

Category Theory

Ivan Murashko

July 26, 2018

Contents

1	Base definitions	7
1.1	Definitions	7
1.1.1	Object	7
1.1.2	Morphism	8
1.1.3	Category	10
1.2	Examples	12
1.2.1	Set category	12
1.2.2	Programming languages	15
1.2.3	Physics	18
2	Objects and morphisms	19
2.1	Equality	19
2.2	Initial and terminal objects	20
2.3	Product and sum	20
2.4	Examples	20
2.4.1	Set category	20
2.4.2	Programming languages	21
3	Functors	23
4	Monads	25
	Index	27

Introduction

There is an introduction to Category Theory.

The first chapter keeps base definition and several examples. The most important one is the example from Set theory. There are also examples from programming languages (Haskell, C++, Scala) and physics.

The source files for programming languages examples (Haskell, C++, Scala) can be found on github repo [\[3\]](#)

Chapter 1

Base definitions

1.1 Definitions

1.1.1 Object

Definition 1.1 (Class). A class is a collection of sets (or sometimes other mathematical objects) that can be unambiguously defined by a property that all its members share.

Definition 1.2 (Object). In category theory object is considered as something that does not have internal structure (aka point) but has a property that makes different objects belong to the same [Class](#)

Remark 1.3 (Class of Objects). The [Class](#) of [Objects](#) will be marked as $\text{ob}(\mathbf{C})$ (see fig. 1.1).



Figure 1.1: Class of objects $\text{ob}(\mathbf{C}) = \{a, b, c, d\}$

1.1.2 Morphism

Morphism is a kind of relation between 2 **Objects**.

Definition 1.4 (Morphism). A relation between two **Objects** a and b

$$f_{ab} : a \rightarrow b$$

is called *morphism*. Morphism assumes a direction i.e. one **Object** (a) is called *source* and another one (b) *target*.

Definition 1.5 (Domain). Given a **Morphism** $f : a \rightarrow b$, the **Object** a is called domain and is denoted as $\text{dom } a$.

Definition 1.6 (Codomain). Given a **Morphism** $f : a \rightarrow b$, the **Object** b is called codomain and is denoted as $\text{cod } a$.

Morphisms have several properties. ¹

Axiom 1.7 (Composition). If we have 3 **Objects** a, b and c and 2 **Morphisms**

$$f_{ab} : a \rightarrow b$$

and

$$f_{bc} : b \rightarrow c$$

then there exists **Morphism**

$$f_{ac} : a \rightarrow c$$

such that

$$f_{ac} = f_{bc} \circ f_{ab}$$

Remark 1.8 (Composition). The equation

$$f_{ac} = f_{bc} \circ f_{ab}$$

means that we apply f_{ab} first and then we apply f_{bc} to the result of the application i.e. if our objects are sets and $x \in a$ then

$$f_{ac}(x) = f_{bc}(f_{ab}(x)),$$

where $f_{ab}(x) \in b$.

Axiom 1.9 (Associativity). The **Morphisms Composition** (*Axiom 1.7*) should follow associativity property:

$$f_{ce} \circ (f_{bc} \circ f_{ab}) = (f_{ce} \circ f_{bc}) \circ f_{ab} = f_{ce} \circ f_{bc} \circ f_{ab}.$$

¹The properties don't have any proof and postulated as axioms

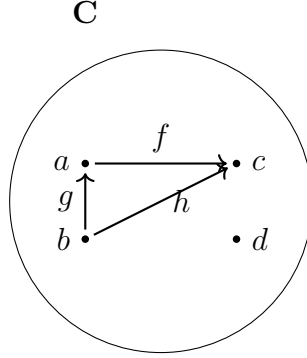


Figure 1.2: Class of morphisms $\text{hom}_{ob(C)} = \{f, g, h\}$, where $h = f \circ g$

Definition 1.10 (Identity morphism). For every **Object** a we define a special **Morphism** $1_a : a \rightarrow a$ with the following properties: $\forall f_{ab} : a \rightarrow b$

$$1_a \circ f_{ab} = f_{ab} \quad (1.1)$$

and $\forall f_{ba} : b \rightarrow a$

$$f_{ba} \circ 1_a = f_{ba}. \quad (1.2)$$

This morphism is called *identity morphism*.

Note that **Identity morphism** is unique, see **Identity is unique** (**Theorem 2.3**) below.

