The Universal Presence of Category Theory

Gabriel Field

26/July/2024

Initial thanks

My thanks to:

~~Math~~ *CT* space	for realising the title of this presentation
Peter Gow	for helping me understand CT
MSS	for providing me a platform
You	for listening to me

Why this presentation?

There's been a big shift in mathematics towards analysing objects using functions, maps or other "arrows" between them.

The modern formalism for these ideas is category theory.

Why this presentation?

There's been a big shift in mathematics towards analysing objects using functions, maps or other "arrows" between them.

The modern formalism for these ideas is category theory.

Because it's "abstract nonsense", category theory is said to be:

Why this presentation?

There's been a big shift in mathematics towards analysing objects using functions, maps or other "arrows" between them.

The modern formalism for these ideas is category theory.

Because it's "abstract nonsense", category theory is said to be:

- Hard to get the gist of

Why this presentation?

There's been a big shift in mathematics towards analysing objects using functions, maps or other "arrows" between them.

The modern formalism for these ideas is category theory.

Because it's "abstract nonsense", category theory is said to be:

- Hard to get the gist of
- Inaccessible without sophisticated mathematical background

Why this presentation?

There's been a big shift in mathematics towards analysing objects using functions, maps or other "arrows" between them.

The modern formalism for these ideas is category theory.

Because it's "abstract nonsense", category theory is said to be:

- Hard to get the gist of
- Inaccessible without sophisticated mathematical background
- Not immediately applicable

Why this presentation?

There's been a big shift in mathematics towards analysing objects using functions, maps or other "arrows" between them.

The modern formalism for these ideas is category theory.

Because it's "abstract nonsense", category theory is said to be:

- Hard to get the gist of
- Inaccessible without sophisticated mathematical background
- Not immediately applicable

Main goal

To informally highlight some of the

context behind **basics of** **applications of**

category theory, so that it isn't as intimidating next time.

Outline

Outline

How do graphs fit together?

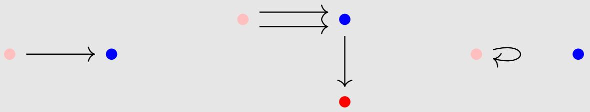
Definition (Graph)

A **graph** G consists of:

a set $V(G)$ of *vertices*, a set $E(G)$ of *edges*

such that each edge has a specified *start* and *end* vertex.

Example



How do graphs fit together?

Remark. We care about the *structure* within a graph, not:

- how the vertices/edges are labelled
- how the vertices/edges are arranged in space

Example

$\bullet \overset{a}{\underset{b}{\rightrightarrows}} \bullet$ is interchangeable with $\bullet \overset{\text{cat}}{\underset{\text{dog}}{\rightrightarrows}} \bullet$

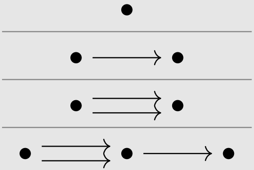
How do graphs fit together?

Question

How do graphs "fit together" amongst each other?

Example

Each graph "forms part of" the graph(s) below it:



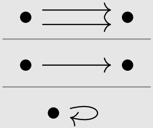
How do graphs fit together?

Question

How do graphs "fit together" amongst each other?

Example

Each graph "collapses onto" the graph(s) below it:



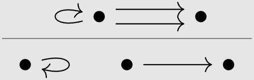
How do graphs fit together?

Question

How do graphs "fit together" amongst each other?

Example

The top graph "collapses onto part of" the graph below it:



How are these relationships encoded?

How do graphs fit together?

Definition (Graph homomorphism)

A **graph homomorphism** $f : G \to H$ consists of:

- A function $V(G) \to V(H)$ (called f)
- A function $E(G) \to E(H)$ (also called f)

such that

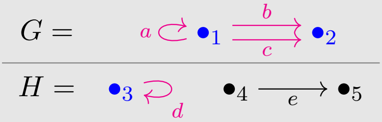
$$
\begin{array}{ccc}
G & \xrightarrow{\ f\ } & H \\
\begin{array}{c} u \\ e \big\downarrow \\ v \end{array} & \xmapsto{\ f\ } & \begin{array}{c} f(u) \\ \big\downarrow f(e) \\ f(v) \end{array}
\end{array}
$$

A graph homomorphism $G \to H$ tells us *how* G "collapses onto part of" H.

How do graphs fit together?

