphisms as concepts and use 2-simplecies to find relations between morphisms, (or relations between relations).
- the naive definition of a higher ontology, then, should be a simplicial set

Benefits of simplicial sets: Higher Ontologies

There are two benefits to simplicial sets, the first is a path to defining Higher Ontologies:

- Ontologies represent concepts and relations between them
- Simplicial Sets allow us to stage morphisms as concepts and use 2-simplecies to find relations between morphisms, (or relations between relations).
- the naive definition of a higher ontology, then, should be a simplicial set

The theory of Simplicial Sets is pretty well developed (almost as well as category theory) and so sSet gives a great starting point for capturing the intuition behind what a higher ontology should be

Benefits of simplicial sets: Computational Representation

The second benefit to using simplicial sets is that, a priori, composition isn't computed, but **given**

Benefits of simplicial sets: Computational Representation

The second benefit to using simplicial sets is that, a priori, composition isn't computed, but **given**
In a 1-category, for any two composable morphisms $f : a \to b, g : b \to c$ there is always a 2-simplex:

Benefits of simplicial sets: Computational Representation

The second benefit to using simplicial sets is that, a priori, composition isn't computed, but **given**

In a 1-category, for any two composable morphisms $f : a \to b, g : b \to c$ there is always a 2-simplex:

$$\begin{tikzcd} a \arrow[d, "f"'] \arrow[dr, "g \circ f"] & \\ b \arrow[r, "g"'] & c \end{tikzcd}$$

Benefits of simplicial sets: Computational Representation

In an abstract simplicial set one need not have a 2-simplex for every composable pair $f : a \to b, g : b \to c$

Benefits of simplicial sets: Computational Representation

In an abstract simplicial set one need not have a 2-simplex for every composable pair $f : a \to b, g : b \to c$
This allows us to get away with representing morphisms without having to compute potentially infinite chains of maps

Benefits of simplicial sets: Computational Representation

In an abstract simplicial set one need not have a 2-simplex for every composable pair $f : a \to b, g : b \to c$
This allows us to get away with representing morphisms without having to compute potentially infinite chains of maps

(for example if $f : a \to a$, $f^{(n)}$ is always defined)

Benefits of simplicial sets: Computational Representation

In an abstract simplicial set one need not have a 2-simplex for every composable pair $f : a \to b, g : b \to c$
This allows us to get away with representing morphisms without having to compute potentially infinite chains of maps

(for example if $f : a \to a$, $f^{(n)}$ is always defined)

Pythonic simplecies

The face and degeneracy maps in a simplicial set can actually be stored as attributes of a class in python: Simplex

- simplex.level = n

Pythonic simplecies

The face and degeneracy maps in a simplicial set can actually be stored as attributes of a class in python: Simplex

- simplex.level = n
- simplex.faces = n-tuple of simplecies with level n-1

Pythonic simplecies

The face and degeneracy maps in a simplicial set can actually be stored as attributes of a class in python: Simplex

- simplex.level = n
- simplex.faces = n-tuple of simplecies with level n-1
- simplex.degeneracy = n-tuple of simplecies with level n+1

Pythonic simplecies

The face and degeneracy maps in a simplicial set can actually be stored as attributes of a class in python: Simplex

- simplex.level = n
- simplex.faces = n-tuple of simplecies with level n-1
- simplex.degeneracy = n-tuple of simplecies with level n+1

A simplicial set is then just a collection of simplex objects containing its degeneracies and faces

Pythonic Functors

A functor between simplicial sets is then just a function F between the collections of simplex objects satisfying simple naturality assertions:

Pythonic Functors

A functor between simplicial sets is then just a function F between the collections of simplex objects satisfying simple naturality assertions:

code

```
for simplex in simpset:
    assert F(simplex.faces) = F(simplex).faces
    assert F(simplex.degeneracies) = F(simplex).degeneracies
```

Pythonic Functors

A functor between simplicial sets is then just a function F between the collections of simplex objects satisfying simple naturality assertions:

code

```
for simplex in simpset:
    assert F(simplex.faces) = F(simplex).faces
    assert F(simplex.degeneracies) = F(simplex).degeneracies
```

you can view my working code at https://github.com/nopounch/golog

Theoretical Goals: faces and degeneracies as ontological expansions

The following is a work in progress:

Theoretical Goals: faces and degeneracies as ontological expansions

The following is a work in progress:
The idea is to consider the collections of face maps and degeneracy maps as small functors:

- $face : \Sigma \to sm(\Sigma)$
- $degen : \Sigma \to sm(\Sigma)$

Theoretical Goals: faces and degeneracies as ontological expansions

The following is a work in progress:
The idea is to consider the collections of face maps and degeneracy maps as small functors:

- $face : \Sigma \to sm(\Sigma)$
- $degen : \Sigma \to sm(\Sigma)$

i.e. $face_n : \Sigma_n \to (\Sigma_{n-1})^n$, $degen_n : \Sigma_n \to (\Sigma_{n+1})^n$

Theoretical Goals: faces and degeneracies as ontological expansions

The following is a work in progress:
The idea is to consider the collections of face maps and degeneracy maps as small functors:

- $face : \Sigma \to sm(\Sigma)$
- $degen : \Sigma \to sm(\Sigma)$

i.e. $face_n : \Sigma_n \to (\Sigma_{n-1})^n$, $degen_n : \Sigma_n \to (\Sigma_{n+1})^n$
The functorality here isn't apparent, and may necessitate working with something close to but not exactly simplicial sets.

Coding Goals: Git Ontologies, Code Ontologies

The goal of programmatically representing ontologies is to actually extract ologs from real world data:

Coding Goals: Git Ontologies, Code Ontologies

The goal of programmatically representing ontologies is to actually extract ologs from real world data:

- Example 1: Extract ontologies from git repositories:
  Objects are actual file structures, morphisms version updates

Coding Goals: Git Ontologies, Code Ontologies

The goal of programmatically representing ontologies is to actually extract ologs from real world data:

- Example 1: Extract ontologies from git repositories:
  Objects are actual file structures, morphisms version updates
- Example 2: Extract ontologies from code:
  Objects are instanciated objects, morphisms are calls

Coding Goals: Git Ontologies, Code Ontologies

The goal of programmatically representing ontologies is to actually extract ologs from real world data:

- Example 1: Extract ontologies from git repositories: Objects are actual file structures, morphisms version updates
- Example 2: Extract ontologies from code: Objects are instanciated objects, morphisms are calls