Definition 1.11 (Commutative diagram). A commutative diagram is a diagram of **Objects** (also known as vertices) and **Morphisms** (also known as arrows or edges) such that all directed paths in the diagram with the same start and endpoints lead to the same result by composition

The following diagram commutes if $f_{ab} = f_{cb} \circ f_{ac}$.



Remark 1.12 (Class of Morphisms). The **Class** of **Morphisms** will be marked as $\text{hom}(C)$ (see fig. 1.2)

Definition 1.13 (Monomorphism). If $\forall g_1, g_2$ the equation

$$f \circ g_1 = f \circ g_2$$

leads to

$$g_1 = g_2$$

then f is called *monomorphism*.

Definition 1.14 (Epimorphism). If $\forall g_1, g_2$ the equation

$$g_1 \circ f = g_2 \circ f$$

leads to

$$g_1 = g_2$$

then f is called *epimorphism*.

Definition 1.15 (Isomorphism). A **Morphism** $f : a \rightarrow b$ is called isomorphism if $\exists g : b \rightarrow a$ such that $f \circ g = \mathbf{1}_a$ and $g \circ f = \mathbf{1}_b$.

Remark 1.16 (Isomorphism). There are can be many different **Isomorphisms** between 2 **Objects**.

1.1.3 Category

Definition 1.17 (Category). A category \mathbf{C} consists of

- **Class** of **Objects** $\text{ob}(\mathbf{C})$
- **Class** of **Morphisms** $\text{hom}(\mathbf{C})$ defined for $\text{ob}(\mathbf{C})$, i.e. each morphism f_{ab} from $\text{hom}(\mathbf{C})$ has both source a and target b from $\text{ob}(\mathbf{C})$

For any **Object** a there should be unique **Identity morphism** $\mathbf{1}_a$. Any morphism should satisfy **Composition** (**Axiom 1.7**) and **Associativity** (**Axiom 1.9**) properties. See fig. 1.3

The **Category** can be considered as a way to represent a structured data. **Morphisms** are the ones to form the structure.

Definition 1.18 (Opposite category). If \mathbf{C} is a **Category** then opposite (or dual) category \mathbf{C}^{op} is constructed in the following way: **Objects** are the same, but the **Morphisms** are inverted i.e. if $f \in \text{hom}(\mathbf{C})$ and $\text{dom } f = a, \text{cod } f = b$, then the corresponding morphism $f^{op} \in \text{hom}(\mathbf{C}^{op})$ has $\text{dom } f^{op} = b, \text{cod } f^{op} = a$ (see fig. 1.4)

Remark 1.19. Composition on \mathbf{C}^{op} As you can see from fig. 1.4 the **Composition** (**Axiom 1.7**) is reverted for **Opposite category**. If $f, g, h = f \circ g \in \text{hom}(\mathbf{C})$ then $f \circ g$ translated into $g^{op} \circ f^{op}$ in opposite category.

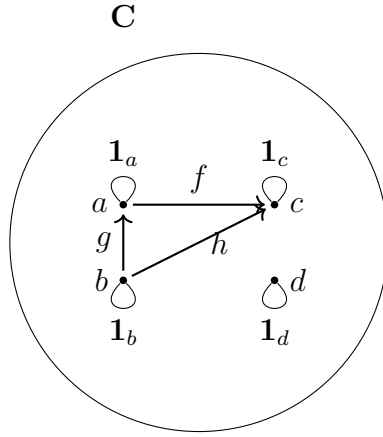


Figure 1.3: Category **C**. It consists of 4 objects $\text{ob}(\mathbf{C}) = \{a, b, c, d\}$ and 7 morphisms $\text{ob}(\mathbf{C}) = \{f, g, h = f \circ g, 1_a, 1_b, 1_c, 1_d\}$

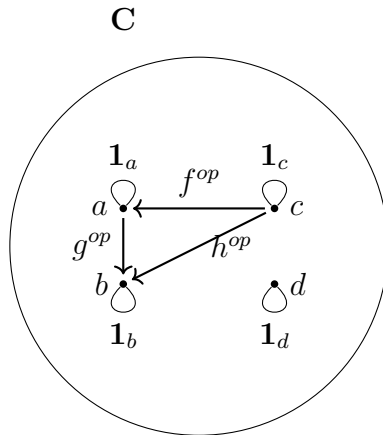


Figure 1.4: Opposite category C^{op} to the category from fig. 1.3 . It consists of 4 objects $\text{ob}(\mathbf{C}^{op}) = \text{ob}(\mathbf{C}) = \{a, b, c, d\}$ and 7 morphisms $\text{hom}(\mathbf{C}^{op}) = \{f^{op}, g^{op}, h^{op} = g^{op} \circ f^{op}, 1_a, 1_b, 1_c, 1_d\}$