Example

G collapses onto part of H…



…witnessed by $f : G \to H$ with

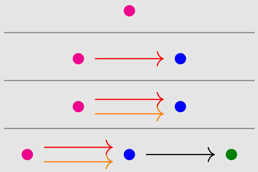
$$f(\bullet_1) = f(\bullet_2) = \bullet_3$$

$$f\left(a \circlearrowleft\right) = f\left(\xrightarrow{b}\right) = f\left(\xrightarrow[c]{}\right) = \circlearrowright d$$

How do graphs fit together?

When $f : G \rightarrowtail H$ is *injective*, there is no “collapse”.
f is a way in which “H is seen from the perspective of G”.

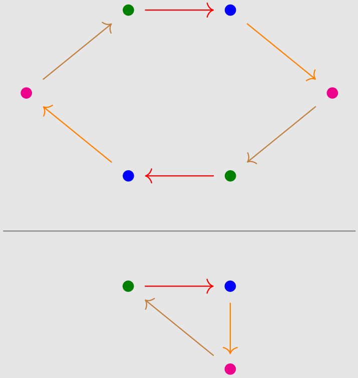
Example



How do graphs fit together?

When $f : G \twoheadrightarrow H$ is *surjective*, then “H is a (perhaps degenerate) version of G”.

Example



A 3-cycle, traversed twice, is a degenerate 6-cycle.

How do graphs fit together?

Observation

Graph homomorphisms give us a powerful way to understand how graphs mimic one another.

How do graphs fit together?

Observation

Graph homomorphisms give us a powerful way to understand how graphs mimic one another.

The same game can be played with:

- Groups and group homomorphisms (e.g. $\mathbb{Z}_{/4} \twoheadrightarrow \mathbb{Z}_{/2}$)
- Topological spaces and continuous maps (e.g. paths are maps out of $[0, 1]$)
- Sets and functions (e.g. $|X| \leq |Y|$ iff $X \rightarrowtail Y$)

In all cases, maps are more important than objects.

Definition (Category)

A **category** $\mathcal{C}$ consists of:

- A collection $\mathrm{ob}(\mathcal{C})$ of **objects**
- $\forall a, b \in \mathrm{ob}(\mathcal{C})$, a collection $\mathcal{C}(a, b)$ of **arrows** from a to b
- $\forall a \in \mathrm{ob}(\mathcal{C})$, an **identity arrow** $1_a \in \mathcal{C}(a, a)$
- $\forall a, b, c \in \mathrm{ob}(\mathcal{C})$, a **composition function**

$$\mathcal{C}(b, c) \times \mathcal{C}(a, b) \xrightarrow{\circ} \mathcal{C}(a, c)$$
$$(g, f) \longmapsto g \circ f$$

Such that for all $a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{h} d$ in $\mathcal{C}$,

$$1_b \circ f = f \circ 1_a = f \qquad h \circ (g \circ f) = (h \circ g) \circ f$$

Remark. The definition of a category emphasises the arrows. "Whenever you introduce new objects, you should specify the arrows between them."

Example

$\mathbf{Set}$ has:

- objects are *sets*
- arrows $X \to Y$ are *functions* $X \to Y$
- identity arrows id_X are *identity functions*
- composition $g \circ f$ is *function composition*

"Structures and structure-preserving maps":

$\mathbf{Monoid}$	$\mathbf{Grp}$	$\mathbf{Ab}$
$\mathbf{Vect}$	$\mathbf{Graph}$	$\mathbf{Top}$

Categories

Example

A poset $(P, \leq)$ determines a category $\mathcal{P}$ where:

- objects are *elements of* P
- if $x \leq y$, then there is a unique arrow $x \to y$ in $\mathcal{P}$
- identity arrows correspond to reflexivity $(x \leq x)$
- composites correspond to transitivity $(x \leq y \leq z \implies x \leq z)$

Categories

Example

A poset $(P, \leq)$ determines a category $\mathcal{P}$ where:

- objects are *elements of* P
- if $x \leq y$, then there is a unique arrow $x \to y$ in $\mathcal{P}$
- identity arrows correspond to reflexivity $(x \leq x)$
- composites correspond to transitivity $(x \leq y \leq z \implies x \leq z)$

A group $(G, \cdot)$ determines a category $\mathrm{B}G$ where:

- there is one object $*$
- $\mathrm{B}G(*, *) = G$
- the identity 1_* is the identity element $e \in G$
- composition $h \circ g := h \cdot g$ uses the group operation

Categories

Category theory provides language and insight for compositionality.

Instantiating general category-theoretic ideas in any particular category often yields interesting results.

Outline

Monomorphisms and epimorphisms

Recall that an *injective* graph hom. captures *substructure*, whereas a *surjective* graph hom. captures *(degenerate) quotient structure*.

