Isomorphic Reasoning: Counting Polymorphic Type Inhabitants

Emily Pillmore, Alexander Konovalov

May 2019

Who are we?

But who are we and what do we do?

Plan

Q: We want to learn how to "count" types, but where do we start?

Plan

Q: We want to learn how to "count" types, but where do we start?
A: Lets start small, in the category of sets, and simple arithmetic.

Plan

As an example, lets prove the following for all $a, b, c \in \mathbb{N}$:

- $a \times (b + c) \cong (a \times b) + (a \times c)$

Plan

As an example, lets prove the following for all $a, b, c \in \mathbb{N}$:

- $a \times (b + c) \cong (a \times b) + (a \times c)$
- $(b + c)^a \cong b^a \times c^a$

Plan

As an example, lets prove the following for all $a, b, c \in \mathbb{N}$:

- $a \times (b + c) \cong (a \times b) + (a \times c)$
- $(b + c)^a \cong b^a \times c^a$
- $a^{b+c} \cong a^b \times a^c$

Plan

As an example, lets prove the following for all $a, b, c \in \mathbb{N}$:

- $a \times (b + c) \cong (a \times b) + (a \times c)$
- $(b + c)^a \cong b^a \times c^a$
- $a^{b+c} \cong a^b \times a^c$
- $(a^b)^c \cong a^{b \times c}$

But How?

Q: But Alex/Emily, how does this apply to Functional Programming? Why are you even here right now teaching me about highschool arithmetic?

But How?

Q: But Alex/Emily, how does this apply to Functional Programming? Why are you even here right now teaching me about rudimentary highschool arithmetic?
A: Stand back my son, I will show you the language of *categories*.

Goals

Here's a sneak peek:

Proposition (LAPC, RAPL)

Any category $\mathbf{C}$ *admits all limits and colimits indexed by a small category* $\mathbf{J}$ *if and only if the constant diagram functor* $\Delta : \mathbf{C} \to \mathbf{C}^{\mathbf{J}}$ *has a right and left adjoint* $\mathit{colim} \dashv \Delta \dashv \mathit{lim}$*:*

$$\mathbf{C} \underset{\mathit{colim}}{\overset{\mathit{lim}}{\underset{\longleftarrow}{\overset{\longleftarrow}{-\Delta\rightarrow}}}} \mathbf{C}^{\mathbf{J}}$$

When these functors exist, we can prove our arithmetic!

Goals

In functional programming we work with a subtle form of arithmetic every day, but it's rare that we actually *understand* what it is we're doing. It turns out to be a very deep process with a very intuitive facade! We will cover the preliminaries necessary to prove the fundamentals of type-driven arithmetic, and we will use this calculus to bolster our understanding of functional programming as a whole.

Steps

Don't worry: we will build the intuition necessary to understand the previous statement.

Steps

Don't worry: we will build the intuition necessary to understand the previous statement. We will do so by understanding the following processes:

Steps

Don't worry: we will build the intuition necessary to understand the previous statement. We will do so by understanding the following processes:

- Step 1: (De-)categorification

Steps

Don't worry: we will build the intuition necessary to understand the previous statement. We will do so by understanding the following processes:

- Step 1: (De-)categorification
- Step 2: The Yoneda lemma

Steps

Don't worry: we will build the intuition necessary to understand the previous statement. We will do so by understanding the following processes:

- Step 1: (De-)categorification
- Step 2: The Yoneda Lemma
- Step 3: Representability

Steps

Don't worry: we will build the intuition necessary to understand the previous statement. We will do so by understanding the following processes:

- Step 1: (De-)categorification
- Step 2: The Yoneda Lemma
- Step 3: Representability
- Step 4: A Preliminary Proof

Steps

Don't worry: we will build the intuition necessary to understand the previous statement. We will do so by understanding the following processes:

- Step 1: (De-)categorification
- Step 2: The Yoneda Lemma
- Step 3: Representability
- Step 4: A Preliminary Proof

And optionally if we have time,

- Step 5: Adjunctions, Limits, Colimits
- Step 6: A Proof Using LAPC, RAPL

Steps

Then, we will count! Lets get started with the preliminaries.

