Una implementación en Coq de conceptos categóricos fundamentales

Daniel F. Osorio

Pontificia Universidad Javeriana Cali

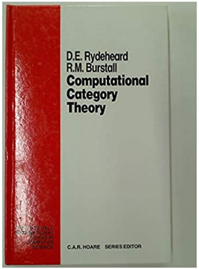
Semillero de lógica computacional y teoría de categorías

Noviembre 11, 2020

Contenido

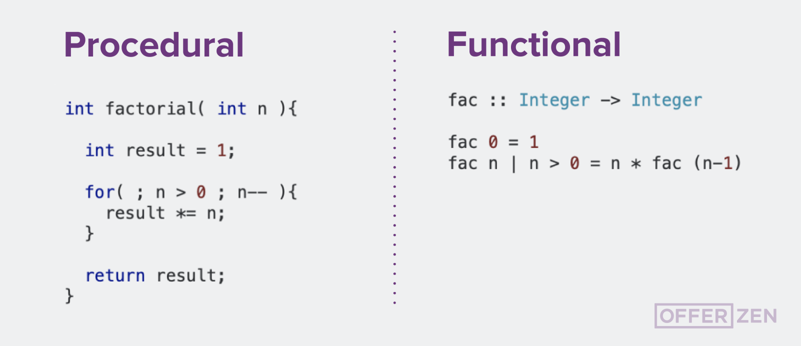
Introducción

Este proyecto se basa en el libro *Computational Category Theory* (Burstall y Rydeheard, 1988), en el cual se muestra una implementación de la teoría de categorías en el lenguaje de programación funcional ML. La idea del proyecto es utilizar el asistente de pruebas y lenguaje funcional Coq, para implementar algunas de las estructuras de la teoría de categorías propuestas en este libro.



Coq y programación funcional

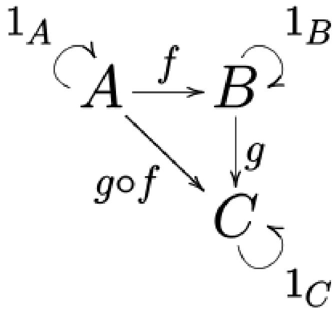
La programación funcional es un paradigma de programación declarativa que se enfoca en el uso de funciones matemáticas. El objetivo es conseguir lenguajes expresivos y matemáticos. Ejemplo de estos lenguajes son Haskell, Scala y Coq.



Coq y programación funcional

Coq a parte de ser un lenguaje de programación funcional, también es un asistente de pruebas, lo que permite formalizar pruebas lógicas y matemáticas. En el caso de este proyecto, el uso de Coq se enfoca en la formalización matemática de la teoría de categorías.

La teoría de categorías nos permite obtener una visión general de las matemáticas, a partir de la abstracción y formalización de estructuras matemáticas, y estableciendo relaciones entre ellas. (También es utilizada en otras áreas como la computación, física o economía)



Categoría matemáticamente

Una categoría C consiste de:

- Una colección ob(C) de **objetos**.
- Por cada $a, b \in$ ob(C), una colección $C(a, b)$ de **mapeos** o **flechas** o **morfismos** de a a b.
- Por cada $a, b, c \in$ ob(C) y flechas $f : a \to b \in C(a, b)$, $g : b \to c \in C(b, c)$, una flecha $g \circ f : a \to c \in C(a, c)$ llamada **composición**.
- Por cada $a \in ob(C)$, una flecha $1_a : a \to a \in C(a, a)$, llamada **identidad** de a.
- **Asociatividad:** sea $f : a \to b \in C(a, b)$, $g : b \to c \in C(b, c)$, $h : c \to d \in C(c, d)$ entonces $(h \circ g) \circ f = h \circ (g \circ f)$.
- **Ley de identidad:** Por cada $f : a \to b \in C(a, b)$ se cumple que $f \circ 1_a = f = 1_b \circ f$

Categoría computacionalmente

```
Inductive category {O A: Type}: Type :=
  |cat (source: A -> O) (target: A -> O) (id: O -> A)
       (comp: A -> A -> A).
Notation "( x , y , z , w )" := (cat x y z w).
```

Objetos en **FinSet**:

```
Definition finEmptySet {X: Type} := @nil X.
Definition finSet {X: Type} := @list X.
```