Monomorphisms and epimorphisms

Recall that an *injective* graph hom. captures *substructure*, whereas a *surjective* graph hom. captures *(degenerate) quotient structure*.

Definition (Monomorphism, Epimorphism)

An arrow $a \xrightarrow{m} b$ is a **monomorphism** just when

$$\forall \left(x \underset{g}{\overset{f}{\rightrightarrows}} a \xrightarrow{m} b \right), \quad m \circ f = m \circ g \implies f = g$$

Monomorphisms and epimorphisms

Recall that an *injective* graph hom. captures *substructure*, whereas a *surjective* graph hom. captures *(degenerate) quotient structure*.

Definition (Monomorphism, Epimorphism)

An arrow $a \xrightarrow{m} b$ is a **monomorphism** just when

$$\forall \left(x \overset{f}{\underset{g}{\rightrightarrows}} a \xrightarrow{m} b \right), \quad m \circ f = m \circ g \implies f = g$$

Dually, an arrow $a \xrightarrow{e} b$ is an **epimorphism** just when

$$\forall \left(a \xrightarrow{e} b \overset{f}{\underset{g}{\rightrightarrows}} x \right), \quad f \circ e = g \circ e \implies f = g$$

Monomorphisms and epimorphisms

Example

In $\mathbf{Set}$, any subset yields an inclusion map:

$$A \subseteq X \implies \begin{array}{rcl} A & \xrightarrow{\iota} & X \\ a & \longmapsto & a \end{array}$$

This map is *monic*.

Monomorphisms and epimorphisms

Example

In $\mathbf{Set}$, any subset yields an inclusion map:

$$A \subseteq X \implies \begin{aligned} A &\xrightarrow{\iota} X \\ a &\longmapsto a \end{aligned}$$

This map is *monic*.

In $\mathbf{Grp}$, any normal subgroup yields a quotient map:

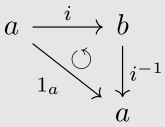
$$N \trianglelefteq G \implies \begin{aligned} G &\xrightarrow{\pi} G/N \\ g &\longmapsto gN \end{aligned}$$

This map is *epic*.

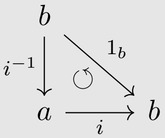
Isomorphisms

Definition (Isomorphism)

An arrow $a \xrightarrow{i} b$ is an **isomorphism** just when $\exists b \xrightarrow{i^{-1}} a$ such that



and



(i.e. $i^{-1} \circ i = 1_a$ and $i \circ i^{-1} = 1_b$.)

Example

Set	Bijection
Top	Homeomorphism
Grp	Isomorphism
Graph	Isomorphism

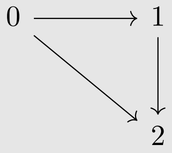
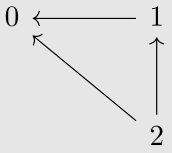
Duality

Definition (Opposite category)

Given a category $\mathcal{C}$, its **opposite category** $\mathcal{C}^{\text{op}}$ consists of:

- Objects: same as $\mathcal{C}$
- Arrows $a \to b$ in $\mathcal{C}^{\text{op}}$: arrows $b \to a$ in $\mathcal{C}$
- Identities: same as $\mathcal{C}$
- Composition: $g \circ^{\text{op}} f := f \circ g$.

Example

$\mathcal{C} = (\{0,1,2\}, \leq)$	$\mathcal{C}^{\text{op}} = (\{0,1,2\}, \geq)$
	

Duality

"Interesting" structures in $\mathcal{C}^{\mathrm{op}}$ are "interesting" in $\mathcal{C}$, too.

Example

Let $b \xrightarrow{e} a$ be a monomorphism in $\mathcal{C}^{\mathrm{op}}$. That is,

$$\forall \left(x \underset{g}{\overset{f}{\rightrightarrows}} b \xrightarrow{e} a \right) \text{ in } \mathcal{C}^{\mathrm{op}}, \quad e \circ^{\mathrm{op}} f = e \circ^{\mathrm{op}} g \implies f = g$$

Duality

"Interesting" structures in $\mathcal{C}^{\mathrm{op}}$ are "interesting" in $\mathcal{C}$, too.

Example

Let $b \xrightarrow{e} a$ be a monomorphism in $\mathcal{C}^{\mathrm{op}}$. That is,

$$\forall \left(x \underset{g}{\overset{f}{\rightrightarrows}} b \xrightarrow{e} a \right) \text{ in } \mathcal{C}^{\mathrm{op}}, \quad e \circ^{\mathrm{op}} f = e \circ^{\mathrm{op}} g \implies f = g$$

Turning all the arrows around,

$$\forall \left(a \xrightarrow{e} b \underset{g}{\overset{f}{\rightrightarrows}} x \right) \text{ in } \mathcal{C}, \quad f \circ e = g \circ e \implies f = g$$

So a monomorphism in $\mathcal{C}^{\mathrm{op}}$ is an epimorphism in $\mathcal{C}$.

