Category Theory

Ivan Murashko

July 23, 2018

Contents

Introduction

There is an introduction to Category Theory.

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 ob(C) (see fig. 1.1).

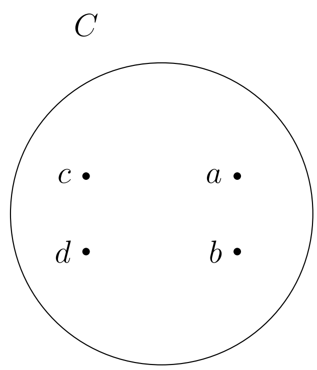

Figure 1.1: Class of objects $\text{ob}(C) = \{a, b, c, d\}$

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

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

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

Morphisms have several properties. [1]

Property 1.5 (Composition)**.** *If we have 3 Objects a, b and c and 2 Morphisms*

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

and

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

then there exists Morphism

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

such that

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

Remark 1.6 (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 $f_{ab}(x) \in b$.

Property 1.7 (Associativity)**.** *The Morphisms Composition (Property 1.5) s should follow associativity property:*

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

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

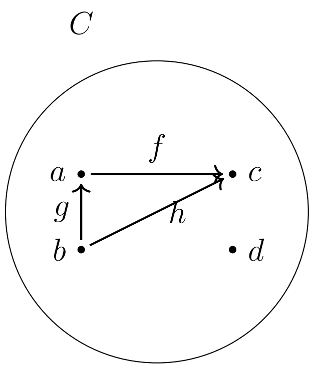

Figure 1.2: Class of morphisms $\hom ob(C) = \{f, g, h\}$, where $h = f \circ g$

Definition 1.8 (Identity morphism)**.** For every Object a we define a special Morphism $\mathbf{1}_a : a \to a$ with the following properties: $\forall f_{ab} : a \to b$

$$\mathbf{1}_a \circ f_{ab} = f_{ab} \tag{1.1}$$

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

$$f_{ba} \circ \mathbf{1}_a = f_{ba}. \tag{1.2}$$

This morphism is called *identity morphism*.

Definition 1.9 (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 $f_{ab} = f_{cb} \circ f_{ac}$.

$$\begin{tikzcd} a \arrow[rr, "f_{ab}"] \arrow[dr, "f_{ac}"'] & & b \\ & c \arrow[ur, "f_{cb}"'] & \end{tikzcd}$$

Remark 1.10 (Class of Morphisms)**.** The Class of Morphisms will be marked as $\hom(C)$ (see fig. 1.2)

Definition 1.11 (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*.

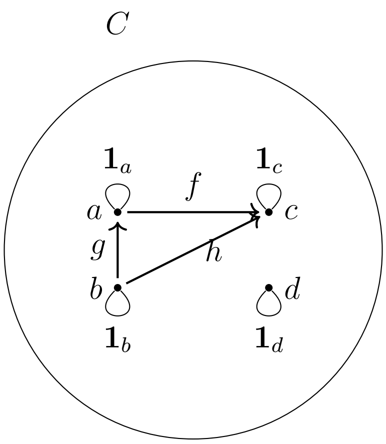

Figure 1.3: Category C. It consists of 4 objects $\mathrm{ob}(C) = \{a, b, c, d\}$ and 7 morphisms $\mathrm{ob}(C) = \{f, g, h = f \circ g, \mathbf{1}_a, \mathbf{1}_b, \mathbf{1}_c, \mathbf{1}_d\}$

Definition 1.12 (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*.

1.1.3 Category

Definition 1.13 (Category)**.** A category **C** consists of

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

For any Object a there should be unique Identity morphism $\mathbf{1}_a$. Any morphism should satisfy Composition (Property 1.5) and Associativity (Property 1.7) 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.

1.2 Examples

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

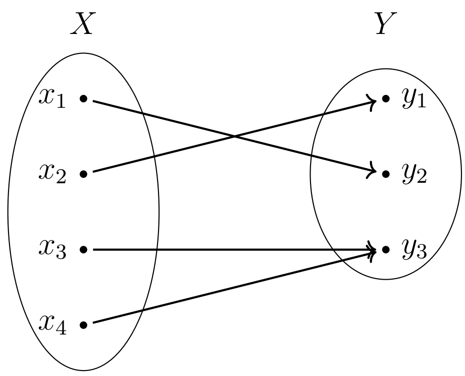

Figure 1.4: A surjective (non-injective) function from domain X to codomain Y

1.2.1 Set category

Definition 1.14 (Set)**.** Set is a collection of distinct object. The objects are called the elements of the set.

