TRABAJO FIN DE GRADO

Teoría de Categorías y Programación Funcional

Diego Pedraza López

tutorizado por

Prof. Tutor José A. Alonso Jiménez

UNIVERSIDAD DE SEVILLA

Teoría de Categorías y Programación Funcional

Diego Pedraza López

2018

Memoria presentada como parte de los requisitos para la obtención del título de Grado en Matemáticas por la Universidad de Sevilla.

Tutorizada por

Prof. Tutor José A. Alonso Jiménez

Dedicado a Mila.

Índice general

Sumario

En esencia, la teoría de categorías es el estudio de la composición. Una categoría es una colección de objetos y morfismos entre ellos de manera que la composición tenga sentido. Este tipo de estructura ressulta ser muy común en la mayoría de los campos de las matemáticas. Es más, tiene un fuerte vínculo con la lógica y la teoría de tipos a través de las categorías cartesianamente cerradas. En programación funcional, algunos diseños como las mónadas son originarias de la teoría de categorías. Para poder hablar de estos temas, primero tendremos que entender las construcciones comunes que pueden ser definidas sobre una categoría o incluso entre categorías. Exploraremos el campo de la teoría de categorías y sus conexiones con otros campos de la matemáticas usando programación funcional, específicamente Haskell.

Abstract

In essence, category theory is the study of the composition. A category is a collection of objects and morphism between them where composition of morphism makes sense. This kind of structure happens to be very common in most fields of mathematics. Furthermore, it has strong links with logic and type theory through cartesian closed categories. In functional programming, some design patterns like monads originate from category theory. In order to discuss this topics, first we'll have to understand common constructions that can be defined over a category or even between categories. We will explore the realm of category theory and its connections with other fields of mathematics using functional programming, specifically Haskell.

1 Introducción

La teoría de categorías nos ofrece al mismo tiempo un lenguaje común por el que se pueden entender matemáticos de distintas ramas a través de una poderosa abstracción a la composición de funciones. Es más, la teoría de categorías ha encontrado un interés en el campo de la ciencia de la computación. Para explorar la teoría de categorías haremos uso de la programación funcional. Más concretamente, usaremos Haskell, que es uno de los lenguajes puramente funcionales más populares.

1.1 Programación funcional

La programación funcional es un estilo de programación que tiene sus orígenes en IPL (1956) y Lisp (1958). Aunque durante mucho tiempo ha tenido un interés más académico que práctico, ha estado teniendo más atención en los últimos años. Lenguajes de gran popularidad como C++ o Java han introducido características de la programación funcional en los últimos años. [1] Algunos lenguajes, como Haskell o Standard ML, están diseñados con programación funcional en mente. Encuentran su inspiración en las matemáticas y sus características principales son:

- *Funciones puras*. Las subrutinas no dejan efectos secundarios en memoria, por lo que sólo dependen en sus argumentos y no en el estado de la máquina. Por lo tanto, las funciones puras son más cercanas a las funciones con las que trabajan los matemáticos.

[1] Since the introduction of the STL (Standard Template Library) – about 1994 – there has been a steady and cautious increase in the use of functional programming techniques in C++. – Bjarne Stroustrup, creador de C++

- *Sistema de tipos fuertes*. El lenguaje categoriza los datos por tipos y restringe la aplicaciones de funciones a ciertos tipos, de manera que la expresión `"texto" + 1` da un error de compilación. A menudo hay un sistema de *inferencia de tipo*, de manera que no es necesario tener que escribir los tipos constantemente.
- *Funciones polimórficas*. En consecuencia del punto anterior y para permitir escribir funciones genéricas, se permite escribir funciones que actúen tipos arbitrarios. Por ejemplo, en Haskell, la declaración de tipo

  ```
  f :: forall a. a -> a
  ```

  describe `f` como una función que puede ser aplicado a todo tipo y devuelve el mismo tipo de salida. Una función equivalente en C++ sería de la forma:

  ```
  template<class A>
  A f(A arg);
  ```

  Sin embargo, la programación funcional permite imponer restricciones a los tipos polimórficos que son validables en compilación a través de mecanismos como las *clases de tipos* en Haskell. Son similares a las *interfaces* de los lenguajes de programación orientada a objetos como Java. En algunos lenguajes funcionales, se permite además imponer restricciones de tipos al tipo polimórfico devuelto de una función, por ejemplo:

  ```
  read :: forall a. (Read a) => String -> a
  ```

  Esta función interpreta una cadena de texto como un tipo cualquiera `a` que sea de la clase de tipo `Read`.
- *Tipos de datos algebraicos*. Pueden representar suma de tipos, además de producto de tipos. Esto permite la existencia de tipos definiciones recursivas como listas o árboles sin hacer uso de punteros. Además no son mutables (por el hecho de que las funciones no pueden tener efectos secundario), por lo que no hay una diferencia esencial entre los datos del constructor y los datos que almacena el tipo de datos.
  Por esta razón, la programación funcional suele resultar algo difícil para programadores acostumbrados a programar con efectos secundarios. Por otro lado, permite procesar los argumentos de una función por patrones.
- *Evaluación perezosa*. Aunque no es característica de todos los lenguajes funcionales –por ejemplo, SML no lo incorpora–, muchos lenguajes funcionales como Haskell y Mirando lo adoptan. La evaluación perezosa permite que los argumentos de una función sean evaluados sólamente

cuando sea necesario. Esto permite trabajar estructuras de datos infinitas, que en todo momento sólo están parcialmente definidas.

1.2 Haskell

Vamos a introducir lo mínimo de Haskell para poder entender el código que aquí se da. No tendremos una imagen completa del lenguaje, pero sí lo suficiente para poder seguir el código. Si ya está familiarizado con Haskell, puede saltarse esta sección.

Haskell nos permite *declarar* una variable y su tipo:

```
x :: Int
```

y *definir* el valor de dicha variable:

```
x = 8
```

Evidentemente, implementa aritmética básica:

```
y = (x + 4) * 5
-- Esto es un comentario
-- y = 60
```

Las funciones se definen de forma similar:

```
duplica :: Int -> Int
duplica x = 2 * x
```

Obsérvese que no se ponen paréntesis alrededor de los parámetros. Tampoco se usan paréntesis cuando aplicamos la función, es decir, `duplica 3` es una expresión correcta.

Las funciones de varios parámetros se definen como funciones de orden superior:

```
multiplica :: Int -> Int -> Int
multiplica x y = x * y
```

Podemos entender que `multiplica` toma dos enteros y los multiplica, pero también se puede interpretar que `multiplica` se aplica a `x` y devuelve una función con tipo `Int -> Int`, que podemos a su vez aplicar al otro argumento. Esta idea nos lleva al concepto de funciones parcialmente aplicadas:

```
duplica' :: Int -> Int
duplica' x = multiplica 2 x
```

Podemos definir funciones polimórficas, como explicamos en la sección anterior:

```
max :: Ord a => a -> a -> a
max x y =
  if x < y
  then x
  else y
```

Aquí vemos además la estructura condicional habitual `if-then-else`.

Trabajaremos a menudo a nivel de funciones, por lo que resultará útil echar un ojo al operador composición:

```
(.) :: (b -> c) -> (a -> b) -> a -> c
f . g = \x -> f (g x)
```

Toda función declarada entre paréntesis, como `(.)`, se corresponde con un operador infijo. Por otro lado, la expresión `\x -> f (g x)` representa una función de un sólo parámetro `x` y que devuelve `f (g x)`. Más específicamente, representa la expresión lambda $\lambda x.f(gx)$.

Hablemos también un poco sobre los tipos de datos algebraicos. Un tipo de dato se define con una expresión de la forma

```
data Dato t1 ... tn =
  Constructor1 a11 ... a1m | ... | ConstructorM aM1 ... aMq
```

donde `t1 ... tn` son argumentos del tipo `Dato`, que tiene distintos constructores `Constructor1,...,ConstructorM` cada uno con sus correspondientes parámetros.

Para entender mejor esto, veamos un ejemplo:

```
data Maybe a = Nothing | Just a
```

El tipo de dato `Maybe a` tiene dos constructores: `Nothing` (sin parámetro) y `Just a`, de parámetro `a`. Recordemos que, en programación funcional, los constructores tienen toda la información sobre el tipo de dato, por lo que podemos identificar todo objeto del tipo `Maybe a` con uno de sus constructores.

Esto nos lleva a la interpretación que un objeto de tipo `Maybe a` es `Nothing` o bien `Just a`. Pero cuidado, no confunda el constructor con el tipo. Tanto `Nothing` como `Just 24` son del tipo `Maybe Int`. Para definir una función que tome `Maybe a` por defecto, suele venir bien definirla por casos:

```
maybe_a_numero :: Maybe Int -> Int
maybe_a_numero Nothing  = 0
maybe_a_numero (Just x) = x
```

Otro tipo que nos resultará de utilidad es el tipo de listas `[a]`. Esto tipo se salta algunas reglas de como está descrito para ser más sencillo de escribir. Podemos definirlo en pseudo-haskell como:

```
data [a] = a : [a] | []
```

Es decir, una lista de elementos del tipo `a` es o bien una lista vacía `[]`, o bien un elemento de tipo `a` y otra lista del mismo tipo.

Podemos definir *clases* sobre los tipos que implementen alguna función particular, por ejemplo:

```
class Eq a where
  (==) :: a -> a -> Bool
```

define la clase de los tipos sobre los que está definido el operador igualdad `(==)`. Si queremos que un tipo esté en la clase `Eq`, tendremos que escribir su instancia. Como ejemplo, supongamos que queremos escribir la instancia de `Maybe a` en la clase `Eq`. Si los dos elementos del tipo `Maybe a` son de la forma `Nothing`, diremos que son iguales. Si no, comparamos el argumento de `Just a`. Pero para ello, el tipo `a` debe ser también de la clase `Eq`. Por ello, una implementación sería:

```
instance Eq a => Eq (Maybe a) where
  Nothing == Nothing = True
  Nothing == Just x  = False
  Just x  == Nothing = False
  Just x  == Just y  = x == y
```

1.3 Referencias

- Thompson, S. (1991). *Type theory & functional programming*, Capítulo 2.
- Pierce, B.C. (1991). *A taste of category theory for computer scientist*, Capítulo 1.
- Lipovaca, M. (2011). *Learn You a Haskell for Great Good!: A Beginner's Guide*.

2 | Fundamentos

2.1 Categorías

La idea empieza por el concepto de categoría, que resulta sorprendentemente ubicua en la matemática. En principio, una categoría es sólo una colección de objetos y flechas entre dichos objetos donde se define una composición de flechas. Veremos que algunas de la categorías más interesantes son las relacionadas con las estructuras matemáticas, como Top, la categoría de espacios topológicos y funciones continuas, o Grp, la categoría de grupos y homomorfismos. En estos casos, la teoría de categorías sirve como un marco de las teorías matemáticas. Las aplicaciones entre categorías –functores– ilustran las conexiones entre distintas teorías matemáticas. Aunque los axiomas que pediremos a las categorías no son muchos, resulta que podemos crear una gran cantidad de resultados interesantes y muy genéricos.

Definición 2.1. *Una* categoría $\mathcal{C}$ *es:*

- *Una colección de* objetos $O(\mathcal{C})$.
- *A cada par de objetos A, B en $O(\mathcal{C})$, una colección de* morfismos $C(A, B)$ *de A a B. Un morfismo f en $C(A, B)$ se denotará $f\colon A \to B$.*
- *A cada par de morfismos $f\colon A \to B$, $g\colon B \to C$, un morfismo de $g \circ f\colon A \to C$ llamado* morfismo composición *de f y g.*

de manera que:

- *Para cada objeto $A \in O(\mathcal{C})$, existe un morfismo* $\mathrm{id}_A \in C(A, A)$ *que llamamos* morfismo identidad.

- *Para todo morfismo* $f\colon A \to B$*:*

$$\mathrm{id}_B \circ f = f \circ \mathrm{id}_A = f$$

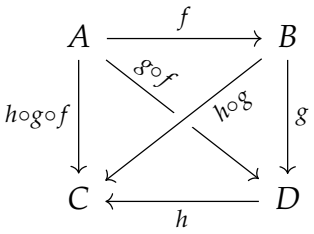
- *Asociatividad de la composición: Para toda tripleta de morfismos* $f\colon A \to B$, $g\colon B \to C$, $h\colon C \to D$*:*

$$(h \circ g) \circ f = h \circ (g \circ f) \tag{2.1}$$

Una notación habitual es usar $Hom(A, B)$ para referirse a la colección de morfismos entre A y B, lo que hemos llamado $C(A, B)$.

A menudo usaremos diagramas para expresar visualmente ciertas propiedades. En Teoría de Categoría, existe una definición precisa de diagrama, pero por ahora, no tenemos las herramientas suficientes para explicarlo. Basta imaginar que es un grafo formado por una cantidad finita de objetos y morfismos de una categoría.

El concepto de conmutatividad de un diagrama es el habitual de otros campos. Como ejemplo, la propiedad (2.1) se puede expresar diciendo que el diagrama



conmuta.

Dado un morfismo $f\colon A \to B$, llamaremos dominio de f a $\mathrm{dom}\, f = A$. De igual manera, el codominio de f será $\mathrm{cod}\, f = B$.

Además, llamaremos a la colección de morfismos de una categoría $M(\mathcal{C})$.

No hay ni mucho menos falta de ejemplos de categorías, ni siquiera si nos limitamos a un campo específicio de la matemática. Algunos ejemplos ilustrativos son los siguientes:

Ejemplo 2.1. La categoría *Set* formada por los conjuntos como objetos y funciones entre conjuntos como morfismos. La composición de morfismos se corresponde, como es de esperar, con la composición de funciones.

Ejemplo 2.2. La categoría Grp formada por los grupos como objetos y homomorfismos entre grupos como morfismos.

Ejemplo 2.3. La categoría Top formada por los espacios topológicos como objetos y funciones continuas como morfismos.

Ejemplo 2.4. Todo *conjunto parcialmente ordenado* (o *poset*) $(X, \leq)$ puede verse como una categoría, donde los objetos son los elementos de X y hay un único morfismo $f : a \to b$ si y sólo si $a \leq b$. Este es nuestro primer ejemplo donde los morfismos no son equivalentes a función. La composición de dos morfismos aquí es equivalente a la propiedad transitiva de la relación de orden $\leq$. Es más, el morfismo identidad es equivalente a la propiedad reflexiva.

También nos interesarán en algunos casos categorías que posean una estructura finita dada, como:

Ejemplo 2.5. La categoría $\mathbb{1}$ formada por un sólo objeto y su morfismo identidad:

$$\overset{\mathrm{id}_A}{\circlearrowright}\ A$$

Ejemplo 2.6. La categoría :

$$\overset{\mathrm{id}_A}{\circlearrowright}\ A \xrightarrow{\ f\ } B \ \overset{\mathrm{id}_B}{\circlearrowright}$$

Las categorías finitas serán de gran uso cuando hablemos más adelante de los diagramas. Por otro lado, las categorías con las que trabajaremos a menudo serán enormes, donde a veces la teoría de conjuntos habitual se nos quedará pequeña. Por ejemplo, en la teoría de conjuntos habitual, el conjunto de todos los conjuntos es una noción inconsistente como consecuencia del Teorema de Cantor. Por esta razón, hablaremos a menudo de «colección» o «familia» y evitaremos la palabra «conjunto» en general.

En ocasiones, sí nos será útil que la colección de objetos o morfismos forme un conjunto. Para ello, introducimos las siguientes definiciones:

Definición 2.2. *Una categoría $\mathcal{C}$ es* pequeña *si $O(\mathcal{C})$ y $M(\mathcal{C})$ forman conjuntos.*

Por ejemplo, toda categoría finita es pequeña, pero Set no es pequeña.