"Interesting" structures in $\mathcal{C}^{\mathrm{op}}$ are "interesting" in $\mathcal{C}$, too.

Example

Let $b \xrightarrow{e} a$ be a monomorphism in $\mathcal{C}^{\mathrm{op}}$. That is,

$$\forall \left(x \underset{g}{\overset{f}{\rightrightarrows}} b \xrightarrow{\;e\;} a \right) \text{ in } \mathcal{C}^{\mathrm{op}}, \quad e \circ^{\mathrm{op}} f = e \circ^{\mathrm{op}} g \implies f = g$$

Turning all the arrows around,

$$\forall \left(a \xrightarrow{\;e\;} b \underset{g}{\overset{f}{\rightrightarrows}} x \right) \text{ in } \mathcal{C}, \quad f \circ e = g \circ e \implies f = g$$

So a monomorphism in $\mathcal{C}^{\mathrm{op}}$ is an epimorphism in $\mathcal{C}$.

Remark. An iso. in $\mathcal{C}^{\mathrm{op}}$ is an iso. in $\mathcal{C}$.

Duality

Duality principle

In any statement that says "$\forall$ categories $\mathcal{C}$", you can replace $\mathcal{C}$ with $\mathcal{C}^{\mathrm{op}}$.

Duality

Duality principle

In any statement that says "$\forall$ categories $\mathcal{C}$", you can replace $\mathcal{C}$ with $\mathcal{C}^{\mathrm{op}}$.

Example

Exercise: in any category $\mathcal{C}$, any isomorphism is monic.

$\Downarrow$ dual (isomorphism) $\Downarrow$ dual (monic)

Therefore: in any category $\mathcal{C}$, any isomorphism is epic.

Functorality

Functorality and naturality were the motive for the first paper in category theory [1].

Functorality

Functorality and naturality were the motive for the first paper in category theory [1].
"Whenever you introduce new objects, you should specify the arrows between them."

Functorality

Functorality and naturality were the motive for the first paper in category theory [1].
"Whenever you introduce new objects, you should specify the arrows between them."

Definition (Functor)

Let $\mathcal{C}, \mathcal{D}$ be categories. A **functor** $F : \mathcal{C} \to \mathcal{D}$ consists of:

- A function $\mathrm{ob}(\mathcal{C}) \to \mathrm{ob}(\mathcal{D})$ (called F)
- $\forall c, c' \in \mathcal{C}$, a function $\mathcal{C}(c, c') \to \mathcal{D}(F(c), F(c'))$ (also F)

subject to:

$$F(1_c) = 1_{F(c)} \qquad\qquad F(g \circ f) = F(g) \circ F(f)$$

Functorality

Example

$$\begin{array}{ccc} \mathbf{Graph} & \xrightarrow{V} & \mathbf{Set} \\ G & & V(G) \\ f \downarrow & \longmapsto & \downarrow f \\ H & & V(H) \end{array}$$

Observe:

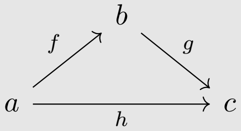
$$1_G \overset{V}{\longmapsto} 1_{V(G)}$$
$$g \circ f \overset{V}{\longmapsto} g \circ f$$

so retrieving vertex sets is *functorial*.

Functorality

Definition (Commutative diagram)

A **commutative diagram** in $\mathcal{C}$ is a graph of objects and arrows, e.g.

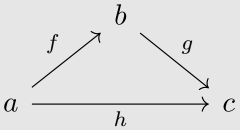


such that any two parallel paths have equal composites (e.g. $g \circ f = h$).

Functorality

Definition (Commutative diagram)

A **commutative diagram** in $\mathcal{C}$ is a graph of objects and arrows, e.g.



such that any two parallel paths have equal composites (e.g. $g \circ f = h$).

Lemma (Functors preserve commutative diagrams)

Take a commutative diagram in $\mathcal{C}$, and apply a functor $F : \mathcal{C} \to \mathcal{D}$ to all objects and arrows in the diagram. Then, the resulting diagram commutes in $\mathcal{D}$.