Flechas en **FinSet**:

```
Inductive finSetArrow {X: Type}: Type :=
  |setArr (set1: @finSet X) (f: X -> X) (set2: @finSet X).
```

Categoría FinSet computacionalmente

Objetos de salida en **FinSet**:

```
Definition finSetSource {X: Type} (s: @finSetArrow X):
                                              @finSet X:=
  match s with
  | setArr (a) (f) (c) => a
  end.
```

Objetos de llegada en **FinSet**:

```
Definition finSetTarget {X: Type} (s: @finSetArrow X):
                                              @finSet X:=
  match s with
  | setArr (a) (f) (c) => c
  end.
```

Categoría FinSet computacionalmente

Identidad en **FinSet**:

```
Definition finSetIdentity {X: Type} (a: @finSet X):
                                            @finSetArrow X :=
setArr (a) (fun x => x) (a).
```

Funcion que verifica si dos flechas pueden ser compuestas en **FinSet**:

```
Definition composable {X: Type} (a1: @finSetArrow X)
(a2: @finSetArrow X) (pred: X -> X -> bool): bool :=
  match a1, a2 with
  |setArr (c) (g) (d), setArr (a) (f) (b) =>
  if finSetEq (b) (c) (pred) then true else false
  end.
```

Categoría FinSet computacionalmente

Composición en **FinSet**:

```
Definition finSetComposition' {X: Type} (a1: @finSetArrow X)
(a2: @finSetArrow X): @finSetArrow X :=
  match a1, a2 with
  |setArr (c) (g) (d), setArr (a) (f) (b) =>
  setArr (a) (fun x => g (f (x))) (d)
  end.
```

Categoría **FinSet**:

```
Definition FinSetCat {X: Type} :=
(@finSetSource X, @finSetTarget X, @finSetIdentity X,
                                   @finSetComposition' X).
```

Categoría finita

Se quiere representar la siguiente categoría finita:

$$\begin{tikzcd}
 & a \arrow[dr, "g"'] \arrow[dr, bend left, "h"] & \\
b \arrow[ur, "f"] \arrow[rr, "k"'] & & c
\end{tikzcd}$$

Categoría finita computacionalmente

Objetos en **finiteCat**:

```
Inductive objects: Type :=
   |a
   |b
   |c
   |none. (*para manejo de errores*)
```

flechas en **finiteCat**:

```
Inductive arrows: Type :=
   |f
   |g
   |h
   |k
   |id (obj: objects)
   |noComp. (*para manejo de errores*)
```

Categoría finita computacionalmente

Objetos de salida en **finiteCat**:

```
Definition s (arrow: arrows): objects :=
  match arrow with
  |f => b
  |g => a
  |h => a
  |k => b
  |id x => x
  |noComp => none
  end.
```

Objetos de llegada en **finiteCat**:

```
Definition t (arrow: arrows): objects :=
  match arrow with
   |f => a
   |g => c
   |h => c
   |k => c
   |id x => x
   |noComp => none
  end.
```

Categoría finita computacionalmente

identidad en **finiteCat**:

```
Definition ident := fun x => id (x).
```

Composición en **finiteCat**:

```
Definition comp (a1 a2: arrows): arrows :=
  match a1, a2 with
   | id x, u => if eqObj (x) (t u) then u else noComp
   (*t(u) debe ser igual a x*)
   | u, id x => if eqObj (x) (s u) then u else noComp
   (*s(u) debe ser igual a x*)
   | g, f => k
   | h, f => k
   | _, _ => noComp
  end
```

Categoría finita computacionalmente

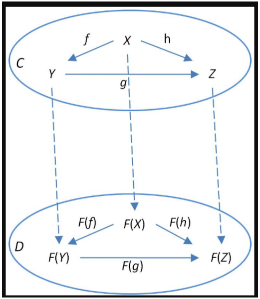
Categoria **finiteCat**:

```
Definition FinCat := (s,t,ident,comp).
```

Functores matemáticamente

Sean C_1, C_2 categorías, un **functor** $F : C_1 \to C_2$ consiste de:

- una función $\mathrm{ob}(C_1) \to \mathrm{ob}(C_2)$ que mapea los objetos de C_1 a C_2.
- para cada $a, a' \in C_1$ una función $C_1(a, a') \to C_2(F(a), F(a'))$ que mapea las flechas de C_1 a flechas de C_2.



