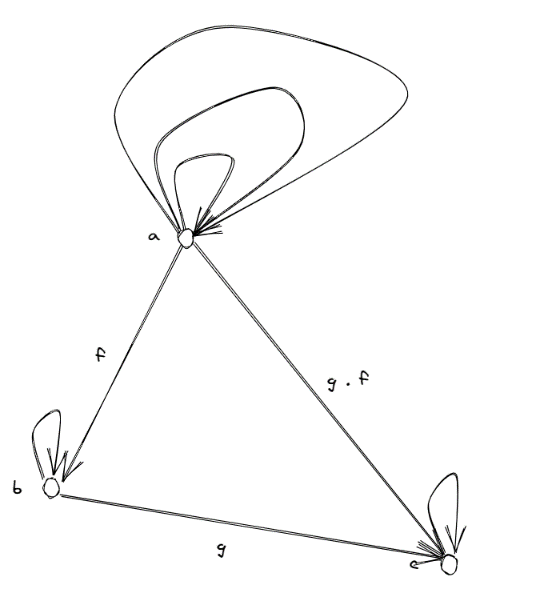
# What is a Function (Part I)

## Introduction

Designing function is an important part of designing and implementing Janiva programming language because it is a templating language with functional programming support. To design functions, we need to fully understand the concept of what functions are, both in mathematical and programmatical ways. In this series, I will quickly go through some core concepts in category theory such as universal construction, functors, and algebraic data types, and explain functions by putting them together. After that, we are going to implement on top of these theories.

## What is Category?

To understand what a function is, we need to first understand category, and categorical way of describing things.

Category theory is a conceptual language used to describe systems of structures. It focuses on objects and morphisms between them, which are relationships. Categories differ from sets in that:

* Categories contain identity morphisms pointing from any object to itself.
* Morphisms in categories can be composed.

However, sets can be represented using categories with objects and identity morphisms only. Such categories are called discrete categories.

形状

描述已自动生成

This is a simple category with 3 objects: a, b, and c. There are also morphisms:

* f :: a -> b
* g :: b ->c
* g . f :: a -> c which is composed by f and g (read as g after f)
* ida :: a -> a
* idb :: b -> b
* idc :: c -> c

Also note that there are three identity morphisms on a, one of which is composed by the other two.

### Category of Real Numbers

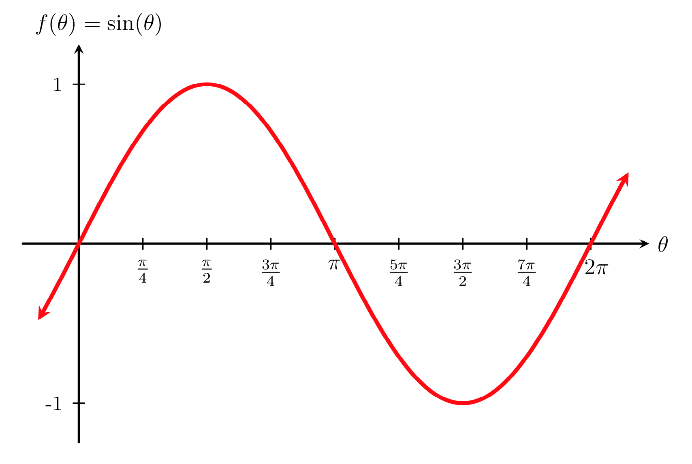
We can construct a category of numbers where objects are real numbers and morphisms are relationships between them. One typical relationship between real numbers is *Greater or Equal to.* Here we have three objects in this category and morphisms are >=. To make it a category, we have to ensure:

* (a >= b, b >= c) derives a >= c, which is the composition rule
* a >= a, which is the identity.

