Category Theory for Programmers

Bartosz Milewski

Unofficial Texinfo Format 0.igaltabachnik0.1 (February 2, 2016),
based on https://github.com/sarabander/sicp-pdf (September 3, 2017).

Preface

> For some time now I've been floating the idea of writing a book about category theory that would be targeted at programmers. Mind you, not computer scientists but programmers — engineers rather than scientists. I know this sounds crazy and I am properly scared. I can't deny that there is a huge gap between science and engineering because I have worked on both sides of the divide. But I've always felt a very strong compulsion to explain things. I have tremendous admiration for Richard Feynman who was the master of simple explanations. I know I'm no Feynman, but I will try my best. I'm starting by publishing this preface — which is supposed to motivate the reader to learn category theory — in hopes of starting a discussion and soliciting feedback.

I will attempt, in the space of a few paragraphs, to convince you that this book is written for you, and whatever objections you might have to learning one of the most abstract branches of mathematics in your "copious spare time" are totally unfounded.

My optimism is based on several observations. First, category theory is a treasure trove of extremely useful programming ideas. Haskell

programmers have been tapping this resource for a long time, and the ideas are slowly percolating into other languages, but this process is too slow. We need to speed it up.

Second, there are many different kinds of math, and they appeal to different audiences. You might be allergic to calculus or algebra, but it doesn't mean you won't enjoy category theory. I would go as far as to argue that category theory is the kind of math that is particularly well suited for the minds of programmers. That's because category theory — rather than dealing with particulars — deals with structure. It deals with the kind of structure that makes programs composable.

…

1

Category: The Essence of Composition

A CATEGORY is an embarrassingly simple concept. A category consists of *objects* and *arrows* that go between them. That's why categories are so easy to represent pictorially. An object can be drawn as a circle or a point, and an arrow… is an arrow. (Just for variety, I will occasionally draw objects as piggies and arrows as fireworks.) But the essence of a category is *composition*. Or, if you prefer, the essence of composition is a category. Arrows compose, so if you have an arrow from object A to object B, and another arrow from object B to object C, then there must be an arrow - their composition - that goes from A to C.

Arrows as Functions

Is this already too much abstract nonsense? Do not despair. Let's talk concretes. Think of arrows, which are also called morphisms, as functions. You have a function `f` that takes an argument of type A and returns a B. You have another function g that takes a B and returns a

C. You can compose them by passing the result of `f` to `g`. You have just defined a new function that takes an A and returns a C.

In math, such composition is denoted by a small circle between functions: $g \circ f$. Notice the right to left order of composition. For some people this is confusing. You may be familiar with the pipe notation in Unix, as in:

```
lsof | grep Chrome
```

Or the chevron >> in F#, which both go from left to right. But in mathematics and in Haskell functions compose right to left. It helps if you read $g \circ f$ as "g after f."