Definitions

Definition (Category)

A **Category** consists of the following data:

- a collection of **objects** $x, y, z, \ldots$
- a collection of **morphisms** $f, g, h, \ldots$

such that

- each morphism has specified **domain** and **codomain** objects. The notation $f : x \to y$ signifies that the morphism f has domain x and codomain y
- each object has an associated **identity morphism** $1_x : x \to y$
- For any pair of morphisms $f : x \to y, g : y \to z$ there exists a morphism $gf : x \to z$

Definitions

This data is subject to the following axioms:

- For any $f : x \to y$, the composites $1_y f$ and $f 1_x$ are both equal to f
- For any composable triple f, g, h, the composites $h(gf)$ and $(hg)f$ are equal, and we will simply denote them hgf.
- Between any two objects x, y in the category $\mathbf{C}$, we may speak of the collection of morphisms with domain x and codomain y called $Hom_{\mathbf{C}}(x, y)$. Sometimes, we will denote this object $\mathbf{C}(X, Y)$ for ease of use.

This is to say that the law of composition is *associative* and *unital* with the morphisms $1_{(-)}$ serve as two-sided identities

Definitions

i **Set** has sets as its objects, and functions with specified domain and codomain as morphisms

ii **Pos** has partially-ordered sets as objects and order-preserving functions as morphisms

iii **Top** has topological spaces as objects and continuous functions as morphisms

iv **Mon** has the single point set 1 as its only object, and its morphisms are the elements of the monoid with composition given by the monoid operation.

v **Hask**... (just kidding)

Definitions

All of these are examples of concrete categories - categories in which the collection of objects and collection of morphisms are both *sets*. However, there are categories with in which one or more of these collections is larger than sets (like the category of categories), so we must introduce some subtlety to the theory in order to avoid category-theoretic versions of Russel's paradox in inopportune places.

Definitions

Lets introduce a few definitions that we'll use sparingly when in need:

Definition

A category $\mathbf{C}$ is...

- **concrete** if its collection of objects and morphisms are both sets.
- **small** if only its collection of morphisms is a set.
- **locally small** if for any two objects x and y, $\mathbf{C}(x, y)$ is a set.

For our purposes, when working with types, we are usually working in a concrete category, if not $\mathbf{C}$ directly.

Definitions

Definition

For every category $\mathbf{C}$ we may speak about its **dual** notion $\mathbf{C}^{op}$, the **opposite category** of $\mathbf{C}$, which consists of the following data:

- the same objects as $\mathbf{C}$
- a morphism f^{op} in $\mathbf{C}^{op}$ for each morphism f in $\mathbf{C}$ with the codomain and domain reversed. i.e. when $f : x \to y$, $f^{op} : y \to x$.

Definitions

The data described in the previous slide defines a category in relation to itself. The process of "turning around the arrows" or "swapping domains and codomains" exhibits a syntactic self-duality for every category, retaining the precisely the same information as $\mathbf{C}$. In this way, we have a unique perspective in Category Theory: For every theorem proven in general for a category, we immediately prove its dual, since the opposite category is a valid category in its own right. More generally, if one proves theorems "for all categories $\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_n}$", then this leads to 2^n dual theorems. In practice though, these dual theorems do not differ meaningfully from the original.

Definitions

We will now introduce the types of morphisms you will encounter in the wild. Much like the injective/bijective/surjective functions on sets that we are used to, we have abstract version notions of these concepts.

Definitions

Definition (Isomorphism)

An **isomorphism** in a category is a morphism $f : x \to y$ for which there exists a morphism $g : y \to x$ such that $gf = 1_x$ and $fg = 1_y$ (g is a two-sided inverse). The objects x and y are said to be **isomorphic**, and we denote this $x \cong y$.