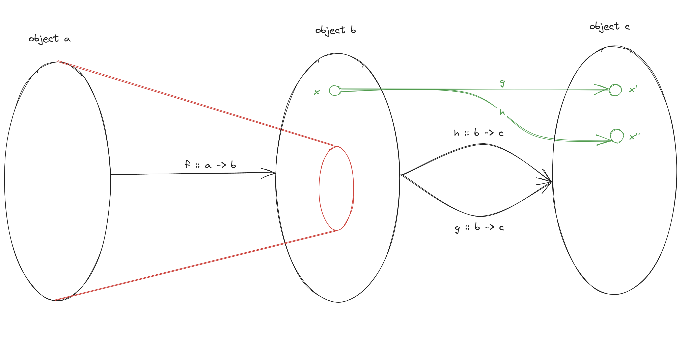
### Category of Sets

There can also be categories of sets, in which sets are objects and relations between sets are morphisms. In mathematics, functions map items between sets, for example, maps ***R*** to ***R,*** but the codomain is much smaller than domain, which is (-1, 1). In programming languages, the concept of (pure) function is similar. You give it a value of some **type**, and it gives you another value back, and that **type** is a set of values. For example, **Bool: {true, false}.** You can write infinitely many functions that map **Int** to **Bool,** such as:

* f1 a = a > 1
* f2 a = a > 1.1
* …

In the category theory of sets (***Set***), we have a slightly different way of describing functions, because sets in ***Set*** are just objects in a category, which is atomic, and you are not able to look inside of it. Therefore, we must use morphisms to describe the properties of sets or functions.

#### Surjective function

Surjective functions ensure that for every element in the codomain, there must be at least one element in the domain corresponding to it. How do we describe such a relation in category? Well, let’s introduce two categories a and b, as shown in this figure, where objects are represented by black circles and morphisms are black arrows. Let’s say that is a surjective function and a morphism from a to b at the same time. Since we cannot look inside objects a and b in category theory, we need to introduce another auxiliary object c to help us describe this. There are two almost identical morphisms g and h, which only differ in that, h maps x to x’ while g maps x to x’’. Any other mappings in g and h are exactly same same. Now let’s compose f with g and h, since f is surjective, as we assume, if and are identical, there’s only one possibility, which is g is identical to h. Let’s think oppositely if f is non-surjective, which only covers a subset of b, then and are identical as long as the difference of g and h lays outside the codomain of f.

Where epimorphism is the category version of surjective.

一群不同颜色的气球

低可信度描述已自动生成

#### Injective function

Injective functions ensure that for each item in the codomain, there’s only one item in the domain mapped to it, which means there is no many-to-one relationships. How do we describe this property in category theory?

This time, we put object c on the other side, having morphism g and h going from c to a. Again, g and h are almost identical, except that g maps x to x’ while h maps x to x’’. If f is injective, which ensures one-to-one mapping, then x’ and x’’ must be mapped to different items. So if we compose (g, h) with f, and , we know that h must be identical to g, otherwise f would map the result to different values. This is written as

Where monomorphism is like injective in category theory. And if a morphism f is monomorphism and epimorphism at the same time, then it is isomorphism.

Now we managed to describe a function's property using a purely categorical way without looking at what’s inside of a set. However, functions under this construction are still in a set called Hom-Set. A Hom-Set is a set which contains all morphisms from a to b. Now let’s see why it’s not enough to have just Hom-Sets to fully understand functions in programming languages.

Functions in programming languages are mappings between data types, and it makes sense that we treat the type system of a programming language as a category and functions as morphisms. However, in some languages which are called functional programming languages, like Haskell, functions are more than just morphisms, because they can be passed like values. Functions become a part of type system in functional programming languages! But the way we used to describe functions doesn’t fit them into the category. In the rest of this blog, I’ll try to use universal construction to solve this problem.

# References

[1] [Category Theory (Stanford Encyclopedia of Philosophy)](https://plato.stanford.edu/entries/category-theory/#:~:text=Category%20theory%20has%20come%20to%20occupy%20a%20central,theory%20of%20structures%20and%20of%20systems%20of%20structures.)