

A First Introduction to Category Theory and Logic

Dirk Pattinson
Institut für Informatik, LMU München

20th March 2002

1 Categories and Functors

1.1 Introduction

What is category theory?

Category theory is the systematic study of similarities. (J. Benabou)

This is the motto, to which we adhere throughout these notes. Concretely, we demonstrate the meaning of this slogan by studying the similarities between two apparently different formalisms: Cartesian closed categories and simply typed lambda calculi. The main result which we are going to establish is the equivalence between (the category of) cartesian closed categories and (the category of) typed lambda calculi.

There's several books covering the topics presented. The book [5] is an introduction to the basic topics, suitable for computer scientists. Similar in spirit but slightly more extensive is [1], who also focus on computer science examples.

The “classical” reference, which examples mostly from mathematics, still is [4]. A very good reference is the tree volume handbook [2]. All slogans (except the first) are taken from [3], the introductory chapter of which also contains a brief introduction to the basic concepts of category theory.

1.2 Definition and Examples

In contrast to set theory, which builds on the basic notion of set and membership, the basic notion in category theory is the notion of function. This is formalised in the first definition: A category is a collection of objects, together with arrows between them, which can be composed.

Definition 1.1. A *category* \mathbb{C} comprises

- A collection $|\mathbb{C}|$ of *objects* item A collection $\mathbb{C}(A, B)$ of *morphisms* for every pair of objects $A, B \in |\mathbb{C}|$ (We write $f : A \rightarrow B$ for $f \in \mathbb{C}(A, B)$ when \mathbb{C} is clear from the context)
- A mapping $\circ_{ABC} : \mathbb{C}(B, C) \times \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C)$ (we generally just write $g \circ f$ instead of $\circ_{ABC}(g, f)$ for $f : A \rightarrow B$ and $g : B \rightarrow C$)

such that the following laws hold:

- For every $A \in |\mathbb{C}|$, there exists $1_A : A \rightarrow A$ such that $f \circ 1_A = f = 1_B \circ f$ for all $A, B \in |\mathbb{C}|$ and all $f \in \mathbb{C}(A, B)$. We call 1_A the *identity mapping*.
- Composition of morphisms is associative, that is, $h \circ (g \circ f) = (h \circ g) \circ f$ for all $f : A \rightarrow B$, $g : B \rightarrow C$ and $h : C \rightarrow D$.

Note that for all objects A in a category \mathbb{C} , the identity morphism is necessarily unique.

This definition is general enough to accommodate lots of mathematical structures. As a slogan, one could say:

May objects of interest in mathematics congregate in categories.

This allows us to compare two classes of structures by relating the categories, in which they congregate. The following examples substantiate the above slogan:

- Example 1.2.**
1. The collection of all sets, with set-theoretic functions as morphisms between them, forms a category, which we denote by **Set**.
 2. Algebraic structures usually form categories: There is a category **Grp** of groups (and group homomorphisms), a category **Mon** of monoids (and monoid homomorphisms), a category **Rng** of rings (and ring homomorphisms) etc. pp.
 3. The collection of topological spaces and continuous functions forms a category (denoted by **Top**)
 4. There's the category of preorders (and order preserving maps), which we denote by **Preord**.
 5. When studying semantics of computation, one often looks at the category **CPO** of complete partial orders (and continuous functions).

Note that categories are often contained in one another: Since every complete partial order is also a preorder, we have $\mathbf{CPO} \subseteq \mathbf{Preord}$. Passing from complete partial orders to topological spaces via the Scott topology, we also have $\mathbf{CPO} \subseteq \mathbf{Top}$.

It is immediate that all the structures from the previous examples form categories. To formally prove this, one has to show that the identity function is a morphism of the required kind and that two morphisms compose. Associativity is inherited from the associativity of function composition.

Apart from being an important tool in maths, categories are also present in many areas of computer science.

Here's a category, which is important in computer science, since its morphisms have to be realisable by a recursive function:

Definition 1.3. | The category ω -Set has pairs (X, E) as objects, where X is a set and $E : X \rightarrow \mathcal{P}(\mathbb{N})$ is a function such that $E(x) \neq \emptyset$ for all $x \in X$. Think of $n \in E(x)$ as a code for x .

The morphisms between ω -sets (X, E) and (Y, F) are those functions $f : X \rightarrow Y$ for which there exists a partial recursive function p which tracks f , in the sense that $p(n)$ is defined and $p(n) \in F(f(x))$ for all $x \in X$ and all $n \in E(x)$.

Unlike in the category Set of sets and functions, a function only qualifies as morphism in the category of ω -Sets, if it can be "realised" by a partial recursive function.

Apart from forming a category, many mathematical entities actually have the structure of a category. Again as a slogan:

Many objects of interest in Mathematics have the structure of a category.