1.2 Examples

There are several examples of categories that will also be used later

1.2.1 Set category

Definition 1.20 (Set). Set is a collection of distinct object. The objects are called the elements of the set.

Definition 1.21 (Function). If A and B are 2 [Sets](#) then a subset of $A \times B$ is called function f between the 2 sets, i.e. $f \subset A \times B$.

Example 1.22 (Set category). In the set category we consider a [Set](#) of [Sets](#) where [Objects](#) are the [Sets](#) and [Morphisms](#) are [Functions](#) between the sets.

The [Identity morphism](#) is trivial function such that $\forall x \in X : \mathbf{1}_X(x) = x$.

Remark 1.23 (Set vs Category). There is an interesting relation between sets and categories. In both we consider objects(sets) and relations between them(morphisms/functions).

In the set theory we can get info about functions by looking inside the objects(sets) aka use “microscope” [2]

Contrary in the category theory we initially don’t have info about object internal structure but can get it using the relation between the objects i.e. using [Morphisms](#). In other words we can use “telescope” [2] there.

Definition 1.24 (Domain). Given a function $f : X \rightarrow Y$, the set X is the domain.

Definition 1.25 (Codomain). Given a function $f : X \rightarrow Y$, the set Y is the codomain.

Definition 1.26 (Surjection). The function $f : X \rightarrow Y$ is surjective (or onto) if $\forall y \in Y, \exists x \in X$ such that $f(x) = y$ (see figs. 1.5 and 1.9).

Remark 1.27 (Surjection vs Epimorphism). [Surjection](#) and [Epimorphism](#) are related each other. Consider a non-surjective function $f : X \rightarrow Y' \subset Y$ (see fig. 1.6). One can conclude that there is not an [Epimorphism](#) because $\exists g_1 : Y' \rightarrow Y'$ and $g_2 : Y \rightarrow Y$ such that $g_1 \neq g_2$ because they operates on different [Domains](#) but from other hand $g_1(Y') = g_2(Y')$. For instance we can choose $g_1 = \mathbf{1}_{Y'}, g_2 = \mathbf{1}_Y$. As soon as Y' is [Codomain](#) of f we always have $g_1(f(X)) = g_2(f(X))$.

As result we can say that an [Surjection](#) is a [Epimorphism](#) in **Set** category. Moreover there is a proof [5] of that fact.

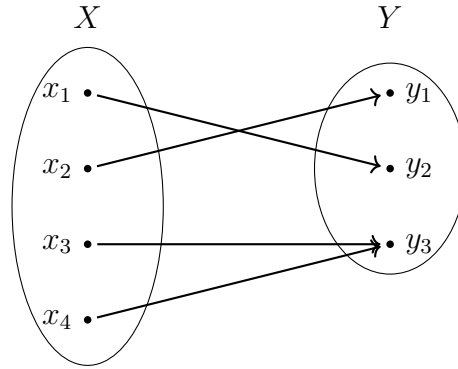


Figure 1.5: A surjective (non-injective) function from domain X to codomain Y

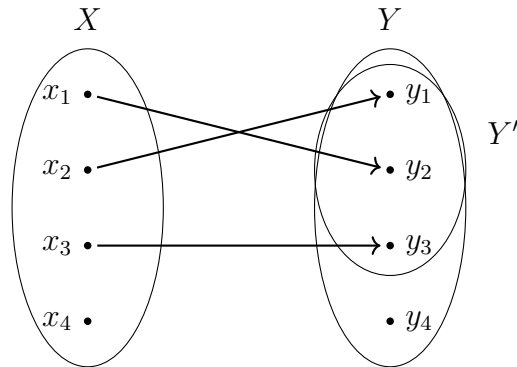


Figure 1.6: A non-surjective function f from domain X to codomain $Y' \subset Y$. $\exists g_1 : Y' \rightarrow Y', g_2 : Y \rightarrow Y$ such that $g_1(Y') = g_2(Y')$, but as soon as $Y' \neq Y$ we have $g_1 \neq g_2$. Using the fact that Y' is codomain of f we got $g_1 \circ f = g_2 \circ f$. I.e. the function f is not epimorphism.

