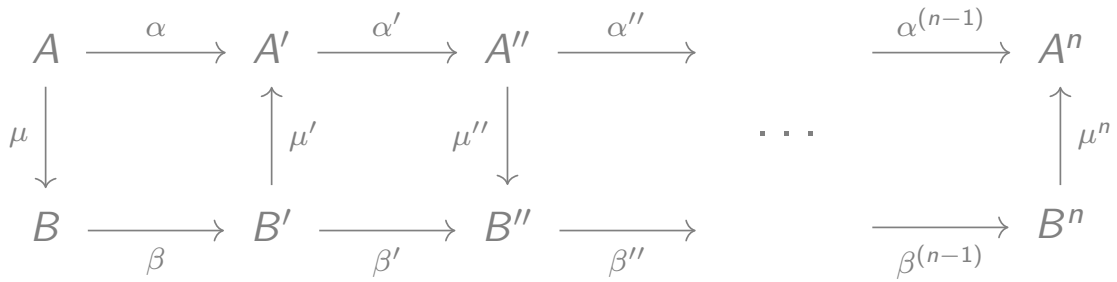
An Investigation of Elementary Category Theory

. . . with Selected Applications in PM and TCS.

Oliver Dixon Matthew Drury Ben Brook

Department of Mathematics, University of York

Spring Term, 2023



Presentation Overview

The Intuition: What are Categories? (1)

- Categories are a remarkably mathematical way of defining abstract objects, and ways to morph between those objects.

The Intuition: What are Categories? (1)

- Categories are a remarkably mathematical way of defining abstract objects, and ways to morph between those objects.
- The power of categories, and the associated theories, lie in their generality: objects can be any mathematically definable structure, and morphisms may be any process or algorithm by which you move from one object to another.

The Intuition: What are Categories? (1)

- Categories are a remarkably mathematical way of defining abstract objects, and ways to morph between those objects.
- The power of categories, and the associated theories, lie in their generality: objects can be any mathematically definable structure, and morphisms may be any process or algorithm by which you move from one object to another.
- At first glance, morphisms appear indistinguishable from functions. Whilst traditional sets and their functions certainly do form a category, as we will see later, the expanse of Category Theory extends far beyond naive Set Theory!

The Intuition: What are Categories? (2)

A useful feature in Category Theory arrives in the form of *commutative diagrams*. Objects are shown as labels, and morphisms are shown, intuitively, as directed arrows between their domain (source object) and codomain (destination object).

The Intuition: What are Categories? (2)

A useful feature in Category Theory arrives in the form of *commutative diagrams*. Objects are shown as labels, and morphisms are shown, intuitively, as directed arrows between their domain (source object) and codomain (destination object).

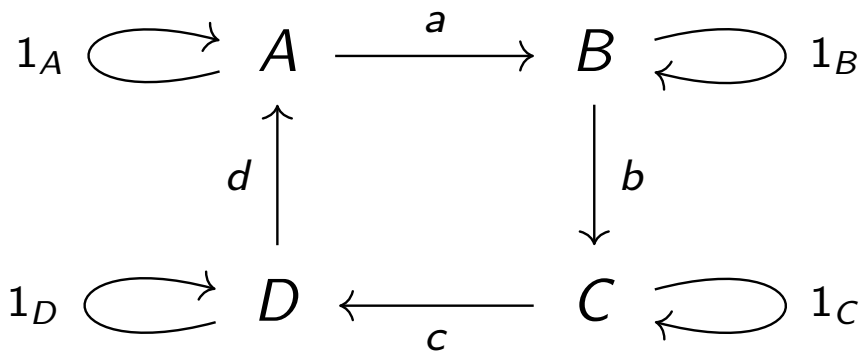


Figure 1.1: A commutative diagram for a simple, four-object category.

Identity morphisms are shown explicitly here, however they are usually omitted to maintain brevity.

The Definition: The Axioms of a Category

For any category $\mathcal{C}$, we need to consider two important sets:

- *The obset*: The set of objects in $\mathcal{C}$, denoted $\mathrm{ob}\,\mathcal{C}$.

The Definition: The Axioms of a Category

For any category $\mathcal{C}$, we need to consider two important sets:

- *The obset*: The set of objects in $\mathcal{C}$, denoted $\operatorname{ob}\mathcal{C}$.
- *The homset*: The set of (homeo)morphisms in $\mathcal{C}$, denoted $\hom\mathcal{C}$. Morphisms are sometimes called "arrows".

The Definition: The Axioms of a Category

For any category $\mathcal{C}$, we need to consider two important sets:

- *The obset*: The set of objects in $\mathcal{C}$, denoted $\operatorname{ob}\mathcal{C}$.
- *The homset*: The set of (homeo)morphisms in $\mathcal{C}$, denoted $\hom\mathcal{C}$. Morphisms are sometimes called "arrows".

In practice, the morphisms are the defining feature of any given category. As with existing algebraic structures, we do need to satisfy some laws to have a well-defined category:

The Definition: The Axioms of a Category

For any category $\mathcal{C}$, we need to consider two important sets:

- *The obset*: The set of objects in $\mathcal{C}$, denoted $\operatorname{ob}\mathcal{C}$.
- *The homset*: The set of (homeo)morphisms in $\mathcal{C}$, denoted $\hom\mathcal{C}$. Morphisms are sometimes called "arrows".

In practice, the morphisms are the defining feature of any given category. As with existing algebraic structures, we do need to satisfy some laws to have a well-defined category:

- Each object A must have an identity morphism $1_A \colon A \to A$.

The Definition: The Axioms of a Category

For any category $\mathcal{C}$, we need to consider two important sets:

- *The obset*: The set of objects in $\mathcal{C}$, denoted $\operatorname{ob}\mathcal{C}$.
- *The homset*: The set of (homeo)morphisms in $\mathcal{C}$, denoted $\hom\mathcal{C}$. Morphisms are sometimes called "arrows".

In practice, the morphisms are the defining feature of any given category. As with existing algebraic structures, we do need to satisfy some laws to have a well-defined category:

- Each object A must have an identity morphism $1_A \colon A \to A$.
- The composition of morphisms, where applicable, must be an associative operation.