Definición 2.3. *Una categoría $\mathcal{C}$ es* localmente pequeña *si para todo par de objetos A y B en $\mathcal{C}$, $\mathcal{C}(A, B)$ forma un conjunto.*

Todos los ejemplos que hemos dado son localmente pequeños. Para poder dar ejemplos de categorías que no son localmente pequeñas, necesitaremos más conceptos. Casi siempre trabajaremos con categorías localmente pequeñas.

También podemos construir categorías a partir de otras categorías. El ejemplo más importante es el de la categoría opuesta:

Definición 2.4. *Dada una categoría $\mathcal{C}$, la* categoría opuesta *o* dual *$\mathcal{C}^{op}$ es la categoría con los mismos objetos que $\mathcal{C}$ y donde un morfismo $f\colon B \to A$ en $\mathcal{C}^{op}$ es un morfismo $f\colon A \to B$ en $\mathcal{C}$.*

Evidentemente se tiene que $(\mathcal{C}^{op})^{op} = \mathcal{C}$. A menudo nos encontraremos con conceptos que están relacionados por el dual de una categoría. Por ejemplo, el objeto terminal de una categoría (que veremos más adelante) no es más que el objeto inicial de su categoría opuesta.

Definición 2.5. *Sea A, B objetos de una categoría $\mathcal{C}$. Un morfismo $f\colon B \to C$ de una categoría es un monomorfismo (o mónico) si para todo morfismo $g\colon A \to B$ y $h\colon A \to B$ tales que $f \circ g = f \circ h$, entonces $g = h$.*

Proposición 2.1. En Set, un morfismo es mónico si y sólo si es inyectivo.

Demostración. Sea $f\colon B \to C$ un monomorfismo. Sean $b, b' \in B$ tales que $f(b) = f(b')$. Sea $A = \{b\}$ y definimos $g\colon A \to B$ como $g(b) = b'$. Entonces $f(\mathrm{id}_A(b)) = f(g(b))$ como b es el único elemento de A, tenemos que:

$$f \circ \mathrm{id}_A = f \circ g$$

luego, como f es monomorfismo:

$$\mathrm{id}_A = g$$

Entonces $b' = g(b) = \mathrm{id}_A(b) = b$.

Consideramos ahora $f\colon B \to C$ un morfismo inyectivo. Sean los morfismos $g, h\colon A \to B$ tales que $f \circ g = f \circ h$. Sea $a \in A$ cualquiera. Como $f(g(a)) = f(h(a))$ y f es inyectiva, entonces $g(a) = h(a)$. Es decir, se tiene que $g = h$. ∎

Definición 2.6. *Un morfismo $f\colon A \to B$ es epimorfismo (o épico) si para cualquier par de morfismos $g\colon B \to C$ y $h\colon B \to C$, se tiene que $g \circ f = h \circ f$ implica que $g = h$.*

Proposición 2.2. En Set, un morfismo es épico si y sólo si es sobreyectivo.

Demostración. Sea $f\colon A \to B$ un epimorfismo. Supongamos que f no fuera sobreyectivo. Entonces existe $b \in B$ que no es imagen de ningún elemento de A. Sean $g, h\colon B \to \{1,2\}$ tal que $g(x) = h(x) = 1$ para todo $x \in B \setminus \{b\}$, $g(b) = 1$ y $h(b) = 2$. Entonces tenemos que $g \circ f = h \circ f$. Como f es epimorfismo, esto implica que $g = h$, pero esto entra en una contradicción.

Sea $f\colon A \to B$ un morfismo sobreyectivo. Sean $g, h\colon B \to C$ tales que $g \circ f = h \circ f$. Para todo $b \in B$, existe $a \in A$ tal que $f(a) = b$, luego: $g(b) = g(f(a)) = h(f(a)) = h(b)$, luego $g = h$. ∎

Definición 2.7. *Un morfismo $f\colon A \to B$ es* isomorfismo *si existe $g\colon B \to A$ tal que $g \circ f = \mathrm{id}_A$ y $f \circ g = \mathrm{id}_B$.*

Dos objetos A y B se dirán isomorfos *si existe un isomorfismo entre ellos.*

Claramente, en *Set*, un morfismo mónico y épico es isomórfico. Sin embargo, todos los morfismos de un poset son épicos y mónicos, pero sólo los morfismos identidad son isomorfismos.

2.2 Functores

Parece que muchos conceptos como *Set* o *Top* forman categorías. De aquí surge la siguiente cuestión: ¿Podríamos hacer una categoría de categorías? Es decir, una categoría donde los objetos mismos sean categorías. Para ello, debemos definir morfismos entre categorías que respetan su estructura interna. Aunque la misma razón por la que no hay conjunto de conjuntos, no hay una categoría de categorías, esta cuestión motiva la definición de functor:

Definición 2.8. *Un* functor *F entre un par de categorías $\mathcal{C}$ y $\mathcal{D}$ es un par de funciones:*

- $F_O : O(\mathcal{C}) \to O(\mathcal{D})$.
- $F_M : M(\mathcal{C}) \to M(\mathcal{D})$.

tal que:

- *Respeta el dominio y codominio de los morfismos: A cada morfismo $f : A \to B$:*

$$F_M(f) : F_O(A) \to F_O(B)$$

- *Respeta el morfismo identidad:*

$$F_M(\mathrm{id}_A) = \mathrm{id}_{F_O(A)}$$

- *Respeta la composición de morfismo: Para todo morfismos $f : A \to B$, $g : B \to C$.*

$$F_M(g \circ f) = F_M(g) \circ F_M(f)$$

Escribiremos $F \colon \mathcal{C} \to \mathcal{D}$ y a menudo usaremos F para referirnos a F_M ó F_O según el contexto.

Viendo los functores como funciones entre categorías, es natural que queramos definir composición entre functores.

Definición 2.9. *Dado dos functores $F \colon \mathcal{C} \to \mathcal{D}$ y $G \colon \mathcal{D} \to \mathcal{E}$, definimos $G \circ F$ como el functor con:*

- $(G \circ F)_O = G_O \circ F_O$.
- $(G \circ F)_M = G_M \circ F_M$.

2.2.1 Functores en Haskell

Recordemos que en el contexto de la programación con tipos, Haskell sólo trabaja con tipos de la categoría `Hask`. Los functores con los que podemos trabajar son de endofunctores de `Hask`, es decir, functores que van de `Hask` a `Hask` – de tipos a tipos.

Haskell implementa los functores a través de sus `typeclasses`.

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

La `f` actúa como F y como F_O en 2.8. Si hay dudas de qué es lo que hace `fmap` aquí, resulta útil añadir unos paréntesis en el lugar adecuado:

```
fmap :: (a -> b) -> (f a -> f b)
```

`fmap` toma un función cualquiera `a -> b` y la transforma en una función del tipo `f a -> f b`. Esto es precisamente lo que antes llamábamos F_M. Con esto en mente, adaptemos las leyes de los functores en Haskell:

```
fmap id = id
fmap (g . f) = fmap g . fmap f
```

Recordemos que en Haskell, la composición de funciones $g \circ f$ se expresa usando `g . f`. Antes de pasar a unos ejemplos, hay que tener en cuenta que Haskell no dispone de ningún mecanismo que comprueba que estas igualdades se cumplen. Queda como responsabilidad del programador que la clase `Functor` no se aplique sobre tipos que no cumplen las leyes de functores.

Ejemplo 2.7. Nuestro primer ejemplo es `Maybe`.

```
data Maybe a = Nothing | Just a
```

Podemos leer esta definición como `Maybe a` contiene o bien un valor constante o bien un valor del tipo `a`. Hay que tener cuidado con la diferencia entre `Maybe` y `Just`. `Maybe` es lo que llamamos un constructor de tipo. Veremos que este constructor de tipo, que asocia a cada tipo `a` el tipo `Maybe a`, es un functor. Por otro lado, `Just` es una función polimórfica –y por lo tanto una familia de morfismos en `Hask`– de `a` a `Maybe a`.

Para ver que `Maybe` es un functor, tenemos que describir como actúa sobre morfismos, ahí es donde entra en juego la función `fmap`.

```
instance Functor Maybe where
  fmap f Nothing = Nothing
  fmap f (Just x) = Just (f x)
```

2.2.2 Functores especiales

Consideramos ahora el concepto de functor contravariante a través de un cambio en la definición de functor:

Definición 2.10. *Un* functor contravariante *F entre un par de categorias $\mathcal{C}$ y $\mathcal{D}$ es un par de funciones:*

- $F_O : O(\mathcal{C}) \to O(\mathcal{D})$.
- $F_M : M(\mathcal{C}) \to M(\mathcal{D})$.

tal que:

- ***Intercambia** el dominio y codominio de los morfismos: A cada morfismo $f : A \to B$:*
$$F_M(f) : F_O(B) \to F_O(A)$$
- *Respeta el morfismo identidad:*
$$F_M(\mathrm{id}_A) = \mathrm{id}_{F_O(A)}$$
- ***Intercambia** la composición de morfismo: Para todo morfismos $f : A \to B$, $g : B \to C$.*
$$F_M(g \circ f) = F_M(f) \circ F_M(g)$$

Los functores con los que hemos trabajado hasta ahora serán llamados *functores covariantes*, pero a menudo prescindiremos del adjetivo. Una razón para esto es que no hay diferencia esencial entre functor contravariante y functor covariante cuando observamos que los «intercambios» en la definición son equivalentes a que el functor parta de $\mathcal{C}^{op}$. Por esta razón, hablaremos de functor contravariante cuando tratemos con un functor covariante de la forma:

$$F\colon \mathcal{C}^{op} \to D$$

Otros functores reciben nombres especiales por su dominio o su codominio:

Definición 2.11. *Un functor $F\colon \mathcal{C} \to \mathcal{C}$ se denomina* endofunctor.

Ejemplo 2.8. Un ejemplo sencillo es el *endofunctor potencia* $\mathcal{P}\colon Set \to Set$ que envía un conjunto A a su conjunto potencia $\mathcal{P}(A)$ y envía a las funciones entre conjuntos $f\colon A \to B$ a la función $\mathcal{P}f\colon \mathcal{P}(A) \to \mathcal{P}(B)$ definida como:

$$\mathcal{P}f(S) = \{f(s)\colon s \in S\}, \quad S \subseteq A$$

Definición 2.12. *Un functor* $F\colon \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{D}$ *se denomina* bifunctor.

Aquí $\mathcal{C}^{op} \times \mathcal{C}$ es la categoría producto de $\mathcal{C}^{op}$ y $\mathcal{C}$, definida más adelante en la sección 1.5.

Definición 2.13. *Un functor* $F\colon \mathcal{C}^{op} \to Set$ *se denomina* prehaz.

2.3 Construcciones universales

Podemos caracterizar ciertos objetos en una categoría por alguna propiedad especial que cumplan. En lugar de pedir que sólo haya un objeto que cumpla dicha propiedad, nos limitamos a pedir que todos los objetos que cumplan la propiedad sean isomorfos. Estas propiedades son llamadas *propiedades universales*.

Definición 2.14. *Decimos que un objeto* $A \in \mathcal{C}$ *es* inicial *si para cada objeto* $B \in \mathcal{C}$*, hay exactamente un morfismo* $f\colon A \to B$.

Análogamente, decimos que un objeto $B \in \mathcal{C}$ *es* terminal *si para cada objeto* $A \in \mathcal{C}$*, hay exactamente un morfismo* $f\colon A \to B$.

Cuando hablemos de un objeto inicial, usaremos a menudo la notación 0. Análogamente, usaremos 1 para referirnos a un objeto terminal.

Ejemplo 2.9. Veamos algunos ejemplos en categorías usuales:

- En *Set*, el conjunto vacío es un objeto inicial y cualquier conjunto unitario es terminal.
- En *Grp*, el grupo trivial es inicial y terminal.
- En *Ring*, el anillo unitario es terminal y el anillo $(\mathbb{Z}, +, \cdot)$ es inicial.
- En un poset, visto como categoría, el objeto inicial es el elemento mínimo global y el objeto terminal es el elemento máximo global, si existen.
- En *Hask*, el objeto terminal se define como el tipo de dato con un sólo valor posible. Se suele usar `()`, que puede entenderse como una 0-tupla. El objeto inicial se define como el tipo de dato sin constructor:

  ```
  data Empty
  ```

De estos ejemplos vemos que:

- Los objetos iniciales y terminales no son necesariamente únicos.
- Un objeto puede ser inicial y terminal. En dicho caso, el objeto en cuestión recibe el nombre de *objeto cero*.
- Una categoría puede no poseer objetos iniciales o terminales.

Aunque no haya unicidad de objetos iniciales o terminales, sí hay una relación entre todos los objetos iniciales o terminales.

Proposición 2.3. Sea $\mathcal{C}$ una categoría. Sean A y B objetos iniciales (o terminales) de $\mathcal{C}$. Entonces existe un único isomorfismo $A \to B$.

Demostración. Como A es inicial, existe un único morfismo $f\colon A \to B$. Como B es inicial, existe un único morfismo $g\colon B \to A$. Entonces $g \circ f\colon A \to A$. Al ser A inicial, sólo hay un morfismo $A \to A$, que debe ser el morfismo identidad, luego $g \circ f = \mathrm{id}_A$. Análogamente, $f \circ g = \mathrm{id}_B$.

Luego f es isomorfismo. ∎

2.4 Diagramas y conos

En estas páginas hemos trabajado a menudo con categorías con infinitos objetos e innumerables morfismos. Sin embargo, resulta de interés estudiar una estructura particular de objetos y morfismos que podamos encontrar dentro de esa categoría. Para ello, formalizaremos un concepto que hemos manejado antes, los diagramas.

Definición 2.15. *Sean $\mathcal{C}$ y $\mathcal{I}$ dos categorías. Un* diagrama *de $\mathcal{C}$ con* forma *$\mathcal{I}$ es un functor $D\colon \mathcal{I} \to \mathcal{C}$.*

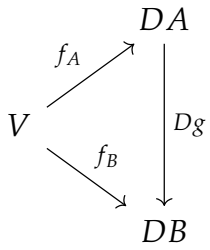
Es decir, técnicamente diagrama será sólo otra palabra para functor. En la práctica, usaremos diagrama cuando querramos distinguir una estructura particular dentro de una categoría. También se llamará a $\mathcal{I}$ *categoría índice* del diagrama.

Ejemplo 2.10. Sea $\mathcal{I}$ la siguiente categoría índice:

$$A \longleftarrow B \longrightarrow C$$

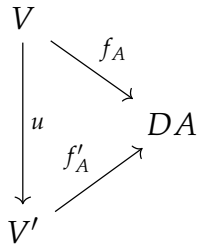
Entonces el diagrama $D\colon \mathcal{I} \to \mathcal{C}$ es llamado *span*.

Definición 2.16. *Sea $\mathcal{C}$ una categoría y $D\colon \mathcal{I} \to \mathcal{C}$ un diagrama de forma $\mathcal{I}$. Un* cono *en el diagrama D es un objeto $V \in \mathcal{C}$ (que llamamos* vértice*) y una familia de morfismos $f_A\colon C \to DA$ para todo $A \in \mathcal{I}$, tal que para todo morfismo $g\colon A \to B$ de $\mathcal{I}$, el siguiente diagrama conmuta:*



Una buena técnica para estudiar una nueva construcción es preguntarse: ¿Podemos formar una categoría con ella?

Lo que nos queda por hacer para crear una categoría de conos de un diagrama fijo D es definir un morfismo entre conos. Resulta intuitivo definirlo a partir de un morfismo entre los vértices. Si C y C' son dos conos con vértices V y V' respectivamente, definimos el morfismo $C \to C'$ como un morfismo $u\colon V \to V'$ tal que el siguiente diagrama conmuta para todo $A \in \mathcal{I}$:



Proposición 2.4. Dado una categoría $\mathcal{C}$ y un diagram $D\colon \mathcal{I} \to \mathcal{C}$, los conos de D forma una categoría llamada $\mathrm{Cono}(D)$ con los morfismos descritos anteriormente.