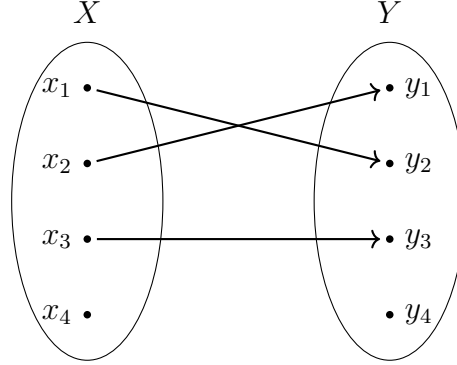


Figure 1.7: A injective (non-surjective) function from domain X to codomain Y

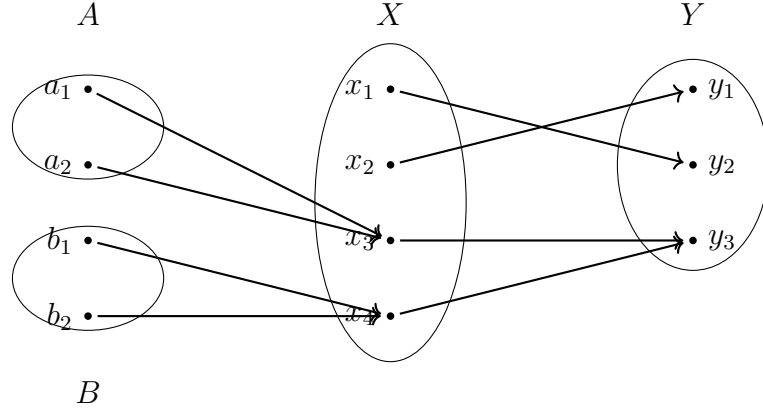


Figure 1.8: A non-injective function f from domain X to codomain Y . $\exists g_1 : A \rightarrow X, g_2 : B \rightarrow X$ such that $g_1 \neq g_2$ but $f \circ g_1 = f \circ g_2$. I.e. the function f is not monomorphism.

Definition 1.28 (Injection). The function $f : X \rightarrow Y$ is injective (or one-to-one function) if $\forall x_1, x_2 \in X$, such that $x_1 \neq x_2$ then $f(x_1) \neq f(x_2)$ (see figs. 1.7 and 1.9).

Remark 1.29 (Injection vs Monomorphism). [Injection](#) and [Monomorphism](#) are related each other. Consider a non-injective function $f : X \rightarrow Y$ (see fig. 1.8). One can conclude that it is not monomorphism because $\exists g_1, g_2$ such that $g_1 \neq g_2$ and $f(g_1(a_1)) = y_3 = f(g_2(b_1))$.

As result we can say that an [Injection](#) is a [Monomorphism](#) in **Set** category. Moreover there is a proof [4] of that fact.

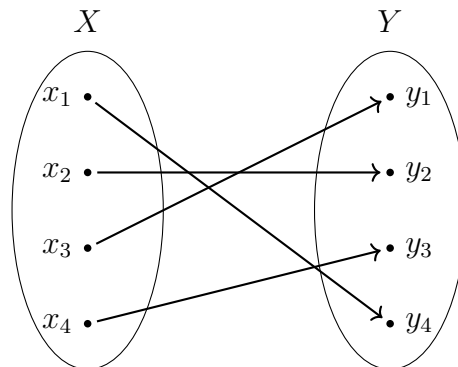


Figure 1.9: An injective and surjective function (bijection)

Definition 1.30 (Bijection). The function $f : X \rightarrow Y$ is bijective (or one-to-one correspondence) if it is an [Injection](#) and a [Surjection](#) (see fig. 1.9).

There is a question what's analog of a single [Set](#). Main characteristic of a category is a structure but the set by definition does not have a structure. Which category does not have any structure? The answer is [Discrete category](#).

Definition 1.31 (Discrete category). Discrete category is a [Category](#) where [Morphisms](#) are only [Identity morphisms](#).