The Definition: The Axioms of a Category

For any category $\mathcal{C}$, we need to consider two important sets:

- *The obset*: The set of objects in $\mathcal{C}$, denoted $\operatorname{ob}\mathcal{C}$.
- *The homset*: The set of (homeo)morphisms in $\mathcal{C}$, denoted $\hom\mathcal{C}$. Morphisms are sometimes called "arrows".

In practice, the morphisms are the defining feature of any given category. As with existing algebraic structures, we do need to satisfy some laws to have a well-defined category:

- Each object A must have an identity morphism $1_A : A \to A$.
- The composition of morphisms, where applicable, must be an associative operation.
- The usual unit law must hold: $f \circ 1_A = f = 1_B \circ f$ for all $f : A \to B$.

Functors (1)

Being able to define multiple categories, it would be useful to have a method of going between them. Just as *functions* serve as mappings between sets, and *linear maps* serve as mappings between vector spaces, *functors* can define relationships between categories.

Functors (1)

Being able to define multiple categories, it would be useful to have a method of going between them. Just as *functions* serve as mappings between sets, and *linear maps* serve as mappings between vector spaces, *functors* can define relationships between categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps the objects and morphisms of $\mathcal{C}$ to corresponding entities in $\mathcal{D}$.

Functors (1)

Being able to define multiple categories, it would be useful to have a method of going between them. Just as *functions* serve as mappings between sets, and *linear maps* serve as mappings between vector spaces, *functors* can define relationships between categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps the objects and morphisms of $\mathcal{C}$ to corresponding entities in $\mathcal{D}$.
- These mappings may be chosen arbitrarily, subject to three coherence conditions:

Functors (1)

Being able to define multiple categories, it would be useful to have a method of going between them. Just as *functions* serve as mappings between sets, and *linear maps* serve as mappings between vector spaces, *functors* can define relationships between categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps the objects and morphisms of $\mathcal{C}$ to corresponding entities in $\mathcal{D}$.
- These mappings may be chosen arbitrarily, subject to three coherence conditions:
  1. Preservation of domains and codomains:
     $F(f : a \to b) = F(f) : F(A) \to F(B)$

Functors (1)

Being able to define multiple categories, it would be useful to have a method of going between them. Just as *functions* serve as mappings between sets, and *linear maps* serve as mappings between vector spaces, *functors* can define relationships between categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps the objects and morphisms of $\mathcal{C}$ to corresponding entities in $\mathcal{D}$.
- These mappings may be chosen arbitrarily, subject to three coherence conditions:
  1. Preservation of domains and codomains: $F(f : a \to b) = F(f) : F(A) \to F(B)$
  2. Preservation of identity: $F(1_A) = 1_F(A)$

Functors (1)

Being able to define multiple categories, it would be useful to have a method of going between them. Just as *functions* serve as mappings between sets, and *linear maps* serve as mappings between vector spaces, *functors* can define relationships between categories.

- A functor $F : \mathcal{C} \to \mathcal{D}$ maps the objects and morphisms of $\mathcal{C}$ to corresponding entities in $\mathcal{D}$.
- These mappings may be chosen arbitrarily, subject to three coherence conditions:
  1. Preservation of domains and codomains: $F(f : a \to b) = F(f) : F(A) \to F(B)$
  2. Preservation of identity: $F(1_A) = 1_F(A)$
  3. Preservation of composition: $F(f \circ g) = F(f) \circ F(g)$

Functors (2)

A pair of connected commutative diagrams can pictorially demonstrate the role of a functor $F : \mathcal{C} \to \mathcal{D}$:

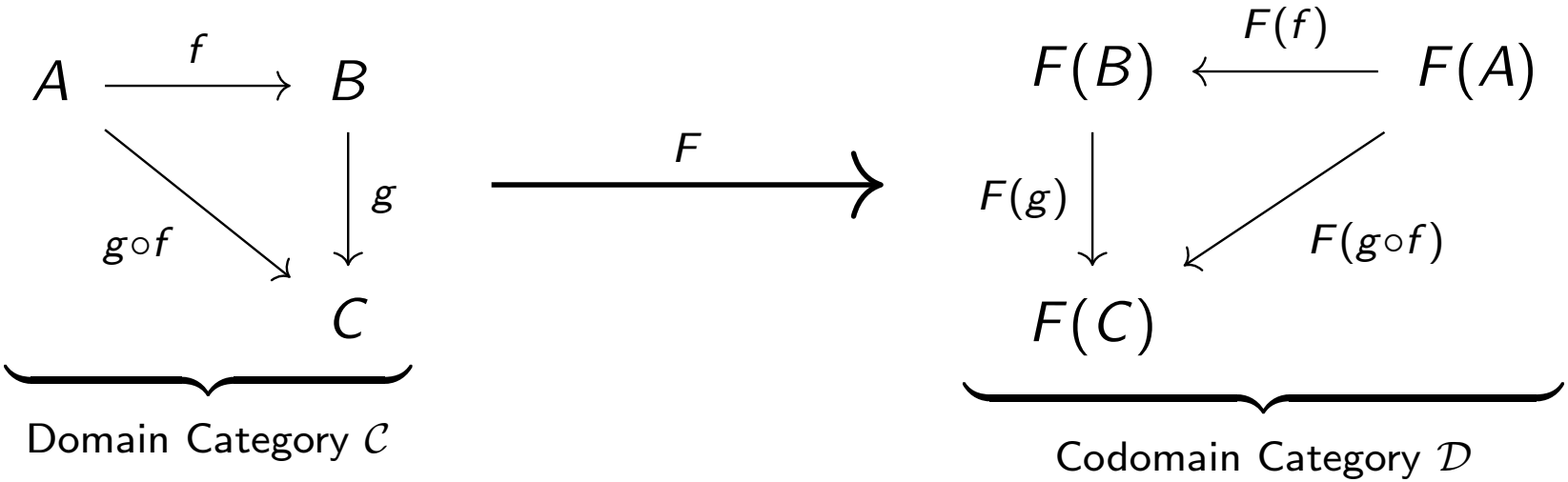


Figure 1.2: A functor F between simple categories $\mathcal{C}$ and $\mathcal{D}$