La demostración es directa gracias a que $\mathcal{C}$ es una categoría.

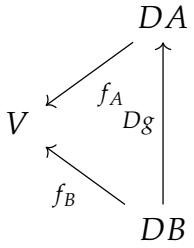
En esta categoría de conos, resulta de especial interés el objeto terminal.

Definición 2.17. *Dado una categoría $\mathcal{C}$ y un diagrama $D\colon \mathcal{I} \to \mathcal{C}$, el* límite del diagrama *D es el objeto terminal de $\mathrm{Cono}(D)$.*

Usaremos la notación $\varprojlim D$ para referirnos al límite de un diagrama si existe.

Podemos considerar el dual de estos conceptos.

Definición 2.18. *Un* cocono *de $\mathcal{C}$ es un cono de $\mathcal{C}^{op}$. En otras palabras, un cocono en un diagrama $D\colon \mathcal{I} \to \mathcal{C}$ es un objeto $V \in \mathcal{C}$ y una familia de morfismos $f_A\colon DA \to V$ para todo $A \in \mathcal{I}$, tal que para todo morfismo $g\colon B \to A$ de $\mathcal{I}$, el siguiente diagrama conmuta:*



Definición 2.19. *Un* colímite *es el objeto terminal de la categoría de coconos de un diagrama. Equivalentemente es el objeto inicial de la categoría de conos.*

Definición 2.20. *Una categoría $\mathcal{C}$ es* (co)completa *si existen todos los (co)límites de diagramas $D\colon \mathcal{I} \to \mathcal{C}$ con $\mathcal{I}$ categoría pequeña.*

Una categoría es bicompleta *si es completa y cocompleta.*

2.5 Productos y coproductos

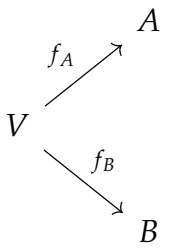
Resulta importante familiarizarse con la idea de límite antes de continuar, porque nos resultará muy útil para definir nuevos conceptos.

Definición 2.21. *Sea $\mathcal{I}$ la categoría de dos objetos I_1 y I_2 y sin morfismos aparte de los morfismos identidad. El* producto *de dos objetos A y B en $\mathcal{C}$ es el límite del diagrama $D\colon \mathcal{I} \to \mathcal{C}$ con $D(I_1) = A$ y $D(I_2) = B$.*

Escribiremos $A \times B$ para denotar el producto de A y B.

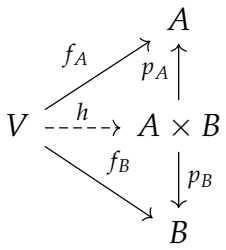
Desenvolvamos las definiciones para entender qué significa realmente el producto de dos objetos A y B. Primero obsérvese que el diagrama D actúa para «seleccionar» los objetos A y B y sus morfismos identidad. Puede que haya algún morfismo entre A y B, pero no son relevantes para el diagrama y, en consecuencia, ni para el límite.

Un cono sobre D es de la forma:



(2.2)

El producto $A \times B$, límite de D, puede ser visto como el cono terminal en la categorías de conos sobre D. Es decir, para todo cono como en 2.2, debe existir un único morfismo de dicho cono al $A \times B$. Desenvolvamos las definiciones aún más. Digamos que el producto está formado por los objetos $A \times B$ (abusando un poco de notación aquí), $p_A \colon A \times B \to A$ y $p_B \colon A \times B \to B$. Para todo V, $f_A \colon V \to A$ y $f_B \colon V \to B$, debe existir un $h \colon V \to A \times B$ tal que el siguiente diagrama conmute:



o algebraicamente:

$$f_A = p_A \circ h$$

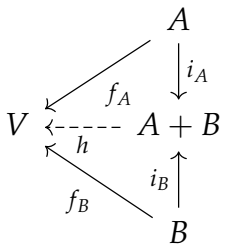
$$f_B = p_B \circ h$$

Es decir, dar un par de morfismos $V \to A$ y $V \to B$ es equivalente a dar un morfismo $V \to A \times B$. Luego, podemos volver a los objetos A y B usando unos morfismos fijos p_A y p_B (que llamamos *proyecciones*).

Definición 2.22. *Sea $\mathcal{I}$ la categoría de dos objetos I_1 y I_2 y sin morfismos aparte de los morfismos identidad. El* coproducto *de dos objetos A y B en $\mathcal{C}$ es el colímite del diagrama $D \colon \mathcal{I} \to \mathcal{C}$ con $D(I_1) = A$ y $D(I_2) = B$.*

Escribiremos $A + B$ para denotar el coproducto de A y B.

Desarrollando como antes la definiciones tenemos que para todo V, $f_A\colon A \to V$, $f_B\colon B \to V$, se tiene que existe $h\colon A + B \to V$ tal que:



donde i_A e i_B son llamadas *inyecciones naturales*.

Ejemplo 2.11. No es de extrañar que en *Set*, el producto de dos objetos es el producto cartesiano de dos conjuntos. Por otro lado, el coproducto de dos conjuntos se corresponde con la unión disjunta.

Ejemplo 2.12. Consideramos la categoría *Pos* de conjuntos parcialmente ordenados (posets) donde los morfismos son funciones monótonas. El producto de dos posets P y Q es un poset $P \times Q$ de pares (p, q) con $p \in P$ y $q \in Q$ ordenado parcialmente por:

$$(p, q) \leq (p', q') \Leftrightarrow p \leq p' \text{ y } q \leq q'$$

Las proyecciones $p_1\colon P \times Q \to P$ y $p_2\colon P \times Q \to Q$ son evidentemente monótonas, luego son morfismos de *Pos*.

Ejemplo 2.13. En *Hask*, el producto de dos tipos `a` y `b` es el par `(a,b)` y las proyecciones son las funciones `fst :: (a,b) -> a` y `snd :: (a,b) -> b`. También tenemos que `(a,b)` es isomorfo a:

```
newtype Pair a b = Pair a b
```

A su vez, el coproducto de dos tipos `a` y `b` es el tipo `Either a b`, cuya definición es:

```
data Either a b = Left a | Right b
```

Es más, tenemos que todos los tipos de datos compuestos son isomorfos a productos y coproductos.

Ejemplo 2.14. En *categoría de categorías pequeñas* Cat, el producto de dos categorías $\mathcal{C}$ y $\mathcal{D}$ es una categoría $\mathcal{C} \times \mathcal{D}$ cuyos objetos son pares ordenados (A, B) con $A \in \mathcal{C}$ y $B \in \mathcal{D}$. Los morfismos, composición e identidad se definen a manera análoga por componentes.

Podemos definir producto n-ario de objetos como el límite de un diagrama con una categoría índice discreta (es decir, sin morfismos a parte de los morfismos identidad) de n objetos.

Observación 2.1.

- Si existen productos binarios, entonces existe cualquier produco n-ario con $n \geq 2$.
- En el caso particular de $n = 1$, el 1-producto de cualquier objeto es precisamente el mismo objeto.
- En el caso particular de $n = 0$, el 0-producto es precisamente el objeto terminal.

Por esta razón, cuando una categoría tiene productos binarios y objeto terminal, se dice que tiene *todos los productos finitos*.

Hemos visto el producto y el coproducto como casos particulares de límites. Otros límites importantes son:

Definición 2.23. *Dado un diagrama* span

$$A \longleftarrow B \longrightarrow C$$

sobre una categoría, su colímite se denomina pushout.

Un uso típico de pushouts está presente en topología:

Ejemplo 2.15. Dado un span

$$X \xleftarrow{f} A \xrightarrow{i} Y$$

en Top, donde $i\colon A \to Y$ es una inclusión, entonces el pushout se llama *espacio de adjunción* y se escribe $X \cup_f Y$. El espacio de adjunción se corresponde con el espacio cociente de la unión disjunta de X e Y bajo la relación de equivalencia generada por $x \sim f(x)$.

Definición 2.24. *Dado un diagrama* cospan *(dual de span)*

$$A \longrightarrow B \longleftarrow C$$

sobre una categoría, su límite se denomina pullback.

Definición 2.25. *Dado un diagrama de* pareja paralela

$$A \rightrightarrows B$$

sobre una categoría, su límite se denomina ecualizador *y su colímite,* coequalizador.

En resumen:

Forma	Límite	Colímite
$\emptyset$	Objeto terminal	Objeto inicial
Discreta finita	Producto	Coproducto
Span	-	Pushout
Cospan	Pullback	-
Pareja paralela	Ecualizador	Coecualizador

2.6 Functores continuos

Claramente un functor envía conos a conos, pero no es cierto que envíe límites a límites en general.

Definición 2.26. *Un functor $F\colon \mathcal{C} \to \mathcal{D}$ se dice que* preserva límites de forma $\mathcal{I}$ *si para todo diagrama $D\colon \mathcal{I} \to \mathcal{C}$ con cono límite $L = \varprojlim D$, el cono FL es el cono límite del diagrama $F \circ D$. Se suele expresar como:*

$$F\left(\varprojlim D\right) = \varprojlim (F \circ D)$$

Análogamente, F se dice que preserva colímites de forma D *si:*

$$F\left(\varinjlim D\right) = \varinjlim (F \circ D)$$

La uso de límites aquí nos recuerda a la definición de funciones continuas de la matemática clásica. Esto lleva a la definición de functor continuo y cocontinuo:

Definición 2.27. *Un functor es* functor (co)continuo *si preserva (co)límites de cualquier forma pequeña.*

En algunos contextos, resulta más útil debilitar la definición de functor continuo a diagramas de forma $\omega = (\mathbb{N}, \leq)$, esto nos lleva a la definición de functor ω-continuo. Un diagrama de forma ω se refiere a una cada de morfismos como:

$$\cdots \to A_3 \to A_2 \to A_1 \Rightarrow A_0$$

En estos casos, se suele identificar el diagrama con la sucesión $\{A_n\}_{n\in\mathbb{N}}$, de manera que el límite se escribe:

$$\varprojlim A_i$$

Definición 2.28. *Un functor es* ω-(co)continuo *si preserva (co)límites de forma* ω.

Ejemplo 2.16. Dado un primo p, los enteros p-ádicos se definen como el límite del diagrama:

$$\cdots \to \mathbb{Z}/p^3\mathbb{Z} \to \mathbb{Z}/p^2\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$$

en la categoría de anillos.

2.7 Transformaciones naturales

Definición 2.29. *Una transformación natural* μ *entre dos functores* $F, G\colon \mathcal{C} \to \mathcal{D}$ *es una colección de morfismos*

$$\mu_A\colon FA \to GA$$

donde $A \in \mathcal{C}$ *tal que para todo* $f\colon A \to B$ *el diagrama:*

$$\begin{array}{ccc} FA & \xrightarrow{\mu_A} & GA \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ FB & \xrightarrow{\mu_B} & GB \end{array}$$

conmuta.

Usaremos la notación $\mu\colon F \Rightarrow G$ para una transformación natural μ. Llamamos μ_A *componente* de μ en A. La conmutatividad del diagrama se denomina *propiedad de naturalidad* y es equivalente a:

$$Gf \circ \mu_A = \mu_B \circ Ff$$

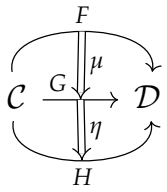
para todo morfismo $f\colon A \to B$ en $\mathcal{C}$.

Hay dos formas de componer transformaciones naturales.

Definición 2.30. *Dado dos transformaciones naturales $\mu\colon F \Rightarrow G$ y $\eta\colon G \Rightarrow H$ donde $F, G, H\colon \mathcal{C} \to \mathcal{D}$ son functores, su composición horizontal $\eta \circ \mu\colon F \Rightarrow H$ viene dada por*

$$(\eta \circ \mu)_A = \eta_A \circ \mu_A$$

para todo objeto $A \in \mathcal{C}$.

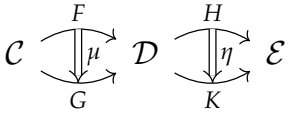


Lema 2.1. Dados dos categorías $\mathcal{C}$ y $\mathcal{D}$, existe una categoría donde los objetos son functores $\mathcal{C} \to \mathcal{D}$ y los morfismos son transformaciones naturales.

Denotamos dicha categoría como $\mathcal{D}^{\mathcal{C}}$ ó $[\mathcal{C}, \mathcal{D}]$.

Demostración. Vemos primero que podemos definir una transformación natural identidad $\iota\colon F \Rightarrow F$ dado por $\iota_A = \mathrm{id}_{FA}$ para todo $A \in \mathcal{C}$. Por otro lado, que la composición de transformaciones naturales es asociativa es trivial. ∎

Nos interesa ahora definir una composición de transformaciones naturales de la forma:



(2.3)

Nuestro objeto será definir una transformación natural que vaya de $H \circ F$ a $G \circ K$. Para ello necesitaremos describir dos formas en las que se pueden componer un functor y una transformación natural:

Definición 2.31. *Si $F\colon \mathcal{C} \to \mathcal{D}$ y $G, H\colon \mathcal{D} \to \mathcal{E}$ son functores y $\mu\colon G \Rightarrow H$ es una transformación natural, el* whiskering izquierdo*:*

$$\mu \circ F\colon G \circ F \Rightarrow H \circ F$$

está definido como:

$$(\mu \circ F)_A = \mu_{FA}$$

La naturalidad de $(\mu \circ F)$ es consecuencia de la naturalidad de μ.

Se suele eliminar $\circ$ y escribir sencillamente μF.

Por otro lado:

Definición 2.32. *Si $F, G\colon \mathcal{C} \to \mathcal{D}$ y $H\colon \mathcal{D} \to \mathcal{E}$ son functores y $\mu\colon F \Rightarrow G$ es una transformación natural, el* whiskering derecho*:*

$$H \circ \mu\colon H \circ F \Rightarrow H \circ G$$

está definido como:

$$(H \circ \mu)_A = H(\mu_A) = H\mu_A$$

Aquí la naturalidad no es tan evidente.

Proposición 2.5. El *whiskering* derecho es una transformación natural.

Demostración. Por la naturalidad de μ, se da la igualdad

$$Gf \circ \mu_A = \mu_B \circ Ff$$

aplicando H y su propiedad de functorialidad:

$$H(Gf) \circ H\mu_A = H\mu_B \circ H(Ff)$$

Lo que prueba la naturalidad de $H \circ \mu$. ∎

Se suele escribir $H\mu$ en lugar de $H \circ \mu$.

Con esto, ya podemos definir la composición horizontal o composición de Godement.

Definición 2.33. *Sean $\mathcal{C}$, $\mathcal{D}$ y $\mathcal{E}$ tres categorías. Consideramos los functores F, G, H y K, y las transformaciones naturales μ y η en la configuración dada en 2.3. Definimos la composición horizontal $\eta * \mu\colon H \circ F \Rightarrow K \circ G$ como:*

$$(\eta * \mu)_A = \eta_{GA} \circ H\mu_A$$

Proposición 2.6. La composición de horizontal $\eta * \mu$ es una transformación natural.