Definitions

The following statements are equivalent

Lemma

(i) $f : x \to y$ *is an isomorphism in* $\mathbf{C}$

(ii) *For all objects* $c \in \mathbf{C}$, *post-composition with* f *defines a bijection*

$$f_* : \mathbf{C}(c, x) \to \mathbf{C}(c, y)$$

(iii) *For all objects* $c \in \mathbf{C}$, *pre-composition with* f *defines a bijection*

$$f^* : \mathbf{C}(y, c) \to \mathbf{C}(x, c)$$

Definitions

Definition

a morphism $f : x \to y$ is...

i a **monomorphism** if for any parallel morphisms $h, k : w \to x, fh = fk$ implies that $h = k$.

ii an **epimorphism** if for any parallel morphisms $h, k : y \to z, hf = kf$ implies $h = k$.

Definitions

This looks foreign, but notice the following:

- When considering a monomorphism f, with $h, k : 1 \to x$, note that this is exactly the criteria for an injection, where $fh = fk$ exactly when $h = k$. ($h(*)$ of course, being a single element in x).
- Likewise, when considering an epimorphism f with $h, k : y \to \{0, 1\}$, then $hf = kf$ exactly when $h = k$.

Definitions

Fun facts: $\{0, 1\}$ is equivalent to $\{\mathit{true}, \mathit{false}\}$, the subobject classifier in the topos of **C**s, which picks out the true or false values within. We can talk more about that after the talk...

Definitions

Definition (Functor)

A **functor** $F : \mathbf{C} \to \mathbf{D}$ between categories $\mathbf{C}$ and $\mathbf{D}$ consists of the following data:

- An object $Fc \in \mathbf{D}$ for each object $c \in \mathbf{C}$
- A morphism $Ff : Fc \to Fc'$ in $\mathbf{D}$ for each morphism $f \in \mathbf{C}$

Additionally, the following must be satisfied:

- For any composable pair f, g in $\mathbf{C}$, $Fg \cdot Ff = F(g \cdot f)$
- For each object $c \in \mathbf{C}$, $F(1_c) = 1_{Fc}$

Definitions

So far, we have defined a functor which acts uniformly on arrows. However, there exist functors which have a tendency to take arrows in the source category and flip the direction of the arrows such that they point the opposite way in the target category i.e. for any such functor $F : \mathbf{C} \to \mathbf{D}$ and any objects c, c' and morphism $f : c \to c'$ in $\mathbf{C}$, we have $Ff : Fc' \to Fc$. Such functors are called **contravariant** functors, in contrast to their siblings which are called **covariant**.

Definitions

Instead of differentiating each functor by variance, note that every contravariant functor is a covariant functor $F : \mathbf{C}^{op} \to \mathbf{D}$. Instead of juggling variance, we will simply be sure to specify the variance of a functor by whether or not it maps from an opposite category.

Definitions

- The power set functor $P : \mathbf{Set} \to \mathbf{Set}$ that sends a set A to its power set $PA = \{U : U \subset A\}$ and a function $f : A \to B$ to the direct image function $f_* : PA \to PB$ which sends $U \subset A$ to $f_*(U) \subset B$
- The contravariant power set functor $P : \mathbf{Set}^{op} \to \mathbf{Set}$ sends a set A to its power set PA, and every function $f : A \to B$ to the inverse-image function $f^{-1} : PB \to PA$ such that $V \subset B \mapsto f^{-1}(V) \subset A$.

Definitions (cont'd)

- For any $c \in \mathbf{C}$ the covariant Hom-functor $\mathbf{C}(c, -) : \mathbf{C} \to \mathbf{C}$ taking objects in $\mathbf{C}$ to their corresponding hom-sets in **Set**, and morphisms to post-composition.
- For any $c \in \mathbf{C}^{op}$ the contravariant Hom-functor $\mathbf{C}(-, c) : \mathbf{C}^{op} \to \mathbf{C}$ taking objects in $\mathbf{C}$ to their corresponding hom-sets in **Set** and morphisms to pre-composition.