Functors (2)

A pair of connected commutative diagrams can pictorially demonstrate the role of a functor $F : \mathcal{C} \to \mathcal{D}$:

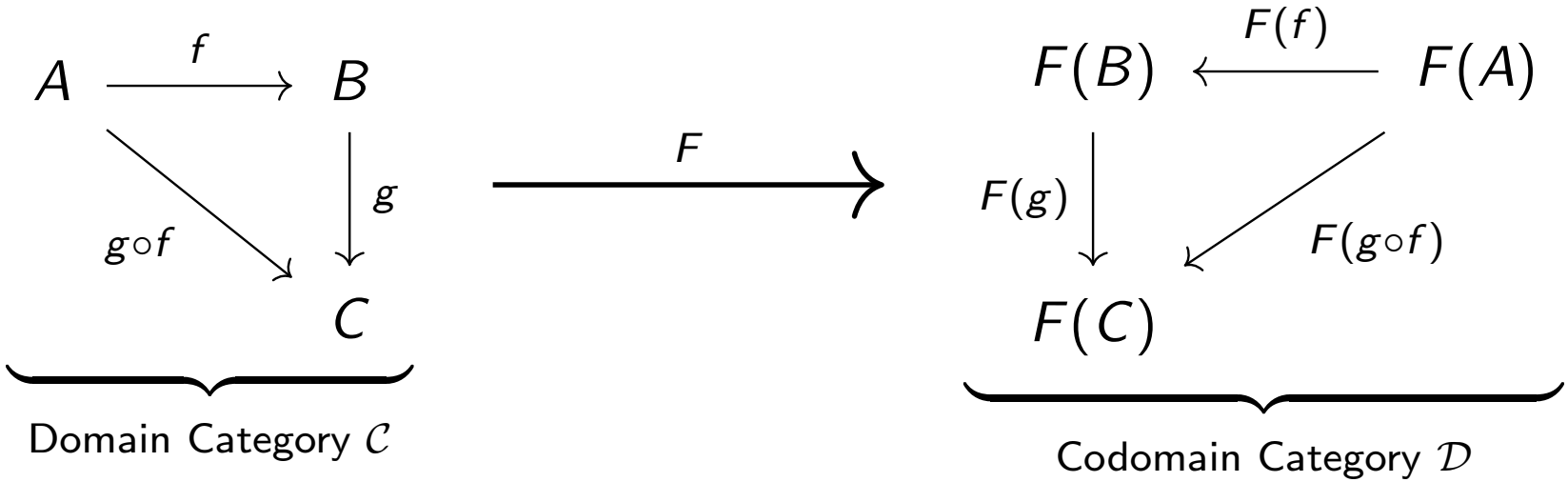


Figure 1.2: A functor F between simple categories $\mathcal{C}$ and $\mathcal{D}$

- An *endofunctor* is a functor whose domain and codomain is the same. This concept is simple, but it will become very important shortly!

The Category Product

It would be similarly useful to combine two categories in some meaningful way to produce a new category.

The Category Product

It would be similarly useful to combine two categories in some meaningful way to produce a new category.

- Analogous to the Cartesian product, we have the *category product*, denoted $\mathcal{C} \times \mathcal{D}$.

The Category Product

It would be similarly useful to combine two categories in some meaningful way to produce a new category.

- Analogous to the Cartesian product, we have the *category product*, denoted $\mathcal{C} \times \mathcal{D}$.
- The objects of $\mathcal{C} \times \mathcal{D}$ have the form (c, d) for $c \in \operatorname{ob}\mathcal{C}$ and $d \in \operatorname{ob}\mathcal{D}$. Morphisms have the form $(f, g) \colon (c, d) \to (c', d')$, where $f : c \to c' \in \operatorname{ob}\mathcal{C}$ and $g : d \to d' \in \operatorname{ob}\mathcal{D}$.

The Category Product

It would be similarly useful to combine two categories in some meaningful way to produce a new category.

- Analogous to the Cartesian product, we have the *category product*, denoted $\mathcal{C} \times \mathcal{D}$.
- The objects of $\mathcal{C} \times \mathcal{D}$ have the form (c, d) for $c \in \operatorname{ob}\mathcal{C}$ and $d \in \operatorname{ob}\mathcal{D}$. Morphisms have the form $(f, g) \colon (c, d) \to (c', d')$, where $f : c \to c' \in \operatorname{ob}\mathcal{C}$ and $g : d \to d' \in \operatorname{ob}\mathcal{D}$.
- $\mathcal{C} \times \mathcal{D}$ can be easily shown to satisfy the category axioms; hence, $\times$ is capable of producing a new category.

The Category Product

It would be similarly useful to combine two categories in some meaningful way to produce a new category.

- Analogous to the Cartesian product, we have the *category product*, denoted $\mathcal{C} \times \mathcal{D}$.
- The objects of $\mathcal{C} \times \mathcal{D}$ have the form (c, d) for $c \in \operatorname{ob}\mathcal{C}$ and $d \in \operatorname{ob}\mathcal{D}$. Morphisms have the form $(f, g) \colon (c, d) \to (c', d')$, where $f : c \to c' \in \operatorname{ob}\mathcal{C}$ and $g : d \to d' \in \operatorname{ob}\mathcal{D}$.
- $\mathcal{C} \times \mathcal{D}$ can be easily shown to satisfy the category axioms; hence, $\times$ is capable of producing a new category.
- We also have two obvious *projection functors* $\pi_{\mathcal{C}}$ and $\pi_{\mathcal{D}}$ from the product to the original constituent categories:

$$\mathcal{C} \xleftarrow[\pi_{\mathcal{C}}]{} \mathcal{C} \times \mathcal{D} \xrightarrow[\pi_{\mathcal{D}}]{} \mathcal{D}$$

Natural Transformations

In the standard tune of categorical thinking, we can push the abstraction further, and consider the *category of functors* between two fixed categories $\mathcal{C}$ and $\mathcal{D}$.