1.2.2 Programming languages

In the programming languages we consider types as [Objects](#) and functions as [Morphisms](#). Particularly we will look into category with 3 objects that are types: `Int`, `Bool`, `String`. There are also several functions between them (see fig. 1.10).

Hask category

Example 1.32 (Hask category). Types in Haskell are considered as [Objects](#). Functions are considered as [Morphisms](#). We are going to implement [Category](#) from fig. 1.10.

The function `isEven` that converts `Int` type into `Bool`.

```
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0
```

There is also [Identity morphism](#) that is defined as follows

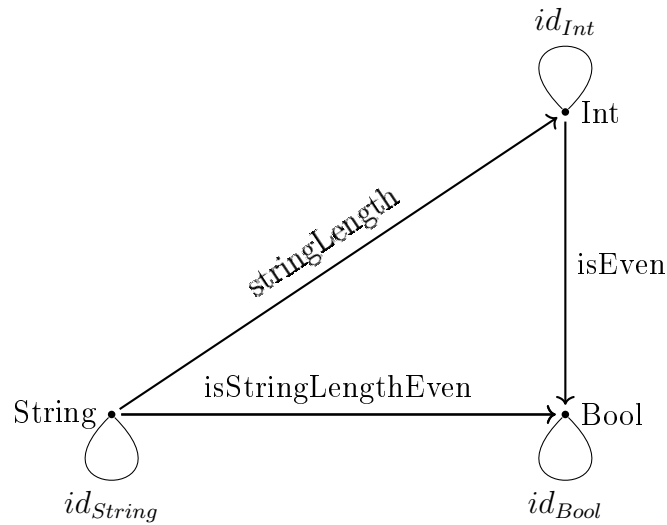


Figure 1.10: Programming language category example. Objects are types: Int, Bool, String. Morphisms are several functions

```
id :: a -> a
id x = x
```

If we have an additional function

```
stringLength :: String -> Int
stringLength x = length x
```

then we can create a [Composition](#) ([Axiom 1.7](#))

```
isStringLengthEven :: String -> Bool
isStringLengthEven = isEven . stringLength
```

Remark 1.33 (Haskell lazy evaluation). Each Haskell type has a special value \perp . The value presents and lazy evaluations make several category law invalid, for instance [Identity morphism](#) behaviour become invalid in specific cases:

The following code

```
seq undefined True
```

produces *undefined* But the following

```
seq (id.undefined) True
seq (undefined.id) True
```


produces *True* in both cases. As result we have (we cannot compare compare functions in Haskell, but if we could we can get the following)

```
id . undefined /= undefined
undefined . id /= undefined,
```

i.e. (1.1) and (1.2) are not satisfied.

C++ category

Example 1.34 (C++ category). We will use the same trick as in [Hask category](#) (Example 1.32) and will assume types in C++ as [Objects](#), functions as [Morphisms](#). We also are going to implement [Category](#) from fig. 1.10.

We also define 2 functions:

```
auto isEven = [](int x) {
    return x % 2 == 0;
};

auto stringLength = [](std::string s) {
    return static_cast<int>(s.size());
};
```

Composition can be defined as follows:

```
// h = g . f
template <typename A, typename B>
auto compose(A g, B f) {
    auto h = [f, g](auto a) {
        auto b = f(a);
        auto c = g(b);
        return c;
    };
    return h;
};
```

The [Identity morphism](#):

```
auto id = [](auto x) { return x; };
```

The usage examples are the following:

```
auto isStringLengthEven = compose<>(isEven, stringLength);

auto isStringLengthEvenL = compose<>(id, isStringLengthEven);

auto isStringLengthEvenR = compose<>(isStringLengthEven, id);
```

Such construction will always provides us the category as soon as we use pure function (functions without effects).

Scala category

Example 1.35 (Scala category). We will use the same trick as in [Hask category](#) ([Example 1.32](#)) and will assume types in Scala as [Objects](#), functions as [Morphisms](#). We also are going to implement [Category](#) from [fig. 1.10](#).

```
object Category {
  def id[A]: A => A = a => a
  def compose[A, B, C](g: B => C, f: A => B):
    A => C = g compose f

  val isEven = (i: Int) => i % 2 == 0
  val stringLength = (s: String) => s.length
  val isStringLengthEven = (s: String) =>
    compose(isEven, stringLength)(s)
}
```