Definition (Natural Transformation)

A **natural transformation** between two functors $F, G : \mathbf{C} \to \mathbf{D}$ consists of the following data:

- To each $c \in C$, a component morphism $\alpha_c : Fc \to Gc$ exists such that the following diagram commutes or any $c, c' \in \mathbf{C}$ and $f : c \to c'$:

$$\begin{array}{ccc} Fc & \xrightarrow{\alpha_c} & Gc \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle Gf} \\ Fc' & \xrightarrow{\alpha_{c'}} & Gc' \end{array}$$

There is such a thing as a **natural isomorphism** as well. How might you define one?

Definition (Natural Transformation)

A **natural transformation** between two functors $F, G : \mathbf{C} \to \mathbf{D}$ consists of the following data:

- To each $c \in C$, a component morphism $\alpha_c : Fc \to Gc$ exists such that the following diagram commutes or any $c, c' \in \mathbf{C}$ and $f : c \to c'$:

$$\begin{array}{ccc} Fc & \xrightarrow{\alpha_c} & Gc \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle Gf} \\ Fc' & \xrightarrow{\alpha_{c'}} & Gc' \end{array}$$

Definition (natural isomorphism)

A **natural isomorphism** is a natural isomorphism where each component is an isomorphism. We'll denote them as $\alpha : F \cong G$.

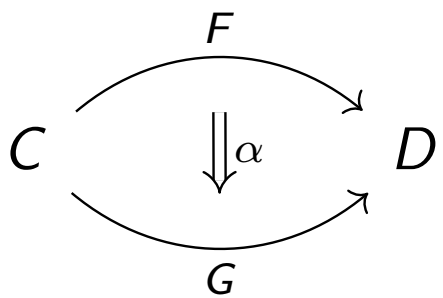
Definitions

In effect, this means that the following naturality condition holds for all $c, c' \in \mathbf{C}$ for a given transformation $\alpha : F \Rightarrow G$:

$$\alpha_{c'} F = G \alpha_c$$

That's it!

Often, we'll just denote this data with either $\alpha : F \Rightarrow G$, or diagrammatically with



Definitions

This is called a *naturality* condition. To say that α is *natural* in c is to say there exists a component morphism α_c providing a relationship between Fc and Gc.

Definitions

Natural transformations are ubiquitous. In a sense, they form "analogies between relationships". If one takes the perspective that functors define an intimate relationship between structures, then natural transformations provide a way of relating those relationships.

Definitions

Examples of natural transformations appear everywhere in the wild. Here are a few:

- There is a natural transformation $\eta_A : 1_{\mathbf{Set}} \Rightarrow P$ from the identity to the covariant power set functor whose components $\eta_A : A \to PA$ are functions that carry $a \in A$ to $a \in PA$.
- The open and closed subset functors are naturally isomorphic when regarded as functors $\mathcal{O}, \mathcal{C} : \mathbf{Top}^{op} \to \mathbf{Set}$. The components are defined by taking an open subset of X to its complement, which is closed. Thus, the "naturality" condition asserts that forming complements commutes with taking preimages (i.e. $f^{-1}(V)^c \cong f^{-1}(V^c)$).

Back to the question

Question

How do we prove something like the following?

$$a \times (b + c) = (a \times b) + (a \times c)$$

Categorification

What is **(de-)categorification**?

- A procedure by which set-theoretic concepts are expressed in terms of category theory (or vice-versa).

Categorification

Lets begin by categorifying arithmetic in the natural numbers. The first step to answering the question is to note some things about structure.