Natural Transformations

In the standard tune of categorical thinking, we can push the abstraction further, and consider the *category of functors* between two fixed categories $\mathcal{C}$ and $\mathcal{D}$.

- Consider categories $\mathcal{C}$, $\mathcal{D}$, and functors $F, G : \mathcal{C} \to \mathcal{D}$. Then, consider the category formed with these functors as objects. Denote this with $\mathbf{Fun}(\mathcal{C}, \mathcal{D})$.

Natural Transformations

In the standard tune of categorical thinking, we can push the abstraction further, and consider the *category of functors* between two fixed categories $\mathcal{C}$ and $\mathcal{D}$.

- Consider categories $\mathcal{C}$, $\mathcal{D}$, and functors $F, G : \mathcal{C} \to \mathcal{D}$. Then, consider the category formed with these functors as objects. Denote this with $\mathbf{Fun}(\mathcal{C}, \mathcal{D})$.
- The formal definition of natural transformations is slightly technical, however, for intuition, it suffices to think of natural transformations as morphisms between functors in $\mathbf{Fun}(\mathcal{C}, \mathcal{D})$.

Natural Transformations

In the standard tune of categorical thinking, we can push the abstraction further, and consider the *category of functors* between two fixed categories $\mathcal{C}$ and $\mathcal{D}$.

- Consider categories $\mathcal{C}$, $\mathcal{D}$, and functors $F, G : \mathcal{C} \to \mathcal{D}$. Then, consider the category formed with these functors as objects. Denote this with $\mathbf{Fun}(\mathcal{C}, \mathcal{D})$.
- The formal definition of natural transformations is slightly technical, however, for intuition, it suffices to think of natural transformations as morphisms between functors in $\mathbf{Fun}(\mathcal{C}, \mathcal{D})$.
- As usual, these morphisms meet certain coherence conditions; most of the time these are trivially satisfiable.

Presentation Overview

Category of Sets

- Let A, B, C, D be sets, and let f, g, h be (set-theoretic) functions such that $f\colon A \to B$, $g\colon B \to C$, and $h\colon C \to D$.

Category of Sets

- Let A, B, C, D be sets, and let f, g, h be (set-theoretic) functions such that $f : A \to B$, $g : B \to C$, and $h : C \to D$.
- Immediately, we have that

$$(f \circ g) \circ h = f \circ (g \circ h),$$

due to the normal associativity property of function composition. We also have the obvious identity morphism $i \in \hom(A, A)$ for some object A, henceforth denoted as 1_A.

Category of Sets

- Let A, B, C, D be sets, and let f, g, h be (set-theoretic) functions such that $f : A \to B$, $g : B \to C$, and $h : C \to D$.
- Immediately, we have that

$$(f \circ g) \circ h = f \circ (g \circ h),$$

  due to the normal associativity property of function composition. We also have the obvious identity morphism $i \in \hom(A, A)$ for some object A, henceforth denoted as 1_A.
- Hence, the category axioms are satisfied, and there is a category whose objects are sets and morphisms are the functions between those sets. We'll denote this category **Set** moving forwards.

Category of Partially Ordered Sets ("Posets")

- A *poset* is a set A with a relation, $\sim_A$, that is reflexive, transitive, and antisymmetric.

Category of Partially Ordered Sets ("Posets")

- A *poset* is a set A with a relation, $\sim_A$, that is reflexive, transitive, and antisymmetric.
- A monotone map $m\colon A \to B$ is an order-preserving function such that

$$(\forall a, b \in A)\,(a \sim_A b) \implies m(a) \sim_B m(b).$$

Category of Partially Ordered Sets ("Posets")

- A *poset* is a set A with a relation, $\sim_A$, that is reflexive, transitive, and antisymmetric.
- A monotone map $m\colon A \to B$ is an order-preserving function such that

$$(\forall a, b \in A)\,(a \sim_A b) \implies m(a) \sim_B m(b).$$

- We have a category **Pos** whose objects are posets and morphisms are monotone maps between those posets. The identity morphism exists for all objects due to the reflexivity of $\sim_A$, and composition follows similarly due to transitivity.

Category of Partially Ordered Sets ("Posets")

- A *poset* is a set A with a relation, $\sim_A$, that is reflexive, transitive, and antisymmetric.
- A monotone map $m\colon A \to B$ is an order-preserving function such that

$$(\forall a, b \in A)\,(a \sim_A b) \implies m(a) \sim_B m(b).$$

- We have a category **Pos** whose objects are posets and morphisms are monotone maps between those posets. The identity morphism exists for all objects due to the reflexivity of $\sim_A$, and composition follows similarly due to transitivity.
- Analogous connexions can be drawn for sets equipped with preording relations (**Ord**), pointed sets (**Based**), etc.

Category of Vector Spaces

- We have a category, K-**Vect**, whose objects are *vector spaces* over some field K and morphisms are *linear mappings* between those vector spaces.

Category of Vector Spaces

- We have a category, K-**Vect**, whose objects are *vector spaces* over some field K and morphisms are *linear mappings* between those vector spaces.
- We take a linear map to be a function, f, between two vector spaces, such that vector addition and scalar multiplication in K is preserved. That is,

$$f(\vec{u}+\vec{v}) = f(\vec{u}) + f(\vec{v}); \qquad \textbf{[Additivity]}$$
$$f(\lambda\vec{u}) = \lambda f(\vec{u}), \qquad \textbf{[Homogeinity]}$$

where $\vec{u}, \vec{v} \in K^n$ and $\lambda \in K$.