Functorality

Lemma (Functors preserve commutative diagrams)

Example

An isomorphism $i : a \simeq b$ in $\mathcal{C}$:

$$\begin{array}{ccc} a & \xrightarrow{i} & b \\ & {\scriptstyle 1_a} \searrow & \downarrow {\scriptstyle i^{-1}} \\ & & a \end{array} \quad \text{and} \quad \begin{array}{ccc} b & & \\ {\scriptstyle i^{-1}} \downarrow & \searrow {\scriptstyle 1_b} & \\ a & \xrightarrow[i]{} & b \end{array} \quad \text{in } \mathcal{C}$$

Applying a functor $F : \mathcal{C} \to \mathcal{D}$, we get *commutative* diagrams

$$\begin{array}{ccc} F(a) & \xrightarrow{F(i)} & F(b) \\ & {\scriptstyle 1_{F(a)}} \searrow & \downarrow {\scriptstyle F(i^{-1})} \\ & & F(a) \end{array} \quad \text{and} \quad \begin{array}{ccc} F(b) & & \\ {\scriptstyle F(i^{-1})} \downarrow & \searrow {\scriptstyle 1_{F(b)}} & \\ F(a) & \xrightarrow[F(i)]{} & F(b) \end{array} \quad \text{in } \mathcal{D}$$

which says that $F(i)$ is an isomorphism $F(a) \simeq F(b)$ in $\mathcal{D}$.

Functorality

Any category $\mathcal{C}$ has an *identity functor*

$$1_{\mathcal{C}} : \quad c \xrightarrow{f} d \quad \longmapsto \quad c \xrightarrow{f} d$$

Functors $\mathcal{A} \xrightarrow{F} \mathcal{B} \xrightarrow{G} \mathcal{C}$ compose:

$$\left(a \xrightarrow{f} a'\right) \quad \xmapsto{G \circ F} \quad \left(G(F(a)) \xrightarrow{G(F(f))} G(F(b))\right)$$

Functorality

Any category $\mathcal{C}$ has an *identity functor*

$$1_{\mathcal{C}} : \quad c \xrightarrow{f} d \quad \longmapsto \quad c \xrightarrow{f} d$$

Functors $\mathcal{A} \xrightarrow{F} \mathcal{B} \xrightarrow{G} \mathcal{C}$ compose:

$$\left(a \xrightarrow{f} a'\right) \quad \xmapsto{G \circ F} \quad \left(G(F(a)) \xrightarrow{G(F(f))} G(F(b))\right)$$

Definition (Cat)

Cat is the category with

- objects: categories
- arrows: functors

and the above identities and composites.

Naturality

"Whenever you introduce new objects, you should specify the arrows between them."

Naturality

"Whenever you introduce new objects, you should specify the arrows between them."

Definition (Natural transformation)

Let $F, G : \mathcal{C} \to \mathcal{D}$ be functors. A **natural transformation** $\alpha : F \Rightarrow G$ consists of:

- A family $\alpha_c : F(c) \to G(c)$ of arrows in $\mathcal{D}$, for each $c \in \mathcal{C}$

such that

$$\forall a \xrightarrow{\forall f} \forall b \qquad \text{in } \mathcal{C}$$

$$\begin{array}{ccc} F(a) & \xrightarrow{F(f)} & F(b) \\ {\scriptstyle \alpha_a}\big\downarrow & & \big\downarrow{\scriptstyle \alpha_b} \\ G(a) & \xrightarrow[G(f)]{} & G(b) \end{array} \qquad \text{in } \mathcal{D}$$

commutes.

Naturality

Remark. **Cat** has:

- objects $\mathcal{C}, \mathcal{D}, \ldots$
- arrows $F, G, \ldots : \mathcal{C} \to \mathcal{D}$
- *2-cells* $\alpha, \ldots : F \Rightarrow G$

which makes it a 2-category.

Universality

$$\forall \text{thing}, \exists! \text{otherThing} : \text{ condition}$$

Universality

$$\forall \text{thing}, \exists! \text{otherThing} : \text{ condition}$$

Example

Let $V, W \in \mathbf{Vect}_K$ and let β be a basis for V.

$$\forall f : \beta \xrightarrow{\text{function}} W,$$

$$\exists! T : V \xrightarrow{\text{linear}} W :$$

$$T|_\beta = f$$

Universality

$$\forall \text{thing}, \exists! \text{otherThing} : \text{ condition}$$

Example

Let $V, W \in \mathbf{Vect}_K$ and let β be a basis for V.

$$\forall f : \beta \xrightarrow{\text{function}} W,$$
$$\exists! T : V \xrightarrow{\text{linear}} W :$$
$$T|_\beta = f$$

$$\mathbf{Vect}_K\left(\operatorname{span}_K(\beta), W\right) \simeq \mathbf{Set}(\beta, W)$$

natural in β, W; **adjunction**.