Demostración. Por la naturalidad de μ y η se tiene que cada cuadrado del siguiente diagrama es conmutativo.

$$\begin{array}{ccccc} H(FA) & \xrightarrow{H\mu_A} & H(GA) & \xrightarrow{\eta_{GA}} & K(GA) \\ \downarrow{\scriptstyle H(Ff)} & & \downarrow{\scriptstyle H(Gf)} & & \downarrow{\scriptstyle K(Gf)} \\ H(FB) & \xrightarrow{H\mu_A} & H(GB) & \xrightarrow{\eta_{GB}} & K(GB) \end{array}$$

Luego, componiendo horizontalmente, tenemos que es conmutativo el diagrama:

$$\begin{array}{ccc} H(FA) & \xrightarrow{\eta_{GA} \circ H\mu_A} & K(GA) \\ \downarrow{\scriptstyle H(Ff)} & & \downarrow{\scriptstyle K(Gf)} \\ H(FB) & \xrightarrow{\eta_{GB} \circ H\mu_A} & K(GB) \end{array}$$

Esto demuestra la naturalidad de $\eta * \mu$. ∎

Véase que la composición horizontal se puede ver como una composición vertical tras aplicar un *whiskering* apropiado a cada transformación natural.

Proposición 2.7. Para una transformación natural $\mu \colon F \Rightarrow G$ y una transformación natural identidad $\iota \colon H \Rightarrow H$:

$$\iota * \mu = H\mu$$
$$\mu * \iota = \mu H$$

Demostración.

$$\begin{aligned} (\iota * \mu)_A &= \iota_{GA} \circ F\mu_A \\ &= \mathrm{id}_{H(GA)} \circ H\mu_A \\ &= H\mu_A \end{aligned}$$

Por otro lado:

$$\begin{aligned} (\mu * \iota)_A &= \mu_{HA} \circ G\iota_A \\ &= \mu_{HA} \circ G\,\mathrm{id}_{HA} \\ &= \mu_{HA} \circ \mathrm{id}_{G(HA)} \\ &= \mu_{HA} \\ &= (\mu H)_A \end{aligned}$$

2.8 Equivalencias y functores adjuntos

Obsérvese que tenemos una noción de "equivalencia" cuando hablamos de objetos en una categoría que no es estrictamente igualdad, sino isomorfía. Que dos objetos sean isomorfos significa que existan morfismos entre ellos inversos el uno del otro. Al nivel de la categoría de categorías pequeñas Cat, podríamos hablar de categorías $\mathcal{C}$ y $\mathcal{D}$ isomorfas si tienen functores $F\colon \mathcal{C} \to \mathcal{D}$ y $G\colon \mathcal{D} \to \mathcal{C}$ tal que $G \circ F = \mathrm{Id}_{\mathcal{C}}$ y $F \circ G = \mathrm{Id}_{\mathcal{D}}$. Sin embargo, esta idea resulta muy estricta para ser aplicada, en lugar de ello hacemos uso de equivalencias. Para relajar esta definición, usaremos transformaciones naturales.

Definición 2.34. *Una* equivalencia *entre dos categorías $\mathcal{C}$ y $\mathcal{D}$ es un par de functores $F\colon \mathcal{C} \to \mathcal{D}$ y $G\colon \mathcal{D} \to \mathcal{C}$ de manera que existen dos isomorfismos naturales $G \circ F \cong \mathrm{Id}_{\mathcal{C}}$ y $F \cong G \cong \mathrm{Id}_{\mathcal{D}}$.*

Aunque las equivalencias resultan útiles en la teoría de categorías de orden superior, no sirve para un tipo de functor muy habitual en diversos campos de la matemáticas.

Aquí entra la idea de adjunción. Antes de entrar formalmente en ellos, veamos un ejemplo que motiva la noción que formalizaremos.

Sea $U\colon Grp \to Set$ el functor que asocia a cada grupo $(G, *)$ el conjunto de sus elementos G. Esta clase de functor, que sólo "olvida" la estructura sobre un objeto, se denomina *functor olvidadizo*. Sea $F\colon Set \to Grp$ el functor que asocia a cada conjunto su correspondiente *grupo libre*. En concreto, para un conjunto S, FS puede verse como el grupo formado por la concatenación de palabras cuyas letras son elementos de S y sus correspondiente letras inversas.

Estos functores no son inversos el uno del otro, ni siquiera forman una equivalencia, pero representan una relajación de inversión. Esta idea se formaliza en la adjunción:

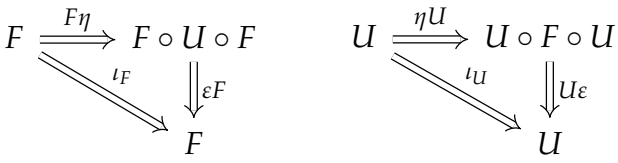
Definición 2.35. *Una* adjunción *entre categorías $\mathcal{C}$ y $\mathcal{D}$ son functores*

$$F\colon \mathcal{C} \to \mathcal{D} \qquad U\colon \mathcal{D} \to \mathcal{C}$$

y dos transformaciones naturales:

$$\eta\colon \mathrm{Id}_{\mathcal{C}} \Rightarrow U \circ F \qquad \varepsilon\colon F \circ U \Rightarrow \mathrm{Id}_{\mathcal{D}}$$

que satisfagan las identidades:



donde $\iota_F\colon F \Rightarrow F$ *es la transformación natural identidad.*

Dicha adjunción se suele expresar como:

$$\mathcal{C} \underset{F}{\overset{U}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} \mathcal{D}$$

o sencillamente $F \dashv U$. Se suele decir que U es la adjunta derecha de F y F es la adjunta izquierda de U. Las transformaciones naturales η y ε se llaman *unidad* y *counidad* respectivamente.

Volviendo a nuestro ejemplo:

$$Set \underset{F}{\overset{U}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} Grp$$

para formar una adjunción basta tomar las transformaciones naturales $\eta_S\colon S \to U(FS)$ que envía cada elemento $s \in S$ a s como elemento de FS y $\varepsilon_G\colon F(UG) \to G$ que envía cada palabra cuyas letras son elementos G a su producto. Con esto es una simple tarea comprobar que $F \dashv U$.

Proposición 2.8. Sea $F\colon \mathcal{C} \to \mathbb{1}$ un functor a la categoría de un objeto $\mathbb{1}$ y A un objeto de $\mathcal{C}$. Dado $U\colon \mathbb{1} \to \mathcal{C}$ cuya imagen es A, entonces 1 es objeto terminal de $\mathcal{C}$ si y sólo si $F \dashv U$.

Demostración. Denominemos $*$ el único objeto de $\mathbb{1}$. Obsérvese primero que $F(U*) = *$ y que $U(FB) = A$ para todo $B \in \mathcal{C}$. Por lo tanto:

$$\eta_B\colon B \to A$$

$$\varepsilon_* = \mathrm{id}_*$$

Entonces, tenemos:

$$F\eta_B = \mathrm{id}_*$$
$$\varepsilon_{FB} = \mathrm{id}_1$$
$$U\varepsilon_* \colon \mathrm{id}_A$$
$$\eta_{U*} = \eta_A \colon A \to A$$

Luego se da siempre una de las dos igualdades triangulares:

$$F\eta \circ \varepsilon F = \iota$$

Supongamos que A es un objeto terminal, entonces η es natural y $\eta_A = \mathrm{id}_A$, luego se da la otra igualdad triangular y se demuestra que $F \dashv U$.

Supongamos que $F \dashv U$, entonces:

$$\mathrm{id}_A = U\varepsilon_1 \circ \eta_A = \mathrm{id}_A \circ \eta_A = \eta_A$$

Sea $f \colon B \to A$ morfismo cualquiera a A. Por la naturalidad de η, se tiene que el siguiente diagrama conmuta:

$$\begin{array}{ccc} B & \xrightarrow{\eta_B} & A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle U(Ff)} \\ A & \xrightarrow{\eta_A} & A \end{array}$$

Obsérvese que $U(Ff) = U(\mathrm{id}_1) = \mathrm{id}_A$, luego tenemos que: $\eta_B = f$, lo que implica la unicidad de los morfismos $B \to A$ y que A es objeto terminal.

3 | Lema de Yoneda

El lema de Yoneda es uno de los resultados más importante, pues el relacionado embebimiento de Yoneda es una herramienta poderosa para demostrar otros resultados.

En este capítulo, siempre estaremos trabajando con categorías localmente pequeñas.

3.1 Hom-Functores

Consideremos la categoría de functores de una categoría $\mathcal{C}$ a Set, que podemos denotar como $Set^{\mathcal{C}}$. En esta categoría, los objetos son functores y los morfismos son transformaciones naturales entre functores.

Como estamos bajo la hipótesis que $\mathcal{C}$ es localmente pequeña, entonces $C(A, B)$ es un conjunto para todo $A \in \mathcal{C}$ y $B \in \mathcal{C}$. Entonces, $C(A, B) \in Set$. De este hecho, podemos crear el functor contravariante:

$$C(A, -) : \mathcal{C} \to Set$$

que a cada objeto $X \in \mathcal{C}$ le asocia el conjunto $C(A, X)$. A cada morfismo $f : X \to Y$ en $\mathcal{C}$, $C(A, f)$ será un morfismo entre los conjuntos $C(A, X)$ y $C(A, Y)$. La forma natural de asociar estos conjuntos es por la composición $g \mapsto f \circ g$. A dicho functor le damos el nombre de *hom-functor*.

Veamos que el hom-functor es realmente un functor.

$$C(A, \mathrm{id}_B)(g) = g \circ \mathrm{id} = g \Rightarrow C(A, \mathrm{id}_B) = \mathrm{id}_{C(A,B)}$$

$$\begin{aligned} C(A, g \circ f)(h) &= (g \circ f) \circ h = g \circ (f \circ h) \\ &= g \circ C(A, f)(h) = C(A, g)(C(A, f)(h)) \\ &= C(A, g) \circ C(A, f)\ (h) \end{aligned}$$

Luego $C(A, -)$ respeta el morfismo unidad y la composición de morfismos, luego es efectivamente un functor de $\mathcal{C} \to Set$. Equivalentemente, tenemos que $C(A, -)$ es un objeto de $Set^{\mathcal{C}}$.

Digamos que tenemos un objeto isomorfo a $C(A, -)$ en $Set^{\mathcal{C}}$. Esto quiere decir que entre el objeto, visto como un functor, y $C(A, -)$ hay una transformación natural invertible, que también llamamos *isomorfismo natural*. De aquí procede la siguiente definición:

Definición 3.1. *Un functor $F\colon \mathcal{C} \to Set$ es* representable *si es isomorfo naturalmente a un hom-functor.*

Es decir, un functor representable puede identificarse con $C(A, -)$ para algún A.

3.1.1 Functores representables en Haskell

En la categoría *Hask*, el hom-functor recibe a menudo el nombre de `Reader`

```
type Reader a x = a -> x
```

Es decir, $C(A, -)$ es equivalente a `Reader a`, que es equivalente a `(->) a`. Para que un functor `F` sea representable necesitamos una transformación natural invertible a `Reader a`, es decir, necesitamos:

```
functorAreader :: f x -> (a -> x)
readerAfunctor :: (a -> x) -> f x
```

tal que cada una sea la inversa de la otra.

Vamos a tomar de la librería para Haskell *adjunctions* de Edward Kmett la siguiente implementación de functores representables:

```
class (Functor f) => Representable f where
  type Rep f :: *
  index :: f x -> Rep f -> x
  tabulate :: (Rep f -> x) -> f x
```

Nuestra función `functorAreader` es aquí `index`, `readerAfunctor` se convierte en `tabulate`. El tipo a pasa a depender del functor `f` a través de `Rep f`. Además, se establece que se deben cumplir:

```
tabulate . return = return
tabulate . index  = id
index . tabulate  = id
```

Las primera condición establece que `tabulate` sea una transformación natural. Las otras dos condiciones son equivalentes a dicha transformaión natural sea invertible con `index`.

3.2 Inmersión de Yoneda

Primero definamos qué es un inmersión.

Recordemos de la definición de functor, que un functor consistía en un par F_O y F_M que actúan sobre los objetos y morfismos respectivamente. Dado un functor $F\colon \mathcal{C} \to \mathcal{D}$ y dos objetos $A, B \in \mathcal{C}$, denotamos por $F_M|_{\mathcal{C}(A,B)}$ o, sencillamente, $F_{A,B}$ a la «restricción» de F_M a la colección de morfismos $\mathcal{C}(A,B)$. Como F_M respeta el dominio y codominio, tenemos que $F_{A,B}$ es una aplicación:

$$F_{A,B}\colon \mathcal{C}(A,B) \to \mathcal{D}(FA, FB)$$

Con esto en mente, definimos las siguientes clases de functores:

Definición 3.2. *Un functor $F\colon \mathcal{C} \to \mathcal{D}$ es*

- lleno *si para cada $A, B \in \mathcal{C}$, $F_{A,B}$ es sobreyectiva.*
- fiel, *si para cada $A, B \in \mathcal{C}$, $F_{A,B}$ es inyectiva.*
- inyectivo en objetos *si F_O es inyectivo.*

Obsérvese que no es lo mismo que F sea lleno o fiel a que F_M sea inyectiva y sobreyectiva. Podríamos decir que la propiedad de ser lleno y fiel es una propiedad «local».

Definición 3.3. *Un functor es una* inmersión *si es fiel e inyectivo en objetos. Si además es lleno, decimos que es una* inmersión llena.

Definición 3.4. *Dada una categoría $\mathcal{C}$, una* subcategoría *es un par $(\mathcal{D}, F)$ donde $\mathcal{D}$ es una categoría y $F\colon \mathcal{D} \to \mathcal{C}$ es una inmersión.*

Si además, F es llena, decimos que $(\mathcal{D}, F)$ es una subcategoría llena.

Lema 3.1 (Lema de Yoneda). Sea $\mathcal{C}$ una categoría localmente pequeña, un functor $F\colon \mathcal{C} \to Set$ y un objeto $A \in \mathcal{C}$. Denotemos por $Nat(\mathcal{C}(A,-), F)$ a la colección de transformaciones naturales de $\mathcal{C}(A,-) \Rightarrow F$. Hay una biyección:

$$Nat(\mathcal{C}(A,-), F) \cong FA$$

que es natural en A y en F.