Categorification

Question

Using what we've just learned, how would we prove something like the following?

$$a \times (b + c) = (a \times b) + (a \times c) \text{ for } a, b, c \in \mathbb{N}$$

- What purpose to the variables serve?
- What do we need in order to prove the statement?

Categorification

First thing to note is that the natural numbers a, b, c denote the cardinality of finite sets:

$$a :\equiv |A| \qquad b :\equiv |B| \qquad c :\equiv |C|$$

Categorification

- What is true when $a = b$?

Categorification

- What is true when $a = b$?

$$A \cong B$$

Categorification

- What is true when $a = b$?

$$A \cong B$$

In fact, this equation asserts the following: $a = b$ if and only is $A \cong B$.

Categorification

It's natural now, to also ask about the operations of $\times$ and $+$.

Categorification

It's natural now, to also ask about the operations of $\times$ and $+$. Ideas?

Categorification

Lets apply the same sort of categorification to the notion of $\times$ and $+$.

Categorification

What are the sizes of the sets $A \times B$ and $B + C$?

Categorification

What are the sizes of the sets $A \times B$ and $B + C$?

Q: What sets have size $a \times b$ and $b + c$?

Categorification

- Note that by simple counting, the set $|A \times B| = a \times b$. We will refer to $\times$ as a *product*.
- Additionally, the disjoint union $B \sqcup C$ has $|B \sqcup C| = b + c$. We will refer to $+$ as a *coproduct*.

Categorification

Hence,

$$A \times B \cong a \times b$$
$$B + C \cong b + c$$

We present this without a rigorous proof. Though, by counting, one can see that this is correct for sets. From now on, we'll call the Product of sets "$\times$", and the disjoint union "$+$".

Categorification

Q: So what is the nature of this (de-)categorification?
A:

- Arithmetic in the naturals can be expressed in terms of natural isomorphisms of finite sets. These natural isomorphism classes restrict to their own category $\mathbf{Fin}_{\mathbf{Iso}}$ called of finite sets and bijections.
- $\mathbf{Fin}_{\mathbf{Iso}}$ serves as the domain for the cardinality functor $|-| : \mathbf{Fin}_{\mathbf{iso}} \to \mathbf{Set}$ which defines a de-categorification procedure under the definition $a :\equiv |A|$ etc.

Categorification

But additionally, one gets the following for free under a similar application of the theorems, which is nice, I guess:

$$A^{B+C} \cong A^B \times A^C \qquad (A \times B)^C \cong A^C \times B^C \qquad (A^B)^C \cong A^{B \times C}$$

Where the exponent A^B denotes the set of functions from A to B. (Can we produce a proof of that?)

Categorification

Q: So what is the nature of (de-)categorification?
A: It's all a miserable pack of lies.

Categorification

Summary:

- $A \times B \cong a \times b$, with $|A \times B| :\equiv a \times b$
- $B + C \cong b + c$ and $|B + C| :\equiv b + c$
- $\times$ refers to a *product* which is the cartesian product of sets, and $+$ refers to a *coproduct*, which is the disjoint union of sets
- $(=)^{(-)}$ refers to $\mathbf{C}(-, =)$

Categorification

So what does this mean?

$$a \times (b + c) = (a \times b) + (a \times c)$$

Categorification

So what does mean?

$$a \times (b + c) = (a \times b) + (a \times c)$$

Answer:

$$A \times (B + C) \cong (A \times B) + (A \times C)$$

Yoneda

A perspective:

An objects is fully determined by its relationship to other objects.

Yoneda

Translation:

An object is fully determined by the morphisms and to and from the object.

Yoneda

Q: But why is this important? Why do we care?
A: To quote from a friend:

"Most statements in elementary category theory can be proved using some variant statement of yoneda together with information about the category of sets"

The Yoneda Lemma

A and B are isomorphic if and only if

- For all X, $\mathbf{C}(A, X) \cong \mathbf{C}(B, X)$
- the isomorphisms $\mathbf{C}(A, X) \cong \mathbf{C}(B, X)$ are *natural* with respect to any function $f : X \to Y$.