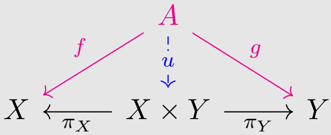
Universality

$$\forall \text{thing}, \exists! \text{otherThing} : \text{ condition}$$

Example

Let $X, Y \in \mathbf{Set}$. The product projections $X \xleftarrow{\pi_X} X \times Y \xrightarrow{\pi_Y} Y$ satisfy

$\forall A, \forall f, \forall g, \exists! u :$



commutes

$(u : a \mapsto (f(a), g(a)).)$ **Limit**.

Remark. The p-adic numbers are the colimit of $\mathbb{Z}_{/p} \hookrightarrow \mathbb{Z}_{/p^2} \hookrightarrow \cdots$

$$\forall \text{thing}, \exists! \text{otherThing} : \text{condition}$$

Example

There is a bijection

$$(n\text{-colour vertex colourings})(G) \simeq \mathbf{Graph}(G, K_n)$$

natural in G. Vertex colourings are **representable**, and form the **universal property** of K_n.

All the universal properties come in this last form.

Yoneda lemma

Scary version:

$$\mathbf{Set}^{\mathcal{C}}\left(\mathcal{C}(x,-), F\right) \simeq_{\mathsf{nat}.F,x} F(x)$$

Useful version:

Lemma (Yoneda)

$$\begin{aligned} &\mathcal{C}(a,x) \simeq_{\mathsf{nat}.x} \mathcal{C}(b,x) \\ \implies \quad & a \simeq_{\mathcal{C}} b \end{aligned}$$

Consequences:

- "Equational" proofs
- Easier handling of universal properties

Outline

Functional programming

Functional programming is a paradigm where nothing can change value.

Example

Imperative program:

```
x := 1; printf("%d", x);
x := 2; printf("%d", x);
// Wdym; 1 != 2?!
// This is clearly evil.
```

Functional programming

Functional programming is a paradigm where nothing can change value.

Example

Imperative program:

```
x := 1; printf("%d", x);
x := 2; printf("%d", x);
// Wdym; 1 != 2?!
// This is clearly evil.
```

Functional program:

```
main = do
  putStrLn "1"
  putStrLn "2"
-- Much "better"
```

Functional programming

Example

Loops aren't functional:

```
for(int i = 0; i < 10; i++) {
  printf("%d", i);
}
// Variable i changes value!
```

Recursion is functional:

```
printer 9 = putStrLn "9"
printer n = putStrLn (show n) >> printer (n + 1)
main = printer 0
```

Functional programming languages are *very good* for recursion.

Functional programming

Imperative:	Functional:
Data types	Data types
Functions	Functions
Variables	*More* functions
Loops	**More** functions
Error handling	**More** functions
. . .	. . .

Functional programming

Imperative:	Functional:
Data types	Data types
Functions	Functions
Variables	*More* functions
Loops	**More** functions
Error handling	**More** functions
...	...

Reasoning about imperative programs is hard.
Reasoning about functional programs is easier.

Categorical influence

Haskell has an associated category $\mathbf{Hask}$, with

- Objects: Data types $A, B, \ldots$
- Arrows $A \to B$: *Computable* functions $A \to B$

Pretend $\mathbf{Hask} = \mathbf{Set}$ if you want to.

For technical (cringe) reasons, this isn't actually a category.

Categorical influence: Algebraic data types

There are some built-in data types:

- `Integer`
- `Char`
- etc.

There's a really cool way to build more.

Example

Categorical influence: Algebraic data types

There are some built-in data types:

- `Integer`
- `Char`
- etc.

There's a really cool way to build more.

Example

```
data IntegerAndChar = ThePair Integer Char
```

Categorical influence: Algebraic data types

There are some built-in data types:

- `Integer`
- `Char`
- etc.

There's a really cool way to build more.

Example

```
data IntegerAndChar = ThePair Integer Char
data IntegerOrChar = Left Integer | Right Char
```

Categorical influence: Algebraic data types

There are some built-in data types:

- `Integer`
- `Char`
- etc.

There's a really cool way to build more.

Example

```
data IntegerAndChar = ThePair Integer Char
data IntegerOrChar = Left Integer | Right Char
data ListInteger = Empty | Both Integer ListInteger
```

Categorical influence: Algebraic data types

There are some built-in data types:

- `Integer`
- `Char`
- etc.

There's a really cool way to build more.