Category of Vector Spaces

- We have a category, K-**Vect**, whose objects are *vector spaces* over some field K and morphisms are *linear mappings* between those vector spaces.
- We take a linear map to be a function, f, between two vector spaces, such that vector addition and scalar multiplication in K is preserved. That is,

$$f(\vec{u}+\vec{v}) = f(\vec{u}) + f(\vec{v}); \qquad \textbf{[Additivity]}$$
$$f(\lambda\vec{u}) = \lambda f(\vec{u}), \qquad \textbf{[Homogeinity]}$$

  where $\vec{u}, \vec{v} \in K^n$ and $\lambda \in K$.
- This only applies to *finite-dimensional* vector spaces. Banach spaces and Functional Analysis are out of the scope of this talk! ($n \in \mathbb{N}$)

The Category of Deduction and Deducability

All of these examples are taken, more or less directly, from constructs expressable purely in non-categorical/algebraic terms. The power of Category Theory lies in its ability to abstract arbitrarily, so what type of non-algebraic structures might we want to represent?

The Category of Deduction and Deducability

All of these examples are taken, more or less directly, from constructs expressable purely in non-categorical/algebraic terms. The power of Category Theory lies in its ability to abstract arbitrarily, so what type of non-algebraic structures might we want to represent?

- How about the Category of Deducability? Objects are mathematical statements embedded in some proof system, and morphisms represent *the existence of a deduction*, in a fixed theory, between antecedents and consequents.

The Category of Deduction and Deducability

All of these examples are taken, more or less directly, from constructs expressable purely in non-categorical/algebraic terms. The power of Category Theory lies in its ability to abstract arbitrarily, so what type of non-algebraic structures might we want to represent?

- How about the Category of Deducability? Objects are mathematical statements embedded in some proof system, and morphisms represent *the existence of a deduction*, in a fixed theory, between antecedents and consequents.
- A richer category is the Category of Deduction, **Deduce**. For every pair of objects, a corresponding morphism in the homset represents a *specific manner of deduction* between its domain and codomain. Deductive cancelling is equivalent to morphism composition!

The True Isomorphism! (1)

Although pervasive throughout Abstract Algebra and Set Theory, the "correct" definition of a general isomorphism is expressed in purely category-theoretic terms, demonstrating the existence of an *undo* morphism, such that we have elements in $\hom(A, C)$ and $\hom(C, A)$.

The True Isomorphism! (1)

Although pervasive throughout Abstract Algebra and Set Theory, the "correct" definition of a general isomorphism is expressed in purely category-theoretic terms, demonstrating the existence of an *undo* morphism, such that we have elements in $\hom(A, C)$ and $\hom(C, A)$.

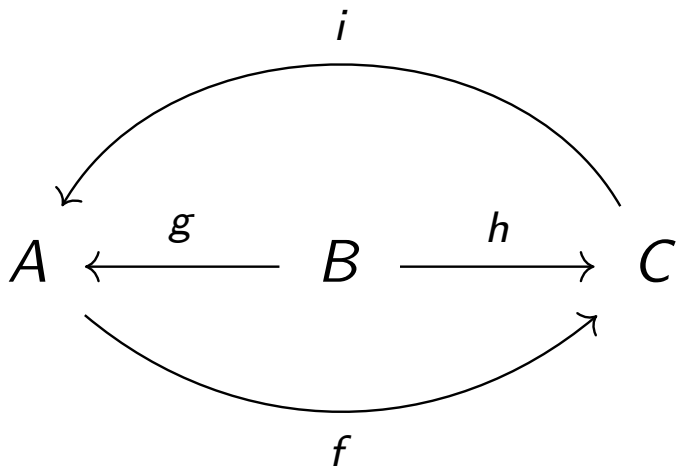


Figure 2.1: f and i are *isomorphisms*; A and C are *isomorphic*.

The True Isomorphism! (1)

Although pervasive throughout Abstract Algebra and Set Theory, the "correct" definition of a general isomorphism is expressed in purely category-theoretic terms, demonstrating the existence of an *undo* morphism, such that we have elements in $\hom(A, C)$ and $\hom(C, A)$.

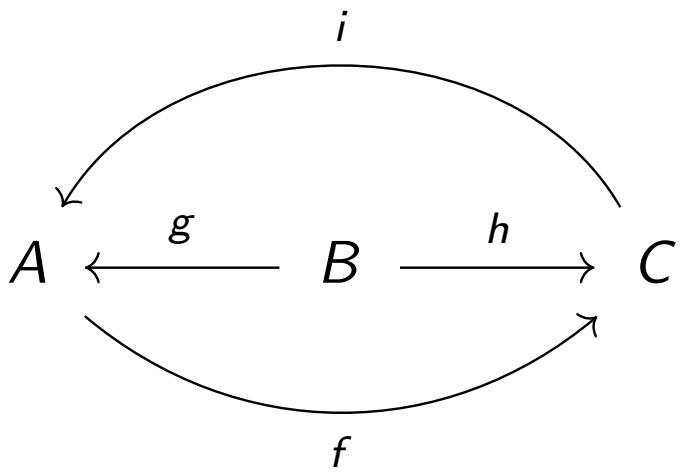


Figure 2.1: f and i are *isomorphisms*; A and C are *isomorphic*.

Isomorphisms are easy to spot with well-drawn commutative diagrams!

The True Isomorphism! (2)

In certain categories, this isomorphic equivalence can be used to deduce particular properties of objects in the corresponding category. Here we consider the Category of Deduction, **Deduce**:

The True Isomorphism! (2)

In certain categories, this isomorphic equivalence can be used to deduce particular properties of objects in the corresponding category. Here we consider the Category of Deduction, **Deduce**:

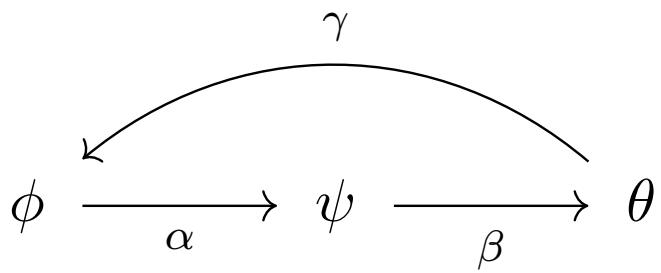


Figure 2.2: ϕ and θ are logically equivalent through deductions $[\alpha, \beta]$ and γ.