The usage example is below

```
class CategorySpec extends Properties("Category") {
  import Category._
  import Prop.forAll

  property("composition") = forAll { (s: String) =>
    isStringLengthEven(s) == isEven(stringLength(s))
  }

  property("right id") = forAll { (i: Int) =>
    isEven(i) == compose(isEven, id[Int])(i)
  }

  property("left id") = forAll { (i: Int) =>
    isEven(i) == compose(id[Boolean], isEven)(i)
  }
}
```

1.2.3 Physics

TBD

Chapter 2

Objects and morphisms

2.1 Equality

The important question is how can we decide whenever an object/morphism is equal to another object/morphism? The trivial answer is possible for if an [Object](#) is a [Set](#). In this case we can say that 2 objects are equal if they contains the same elements. Unfortunately we cannot do the same for default objects as soon as they don't have any internal structure. We can use the same trick as in [Set vs Category](#) ([Remark 1.23](#)) : if we cannot use “microscope” lets use “telescope” and define the equality of objects and morphisms of a category \mathbf{C} in the terms of whole $\text{hom}(\mathbf{C})$.

Definition 2.1 (Objects equality). Two [Objects](#) a and b in [Category](#) C are equal if there exists an unique [Isomorphism](#) $f : a \rightarrow b$. This also means that also exist unique isomorphism $g : b \rightarrow a$. These two [Morphisms](#) are related each other via the following equations: $f \circ g = \mathbf{1}_a$ and $g \circ f = \mathbf{1}_b$.

Unlike [Functions](#) between [Sets](#) we don't have any additional info ¹ about [Morphisms](#) except category theory axioms which the morphisms satisfied [1]. This leads us to the following definition for morphisms equality:

Definition 2.2 (Morphisms equality). Two [Morphisms](#) f and g in [Category](#) C are equal if the equality can be derived from the base axioms:

- [Composition](#) ([Axiom 1.7](#))
- [Associativity](#) ([Axiom 1.9](#))
- [Identity morphism](#): (1.1), (1.2)

¹ for instance info about sets internals. i.e. which elements of the sets are connected by the considered functions

or [Commutative diagrams](#) which postulate the equality.

As an example lets proof the following theorem

Theorem 2.3 (Identity is unique). *The [Identity morphism](#) is unique.*

Proof. Consider an [Object](#) a and it's [Identity morphism](#) 1_a . Let $\exists f : a \rightarrow a$ such that f is also identity. In the case (1.1) for f as identity gives

$$f \circ 1_a = 1_a.$$

From other side (1.2) for 1_a satisfied

$$f \circ 1_a = f$$

i.e. $f = 1_a$. □

2.2 Initial and terminal objects

Definition 2.4 (Initial object). Let \mathbf{C} is a [Category](#), the [Object](#) $i \in \text{ob}(\mathbf{C})$ is called *initial object* if $\forall x \in \text{ob}(\mathbf{C}) \exists ! f_x : i \rightarrow x \in \text{hom}(\mathbf{C})$.

Definition 2.5 (Terminal object). Let \mathbf{C} is a [Category](#), the [Object](#) $t \in \text{ob}(\mathbf{C})$ is called *terminal object* if $\forall x \in \text{ob}(\mathbf{C}) \exists ! g_x : x \rightarrow t \in \text{hom}(\mathbf{C})$.

As you can see the initial and terminal objects are opposite each other. I.e. if i is an [Initial object](#) in \mathbf{C} then it will be [Terminal object](#) in the [Opposite category](#) \mathbf{C}^{op} .

Theorem 2.6 (Initial object is unique). *Let \mathbf{C} is a category and $i, i' \in \text{ob}(\mathbf{C})$ two [Initial objects](#) then there exists an unique [Isomorphism](#) $u : i \rightarrow i'$ (see [Objects equality](#))*