Example

```
data IntegerAndChar = ThePair Integer Char
data IntegerOrChar = Left Integer | Right Char
data ListInteger = Empty | Both Integer ListInteger
```

Example

```
data Pair   a b = ThePair a b
data Either a b = Left    a   | Right b
data [a]        = []          | a : [a]
```

Categorical influence: Algebraic data types

Example

```
data Pair   a b = ThePair a b
data Either a b = Left    a   | Right b
data [a]        = []          | a : [a]
```

Polymorphic data types are functors:

$$\texttt{Pair} : \mathbf{Hask} \times \mathbf{Hask} \longrightarrow \mathbf{Hask}$$
$$\texttt{Either} : \mathbf{Hask} \times \mathbf{Hask} \longrightarrow \mathbf{Hask}$$
$$[-] : \mathbf{Hask} \longrightarrow \mathbf{Hask}$$

Categorical influence: Algebraic data types

Example

```
data Pair   a b = ThePair a b
data Either a b = Left    a   | Right b
data [a]        = []          | a : [a]
```

Polymorphic data types are functors:

$$\texttt{Pair} : \mathbf{Hask} \times \mathbf{Hask} \longrightarrow \mathbf{Hask}$$
$$\texttt{Either} : \mathbf{Hask} \times \mathbf{Hask} \longrightarrow \mathbf{Hask}$$
$$[-] : \mathbf{Hask} \longrightarrow \mathbf{Hask}$$

Example

```
(+1) <$>     [] =     []
(+1) <$>   1:[] =   2:[]
(+1) <$> 1:2:[] = 2:3:[]
```

Categorical influence: Algebraic data types

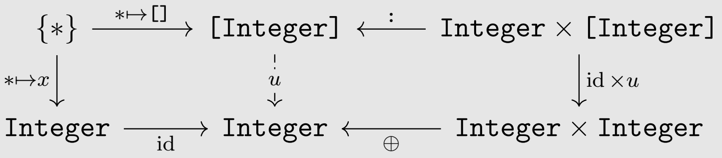
Example

```
data Pair    a b = ThePair a b
data Either a b = Left    a     | Right b
data [a]        = []            | a : [a]
```

Polymorphic data types have a universal property:

Example

$\forall x, \forall \oplus, \exists! u :$



Categorical influence: Algebraic data types

Example

```
data Pair   a b = ThePair a b
data Either a b = Left    a   | Right b
data [a]        = []          | a : [a]
```

Polymorphic data types have a universal property:

Example

$\forall x, \forall \oplus, \exists ! u :$

$$
\begin{array}{ccccc}
\{*\} & \xrightarrow{* \mapsto []} & [\texttt{Integer}] & \xleftarrow{:} & \texttt{Integer} \times [\texttt{Integer}] \\
{\scriptstyle * \mapsto x}\downarrow & & \downarrow{\scriptstyle !\, u} & & \downarrow{\scriptstyle \mathrm{id} \times u} \\
\texttt{Integer} & \xrightarrow[\mathrm{id}]{} & \texttt{Integer} & \xleftarrow[\oplus]{} & \texttt{Integer} \times \texttt{Integer}
\end{array}
$$

With $x := 0$ and $\oplus := +$, we get $u = \texttt{sum}$.

Categorical influence: Monads

Error handling in Haskell is sick.

Maybe

```
data Maybe a = Nothing | Just a
```

Categorical influence: Monads

Error handling in Haskell is sick.

Maybe

```
data Maybe a = Nothing | Just a
```

As a functor, `Maybe` : $\mathbf{Hask} \to \mathbf{Hask}$.

Categorical influence: Monads

Error handling in Haskell is sick.

Maybe

```
data Maybe a = Nothing | Just a
```

As a functor, $\texttt{Maybe} : \mathbf{Hask} \to \mathbf{Hask}$.
But really, `Maybe` fakes the behaviour

$$\mathbf{Hask} \longrightarrow \mathbf{Hask}_*$$

by sending A to $(A \sqcup \{\texttt{Nothing}\}, \texttt{Nothing})$.
The categorical way of "faking" this behaviour is a monad.

Categorical influence: Monads

Example

```
main = do
  putStrLn "gimme a number"
  number <- (read @Integer) <$> getLine
  putStrLn $ "you said " ++ show number

>> main
out> gimme a number
in>  69
out> you said 69
```

Monoidal categories

Categories tell us about moving from one object to another.
Monoidal categories tell us about combining objects together.