Definition 1.15 (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 1.16 (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 1.17 (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" [1]

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" [1] there.

Definition 1.18 (Domain)**.** Given a function $f : X \to Y$, the set X is the domain.

Definition 1.19 (Codomain)**.** Given a function $f : X \to Y$, the set Y is the codomain.

Definition 1.20 (Surjection)**.** The function $f : X \to Y$ is surjective (or onto) if $\forall y \in Y$, $\exists x \in X$ such that $f(x) = y$ (see figs. 1.4 and 1.8).

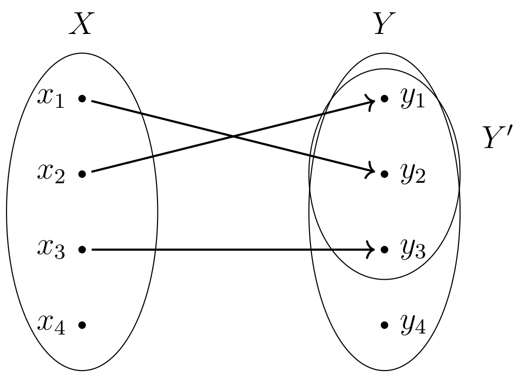

Figure 1.5: A non-surjective function f from domain X to codomain $Y' \subset Y$. $\exists g_1 : Y' \to Y', g_2 : Y \to Y$ such that $g_1(Y') = g_2(Y')$, but as soon as $Y' \neq Y$ we have $g_1 \neq g_2$. Using the fact that Y' is codomain of f we got $g_1 \circ f = g_2 \circ f$. I.e. the function f is not epimorphism.

Remark 1.21 (Surjection vs Epimorphism)**.** Surjection and Epimorphism are related each other. Consider a non-surjective function $f : X \to Y' \subset Y$ (see fig. 1.5). One can conclude that there is not an Epimorphism because $\exists g_1 : Y' \to Y'$ and $g_2 : Y \to Y$ 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 [3] of that fact.

Definition 1.22 (Injection)**.** The function $f : X \to Y$ 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. 1.6 and 1.8).

Remark 1.23 (Injection vs Monomorphism)**.** Injection and Monomorphism are related each other. Consider a non-injective function $f : X \to Y$ (see fig. 1.7). 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 [2] of that fact.

Definition 1.24 (Bijection)**.** The function $f : X \to Y$ is bijective (or one-to-one correspondence) if it is an Injection and a Surjection (see fig. 1.8).

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 struc-

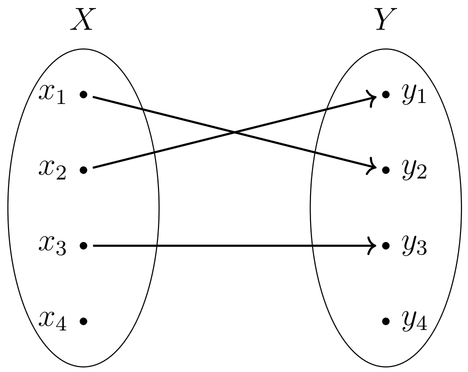

Figure 1.6: A injective (non-surjective) function from domain X to codomain Y

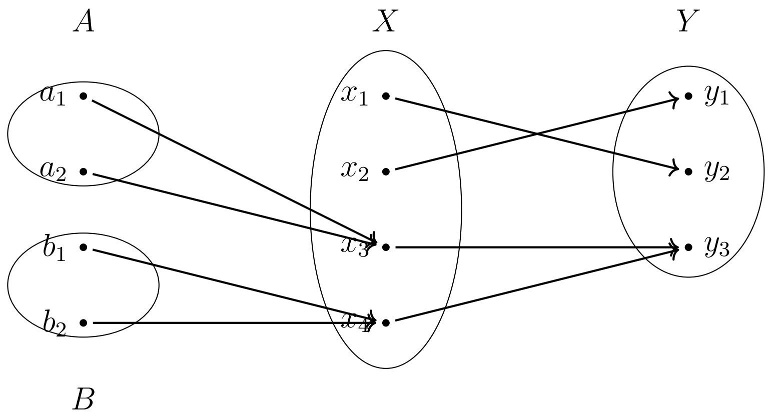

Figure 1.7: A non-injective function f from domain X to codomain Y. $\exists g_1 : A \to X, g_2 : B \to X$ such that $g_1 \neq g_2$ but $f \circ g_1 = f \circ g_2$. I.e. the function f is not monomorphism.