Proof. Consider the following [Commutative diagram](#) (see fig. 2.1) □

2.3 Product and sum

TBD

2.4 Examples

2.4.1 Set category

TBD

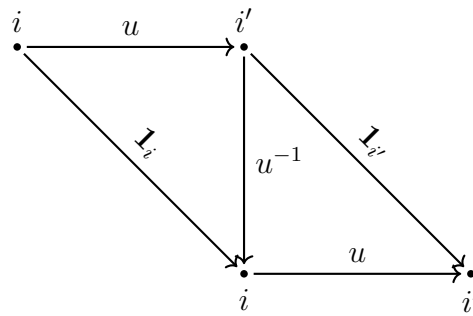


Figure 2.1: Commutative diagram

2.4.2 Programming languages

Hask category

TBD

C++ category

TBD

Chapter 3

Functors

TBD

Chapter 4

Monads

TBD

Index

- C++** category
 - example, [17](#)
- Hask** category
 - example, [15](#)
- Hask** category example, [17](#), [18](#)
- Scala** category
 - example, [18](#)
- Set category
 - example, [12](#)
- Associativity axiom, [10](#), [19](#)
 - declaration, [8](#)
- Bijection
 - definition, [14](#)
- Category, [10](#), [15](#), [17–20](#)
 - definition, [10](#)
 - dual, [10](#)
 - opposite, [10](#)
- Class, [7](#), [9](#), [10](#)
 - definition, [7](#)
- Class of Morphisms
 - remark, [9](#)
- Class of Objects
 - remark, [7](#)
- Codomain, [12](#)
 - definition, [8](#), [12](#)
- Commutative diagram, [20](#)
 - definition, [9](#)
- Composition
 - opposite category, [10](#)
 - remark, [8](#)
- Composition axiom, [8](#), [10](#), [16](#), [19](#)
 - declaration, [8](#)
- Discrete category, [15](#)
 - definition, [15](#)
- Domain, [12](#)
 - definition, [8](#), [12](#)
- Epimorphism, [12](#)
 - definition, [10](#)
- Function, [12](#), [19](#)
 - definition, [12](#)
- Haskell lazy evaluation
 - remark, [16](#)
- Identity is unique theorem, [9](#)
 - declaration, [20](#)
- Identity morphism, [9](#), [10](#), [12](#),
[15–17](#), [19](#), [20](#)
 - definition, [9](#)
- Initial object, [20](#)
 - definition, [20](#)
- Initial object is unique theorem
 - declaration, [20](#)
- Injection, [14](#), [15](#)
 - definition, [14](#)
- Injection vs Monomorphism
 - remark, [14](#)
- Isomorphism, [10](#), [19](#), [20](#)

- definition, [10](#)
- remark, [10](#)
- Monomorphism, [14](#)
 - definition, [10](#)
- Morphism, [8–10](#), [12](#), [15](#), [17–19](#)
 - C++** example, [17](#)
 - Hask** example, [15](#)
 - Scala** example, [18](#)
 - Set** example, [12](#)
 - definition, [8](#)
- Morphisms equality
 - definition, [19](#)
- Object, [7–10](#), [12](#), [15](#), [17–20](#)
 - C++** example, [17](#)
 - Hask** example, [15](#)
 - Scala** example, [18](#)
 - Set** example, [12](#)
 - definition, [7](#)
- Objects equality, [20](#)
 - definition, [19](#)
- Opposite category, [10](#), [20](#)
 - definition, [10](#)
- Set, [12](#), [15](#), [19](#)
 - definition, [12](#)
- Set vs Category
 - remark, [12](#)
- Set vs Category remark, [19](#)
- Surjection, [12](#), [15](#)
 - definition, [12](#)
- Surjection vs Epimorphism
 - remark, [12](#)
- Terminal object, [20](#)
 - definition, [20](#)

Bibliography

- [1] (https://math.stackexchange.com/users/142355/david_myers), D. M. How should i think about morphism equality? — Mathematics Stack Exchange. — URL:<https://math.stackexchange.com/q/1346167> (version: 2015-07-01). <https://math.stackexchange.com/q/1346167>.
- [2] Milewski, B. Category Theory for Programmers / B. Milewski. — Bartosz Milewski, 2018. — <https://github.com/hmemcpy/milewski-ctfp-pdf/releases/download/v0.7.0/category-theory-for-programmers.pdf>.
- [3] Murashko, I. Category theory. — <https://github.com/ivanmurashko/articles/tree/master/cattheory/src>. — 2018.
- [4] ProofWiki. Injection iff monomorphism in category of sets / ProofWiki. — 2018. — https://proofwiki.org/wiki/Injection_iff_Monomorphism_in_Category_of_Sets.
- [5] ProofWiki. Surjection iff epimorphism in category of sets / ProofWiki. — 2018. — https://proofwiki.org/wiki/Surjection_iff_Epimorphism_in_Category_of_Sets.