Yoneda

Q: But what does this mean?
A: The equivalent statements say the following: the objects A and B are isomorphic if and only if their relationships to any other X are in correspondence.

Yoneda

The Yoneda Lemma

A and B are isomorphic if and only if the sets $\mathbf{C}(A, X)$ and $\mathbf{C}(B, X)$ are naturally isomorphic.

Yoneda

The Yoneda Lemma

A and B are isomorphic if and only if the sets $\mathbf{C}(A, X)$ and $\mathbf{C}(B, X)$ are naturally isomorphic.

Proof ($\Leftarrow$).

Let $\mathbf{C}(A, X) \cong \mathbf{C}(B, X)$ for every X. Let $X = A$, and $X = B$. Then, we prove this using the following bijections and a diagram chase (see board):

$$\mathbf{C}(A, A) \cong \mathbf{C}(B, A) \qquad \mathbf{C}(B, A) \cong \mathbf{C}(B, B)$$

The Yoneda Lemma

A and B are isomorphic if and only if the sets $\mathbf{C}(A, X)$ and $\mathbf{C}(B, X)$ are naturally isomorphic.

Proof ($\Rightarrow$).

Note that $\mathbf{C}(A, -)$ is a functor $\mathbf{C} \to \mathbf{C}$. Functors preserve isomorphisms. Hence, $\mathbf{C}(A, X) \cong \mathbf{C}(B, X)$ for all X.

Yoneda

There was actually an even faster way to do this using the full/faithfulness of the hom-functor (fully faithful functors *reflect* isomorphisms as well as preserve them i.e. $FA \cong FB \Rightarrow A \cong B$!), but we can put that away for another day

Yoneda

Recap:

With a categorification, we understand that

- $a \times (b + c) = (a \times b) + (a \times c)$ is the same as saying $A \times (B + C) \cong (A \times B) + (A \times C)$

Yoneda

Recap:

With a categorification, we understand that

- $a \times (b + c) = (a \times b) + (a \times c)$ is the same as saying $A \times (B + C) \cong (A \times B) + (A \times C)$

And the Yoneda lemma helps us realize that this is equivalent to the following statement:

- $\mathbf{C}(A \times (B + C), X) \cong \mathbf{C}((A \times B) + (A \times C), X)$ for all X.

Plan

If we want to prove the above, we should want to break it up and study each component on their own. First, we need to learn about the nature of products and coproducts.

Meditations on $\times$

Suppose we wanted to talk about $\mathbf{C}(A \times B, X)$.

Meditations on $\times$

Q: What does a function $A \times B \to X$ look like?

Meditations on $\times$

Q: What does a function $A \times B \to X$ look like?
A: Well, for one, we don't really have a definition for products yet!
Lets define it.

Meditations on $\times$

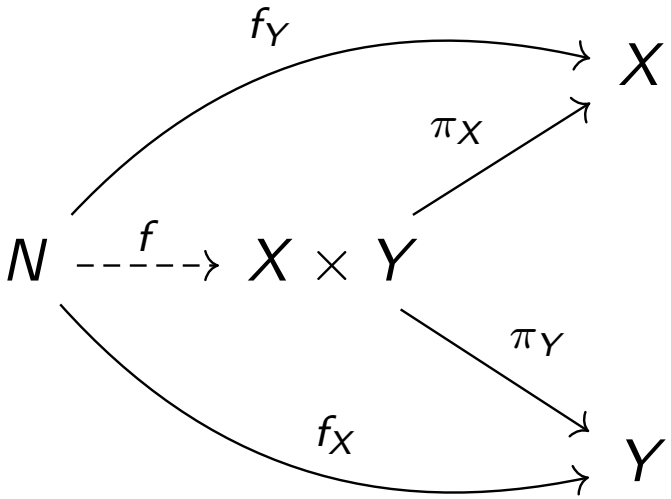
Definition

Product Let $\mathbf{C}$ be a category, and let $X, Y \in \mathbf{C}$. A product P is an object of $\mathbf{C}$ together with functions $\pi_X : P \to X$ and $\pi_Y : P \to Y$ such that the following holds:

- For any other $N \in \mathbf{C}$ and morphisms $f_X : N \to X$ and $f_Y : N \to Y$, we have that a unique, induced morphism $f : N \to P$ such that $f_X = \pi_X f$ and $f_Y = \pi_Y f$.