Vamos a precisar:

- Hemos visto previamente que $\mathcal{C}(A,-)\colon \mathcal{C} \to Set$ es un functor, luego puede haber una colección de transformaciones naturales de $\mathcal{C}(A,-)$ al functor $F\colon \mathcal{C} \to Set$.
- En la categoría de functores $Set^{\mathcal{C}}$, donde $\mathcal{C}(A,-)$ y F son objetos, $Nat(\mathcal{C}(A,-), F)$ es precisamente la colección de morfismos $Set^{\mathcal{C}}(\mathcal{C}(A,-), F)$.
- La naturalidad en F quiere decir que para toda transformación natural $\theta\colon F \Rightarrow G$, se tiene que el siguiente diagrama es conmutativo:

$$\begin{array}{ccc} Nat(\mathcal{C}(A,-), F) & \xrightarrow{\cong} & FA \\ \downarrow{\scriptstyle Nat(\mathcal{C}(A,-),\theta)} & & \downarrow{\scriptstyle \theta_A} \\ Nat(\mathcal{C}(A,-), G) & \xrightarrow{\cong} & GA \end{array}$$

- La naturalidad en A quiere decir que para todo $f\colon A \to B$, se tiene que el

siguiente diagrama conmuta:

$$\begin{array}{ccc} Nat(\mathcal{C}(A,-),F) & \xrightarrow{\cong} & FA \\ \downarrow{\scriptstyle Nat(\mathcal{C}(f,-),F)} & & \downarrow{\scriptstyle Ff} \\ Nat(\mathcal{C}(B,-),F) & \xrightarrow{\cong} & FB \end{array}$$

donde para una transformación natural $\mu\colon \mathcal{C}(A,-) \Rightarrow F$ y morfismo $h\colon B \to C$:

$$(Nat(\mathcal{C}(f,-),F)\mu)_C(h) = \mu_C \circ \mathcal{C}(f,C)(h) = \mu_C(h \circ f)$$

Demostración. Primero miramos que para una transformación natural $\mu \in Nat(\mathcal{C}(A,-),F)$, es decir, $\mu\colon \mathcal{C}(A,-) \Rightarrow F$, su componente en $A \in \mathcal{C}$ es una función entre conjuntos

$$\mu_A\colon \mathcal{C}(A,A) \to FA$$

Sabemos además que $\mathcal{C}(A,A)$ es un conjunto no vacío, pues al menos contiene id_A

Consideramos la función $\phi\colon Nat(\mathcal{C}(A,-),F) \to FA$ definida por:

$$\begin{aligned} \phi\colon Nat(\mathcal{C}(A,-),F) &\to FA \\ \alpha &\mapsto \alpha_A(\mathrm{id}_A) \end{aligned}$$

Dado cualquier $x \in FA \in Set$, definimos la transformación natural

$$\psi(x)\colon \mathcal{C}(A,-) \Rightarrow F$$

estableciendo su componente para $B \in \mathcal{C}$ cualquiera:

$$\begin{aligned} \psi(x)_B\colon \mathcal{C}(A,B) &\to FB \\ \psi(x)_B(f) &\mapsto (Ff)(x) \end{aligned}$$

Veamos que $\psi(x)$ cumple la naturalidad. Para un $f\colon B \to C$:

$$\begin{array}{ccc} \mathcal{C}(A,B) & \xrightarrow{\psi(x)_B} & FB \\ \downarrow{\scriptstyle \mathcal{C}(A,f)} & & \downarrow{\scriptstyle Ff} \\ \mathcal{C}(A,C) & \xrightarrow{\psi(x)_C} & FC \end{array} \tag{3.1}$$

Tenemos que para todo $h\colon A \to B$:

$$\begin{aligned}(\psi(x)_C \circ \mathcal{C}(A,f))(h) &= \psi(x)_C(f \circ h)\\ &= (F(f \circ h))(x)\\ &= (Ff) \circ (Fh)(x)\\ &= (Ff)(\psi(x)_B(h))\\ &= (Ff) \circ (\psi(x)_B)(h)\end{aligned}$$

Luego el diagrama 3.1 conmuta.

Veamos que ϕ y ψ son inversas. Para $x \in FA$:

$$\begin{aligned}\phi(\psi(x)) &= \psi(x)_A(\mathrm{id}_A)\\ &= (F(\mathrm{id}_A))(x)\\ &= \mathrm{id}_{FA}(x) && \text{por definición de functor}\\ &= x\end{aligned}$$

Para $\mu\colon \mathcal{C}(A,-) \Rightarrow F$, $B \in \mathcal{C}$ y $h\colon A \to B$:

$$\begin{aligned}\psi(\phi(\mu))_B(h) &= \psi(\mu_A(\mathrm{id}_A))_B(h)\\ &= (Fh)(\mu_A(\mathrm{id}_A))\\ &= (\mu_B)(C(A,h)(\mathrm{id}_A)) && \text{por naturalidad de } \mu\\ &= \mu_B(h \circ \mathrm{id}_A)\\ &= \mu_B(h)\end{aligned}$$

Luego ϕ y ψ son inversas y $Nat(\mathcal{C}(A,-),F) \cong FA$. Veamos la naturalidad en F y en A:

$$\begin{array}{ccc} Nat(\mathcal{C}(A,-),F) & \xrightarrow{\phi_F} & FA \\ \Big\downarrow{\scriptstyle Nat(\mathcal{C}(A,-),\theta)} & & \Big\downarrow{\scriptstyle \theta_A} \\ Nat(\mathcal{C}(A,-),G) & \xrightarrow{\phi_G} & GA \end{array} \tag{3.2}$$

Sea $B \in \mathcal{C}$ y $\mu\colon \mathcal{C}(A,-) \Rightarrow F$:

$$\begin{aligned}\theta_A(\phi_F(\mu)) &= \theta_A(\mu_A(\mathrm{id}_A))\\ &= (\theta \circ \mu)_A(\mathrm{id}_A)\\ &= \phi_G(\theta \circ \mu)\end{aligned}$$

que demuestra que el diagrama 3.2 conmuta.

Sea $f\colon A \to B$ y $\mu\colon \mathcal{C}(A,-) \Rightarrow F$:

$$\begin{array}{ccc} Nat(\mathcal{C}(A,-),F) & \xrightarrow{\phi_A} & FA \\ {\scriptstyle Nat(\mathcal{C}(f,-),F)}\downarrow & & \downarrow{\scriptstyle Ff} \\ Nat(\mathcal{C}(B,-),F) & \xrightarrow{\phi_B} & FB \end{array} \tag{3.3}$$

$$\begin{aligned} (Ff)(\phi_A(\mu)) &= (Ff)(\mu_A(\mathrm{id}_A)) \\ &= (\mu_B)(C(A,f)(\mathrm{id}_A)) && \text{por naturalidad de } \mu \\ &= (\mu_B)(f \circ \mathrm{id}_A) \\ &= (\mu_B)(\mathrm{id}_B \circ f) \\ &= (\mu_B)(C(f,B)(\mathrm{id}_B)) \\ &= (\mu \circ C(f,-))_B(\mathrm{id}_B) \\ &= (\phi_B)(\mu \circ C(f,-)) \end{aligned}$$

que demuestra que el diagrama 3.3 conmuta y acaba la demostración. ∎

Observación 3.1. Considerando la categoría opuesta, se tiene que para todo objeto $A \in \mathcal{C}$ y functor contravariante $F\colon \mathcal{C}^{op} \to Set$ hay una biyección

$$Nat(\mathcal{C}(-,A),F) \cong FA$$

natural en A y F.

Nuestra primera aplicación del lema de Yoneda es usando $F = \mathcal{C}(B,-)$. Se tiene que:

$$Nat(\mathcal{C}(A,-),\mathcal{C}(B,-)) \cong \mathcal{C}(B,A)$$

Análogamente:

$$Nat(\mathcal{C}(-,A),\mathcal{C}(-,B)) \cong \mathcal{C}(A,B)$$

Esto motiva la inmersión de Yoneda:

Definición 3.5. *Las* inmersiones de Yoneda *covariante y contravariante son los morfismos:*

$$\begin{aligned} Y\colon \mathcal{C} &\to Set^{\mathcal{C}^{op}} \\ A &\mapsto \mathcal{C}(-,A) \\ f\colon A \to B &\mapsto \mathcal{C}(-,f)\colon \mathcal{C}(-,A) \to \mathcal{C}(-,B) \end{aligned}$$

$$\begin{aligned} K\colon \mathcal{C}^{op} &\to Set^{\mathcal{C}} \\ A &\mapsto \mathcal{C}(A,-) \\ f\colon A \to B &\mapsto \mathcal{C}(f,-)\colon \mathcal{C}(B,-) \to \mathcal{C}(A,-) \end{aligned}$$

es decir, las currificaciones del bifunctor $\mathcal{C}(-,-)\colon \mathcal{C}^{op} \times \mathcal{C} \to Set$.

Obsérvese que K es un functor contravariante y que KA es un functor covariante $\mathcal{C}^{op} \to Set$. Análogamente, Y es un functor covariante a los functores contravariantes $\mathcal{C}^{op} \to Set$.

Veremos ahora que el uso de «inmersión» está justificado:

Proposición 3.1. Las inmersiones de Yoneda son inmersiones llenas.

Demostración. Lo probamos para Y. La inyectividad sobre objetos es evidente. Observamos que, por definición, $\mathcal{C}(A,B) = (YB)A$. Aplicando el lema de Yoneda:

$$\mathcal{C}(A,B) = (YB)A \cong Set^{\mathcal{C}^{op}}(YA, YB)$$

Luego Y es inmersión llena. La prueba para K es análoga. ∎

Un functor contravariante en $Set^{\mathcal{C}^{op}}$ recibe el nombre de *prehaz*. Una consecuencia de la anterior proposición es que toda categoría localmente pequeña es una subcategoría llena de la categoría de prehaces.

Proposición 3.2.

3.3 Referencias

- Milewski, B. (2014). *Category Theory for Programmers*, Capítulos 14-16.
- Riehl, E. (2014). *Category Theory in Context*, Capítulo 2.
- Awodey, S. (2006). *Category Theory*, Capítulo 8.

4 | Categorías cartesianamente cerradas

4.1 Exponencial

Si $\mathcal{C}$ tiene productos binarios, entonces el *exponencial* de los objetos B y C de $\mathcal{C}$ es el objeto

$$C^B$$

y un morfismo:

$$\varepsilon\colon C^B \times B \to C$$

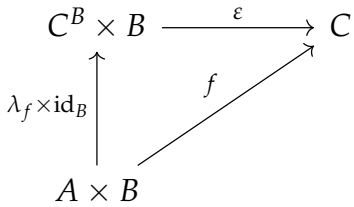
tal que para todo objeto A y morfismo

$$f\colon A \times B \to C$$

hay un único morfismo

$$\lambda_f\colon A \to C^B$$

tal que el siguiente diagrama conmuta:



El morfismo ε se denomina *evaluación*. El morfismo λ_f se denomina *transposición* o *currificación* de f.

Obsérvese que

$$\begin{aligned} \lambda \colon C(A \times B, C) &\to C(A, C^B) \\ f &\mapsto \lambda_f \end{aligned}$$

es un isomorfismo por la existencia e unicidad de λ_f por cada f.

En programación funcional, la currificación resulta de gran utilidad. En Haskell, por ejemplo, se usa la función `curry` de tipo:

```
curry :: ((a, b) -> c) -> a -> b -> c
```

Su inversa es `uncurry`, de tipo:

```
uncurry :: (a ->  b -> c) -> (a, b) -> c
```

4.2 Categorías cerradas cartesianas

Definición 4.1. *Decimos que una categoría es* cartesianamente cerrada *si cumple las siguientes condiciones:*

1. *Tiene productos finitos.*
2. *Tiene el exponencial de dos objetos cualesquiera.*

Si además tiene coproductos finitos, se denomina bicartesianamente cerrada

Ejemplo 4.1. Consideramos la categoría *Pos* de conjuntos parcialmente ordenados donde los morfismos son funciones monótonas del que hablamos en 2.12. Sean P y Q posets, veamos que $Q^P = \{f \colon Q \to P \mid f \text{ monótona}\}$ con el orden:

$$f \leq g \Leftrightarrow f(p) \leq g(p) \text{ para todo } p \in P$$

El morfismo evaluación es:

$$\varepsilon(f, p) = f(p)$$

Veamos que efectivamente ε es un morfismo. Supongamos que $(f,p) \leq (f',p')$, entonces:

$$\begin{aligned} \varepsilon(f,p) &= f(p) \\ &\leq f(p') && \text{por ser } f \text{ monótona y } p \leq p' \\ &\leq f'(p') && \text{por ser } f \leq f' \\ &= \varepsilon(f',p') \end{aligned}$$

Luego ε es un morfismo *Pos*. Sea $f\colon X \times P \to Q$ monótona y sea $x \leq x'$.

$$\begin{aligned} \lambda_f(x)(p) &= f(x,p) \\ &\leq f(x',p) && \text{pues } (x,p) \leq (x',p) \\ &\leq f'(x',p) && \text{pues } f \leq f' \\ &= \lambda_f(x')(p) \end{aligned}$$

Lema 4.1. En una categoría cartesianamente cerrada $\mathcal{C}$, la exponenciación por un objeto fijo es un functor.

Demostración. Queremos probar que la exponenciación:

$$-^Z\colon \mathcal{C} \to \mathcal{C}$$

para un objeto fijo Z es un endofunctor. Como $\mathcal{C}$ es cartesianamente cerrada, la exponenciación de A^Z está bien definida para todo objeto A de $\mathcal{C}$. Consideremos ahora un morfismo $f\colon A \to B$, tenemos que definir $f^Z\colon A^Z \to B^Z$. Primero observamos que:

$$\varepsilon\colon A^Z \times Z \to A$$

Luego:

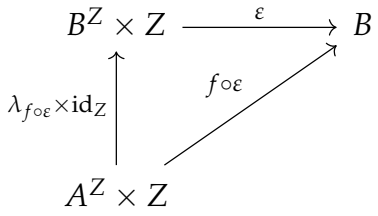
$$f \circ \varepsilon\colon A^Z \times Z \to B$$

Transponiendo:

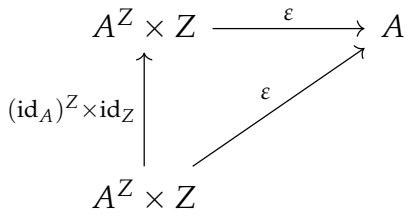
$$\lambda_{f\circ\varepsilon}\colon A^Z \to B^Z$$

Así que definimos f^Z como $\lambda_{f\circ\varepsilon}$.

Tenemos el siguiente diagrama conmutativo consecuencia de la transposición:

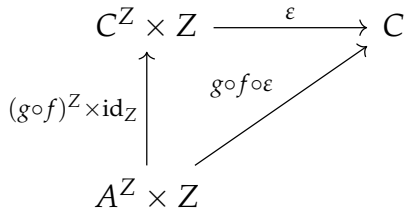


Luego considerando $(\mathrm{id}_A)^Z$:

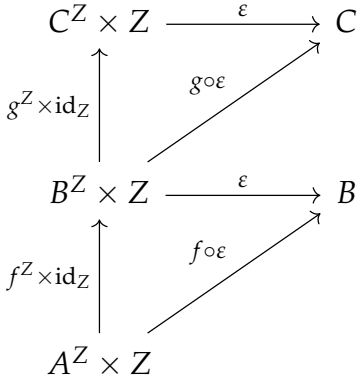


Como id_{A^Z} cumple también el anterior diagrama y tenemos que el morfismo que cumple el diagrama es único, $(\mathrm{id}_A)^Z = \mathrm{id}_{A^Z}$.

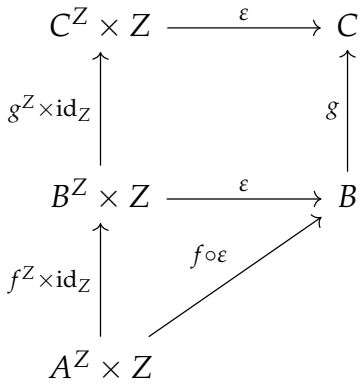
Por otro lado, si tenemos morfismos: $f\colon A \to B$ y $g\colon B \to C$, tenemos que $(g \circ f)^Z$ es el único que conmuta el diagrama:



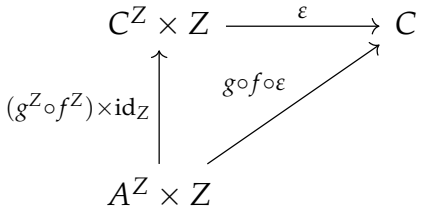
Sin embargo, observamos que también conmuta:



Podemos reemplazar la flecha diagonal $g \circ \varepsilon$ obteniendo:



Finalmente, este diagrama implica que:



conmuta, luego prueba por unicidad que $(g \circ f)^Z = g^Z \circ f^Z$.

Esto demuestra que $-^Z$ es un endofunctor. ∎

4.3 Lógica

Definición 4.2. *Un* sistema deductivo *es un grafo:*

1. *Con flechas* $A \xrightarrow{1_A} A$
2. *Con una operación binaria sobre flechas* $A \xrightarrow{1_A} A$*:*

$$\frac{A \xrightarrow{f} B \qquad B \xrightarrow{g} C}{A \xrightarrow{gf} C}$$

Los nodos de los grafos de un sistema deductivo se denominan *fórmulas* y las flechas, *pruebas*.

Hasta aquí es evidente que el concepto de sistema deductivo es equivalente a categoría.