The True Isomorphism! (2)

In certain categories, this isomorphic equivalence can be used to deduce particular properties of objects in the corresponding category. Here we consider the Category of Deduction, **Deduce**:

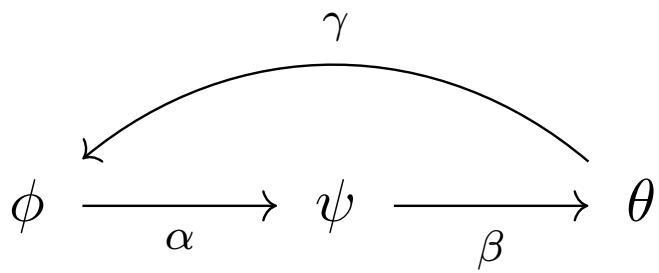


Figure 2.2: ϕ and θ are logically equivalent through deductions $[\alpha, \beta]$ and γ.

Using deductive cancelling, α and β can be forged into a single morphism, with ψ being embedded into the environmental axioms of ϕ. Given our knowledge of the nature of ob (**Deduce**), we know that

$$(\phi \vdash \theta) \wedge (\theta \vdash \phi) \iff \phi \equiv \theta \iff \phi \cong \theta.$$

Presentation Overview

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$
- A bifunctor $\otimes : \mathcal{C}_0 \times \mathcal{C}_0 \to \mathcal{C}_0$

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$
- A bifunctor $\otimes : \mathcal{C}_0 \times \mathcal{C}_0 \to \mathcal{C}_0$
- An identity from our base category, $I \in \mathrm{ob}\,\mathcal{C}_0$

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$
- A bifunctor $\otimes : \mathcal{C}_0 \times \mathcal{C}_0 \to \mathcal{C}_0$
- An identity from our base category, $I \in \mathrm{ob}\,\mathcal{C}_0$
- An associativity natural transformation: $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$
- A bifunctor $\otimes : \mathcal{C}_0 \times \mathcal{C}_0 \to \mathcal{C}_0$
- An identity from our base category, $I \in \mathrm{ob}\,\mathcal{C}_0$
- An associativity natural transformation: $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$
- A left-identity natural transformation: $\lambda_A : I \otimes A \to A$

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$
- A bifunctor $\otimes : \mathcal{C}_0 \times \mathcal{C}_0 \to \mathcal{C}_0$
- An identity from our base category, $I \in \mathrm{ob}\,\mathcal{C}_0$
- An associativity natural transformation: $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$
- A left-identity natural transformation: $\lambda_A : I \otimes A \to A$
- A right-identity natural transformation: $\rho_A : A \otimes I \to A$.

Monoidal Categories (1)

To construct a monoidal category $\mathcal{C}$, we need to consider six elements:

- A base category $\mathcal{C}_0$
- A bifunctor $\otimes : \mathcal{C}_0 \times \mathcal{C}_0 \to \mathcal{C}_0$
- An identity from our base category, $I \in \mathrm{ob}\,\mathcal{C}_0$
- An associativity natural transformation: $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$
- A left-identity natural transformation: $\lambda_A : I \otimes A \to A$
- A right-identity natural transformation: $\rho_A : A \otimes I \to A$.

Then, $\mathcal{C} = (\mathcal{C}_0, \otimes, I, \alpha, \lambda, \rho)$, or just $\mathcal{C} = (\mathcal{C}_0, \otimes, I)$.

Monoidal Categories (2)

$$\begin{array}{ccccc}
[(A \otimes B) \otimes C] \otimes D & \xrightarrow{\alpha} & (A \otimes B) \otimes (C \otimes D) & \xrightarrow{\alpha} & A \otimes [B \otimes (C \otimes D)] \\
{\scriptstyle \alpha \otimes \mathrm{id}\, D}\downarrow & & & & \uparrow{\scriptstyle \mathrm{id}\, A \otimes \alpha} \\
[A \otimes (B \otimes C)] \otimes D & & \xrightarrow[\alpha]{} & & A \otimes [(B \otimes C) \otimes D]
\end{array}$$

Figure 3.1: Associativity induced by α

Monoidal Categories (2)

$$\begin{array}{ccccc}
[(A \otimes B) \otimes C] \otimes D & \xrightarrow{\alpha} & (A \otimes B) \otimes (C \otimes D) & \xrightarrow{\alpha} & A \otimes [B \otimes (C \otimes D)] \\
{\scriptstyle \alpha \otimes \mathrm{id}\, D}\downarrow & & & & \uparrow{\scriptstyle \mathrm{id}\, A \otimes \alpha} \\
[A \otimes (B \otimes C)] \otimes D & & \xrightarrow[\alpha]{} & & A \otimes [(B \otimes C) \otimes D]
\end{array}$$

Figure 3.1: Associativity induced by α

$$\begin{array}{ccc}
(B \otimes I) \otimes C & \xrightarrow{\alpha} & B \otimes (I \otimes C) \\
& {\scriptstyle \rho \otimes \mathrm{id}\, C} \searrow \quad \swarrow {\scriptstyle \mathrm{id}\, B \otimes \lambda} & \\
& B \otimes C &
\end{array}$$

Figure 3.2: Left- and right-identities induced by λ and ρ

Monoids (1)

A monoid (M, μ, η) is composed of:

- Some object $M \in \mathrm{ob}\,\mathcal{C}_0$

Monoids (1)

A monoid (M, μ, η) is composed of:

- Some object $M \in \mathrm{ob}\,\mathcal{C}_0$
- An associated "multiplication" bifunctor $\mu \in \hom_{\mathcal{C}}(M \otimes M, M)$

Monoids (1)

A monoid (M, μ, η) is composed of:

- Some object $M \in \mathrm{ob}\,\mathcal{C}_0$
- An associated "multiplication" bifunctor $\mu \in \hom_{\mathcal{C}}(M \otimes M, M)$
- An associated "unit" identity $\eta \in \hom_{\mathcal{C}}(I, M)$.