P is usually denoted $X \times Y$.

Or, more succinctly, the following diagram commutes for any given N and morphisms f_Y, f_X:



Meditations on $\times$

So, back to the question.

Meditations on $\times$

A function $f : A \times B \to X$ contains the following data:

Meditations on $\times$

A function $f : A \times B \to X$ contains the following data:

- For $A \times B$, the function f must specify an element $f(a, b) \in X$.
- Additionally, for any $a \in A$, the function $f(a, -)$ must specify a function $B \to X$ sending b to $f(a, b)$

Meditations on $\times$

Thus, all that needed in order to define a function $A \times B \to X$ is the data that sends a $a \in A$ to the partial application of the function f to form a function $A \to \mathbf{C}(B, X) : a \mapsto f(a, -)$.

Meditations on $\times$

This should look familiar. *Because it's currying*. The full statement of currying is the following:

Currying

$$\mathbf{C}(A \times B, C) \cong \mathbf{C}(A, \mathbf{C}(B, X))$$

Meditations on $+$

Likewise, we can ask what the function $B + C \to X$ looks like.

Meditations on $+$

First, let us define coproducts.

Meditations on $+$

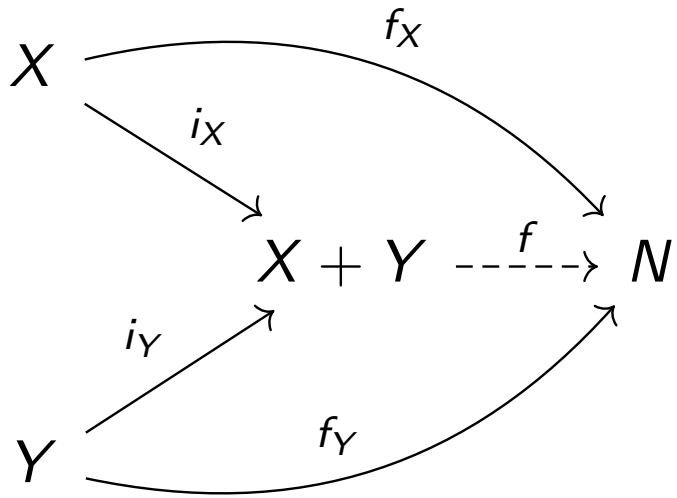
Definition

Coproduct Let C be a category and let $X, Y \in \mathbf{C}$. A *coproduct* C is an object of $\mathbf{C}$ together with functions $i_X : X \to C$ and $i_Y : Y \to C$ such that the following holds:

- For any other $N \in \mathbf{C}$, and morphisms $f_X : X \to N$ and $f_Y : Y \to N$, there is a unique, induced morphism $f : C \to N$ such that $f_X = fi_X$ and $f_Y = fi_Y$.

The coproduct of C is generally denoted $X + Y$

Or, more succinctly, the following diagram commutes for any given N and morphisms f_Y, f_X:



Meditations on $+$

Note that all that's needed to define a function $f : B + C \to X$ is then to define a pair of functions $g : B \to X$ and $h : C \to X$ associating each $b \in B$ and $c \in C$ with an element of X.