Definición 4.3. *Un* cálculo de conjunción *es un sistema deductivo donde existe una fórmula* $\top$*, una operación binaria* $\wedge$ *llamada conjunción y las siguientes reglas:*

1. $A \xrightarrow{O_A} \top$.
2. $A \wedge B \xrightarrow{\pi^1_{A,B}} A$.
3. $A \wedge B \xrightarrow{\pi^2_{A,B}} B$.
4. $$\frac{C \xrightarrow{f} A \qquad C \xrightarrow{g} C}{C \xrightarrow{\langle f,g\rangle} A \wedge B}$$

Proposición 4.1. Una categoría con productos finitos es un cálculo de conjunción donde $\top$ es el objeto terminal y $\wedge$ es el producto de objetos.

Demostración. Tomamos O_A como el morfismo $A \to \top$. Igualmente, tomamos π^1 y π^2 como las proyecciones del objeto producto $A \times B$ a sus factores. Para dos morfismo $f\colon C \to A$ y $g\colon C \to B$, sabemos que existe un único morfismo $f\colon C \to A \times B$, que se corresponde con $\langle f, g\rangle$.

Definición 4.4. *Un* cálculo proposicional intuicionista positivo *es un cálculo de conjunción con una operación binaria* $\Rightarrow$ *sobre fórmulas y:*

1. $(A \Rightarrow B) \wedge A \xrightarrow{\varepsilon_{A,B}} B$.
2. $$\frac{C \wedge A \xrightarrow{h} B}{C \xrightarrow{h^*} A \Rightarrow B}$$

Proposición 4.2. Una categoría cartesianamente cerrada es un cálculo proposicional intuicionista positivo donde $\Rightarrow$ se corresponde con la exponenciación.

Demostración. El morfismo $\varepsilon\colon B^A \times A \to B$ de la evaluación de la exponenciación sirve como $\varepsilon_{A,B}$. Por otro lado h^* se corresponde con la currificación λ_h.

Definición 4.5. *Una* cálculo intuicionista *es un cálculo proposicional intuicionista positivo con una fórmula* $\bot$*, una operación disyunción* $\vee$ *sobre fórmulas y unas flechas:*

1. $\bot \xrightarrow{!} A$.
2. $A \xrightarrow{\kappa^1_{A,B}} A \vee B$

3. $B \xrightarrow{\kappa^2_{A,B}} A \vee B$
4. $$\frac{A \xrightarrow{f} C \qquad B \xrightarrow{g} C}{A \vee B \xrightarrow{[f,g]} C}$$

Proposición 4.3. Una categoría bicartesianamente cerrada es un cálculo intuicionista donde $\bot$ se corresponde con el objeto inicial y $\vee$ se corresponde con el coproducto.

Demostración. Las primera regla es consecuencia de la definición de objeto inicial. Las otras tres reglas están dadas por las propiedades del coproducto. ∎

Este vínculo entre lógica intuicionista y categorías bicartesianamente cerradas nos permite movernos entre las dos teorías cuando queramos demostrar algún resultado, por ejemplo:

Lema 4.2. En cálculo intuicionista proposicional hay como mucho una demostración $A \to \bot$ salvo equivalencia de pruebas.

Demostración. En una categoría bicatersianamente cerrada $\mathcal{C}$ con objeto inicial 0, para cualquier par de objetos A y B cualesquiera:

$$\mathcal{C}(0 \times A, B) \cong \mathcal{C}(0, B^A)$$

Como hay un único morfismo en $\mathcal{C}(0, B^A)$ por ser 0 inicial, entonces hay un único morfismo $0 \times A \to B$. Luego $0 \times A$ es objeto inicial. Supongamos que hubiera un morfismo $f\colon A \to 0$. Como también hay un morfismo id_A, debe haber un morfismo $h\colon A \to 0 \times A$ por la definición de objeto producto. Si $\pi_2\colon 0 \times A \to A$ es la segunda proyección, entonces $\pi_2 \circ h = \mathrm{id}_A$ por la definición de producto y $h \circ \pi_2 = \mathrm{id}_{0\times A}$ porque sólo hay un morfismo $0 \times A \to 0 \times A$. Por lo tanto $A \cong 0$ y sólo f es el único morfismo. ∎

Ahora que hemos construido en paralelo un sistema lógico intuicionista y una categoría equivalente, nos podemos preguntar si podemos dar un paso más y llegar a la lógica clásica. Para ello hay que añadir el principio del tercero excluso, que es equivalente a la regla de eliminación de doble negación. Para ello, definimos $\neg A$ como la fórmula $A \Rightarrow \bot$.

$$\frac{\neg\neg A}{\mathrm{A}}$$

¿Qué ocurriría si hubiera un isomorfismo $\neg\neg A \to A$ en una categoría bicartesianamente cerrada? Primero observémos que $\neg\neg A$ es equivalente a 0^{0^A}.

Teorema 4.1. *Sea $\mathcal{C}$ una categoría cartesianamente cerrada con objeto inicial 0. Si para todo objeto A se tiene $0^{0^A} \cong A$, entonces $\mathcal{C}$ es fina, es decir, tiene como mucho un morfismo entre dos objetos.*

Demostración. Por definición, para objetos B y C:

$$\mathcal{C}(B, 0^C) \cong \mathcal{C}(B \times C, 0)$$

Como hay como mucho un morfismo en $\mathcal{C}(B \times C, 0)$, entonces hay como mucho un morfismo $B \to 0^C$. Tomando $C = 0^A$, se tiene que hay como mucho un morfismo $B \to 0^{0^A} \cong A$.

Por lo tanto, cuando añadimos la regla de eliminación de doble negación perdemos la capacidad de distinguir entre distintas pruebas en nuestra categoría. Esto vuelve poco interesante estudiar categorías de este tipo.

Por otro lado, la *correspondencia de Curry-Howard* establece una equivalencia entre el cálculo intuicionista y el λ-cálculo con tipos simples. Las fórmulas se corresponden con tipos y una fórmula es cierta (en un sentido intuicionista) si está habitada, es decir, si existe algún término con dicho tipo. Esto resulta importante para nosotros, pues Haskell implementa λ-cálculo con tipos simples. Por lo tanto, podemos hacer deducciones en cálculo intuicionista con Haskell.

De hecho, tenemos la siguiente tabla de equivalencias:

Lógica	Categoría	Haskell
$\bot$	0	`Empty`
$\top$	1	`()`
$\neg$	0^-	`a -> Empty`
$p \wedge q$	$A \times B$	`(a, b)`
$p \vee q$	$A + B$	Either a b

Con esto en mente, podemos hacer algunas demostraciones sencillamente escribiendo una función que habite el tipo correspondiente.

```
modus_ponens :: a -> (a -> b) -> b
modus_ponens x f = f x
```

```
prueba_por_negacion :: (a -> Empty) -> Not a
prueba_por_negacion f = f

de_morgan :: Either (Not a) (Not b) -> Not (a, b)
de_morgan (Left f) = \(x, y) -> f x
de_morgan (Right f) = \(x, y) -> f y

otro_ejemplo :: (a -> (b -> c)) -> (a -> b) -> (a -> c)
otro_ejemplo f g = \x -> f x (g x)
```

4.4 Referencias

- Awodey, S. (2006). *Category Theory*, Capítulo 6.
- Lambek, J. (1986). *Introduction to Higher Order Categorical Logic*, Parte I.
- Low, Z.L. (2011). *Bicartesian closed categories and logic*.

5 | Monoides

"A monad is a monoid in the category of endofunctors, what's the problem?"

A Brief, Incomplete, and Mostly Wrong History of Programming Languages
James Iry

5.1 Monoides clásicos

En álgebra, un *monoide* $(M, \cdot)$ es un conjunto M junto a una operación binaria $\cdot\colon M \times M \to M$ asociativa y con elemento unidad en M. Una forma alternativa de ver un monoide es como una categoría de un sólo objeto. Si llamamos A al único objeto de dicha categoría, identificamos los elementos de M con los morfismos $f\colon A \to A$ y la composición de morfismos con la operación binaria. El elemento identidad se identifica con el morfismo identidad.

Por ejemplo, el monoide $(\mathbb{N}, +)$ se corresponde con la categoría:

A su vez, para cualquier categoría localmente pequeña $\mathcal{C}$, todo objeto $A \in \mathcal{C}$ induce un monoide $\mathcal{C}(A, A)$.

5.2 Monoides en Haskell

En Haskell, los monoides están representando por la siguiente clase:

```
class Monoid m where
  mappend :: m -> m -> m
  mempty :: m
```

donde `mappend` debe ser asociativa y `mempty` debe ser su elemento unidad. En Haskell, se suele usar el operador infijo `<>` como sinónimo de `mappend` para facilitar la lectura. Un ejemplo de monoide habitual es el de la listas de tipo un tipo cualquiera `a` donde `mappend` se corresponde con la concatenación.

```
> [2,3,4] <> [5,6,7]
[2,3,4,5,6,7]
```

La interpretación de un monoide como la estructura de los endomorfismo de un objeto se puede ver aquí cuando aplicamos parcialmente `mappend` a un objeto `x` de tipo `m`:

```
mappend x :: m -> m
```

Tenemos que `mappend x` es un endomorfismo en el tipo `m` y que `mappend mempty` es equivalente al morfismo identidad. Es decir `mappend` se puede ver como una función que asocia cada elemento del monoide (como grupo) con el endomorfismo correspondiente del monoide (como endomorfismos de `m`).

5.3 Categorías monoidales

Otra forma de ver monoides en teoría de categorías es através de *objeto monoide* en una categoría.

Definición 5.1. *Una* categoría monoidal *es una categoría $\mathcal{C}$ con un bifunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ y un objeto unidad $I \in \mathcal{C}$ con isomorfismos naturales:*

- *El asociador:*
$$\alpha_{ABC} \colon A \otimes (B \otimes C) \Rightarrow (A \otimes B) \otimes C$$
- *El unidor izquierdo:*
$$\lambda_A \colon I \otimes A \Rightarrow A$$
- *El unidor derecho:*
$$\rho_A \colon A \otimes I \Rightarrow A$$

de manera que cumpla las leyes de coherencia, es decir, que los siguientes diagramas conmuten (entiéndase que estamos tomando las correctas componentes de α, λ y ρ):

$$\begin{array}{ccccc} A \otimes (B \otimes (C \otimes D)) & \xrightarrow{\alpha} & (A \otimes B) \otimes (C \otimes D) & \xrightarrow{\alpha} & ((A \otimes B) \otimes C) \otimes D \\ \downarrow{\scriptstyle \mathrm{id} \otimes \alpha} & & & & \uparrow{\scriptstyle \alpha \otimes \mathrm{id}} \\ A \otimes ((B \otimes C) \otimes D) & & \xrightarrow{\qquad\qquad\alpha\qquad\qquad} & & (A \otimes (B \otimes C)) \otimes D \end{array}$$

$$\begin{array}{ccc} A \otimes (I \otimes C) & \xrightarrow{\alpha} & (A \otimes I) \otimes C \\ & {\scriptstyle \mathrm{id} \otimes \lambda} \searrow \quad \swarrow {\scriptstyle \rho \otimes \mathrm{id}} & \\ & A \otimes C & \end{array}$$

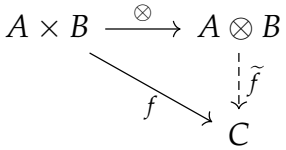
y por último:
$$\lambda_I = \rho_I$$

Un ejemplo sencillo de categoría monoidal es cualquier categoría con productos finitos, tomando $A \otimes B = A \times B$. Es más, en dicho caso, existe un isomorfismo natural $A \otimes B \cong B \otimes A$. Una categoría monoidal con dicho isomorfismo natural se denomina *categoría monoidal simétrica*. Por lo tanto, *Set* forma una categoría monoidal simétrica con el producto cartesiano y un conjunto unitario cualquiera como identidad del producto. También, *Cat* forma una categoría monoidal simétrica con el producto y una categoría unitaria como $\mathbb{1}$ como identidad del producto.

Veamos otro ejemplo más sofisticado.

Ejemplo 5.1. Consideramos la categoría de grupos abelianos *Ab*. Para dos grupos A y B, definimos $A \otimes B$ como su producto tensorial como $\mathbb{Z}$-módulos. Más explícitamente, $A \otimes B$ es un grupo abeliano con un producto bilineal $\otimes$ tal que

para todo grupo abeliano C y toda aplicación bilineal $f\colon A \times B \to C$, existe un único homomorfismo de grupos $\widetilde{f}$ tal que el siguiente diagrama conmuta:



Por la unicidad de la construcción, hay un único isomorfismo natural $\alpha_{ABC}\colon A \otimes (B \otimes C) \to (A \otimes B) \otimes C$.

Veamos que $\mathbb{Z}$ es el objeto unidad. Para ello, consideramos una aplicación bilineal $f\colon A \times \mathbb{Z} \to B$ cualquiera. Definimos la aplicación lineal

$$\begin{aligned} \phi\colon A \times \mathbb{Z} &\to A \\ \phi\colon (a,n) &\mapsto \overbrace{a + \cdots + a}^{\text{n veces}} \end{aligned}$$

y el homomorfismo de grupo:

$$\begin{aligned} \widetilde{f}\colon A &\to B \\ \widetilde{f}\colon a &\mapsto f(a,1) \end{aligned}$$

Entonces, por linealidad sobre $\mathbb{Z}$:

$$f(a,n) = f(a, \overbrace{1+1+\cdots+1}^{\text{n veces}}) = \overbrace{f(a,1)+\cdots+f(a,1)}^{\text{n veces}} = (\widetilde{f} \circ \phi)(a,n)$$

Luego $A \otimes \mathbb{Z} \cong_{\lambda_A} A \cong_{\rho_A} \mathbb{Z} \otimes G$.

Es sencillo comprobar que las leyes de coherencia se cumplen.

5.4 Categorías enriquecidas

Recordemos que una categoría $\mathcal{C}$ es localmente pequeña si para todo pares de objetos $A, B \in \mathcal{C}$, se tiene que $\mathcal{C}(A,B) \in Set$. Una idea esencial en la teoría de categorías de orden superior es el concepto de *categoría enriquecida*, que consiste en remplazar Set de la definición anterior con cualquier otra categoría monoidal simétrica. Informalmente, dada una categoría monoidal simétrica $\mathcal{V}$, una categoría $\mathcal{C}$ se dice que está $\mathcal{V}$-enriquecida si para cualquier par de objetos $A, B \in \mathcal{C}$, $\mathcal{C}(A,B) \in \mathcal{V}$. A esta definición hay que añadir unas condiciones de compatibilidad con la composición, de las que hablamos a continuación:

Definición 5.2. *Dada una categoría monoidal simétrica $(\mathcal{V}, \otimes, I)$, una $\mathcal{V}$-categoría $\mathcal{C}$ consiste en:*

- *Una colección de objetos.*
- *Para cada par de objetos A, B en $\mathcal{C}$, un objeto llamado* hom-objeto $\mathcal{C}(A,B) \in \mathcal{V}$.
- *Para cada $A \in \mathcal{C}$, un morfismo $\mathrm{Id}_A \colon I \to \mathcal{C}(A,A)$ en $\mathcal{V}$.*
- *Para todo $A, B, C \in \mathcal{C}$, un morfismo $\circ \colon \mathcal{C}(B,C) \otimes \mathcal{C}(A,B) \to \mathcal{C}(A,C)$ en $\mathcal{V}$.*

de manera que los siguientes diagramas conmuten:

$$\begin{array}{ccc} \mathcal{C}(C,D) \otimes \mathcal{C}(B,C) \otimes \mathcal{C}(A,B) & \xrightarrow{\mathrm{id} \otimes \circ} & \mathcal{C}(C,D) \otimes \mathcal{C}(A,C) \\ \downarrow{\scriptstyle \circ \otimes \mathrm{id}} & & \downarrow{\scriptstyle \circ} \\ \mathcal{C}(B,D) \otimes \mathcal{C}(A,B) & \xrightarrow{\quad \circ \quad} & \mathcal{C}(A,D) \end{array}$$

$$\begin{array}{ccc} \mathcal{C}(A,B) \otimes I & \xrightarrow{\mathrm{id} \otimes \mathrm{Id}_A} & \mathcal{C}(A,B) \otimes \mathcal{C}(A,A) \\ & \searrow^{\cong} & \downarrow{\scriptstyle \circ} \\ & & \mathcal{C}(A,B) \end{array} \qquad \begin{array}{ccc} \mathcal{C}(B,B) \otimes \mathcal{C}(A,B) & \xleftarrow{\mathrm{Id}_B \otimes \mathrm{id}} & I \otimes \mathcal{C}(A,B) \\ \downarrow{\scriptstyle \circ} & \swarrow_{\cong} & \\ \mathcal{C}(A,B) & & \end{array}$$

Ejemplo 5.2. Dotemos ahora a Cat de estructura de Cat-categoría, es decir, 2-categoría.