Monoids (2)

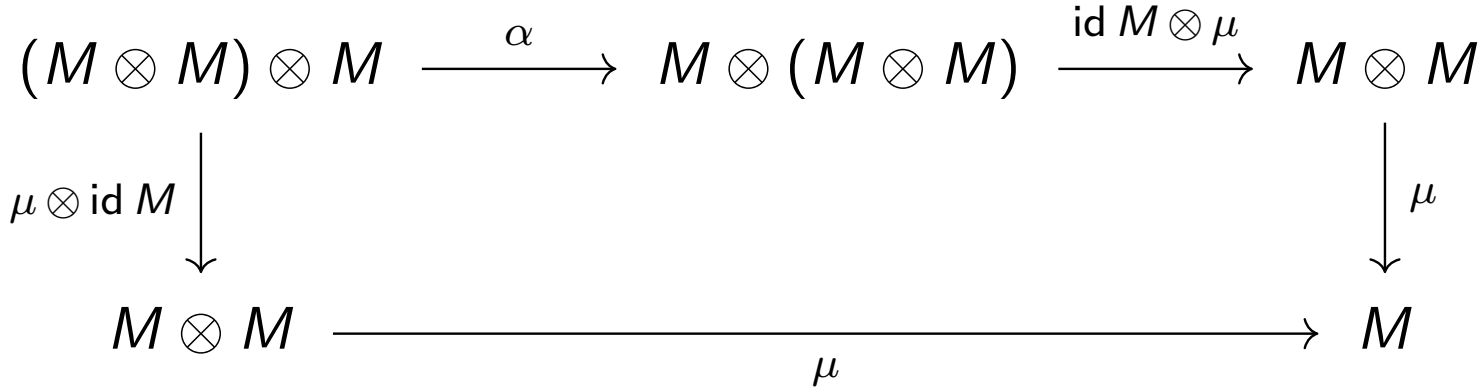


Figure 3.3: Associativity of the monoid

Monoids (2)

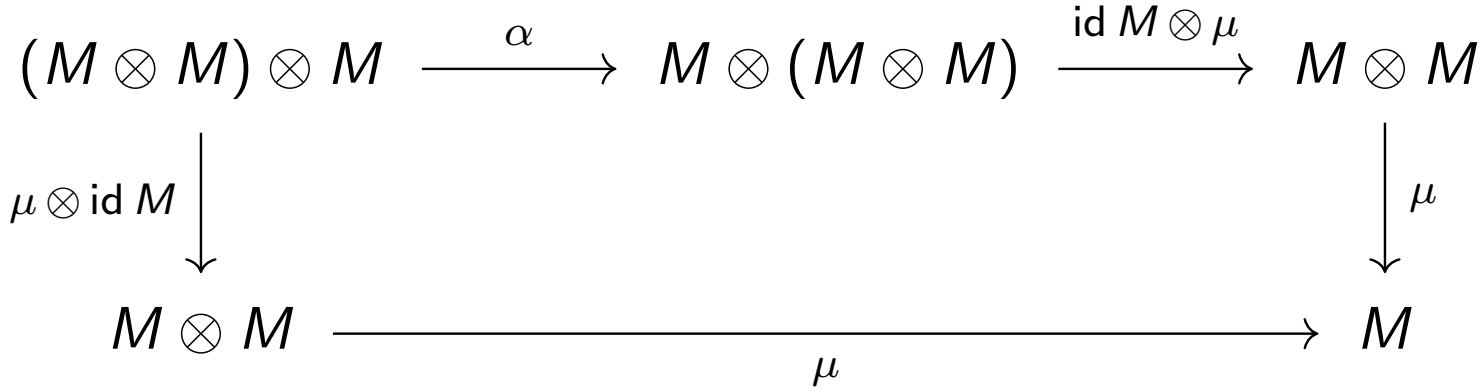


Figure 3.3: Associativity of the monoid

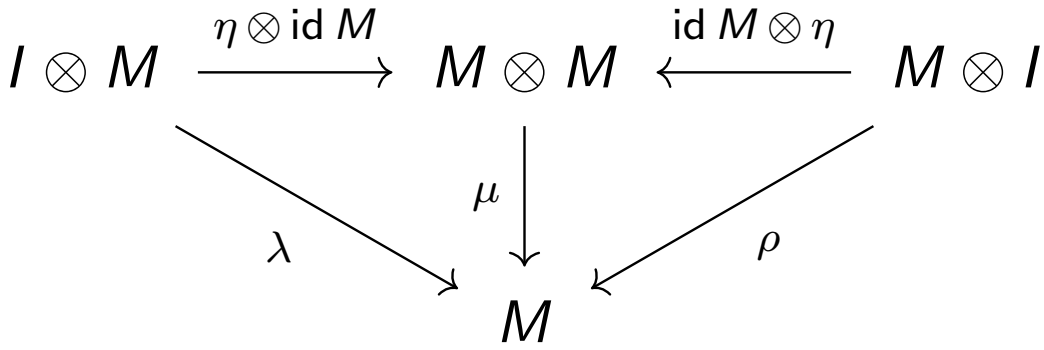


Figure 3.4: Left- and right-identities of the monoid

Functional Programming Languages

- In *purely* functional programming languages, functions must not cause side-effects: for some input, they must always return the same output, independent of the state of the wider environment.

Functional Programming Languages

- In *purely* functional programming languages, functions must not cause side-effects: for some input, they must always return the same output, independent of the state of the wider environment.
- There is no allowance for a shared mutable state. At first glance, this causes problems, since many common operations that are made easy in the imperative world, are in direct contravention with this safety principle.

Functional Programming Languages

- In *purely* functional programming languages, functions must not cause side-effects: for some input, they must always return the same output, independent of the state of the wider environment.
- There is no allowance for a shared mutable state. At first glance, this causes problems, since many common operations that are made easy in the imperative world, are in direct contravention with this safety principle.
- How can monoids and monodial categories help us with printing "Hello World"?

An Especially Useful Monoidal Category

Let's consider an example: the category of *endofunctors*! We have already encountered endofunctors in the first section.

- The endofunctors form a (strict) monoidal category.

An Especially Useful Monoidal Category

Let's consider an example: the category of *endofunctors*! We have already encountered endofunctors in the first section.

- The endofunctors form a (strict) monoidal category.
- The normal functor composition operation becomes the associated bifunctor $\otimes$.