Meditations on $+$

Note that all that's needed to define a function $f : B + C \to X$ is then to define a pair of functions $g : B \to X$ and $h : C \to X$ associating each $b \in B$ and $c \in C$ with an element of X.
Thus, we obtain the following:

Pairing

$$\mathbf{C}(B + C, X) \cong \mathbf{C}(B, X) \times \mathbf{C}(C, X)$$

Representability

Note the duality of it all! The diagrams obtained in the previous slides are dual universal properties of the product and coproduct respectively.

Representability

Q: But what do we mean by "representability" here?

Representability

Q: But what do we mean by "representability" here?
A: We mean actions of the natural isomorphisms.

Representability

In fact, we can make use of these natural isomorphisms because we have been working with things called representable functors!

Representability

Definition (Functor Category)

Let $\mathbf{C}$ and $\mathbf{D}$ be categories. A *functor category*, denoted $[\mathbf{C}, \mathbf{D}]$, is defined to to be a category with functors $F : \mathbf{C} \to \mathbf{D}$ as objects and natural transformations as morphisms.

Representability

Definition (Representable Functors)

A functor $F \in [\mathbf{C}, \mathbf{Set}]$ is called *representable* if it is naturally isomorphic to a hom-functor $\mathbf{C}(A, -)$.

Representability

This is interesting. Notice how similar it looks to the Yoneda lemma above!

Representability

Indeed, the definition of Yoneda can be refactored into a statement about the class of natural isomorphisms $\mathbf{C}(A, -) \cong F$.

Definition (Yoneda Lemma)

Let $F \in [\mathbf{C}, \mathbf{Set}]$ be a representable functor, and let $A \in C$. The set of natural transformations $[\mathbf{C}, \mathbf{Set}](\mathbf{C}(A, -), F)$ is in bijection with FA for all $A \in \mathbf{C}$. This bijection associates a morphism (natural transformation) $\alpha : \mathbf{C}(A, -) \Rightarrow F$ to the identity $\alpha(1_A) \in FA$

Representability

Indeed, the statement actually holds for $F \in [\mathbf{C}^{op}, \mathbf{Set}]$ as well, only, since the variance is flipped, $FX \cong \mathbf{C}^{op}(-, X)$ is in correspondence.

Representability

It turns out that our work has been about the correspondence between representable functors and natural isomorphisms all along!

Proof

We can now prove the original statement:

$$a \times (b + c) = (a \times b) + (a \times c)$$

Proof

We can now prove the original statement:

$$a \times (b + c) = (a \times b) + (a \times c)$$

Proof

Proof.

$$
\begin{aligned}
a \times (b + c) &\cong A \times (B + C) && (\textit{categorification}) \\
&\cong \mathbf{C}((B + C) \times A, X) && (\textit{swap} + \textit{yoneda}) \\
&\cong \mathbf{C}(B + C, \mathbf{C}(A, X)) && (\textit{curry}) \\
&\cong \mathbf{C}(B, \mathbf{C}(A, X))) \times \mathbf{C}(C, \mathbf{C}(A, X)) && (\textit{pairing}) \\
&\cong \mathbf{C}(A \times B, X) \times \mathbf{C}(A \times C, X) && (\textit{uncurry}) \\
&\cong \mathbf{C}((A \times B) + (A \times C), X) && (\textit{unpair}) \\
&\cong (A \times B) + (A \times C) && (\textit{yoneda}) \\
&\cong (a \times b) + (a \times c) && (\textit{decategorification})
\end{aligned}
$$

Adjunctions

This section is optional, but a wonderful insight into the nature of duality and the previously discussed theory.

Adjunctions

This section is optional, but a wonderful insight into the nature of duality and the previously discussed theory.

Adjunctions

"Adjoint Functors arise everywhere" - Saunders MacLane

Adjunctions

In fact, they are more than pertinent to the study of functional programming, as it is a classic theorem of adjoint functors (also called adjunctions) that every adjunction gives rise to a monad/comonad pair.

Adjunctions

Lets begin with some definitions, and build towards adjunctions.

Adjunctions

Definition (Diagram)