This slogan also calls for examples:

- Example 1.4.**
1. Every set S gives rise to a category \mathbb{S} by putting $|\mathbb{S}| = S$ and $\mathbb{S}(x, y) = \begin{cases} 1_x & \text{if } x = y \\ \emptyset & \text{o/w} \end{cases}$. Categories like this one, which have no morphisms except for the identity morphisms, are also called *discrete*.
 2. Every preorder (P, \leq) gives rise to a category \mathbb{P} by putting $|\mathbb{P}| = P$ and $\mathbb{P}(p, q) = \begin{cases} (p, q) & \text{if } p \leq q \\ \emptyset & \text{o/w} \end{cases}$. We put $(q, r) \circ (p, q) = (p, r)$.
 3. A monoid (M, \bullet) gives rise to a category \mathbb{M} as follows: Put $|\mathbb{M}| = \{0\}$ and $\mathbb{M}(0, 0) = M$. For $m, n \in \mathbb{M}(0, 0)$ we put $m \circ n = m \bullet n$.

We can be even more concrete and instantiate the example above with a concrete preorder:

Example 1.5. Suppose \mathcal{L} is the language of propositional logic some set of propositional variables. Then \mathcal{L} gives rise to a category \mathbb{L} as follows: The objects of \mathbb{L} are the formulas of \mathcal{L} and then existence of a morphism $\phi \rightarrow \psi$ is equivalent to $\phi \vdash \psi$ by means of propositional logic.

The above example “categorifies” a proof irrelevant version of propositional logic. That is, the only thing we are interested in is that ϕ is provable from ψ , but we disregard the actual proof. The simply typed lambda calculus, which we shall introduce later, is a proof relevant version of the same theme.

1.3 Morphisms between Categories: Functors

If categories are of interest in mathematics, they should better from a category themselves. In order to define a category of categories, we need to define a notion of morphism between categories. These morphisms, which are called functors, are structure preserving mappings between categories.

Definition 1.6. Suppose \mathbb{C} and \mathbb{D} are categories. A *functor* $F : \mathbb{C} \rightarrow \mathbb{D}$ between \mathbb{C} and \mathbb{D} is given by

- An assignment $F_o : |\mathbb{C}| \rightarrow |\mathbb{D}|$ (We mostly write FA instead of $F_o(A)$.)
- For all $A, B \in \mathbb{C}$, an assignment $F_{AB} : \mathbb{C}(A, B) \rightarrow \mathbb{D}(FA, FB)$ (We just write Ff for $F_{AB}(f)$ in the sequel.)

such that $F(g \circ f) = Fg \circ Ff$ and $F1_A = 1_{FA}$ for all $A, B, C \in \mathbb{C}$ and all $f : A \rightarrow B, g : B \rightarrow C$.

One readily shows that the identity is a functor and that functors compose:

Lemma 1.7. *Categories and Functors form a category.*

This category, the category of all categories, will be denoted by Cat in the sequel.

We give a concrete example of a functor in the concept of functional programming: we show that functors capture (some aspects of) data types. In functional programming, functors can be used to model data types.

Example 1.8. Functors can be used to model type constructors in functional programming languages. To see an example of this, consider the category \mathbf{Set} of sets and functions. To every type X (which we take as represented by the set of its elements) one can associate the type $\mathbf{List}(X)$ of lists over elements of X . Defining $\mathbf{List}f$ by $\mathbf{List}f([x_0, \dots, x_n]) = [f(x_0), \dots, f(x_n)]$, we obtain a functor $\mathbf{List} : \mathbf{Set} \rightarrow \mathbf{Set}$. Note that $\mathbf{List}(f)$ is precisely the construct $\mathbf{map}(f)$, as known in functional programming.

Other examples of functors can be obtained by looking at the functors between the categories constructed in Example 1.4. There, one can compare the “natural” notion of morphisms between these structures and the functors between the induced categories.

Exercise 1.9. Suppose X and Y are sets (preorders / monoids). Then the functions (order-preserving functions / homomorphisms of monoids) are in 1-1 correspondence with the functors $\mathbb{X} \rightarrow \mathbb{Y}$ between X and Y regarded as categories.

In their book, Lambek and Scott formulate another slogan, saying that many objects of interest in mathematics arise as functors from categories to sets. One such example is given in the next exercise.

Exercise 1.10. Let \mathbb{C} be the category $\bullet \begin{array}{c} \rightrightarrows \\ \leftleftarrows \end{array} \bullet$ (with identity arrows omitted). Show that every functor $\mathbb{C} \rightarrow \mathbf{Set}$ specifies a directed graph and vice versa.

When defining functors, categorists have the dangerous habit of only specifying what a functor does to objects of the source category and leave the action on morphisms implicit. The next exercise shows, that the action of a functor on objects by no means determines its action on morphisms:

Exercise 1.11. Find two distinct functors $F, G : \mathbf{Grp} \rightarrow \mathbf{Grp}$, whose object part is the identity, ie. such that $F_o(X) = G_o(X) = X$ for all groups X .

References

- [1] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, 1989.
- [2] F. Borceux. *Handbook of Categorical Algebra*. Cambridge University Press, 1994. 3 Volumes.

- [3] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1986.
- [4] S. MacLane. *Categories for the Working Mathematician*. Springer, 1971.
- [5] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991.