Primero recordemos que $(Cat, \times, \mathbb{1})$ es una categoría monoidal simétrica. Por otro lado, para un par de categorías pequeñas $\mathcal{C}$ y $\mathcal{D}$, los functores $\mathcal{C} \to \mathcal{D}$ forman una categoría pequeña $\mathcal{D}^{\mathcal{C}}$ como comentamos en 2.1. En esta categoría, los morfismos entre dos functores F y G eran las transformaciones naturales $F \Rightarrow G$.

Además, consideramos la transformación natural $\mathrm{Id}_A \colon \mathbb{1} \to \mathcal{C}^{\mathcal{C}}$ que asocia el único objeto de $\mathbb{1}$ con el functor identidad. Por último, para unas categorías cualesquiera $\mathcal{C}$, $\mathcal{D}$ y $\mathcal{E}$ tenemos que definir un morfismo.

$$\circ \colon \mathcal{E}^{\mathcal{D}} \times \mathcal{D}^{\mathcal{C}} \to \mathcal{E}^{\mathcal{C}}$$

Ojo, un morfismo en Cat es un functor. Sea $F \in \mathcal{D}^{\mathcal{C}}$ y $G \in \mathcal{E}^{\mathcal{D}}$, definimos $G \circ F$ como la composición habitual de functores. Ahora bien, como $\circ$ es functor, debe actuar también sobre los morfismos, que aquí son transformaciones

naturales. Para transformaciones naturales $\mu\colon F \Rightarrow G$ en $\mathcal{D}^{\mathcal{C}}$ y $\eta\colon H \Rightarrow K$ en $\mathcal{E}^{\mathcal{D}}$, basta definir $\eta \circ \mu$ como su composición horizontal.

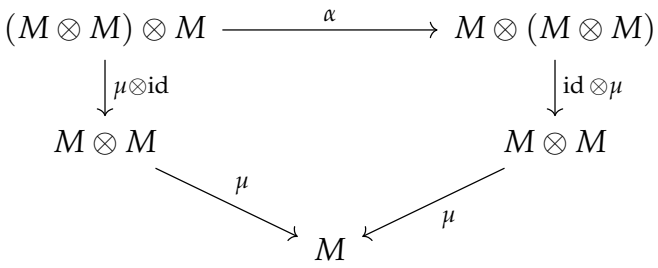
Con estas definiciones se cumple la conmutatividad de los diagramas de las categoría enriquecidas y deducimos que *Cat* es una 2-categoría.

5.5 Monoides

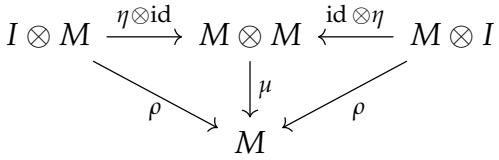
Volviendo, al tema de los monoides, ahora podemos generalizar el concepto de monoide.

Definición 5.3. *Un* monoide *es un objeto M de una categoría monoidal $(\mathcal{C}, \otimes, I)$ con dos morfismos: $M \otimes M \xrightarrow{\mu} M \xleftarrow{\eta} I$ que cumplan:*

- *La propiedad asociativa:*



- *La propiedad de la unidad:*



Ejemplo 5.3. Algunos ejemplos de monoides son:

- Un monoide sobre la categoría monoidal $(Set, \times, \{1\})$ es un monoide en el sentido algebraico.
- Un monoide sobre la categoría monoidal de grupos abelianos $(Ab, \otimes, \mathbb{Z})$ descrita en el ejemplo 5.1 es un *anillo*.
- Un monoide sobre la categoría monoidal de k-espacios vectoriales $(Vect_k, \otimes_k, k)$ es una *k-álgebra*.

Un ejemplo más será de gran importancia para nosotros, las mónadas.

5.6 Referencias

- Awodey, S. (2006). *Category Theory*, Capítulo 4.
- Mac Lane, S. (1997). *Categories for the Working Mathematician*, Capítulo 7.
- Riehl, E. (2014). *Category Theory in Context*, Capítulo 1.
- Riehl, E. (2014). *Categorical homotopy theory*, Capítulo 3.

6 | Mónadas

6.1 Mónadas de una categoría

Dada una categoría $\mathcal{C}$, consideramos la categoría $\mathcal{C}^{\mathcal{C}}$, donde los objetos son endofunctores y los morfismos vienen dados por la composición entre functores.

Proposición 6.1. $\mathcal{C}^{\mathcal{C}}$ forma una categoría monoidal con la composición como producto tensorial y functor unidad Id como objeto unidad.

Demostración. Esto es consecuencia de que Cat sea una 2-categoría, como vimos en 5.2.

Por lo tanto, podemos definir monoides sobre $\mathcal{C}^{\mathcal{C}}$. Dichos monoides se denominan *mónadas*. Tratemos de dar una definición más explícita, especializando la definición de monoide teniendo en cuenta que:

- $\circ$ es asociativa.
- El functor identidad $\mathrm{Id}\colon \mathcal{C} \to \mathcal{C}$ cumple que:

$$\mathrm{Id} \circ F = F$$

$$F \circ \mathrm{Id} = F$$

- El bifunctor $\circ$ actuando sobre transformaciones naturales es la composición horizontal $*$.
- Para la transformación natural identidad $\iota\colon F \Rightarrow F$:

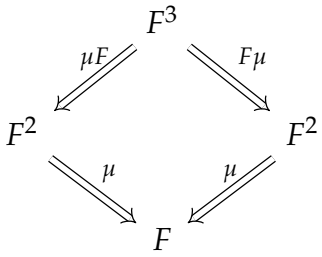
$$\mu * \iota = \mu F$$

$$\iota * \mu = F\mu$$

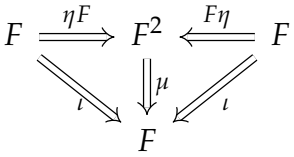
- Escribiremos F^n para referirnos a $\underbrace{F \circ \cdots \circ F}_{n \text{ veces}}$.

Definición 6.1. *Una* mónada *es un endofunctor F de una categoría $\mathcal{C}$ con dos transformaciones naturales:* $F^2 \xrightarrow{\mu} F \xleftarrow{\eta} \mathrm{Id}$ *que cumplan:*

- *La propiedad asociativa:*



- *La propiedad de la unidad:*



Ejemplo 6.1. Sea $\mathcal{P}\colon Set \to Set$ el endofunctor potencia y consideramos las transformaciones naturales.

$$\begin{aligned} \mu_A \colon \mathcal{P}^2(A) &\to \mathcal{P}(A) \\ \{S_i\}_{i\in I} &\mapsto \bigcup_{i\in I} B_i \end{aligned}$$

$$\begin{aligned} \eta_A \colon \mathrm{Id}\, A = A &\to \mathcal{P}(A) \\ a &\mapsto \{a\} \end{aligned}$$

Veamos que cumple las condiciones de mónada. Sea A un conjunto cualquiera. Obsérvese que $\mu_{\mathcal{P}(A)}(\mathcal{P}(A)) = A$, pues $A \in \mathcal{P}(A)$ y todo elemento de $\mathcal{P}(A)$ está contenido en A.

$$\begin{aligned} (\mu \circ \mu\mathcal{P})(\mathcal{P}^3(A)) &= \mu_{\mathcal{P}^2(A)} \left(\mu_{\mathcal{P}^3(A)} \mathcal{P}^3(A) \right) \\ &= \mu_{\mathcal{P}^2(A)} \mathcal{P}^2(A) \\ &= \mathcal{P}(A) \end{aligned}$$

$$\begin{aligned}(\mu \circ \mathcal{P}\mu)(\mathcal{P}^3(A)) &= \mu_{\mathcal{P}^2(A)} \left((\mathcal{P}\mu_{\mathcal{P}^2(A)})\mathcal{P}^3(A) \right) \\ &= \mu_{\mathcal{P}^2(A)}\mathcal{P}^2(A) \\ &= \mathcal{P}(A)\end{aligned}$$

$$\begin{aligned}(\mu \circ \eta\mathcal{P})(\mathcal{P}(A)) &= \mu_{\mathcal{P}^2(A)}\eta_{\mathcal{P}(A)}\mathcal{P}(A) \\ &= \mu_{\mathcal{P}^2(A)}(\{\mathcal{P}(A)\}) \\ &= \mathcal{P}(A)\end{aligned}$$

$$\begin{aligned}(\mu \circ \mathcal{P}\eta)(\mathcal{P}(A)) &= \mu_{\mathcal{P}^2(A)} \left(\mathcal{P}\eta_{\mathcal{P}(A)}(\mathcal{P}(A)) \right) \\ &= \mu_{\mathcal{P}^2(A)} \left(\{\mathcal{P}(A)\} \right) \\ &= \mathcal{P}(A)\end{aligned}$$

6.2 Categoría de Kleisli

La siguiente construcción nos llevará de vuelta al dominio de Haskell.

Definición 6.2. *Sea (F, η, μ) una mónada sobre una categoría $\mathcal{C}$. La* categoría de Kleisli *$\mathcal{C}_F$ es la categoría donde:*

- *Sus objetos son los objetos de $\mathcal{C}$.*
- *Un morfismo $A \to_F B$ en $\mathcal{C}_F$ es un morfismo de la forma $A \to FB$ en $\mathcal{C}$.*
- *La composición entre dos morfismos $f\colon A \to_F B$ y $g\colon B \to_F C$ viene dado por la conmutatividad del siguiente diagrama:*

$$\begin{array}{ccc} A & \xrightarrow{g \circ_F f} & FC \\ {\scriptstyle f}\downarrow & & \uparrow{\scriptstyle \mu_C} \\ FB & \xrightarrow[Fg]{} & F^2C \end{array}$$

- *El morfismo identidad* id_A *en $\mathcal{C}_F$ se correspoonde con η_A en $\mathcal{C}$.*

Para demostrar que la categoría de Kleisli es realmente un categoría, nos limitamos a comprobar la asociatividad:

Para $h\colon C \to FD$, $g\colon B \to FC$ y $f\colon A \to FB$, se tiene:

$$\begin{aligned}(h \circ_F g) \circ f = (\mu_D \circ Fh \circ g) \circ_F f \\ &= \mu_D \circ F(\mu_D \circ Fh \circ g) \circ g \\ &= \mu_D \circ F\mu_D \circ F^2 h \circ Fg \circ g\end{aligned}$$

Por la naturalidad de μ.

6.3 Mónadas en Haskell

¿Cómo vuelve todo esto a la programación funcional? Consideremos el siguiente ejemplo:

```
convertir :: String -> Maybe Float
inversa :: Float -> Maybe Float
```

donde `convertir` intenta convertir una cadena de texto a un número e inversa intenta dar la inversa de un número. En estos dos ejemplos, hay casos donde la función fallaría. Cuando `String` no se puede convertir a un número, o cuando `inversa` intenta invertir el número `0`. Imaginemos que quisieramos combinar estas dos funciones en una nueva función:

```
convertir_inversa :: String -> Maybe Float
```

Donde `convertir_inversa` primero usa `convertir` y luego usa `inversa` sobre el posible resultado. ¿Cómo podríamos implementar esta función? Un camino sería el siguiente:

```
convertir_inversa str =
  let conv = convertir str in
  if isNothing conv
  then Nothing
  else inversa (fromJust conv)
```

Recuérdese que `Maybe` es un functor, luego podemos aplicar `Maybe` a la aplicación inversa para obtener:

```
fmap inversa :: Maybe Float -> Maybe (Maybe Float)
```

ahora que el dominio de `fmap inversa` coincide con el codominio de `convertir`, podemos hacer su composición:

```
fmap inversa . convertir :: String -> Maybe (Maybe Float)
```

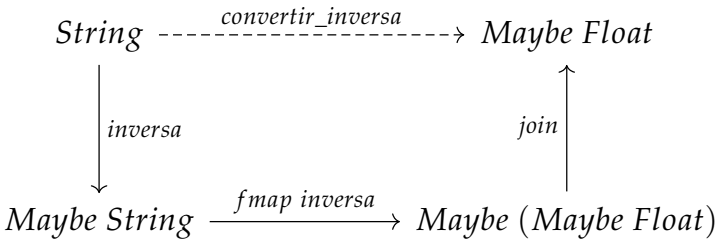
Aquí entra en juego la transformación natural $\mu\colon F^2 \Rightarrow F$, que en Haskell recibe el nombre de `join`, de tipo

```
join :: Monad m => m (m a) -> m a
```

`join` nos permite transformar `Maybe (Maybe Float)` en `Maybe Float`. Luego podríamos implementar `convertir_inversa` como la composición de estas tres funciones:

```
convertir_inversa = join . fmap inversa . convertir
```

Estamos definiendo `convertir_inversa` de manera que el siguiente diagram conmute:



Se puede reconocer aquí el diagrama que define la composición de Kleisli $\circ_F$. En Haskell, dicha composición se denota a menudo como `>=>`, llamado comúnmente *operador pez*, con tipo:

```
(>=>) :: Monad m => (a -> m b) -> (b -> m c) -> a -> m c
```

y nos permite dar la siguiente definición sucinta de `convertir_inversa`

```
convertir_inversa = convertir >=> inversa
```

Las mónadas en Haskell están definidas como una subclase de `Applicative`, que a su vez es una subclase de `Functor`. Aunque no entraremos mucho en ello, `Applicative` representa los functores que mantienen la estructura monoidal de *Hask*, y están definidos como:

```
class Functor f => Applicative f where
    pure :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

Las mónadas se definen por la clase `Monad`:

```
class Applicative m => Monad m where
  (>>=)       :: m a -> (a -> m b) -> m b
  return      :: a -> m a
```

Además, se deben cumplir las llamadas leyes de mónadas:

```
return v >>= k = k v
x >>= return = x
m >>= (\y -> k y >>= h) = (m >>= k) >>= h
```

El operador `(»=)` se puede definir en terminos de `join :: m (m a) -> m a`, que era la implementación de la transformación natural μ, y viceversa.

```
x >>= f = join (fmap f x)
join x = x >>= id
```

También se puede implementar el operador pez de la categoría de Kleisli `(>=>)` con `(»=)`.

