Functional Programming & Category Theory

An Informal Introduction

Aadi Rave
Purdue University
September 19, 2024

A Quick Intro to Functional Programming

Function composition, higher-order functions & immutability

The Commandments of FP

- Thou shalt use **pure functions**
- Thou shalt avoid **mutable state**
- Thou shalt embrace **first-class functions**
- Thou shalt use **higher-order functions**
- Thou shalt favor **function composition**
- Thou shalt use **recursion**
- Thou shalt swear allegiance to **monads**
- Thou shalt keep code **declarative**

The Important Stuff

- **Immutability** Once data is created, it should not be modified.
- **Pure Functions** Do not depend on program state, and do not modify program state.
- **First-Class Functions** Treat functions as values – return them, pass them as arguments, assign them to variables
- **Higher-Order Functions** Compose simple functions to do complex things

A Quick Demo of Function Composition

Consider the simple problem of taking the sum of squares of even numbers in a list. Let's compare the C and OCaml ways of solving the problem.

An Informal Intro to Category Theory

Objects, Morphisms, Functors & Monads

What is a category?

Think of category theory as the mathematics of **structures** and their **relations.** A category consists of

- **objects**
- **morphisms**

Properties of Categories

- **Composition**

$$\forall f : A \to B, g : B \to C, \exists g \cdot f : A \to C$$

- **Associativity**

$$(h \cdot g) \cdot f = h \cdot (g \cdot f)$$

- **Identity**

$$\exists id_A : A \to A, id_B : B \to B \ni f \cdot id_A = f, id_B \cdot f = f, \forall f : A \to B$$

Partial Ordering as a Category

Partial Ordering has three properties:

- **Reflexivity**

$$a \leq a$$

- **Antisymmetry**

$$a \leq b, b \leq a \implies a = b$$

- **Transitivity**

$$a \leq b, b \leq c \implies a \leq c$$

Think of the properties of a category: composition, associativity, and identity. Let's see how partial ordering fulfils each of them.

Functors

A **functor** is a morphism between categories, ie. a functor $T : A \to B$ maps each object of category A to an object in category B and each morphism in category A to a morphism in category B.

Monads

A **monad** is like a pattern that helps you manipulate data types, and actions, while dealing with their side effects elegantly.

A monad has three parts:

- A functor T to add extra context to a value
- A unit η which takes a value and wraps it in T
- A multiplication operator μ to unwrap a value from the context

In the real world, we change what our components do and what our functor is, depending on requirements. Let's look at Option types using a simple Python implementation.

Bonus: Kliesli Categories

What if you want to compose monads? If you apply the η function on an object A twice, we get $M(M(A))$, which we can't deal with using μ! The solution? A **Kliesli Category.** Intuitively, we use these to "flatten" objects from $M(M(A)) \rightarrow M(A)$, allowing us to compose monads! Let's go back to our previous example and see it implemented in Python.

Monads Everywhere

Every problem in programming can be solved using abstraction – except using too much abstraction. With great power comes great responsiblity!

Applications

- Error Handling
- Concurrency
- List comprehension

Error Handling with Monads

Here's some error checked OCaml code to extract an integer from a string:

```
let safe_parse_int s =
  try Some (int_of_string s)
  with Failure _ -> None
```

Compare this to error checking a C function like `strtol`!

Languages like **Rust** and **Scala** have adopted the monadic way of handling errors. Pointers in newer languages like **Zig** can be made optional using monadic logic behind the scenes.

Further Reading

Textbooks

Ordered from least formal to most formal, some resources to check out are:

- Category Theory for Programmers (B. Milewski)
- Basic Category Theory for Computer Scientists (B. Pierce)
- Algebra: Chapter Zero (P. Aluffi)
- Category Theory (S. Awodey)