An Especially Useful Monoidal Category

Let's consider an example: the category of *endofunctors*! We have already encountered endofunctors in the first section.

- The endofunctors form a (strict) monoidal category.
- The normal functor composition operation becomes the associated bifunctor $\otimes$.
- The identity functor becomes the identity element I.

An Especially Useful Monoidal Category

Let's consider an example: the category of *endofunctors*! We have already encountered endofunctors in the first section.

- The endofunctors form a (strict) monoidal category.
- The normal functor composition operation becomes the associated bifunctor $\otimes$.
- The identity functor becomes the identity element I.
- By our previous definitions, an endofunctor can be interpreted as a monoid.

An Especially Useful Monoidal Category

Let's consider an example: the category of *endofunctors*! We have already encountered endofunctors in the first section.

- The endofunctors form a (strict) monoidal category.
- The normal functor composition operation becomes the associated bifunctor $\otimes$.
- The identity functor becomes the identity element I.
- By our previous definitions, an endofunctor can be interpreted as a monoid.
- A *monad* is a monoid in this category.

An Especially Useful Monoidal Category

Let's consider an example: the category of *endofunctors*! We have already encountered endofunctors in the first section.

- The endofunctors form a (strict) monoidal category.
- The normal functor composition operation becomes the associated bifunctor $\otimes$.
- The identity functor becomes the identity element I.
- By our previous definitions, an endofunctor can be interpreted as a monoid.
- A *monad* is a monoid in this category.
- We can specify the base category $\mathcal{C}_0$ with $\mathbf{Endo}\,(\mathcal{C}_0)$.

Haskell, **Hask**, and **Endo(Hask)**

The *Haskell* language is a useful demonstration tool, however the type systems of most languages will be equally valid for these purposes. (Early versions of Haskell weren't capable of I/O!)

Haskell, **Hask**, and **Endo(Hask)**

The *Haskell* language is a useful demonstration tool, however the type systems of most languages will be equally valid for these purposes. (Early versions of Haskell weren't capable of I/O!)

- Consider the category **Hask**: its objects are Haskell types, and its morphisms are functions between types. Haskell is a *purely* functional language, so **Hask** is well-defined.

Haskell, **Hask**, and **Endo(Hask)**

The *Haskell* language is a useful demonstration tool, however the type systems of most languages will be equally valid for these purposes. (Early versions of Haskell weren't capable of I/O!)

- Consider the category **Hask**: its objects are Haskell types, and its morphisms are functions between types. Haskell is a *purely* functional language, so **Hask** is well-defined.
- What about **Endo(Hask)**?

Haskell, **Hask**, and **Endo(Hask)**

The *Haskell* language is a useful demonstration tool, however the type systems of most languages will be equally valid for these purposes. (Early versions of Haskell weren't capable of I/O!)

- Consider the category **Hask**: its objects are Haskell types, and its morphisms are functions between types. Haskell is a *purely* functional language, so **Hask** is well-defined.
- What about **Endo(Hask)**?
- Its objects are endofunctors on **Hask**, and its morphisms are natural transformations between these functors.

Haskell, **Hask**, and **Endo(Hask)**

The *Haskell* language is a useful demonstration tool, however the type systems of most languages will be equally valid for these purposes. (Early versions of Haskell weren't capable of I/O!)

- Consider the category **Hask**: its objects are Haskell types, and its morphisms are functions between types. Haskell is a *purely* functional language, so **Hask** is well-defined.
- What about **Endo(Hask)**?
- Its objects are endofunctors on **Hask**, and its morphisms are natural transformations between these functors.
- Thus, endofunctors on **Hask** with natural transformations μ and η form monads.

In Closing: Practical Notes

We have established a tangential, perhaps contrived, link between Category Theory and functional programming, the latter of which is based on the λ-calculus. Why is this interdisciplinary observation useful?

In Closing: Practical Notes

We have established a tangential, perhaps contrived, link between Category Theory and functional programming, the latter of which is based on the λ-calculus. Why is this interdisciplinary observation useful?

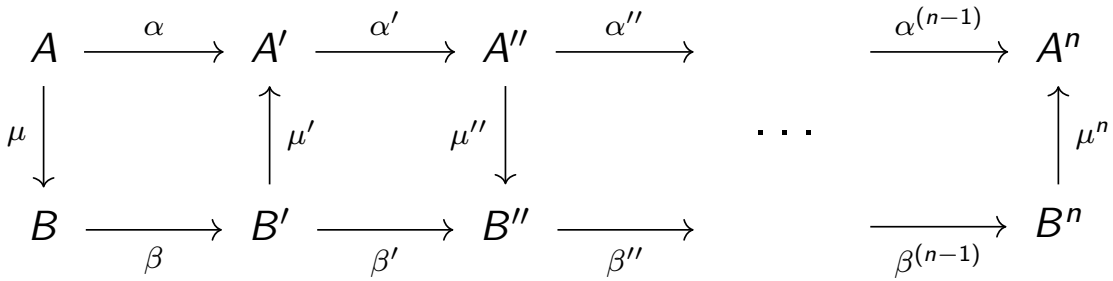
- Akin to "following the arrows" on a commutative diagram of **Endo** (**Hask**), we use the category product **Hask** $\times$ **Hask** to pass the desired parameter with the current state, and returning the transformed state, recalling that the endofunctors are morphisms in **Hask**.

In Closing: Practical Notes

We have established a tangential, perhaps contrived, link between Category Theory and functional programming, the latter of which is based on the λ-calculus. Why is this interdisciplinary observation useful?

- Akin to "following the arrows" on a commutative diagram of **Endo** (**Hask**), we use the category product **Hask** $\times$ **Hask** to pass the desired parameter with the current state, and returning the transformed state, recalling that the endofunctors are morphisms in **Hask**.
- In practice, monads allow stateful calculation. Unlike the mindless emulation of imperative languages, this is achieved without breaking function purity. This can be seen intuitively as *threading state though a chain of functions*.

Any Questions?



"*The Categorical Snake of Morphistic Infinitude*"