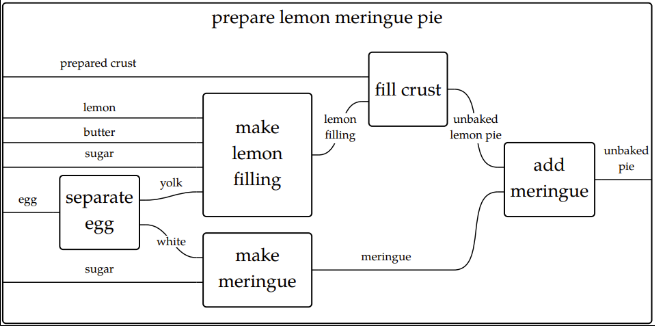


Figure: Wiring diagram for preparing a lemon meringue pie

src: Fong and Spivak's *Seven Sketches* [2]

Enrichment

Monoidal categories serve as bases for enriched categories.

Definition ($\mathcal{V}$-Category)

Fix a monoidal category $\mathcal{V}$.
A $\mathcal{V}$-**category** $\mathcal{C}$ consists of:

- A collection $\mathrm{ob}(\mathcal{C})$ of **objects**
- $\forall a, b \in \mathrm{ob}(\mathcal{C})$, a **hom-object** $\mathcal{C}(a, b) \in \mathcal{V}$
- *(and identities and composition)*

subject to some coherence conditions

Key point: Replace set of arrows $\mathcal{C}(a, b)$ with $\mathcal{V}$-thing of arrows $\mathcal{C}(a, b)$.

Enrichment

Example

$\mathbf{Bool} = (\texttt{false} \to \texttt{true})$ is a monoidal category.
A $\mathbf{Bool}$-category is a preorder:

$$a \leq b \iff \mathcal{C}(a, b) = \texttt{true}$$

Enrichment

Example

$\mathbf{Bool} = (\texttt{false} \to \texttt{true})$ is a monoidal category.
A $\mathbf{Bool}$-category is a preorder:

$$a \leq b \iff \mathcal{C}(a, b) = \texttt{true}$$

Example

An $\mathbf{Ab}$-category is an **additive category**: $\mathcal{C}(a, b)$ is an abelian group, so arrows $f, g : a \to b$ have a *sum* $f + g : a \to b$.

Enrichment

Example

$\mathbf{Bool} = (\texttt{false} \to \texttt{true})$ is a monoidal category.
A $\mathbf{Bool}$-category is a preorder:

$$a \leq b \iff \mathcal{C}(a, b) = \texttt{true}$$

Example

An $\mathbf{Ab}$-category is an **additive category**: $\mathcal{C}(a, b)$ is an abelian group, so arrows $f, g : a \to b$ have a *sum* $f + g : a \to b$.

Example

A $\mathbf{Cat}$-category is a 2-category; e.g. $\mathbf{Cat}$.

Abelian categories

Definition (Abelian category)

An **abelian category** is an $\mathbf{Ab}$-category with the **first isomorphism theorem**.

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow\!\!\downarrow & & \uparrow \\ \operatorname{coim}(f) & \overset{\simeq}{\dashrightarrow} & \operatorname{im}(f) \end{array}$$

These have use in homological algebra, especially in cohomology [3].

Outline

Conclusions

Main goal

To informally highlight some of the

context behind **basics of** **applications of**

category theory, so that it isn't as intimidating next time.

Conclusions

Main goal

To informally highlight some of the

context behind **basics of** **applications of**

category theory, so that it isn't as intimidating next time.

Context:

- Studying functions
- Regarding arrows as more important than objects

Conclusions

Main goal

To informally highlight some of the

context behind **basics of** **applications of**

category theory, so that it isn't as intimidating next time.

Basics:

- Monos, epis, isos
- Duality
- Functors and natural transformations
- Universality
- Yoneda lemma

Conclusions

Main goal

To informally highlight some of the

context behind **basics of** **applications of**

category theory, so that it isn't as intimidating next time.

Applications:

- Functional programming, esp. in the Haskell type system
- Monoidal categories and "combining objects"
- Enrichment and abelian categories, esp. in homology

Terminal thanks

Thanks 4 watching.

References.

[1] Samuel Eilenberg and Saunders Maclane. "General theory of natural equivalences". In: *Transactions of the American Mathematical Society* 58 (1945), pp. 231–294. URL: https://api.semanticscholar.org/CorpusID:10673176.

[2] Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 2019.

[3] Alexander Grothendieck. "Sur quelques points d'algèbre homologique, I". In: *Tohoku Mathematical Journal* 9.2 (1957), pp. 119–221. DOI: 10.2748/tmj/1178244839. URL: https://doi.org/10.2748/tmj/1178244839.