```
f >=> g = \x -> f x >>= g
```

Haskell añade una forma de hacer operaciones con mónadas de forma aparentemente imperativa:

```
convertir_inversa x = do
  y <- convierte x
  inversa y
```

Esta expresión es equivalente a:

```
convertir_inversa x =
  convierte x >>= \y ->
    inversa y
```

Hemos usado implícitamente que `Maybe` forma una mónada, comprobémoslo:

Ejemplo 6.2. El functor `Maybe` forma una mónada:

```
instance Monad Maybe where
  (Just x) >>= f = f x
  Nothing >>= _ = Nothing

  return x = Just x
```

Veamos que cumple las leyes de las mónadas:

```
  return v >>= k
= { definición de return }
  Just x >>= k
= { definición de (>>=) }
  k x
```

Esto demuestra la primera ley. Demostraremos las otras dos leyes por casos:

```
  Nothing >>= return
= { definición de (>>=) }
  Nothing

  (Just x) >>= return
= { definición de (>>=) }
```

```
  return x
= { definición de return }
  Just x
```

Esto demuestra la segunda ley. Finalmente:

```
  Nothing >>= (\y -> k y >>= h)
= { definición de (>>=) }
  Nothing
= { definición de (>>=) }
  Nothing >>= h
= { defincón de (>>=) }
  (Nothing >>= k) >>= h
```

```
  (Just x) >>= (\y -> k y >>= h)
= { definición de (>>=) }
  (\y -> k y >>= h) x
= { reducción beta }
  k x >>= h
= { definición de (>>=) }
  (Just x >>= k) >>= h
```

Por lo que `Maybe` forma una mónada.

6.4 Referencias

- Awodey, S. (2006). *Category Theory*, Capítulos 7 y 10.
- Buurlage, J. (2017). *Categories and Haskell*, Capítulo 7.
- Milewski, B. (2014). *Category Theory for Programmers*, Capítulo 20.

7 | F-álgebras

7.1 Functores polinomiales

Nuestro objetivo va a ser describir una cierta clase de endofunctores que pueden ser descritos por producto y coproductos. Por esto, supongamos que estamos en una categoría $\mathcal{C}$ con productos y coproductos finitos. Recordemos que esto implica, en particular que existen objetos iniciales y finales.

Definición 7.1. *Un* functor polinomial *es un miembro de la menor colección de functores que:*

- *Contiene el functor identidad* Id.
- *Contiene todo functor constante Δ_A, que asocia todo elemento al objeto A.*
- *Es cerrada bajo composición, producto y coproducto, donde el producto de dos functores se corresponde con el functor:*

$$(F \times G)(X) = F(X) \times G(X)$$

y el coproducto se corresponde con:

$$(F + G)(X) = F(X) + G(X)$$

Estos tipos de functores se expresan como un polinomio. Por ejemplo, $F(X) = 1 + A \times X$ se debe entender como el functor que asocia a todo objeto X el coproducto del producto de A y X con el objeto terminal 1.

Consideremos el functor $F(X) = 1 + X$ y una F-álgebra (A, f) donde $f\colon F(A) \to A$. Por la definición de coproducto, este morfismo f puede ser identificado con alguno de dos morfismos $f_1\colon 1 \to U$ y $f_2\colon A \to A$. Viendo $F(X)$ como un tipo parametrizado por X, tiene sentido considerar f_1 y f_2 los *constructores* de $F(X)$.

Recordemos que en Haskell, el producto de dos tipos es su combinación en un par. El objeto terminal se corresponde con `()` y el objeto inicial se define como

```
data Empty
```

Es decir, un tipo sin constructor. En este contexto, podemos identificar $F(X)$ con el functor `Maybe a` y los morfismos f_1 y f_2 con `Nothing` y `Just`.

7.2 F-álgebras

Definición 7.2. *Sea $\mathcal{C}$ una categoría y $F\colon \mathcal{C} \to \mathcal{C}$ un endofunctor. Una F-álgebra es un par (A, α), donde $A \in \mathcal{C}$ y $\alpha\colon Fa \to a$ es un morfismo de $\mathcal{C}$.*

En algunos libros, también se le llama *punto prefijo*. Aunque no usaremos el nombre de punto prefijo, sí diremos que si α es isomorfismo, al par (A, α) par se le llama también *punto fijo*, señalando su similitud con la idea de punto fijo habitual de análisis.

Puede ser interesante darle una estructura de categoría a la colección de F-álgebras. Para ello, necesitamos definir morfismos de un (A, α) a (B, β). Suponiendo que tenemos un morfismo $f\colon A \to B$, observamos que $Ff\colon FA \to FB$, que puede ser compuesta con β:

$$\beta \circ Ff\colon FA \to B$$

Por otro lado:

$$f \circ \alpha\colon FA \to B$$

La igualdad de estos dos morfismos nos dará la fundación de la categoría de F-álgebras. Es más, como estos morfismos preservan bien la estructura de F-álgebras, les daremos el nombre (honorífico) de F-homomorfismo:

Proposición 7.1. Sea $\mathcal{C}$ una categoría y F un endomorfismo en $\mathcal{C}$. Sea Alg_F la colección de F-álgebras con homomorfismos $f\colon (A, \alpha) \to (B, \beta)$ tal que:

- $f\colon A \to B$ es un morfismo de $\mathcal{C}$.

- El siguiente diagrama conmuta

$$\begin{array}{ccc} FA & \xrightarrow{\alpha} & A \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle f} \\ FB & \xrightarrow{\beta} & B \end{array}$$

Entonces Alg_F forma una categoría.

Demostración. Basta ver que la composición está bien definida, pues las demás condiciones se cumplen trivialmente. Sean $f\colon (A,\alpha) \to (B,\beta)$ y $g\colon (B,\beta) \to (C,\gamma)$ homomorfismos de F-álgebras. Entonces cada cuadrado del siguiente diagrama conmuta:

$$\begin{array}{ccc} FA & \xrightarrow{\alpha} & A \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle f} \\ FB & \xrightarrow{\beta} & B \\ \downarrow{\scriptstyle Fg} & & \downarrow{\scriptstyle g} \\ FC & \xrightarrow{\gamma} & C \end{array}$$

Tenemos que:

$$g \circ f \circ \alpha = g \circ \beta \circ Ff = \gamma \circ Fg \circ Ff$$

Como $Fg \circ Ff = F(g \circ f)$, tenemos que el siguiente diagrama conmuta:

$$\begin{array}{ccc} FA & \xrightarrow{\alpha} & A \\ \downarrow{\scriptstyle F(g\circ f)} & & \downarrow{\scriptstyle g\circ f} \\ FC & \xrightarrow{\gamma} & C \end{array}$$

Luego $g \circ f$ es un homomorfismo de F-álgebras. ∎

Una vez hemos determinado una nueva categoría, podemos empezar a buscar qué aspecto tienen nuestra construcciones universales en esta nueva categoría. Resultará de particular interés el objeto inicial de una Alg_F por el siguiente lema debido a Lambek:

Lema 7.1. Sea $F\colon \mathcal{C} \to \mathcal{C}$ un endofunctor. Si (A,α) es una F-álgebra inicial de Alg_F, entonces (A,α) es un punto fijo de F.

Demostración. Sea (A,α) un objeto inicial en Alg_F. Para decir que (A,α) es un punto fijo, debemos ver que α es un isomorfismo. Para ello consideramos

el F-álgebra $(FA, F\alpha)$. Como (A, α) es inicial, existe un único homomorfismo $u\colon (A, \alpha) \to (FA, F\alpha)$.

Por un lado, $\alpha \circ u\colon (A, \alpha) \to (A, \alpha)$, pero el único homomorfismo de un objeto inicial a sí mismo es el morfismo identidad, luego $\alpha \circ u = \mathrm{id}_{(A,\alpha)}$. Por definición de homomorfismo, el siguiente diagram conmuta:

$$\begin{array}{ccc} FA & \xrightarrow{\alpha} & A \\ \downarrow{\scriptstyle Fu} & & \downarrow{\scriptstyle u} \\ F(FA) & \xrightarrow{F\alpha} & FA \end{array}$$

Entonces:

$$u \circ \alpha = F\alpha \circ Fu = F(\alpha \circ u) = F(\mathrm{id}_{(A,\alpha)}) = \mathrm{id}_{(FA,F\alpha)}$$

Luego α tiene inversa y debe ser isomorfismo. ∎

Dada una F-álgebra inicial (A, α), para toda otra F-álgebra (B, β), el único homomorfismo de (A, α) a (B, β) recibirá el imponente nombre de *catamorfismo*. Por ejemplo, el homomorfismo u de la demostración era un catamorfismo a $(FA, F\alpha)$.

7.3 Catamorfismos en Haskell

Debemos empezar implementando álgebras en Haskell:

```
type Algebra f a = f a -> a
```

Hay que tener en cuenta que no estamos imponiendo que `f` sea un functor, ya que Haskell no permite imponer restricciones a constructores de datos sin recurrir a GADTs[1].

Lo siguiente que necesitamos es implementar el punto fijo de `f`:

```
newtype Fix f = U (f (Fix f))
```

[1] `https://wiki.haskell.org/Data_declaration_with_constraint`

Como `Fix` tiene un único constructor, hay un isomorfismo entre `Fix f` y `f (Fix f)`. Este isomorfismo es `U`, y su inversa es:

```
unFix :: Fix f -> f (Fix f)
unFix (U f) = f
```

Ahora que sabemos que (`Fix f`, `U`) es el álgebra inicial, sabemos que existe un catamorfismo de este álgebra a todas las demás álgebras. Consideramos cualquier otra álgebra (`b`, `g`), con `g :: Algebra f b`. Sea `m :: Fix f -> b` el correspondiente catamorfismo, entonces por 7.1:

$$\begin{array}{ccc} f(Fixf) & \xrightarrow{fmap\ m} & fb \\ \downarrow{\scriptstyle U} & & \downarrow{\scriptstyle g} \\ Fixf & \xrightarrow{m} & b \end{array}$$

Podemos invertir `U` con `unFix`, luego la conmutatividad del diagrama es equivalente a que:

```
m = g . fmap m . unFix
```

Usamos esto para definir recursivamente el catamorfismo:

```
cata :: (Functor f) => (Algebra f b) -> (Fix f -> b)
cata g = g . fmap (cata g) . unFix
```

Veamos un ejemplo con el functor `Maybe`

Ejemplo 7.1. Especialicemos el código anterior para `Maybe`:

```
type AlgebraM a = Algebra Maybe a
type FixM = Fix Maybe
```

Por ahora, no dice mucho. Pero al añadir las siguientes de líneas obtenemos una perspectiva interesante:

```
type Nat = FixM

cero :: Nat
cero = U Nothing
suc :: Nat -> Nat
suc = U . Just
```

Es decir, los naturales pueden verse como el punto fijo del functor `Maybe`.

7.4 F-coálgebras en Haskell

Una *F-coálgebra* es, naturalmente, el dual de una F-álgebra. Más precisamente, es un par (A, α), donde $\alpha \colon A \to FA$. En Haskell:

```
type Coalgebra f a = a -> f a
```

Al igual que con las álgebras, las F-coálgebras forman una categoría $Coalg_F$. El lema de Lambek (7.1) en el contexto de las coálgebras dice así:

Lema 7.2. 7.1 Sea $F \colon \mathcal{C} \to \mathcal{C}$ un endofunctor. Si (A, α) es una F-coálgebra terminal de $Coalg_F$, entonces (A, α) es un punto fijo de F.

El único homomorfismo de una F-coálgebra a la F-coálgebra terminal se denomina *anamorfismo*. Como en el caso del catamorfismo, podemos definirlo recursivamente:

```
ana :: (Functor f) => (Coalgebra f b) -> (b -> Fix f)
ana g = U . fmap (ana g) . g
```

Ejemplo 7.2. Consideramos el functor $F(X) = 1 + A \times X$, que se puede ver como la composición de $G(X) = A \times X$ con $H(X) = 1 + X$. En términos de Haskell, la composición de `(a,)` con `Maybe`.

```
data F a x = Maybe (a,x)
```

Su punto fijo es un tipo de la forma $X = 1 + A \times X$, que se corresponde con el functor lista `[a]`.

Por lo tanto, en este contexto `ana` tiene la forma:

```
ana :: (x -> Maybe (a,x)) -> (x -> [a])
```

`ana` nos permite construir una lista (posiblemente infinita) a partir de una «semilla» `x` y una función que devuelva `a`, el valor generado, y `x`, una nueva semilla. Una implementación de este anamorfismo del paquete básico de Haskell es la función `unfoldr`.

7.5 Referencias

- Pierce, B.C. (1991). *A taste of category theory for computer scientist*, capítulo 3.4.
- Milewski, B. (2014). *Category Theory for Programmers*, capítulo 24.
- Lambek, J. (1968). A fixed point theorem for complete categories. *Mathematische Zeitschrift 103*, páginas 151–161.

Epílogo

A estas alturas hemos visto las piezas básicas con las que se construye la teoría de categorías y cómo aparecen en programación funcional. Sin embargo, la teoría de categoría es un campo mucho más grande que lo podemos abarcar en este trabajo. Aquí damos un punto de comienzo para seguir el estudio de teoría de categorías.

En el capítulo 4, hemos identificado qué propiedades debe cumplir una categoría para que sea equivalente a la lógica proposicional intuicionista. Añadiendo más propiedades a las categorías podemos obtener distintas lógicas. Por ejemplo, podemos "mejorar" esta lógica a lógica intuicionista de orden superior. Para ello, necesitaremos definir los *clasificador de subobjetos*. En pocas palabras, un subobjeto es cualquier objeto S con un monomorfismo $S \to X$. Un clasificador de subobjetos de X es una colección de morfismos que caracterizan cada uno de sus subobjetos. En pocas palabras, es una generalización de la idea de función característica de los conjuntos:

$$\chi_S \colon X \to \{0, 1\}$$
$$\chi_S(x) = \begin{cases} 0 & \text{si } x \notin S \\ 1 & \text{si } x \in S \end{cases}$$

Una categoría cerrada cartesianamente con un clasificador de subobjetos se denomina *topos elemental*. Resulta que un *topos elemental* es equivalente a la lógica intuicionista de orden superior.

En los capítulos 5, hemos introducido el concepto de monoide, pero también el concepto de categoría enriquecida y 2-categoría. Siguiendo esta idea llegamos a la teoría de categorías de orden superior. El libro de Lurie sirve de introducción a este tema. Recordemos que en una 2-categoría, tenemos 1-morfismos entre 0-morfismos (objetos) y 2-morfismos entre 1-morfismos. Con

esto en mente, definimos las ∞-categorías describiendo los k-morfismos entre $k-1$-morfismos. Si además establecemos un particular concepto de equivalencia entre k-morfismos basado en nuestra descripción de equivalencias en la sección 2.8, se introduce la idea de $(\infty, 1)$-categorías, donde todos los k-morfismos son equivalentes para $k > 1$. El estudio de estas $(\infty, 1)$-categorías y su relación con la teoría de homotopía de tipos es un campo muy activo en los últimos años. Véase el libro colaborativo

Bibliografía

- AWODEY, STEVE, *Category Theory*, Oxford University Press, 2006.
- BUURLAGE, JAN-WILLEM, *Lecture Notes on Categories and Haskell*, 2017. Disponible en `https://jwbuurlage.github.io/`.
- LAMBEK, JOACHIM y SCOTT, PHILIP J., *Introduction to Higher Order Categorical Logic*, Cambridge University Press, 1986.
- LIPOVAČA, MIRAN, *Learn You a Haskell for Great Good!: A Beginner's Guide*, No Starch Press (2011). Disponible en `http://learnyouahaskell.com/`.
- MAC LANE, SAUNDER, *Categories for the Working Mathematician*, Springer, 2ª edición, 1998.
- MILEWSKI, BARTOSZ, *Category Theory for Programmers*, 2018. Disponible en `https://github.com/hmemcpy/milewski-ctfp-pdf`.
- PIERCE, BENJAMIN C., *A taste of category theory for computer scientist*, Carnegie Mellon University, 1988.
- RIEHL, EMILY, *Categorical homotopy theory*, Cambridge University Press, 2014.
- RIEHL, EMILY, *Category Theory in Context*, Cambridge University Press, 2016.
- THOMPSON, SIMON, *Type theory & functional programming*, Addison-Wesley, 1991.

Índice alfabético