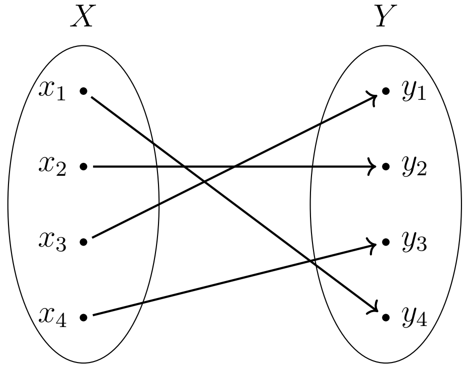

Figure 1.8: An injective and surjective function (bijection)

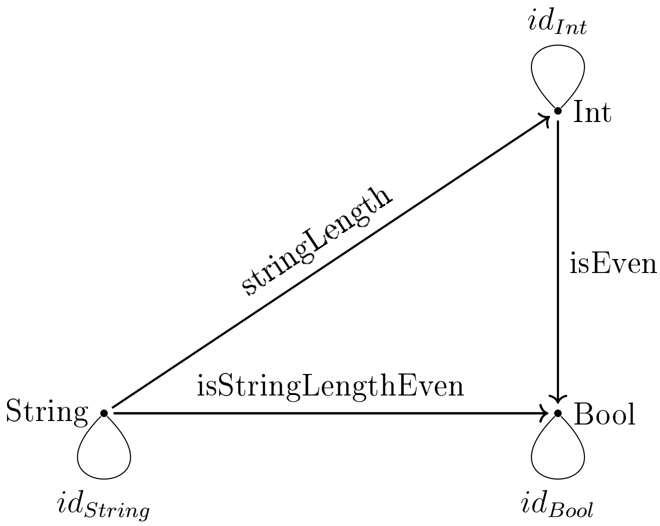

Figure 1.9: Programming language category example. Objects are types: Int, Bool, String. Morphisms are several functions

ture. Which category does not have any structure? The answer is Discrete category.

Definition 1.25 (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. 1.9).

Hask category

Example 1.26 (**Hask** category)**.** Types in Haskell are considered as Objects. Functions are considered as Morphisms. We are going to implement Category from fig. 1.9.

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

```
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 (Property 1.5)

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

Remark 1.27 (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 1.28 (C++ category)**.** We will use the same trick as in **Hask** category (Example 1.26) and will assume types in C++ as Objects, functions as Morphisms. We also are going to implement Category from fig. 1.9.

We also define 2 functions:

```
std::function<bool(int)> isEven =
[](int x)
{
  return x % 2 == 0;
```

```
};

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

Composition can be defined as follows:

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

The Identity morphism:

```
std::function<bool(bool)> id_bool =
[](bool x)
{
  return x;
};

std::function<std::string(std::string)> id_string =
[](std::string x)
{
  return x;
};
```

The usage examples are the following:

```
std::function<bool(std::string)> isStringLengthEven =
compose<>(isEven, stringLength);

std::function<bool(std::string)> isStringLengthEvenLeft =
```

```
compose<>(id_bool, isStringLengthEven);

std::function<bool(std::string)> isStringLengthEvenRight =
compose<>(isStringLengthEven, id_string);
```

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

Chapter 2

Objects and morphisms

2.1 Equality

2.1.1 Equality of objects

via unique isomorphism

2.1.2 Equality of morphisms

TBD

2.2 Initial and terminal objects

TBD

2.3 Product and sum

TBD

2.4 Examples

2.4.1 Set category

TBD

2.4.2 Programming languages

Hask category

TBD

C++ category

TBD

Chapter 3

Functors

TBD

Chapter 4

Monads

TBD

Index

Bibliography

[1] 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.

[2] ProofWiki. Injection iff monomorphism in category of sets / ProofWiki. — 2018. — https://proofwiki.org/wiki/Injection_iff_Monomorphism_in_Category_of_Sets.

[3] ProofWiki. Surjection iff epimorphism in category of sets / ProofWiki. — 2018. — https://proofwiki.org/wiki/Surjection_iff_Epimorphism_in_Category_of_Sets.