Functores computacionalmente

```
Inductive functor {oA aA oB aB: Type}: Type :=
|func (catA: @category oA aA) (oFun: oA -> oB)
 (aFun: aA -> aB) (catB: @category oB aB).
```

Functor $F : a \to a \times a$ matemáticamente

Sea a un conjunto finito y $f : a \to a$. El functor $F : a \to a \times a$ es de la forma:

- Una función $f_1 : a \to a \times a$ que convierte a a en su producto cartesiano
- Una función $f_2 : f \to (f, f)$ que convierte la función f, en la pareja de funciones (f, f).

Definición de **producto cartesiano**

```
Fixpoint aux {X Y: Type} (l1: @list X) (l2: @list Y):=
  match l1, l2 with
  |h1::t1, h2::t2 => (h1 , h2)::(aux (l1) (t2))
  |h1::t1, nil => nil
  |nil, _ => nil
  end.

Fixpoint cartesianProduct {X Y: Type} (l1: @finSet X)
                                      (l2: @finSet Y) :=
  match l1, l2 with
  |h1::t1, h2::t2 => app (aux l1 l2) (cartesianProduct t1 l2)
  |nil, _ => nil
  |_, nil => nil
  end.
```

Definición de la pareja de funciones:

```
Definition pairArrow {X Y: Type} (f: X -> X)
             (g: Y -> Y) (pareja: @prod X Y): @prod X Y :=
  match pareja with
  |(x , y) => ( (f x) , (g y) )
  end.
```

función f_2:

```
Definition prodArrow {X Y: Type} (s1: @finSetArrow X)
(s2: @finSetArrow Y) : finSetArrow :=
  match s1, s2 with
  |setArr A f B, setArr C g D =>
                       setArr (cartesianProduct A C)
                       (pairArrow f g)
                       (cartesianProduct B D)
  end.
```

Functor $F : a \to a \times a$ computacionalmente

Definición del functor **functAtoAXA**:

```
Definition functAtoAXA {X: Type} :=
                            func (@FinSetCat X)
                            (fun A => cartesianProduct A A)
                            (fun f => prodArrow f f)
                            (@FinSetCat (@prod X X)).
```

Coproducto matemáticamente

Un **coproducto** de objetos a, b en una categoría es un objeto $a + b$ junto con flechas $f : a \to a + b$ y $g : b \to a + b$ tal que para cualquier objeto c' y flechas $f : a \to c'$ y $g : b \to c'$ hay una única flecha : $a + b \to c'$ tal que el siguiente diagrama conmute:

$$
\begin{tikzcd}
a \arrow[r, "f"] \arrow[dr, "f'"'] & a+b \arrow[d, "u"] & b \arrow[l, "g"'] \arrow[dl, "g'"] \\
& c' &
\end{tikzcd}
$$

Coproducto computacionalmente

```
Definition coproductCoCone {X Y: Type}: Type :=
                        (X * Y * Y) * (X * Y * Y -> Y).

Definition coproduct {X Y: Type}: Type :=
                        (X * X) -> @coproductCoCone X Y.
```

Coproducto FinSet computacionalmente

Definición de **unión disjunta** en FinSet:

```
Inductive Tag {X: Type} : Type :=
  |just (x: X)
  |setA (x: @Tag X)
  |setB (x: @Tag X).

Definition disjointU {X: Type} (a b: @finSet (@Tag X)) :=
                          app (map setA a) (map setB b).
```

Definición de la función **[f,g]**:

```
Definition fg {X: Type} (f g: @Tag X -> @Tag X)
                            (tag: @Tag X): @Tag X  :=
  match tag with
  |setA x => f (x)
  |setB x => g (x)
  |just x => tag
  end.
```

Definición de la función **u**:

```
Definition univ {X: Type} (a b c: @finSet (@Tag X))
                        (s1 s2: @finSetArrow (@Tag X)) :=
  match s1, s2 with
  |setArr _ f _ , setArr _ g _ =>
   setArr (disjointU a b) (fg f g) (c)
  end.
```

Coproducto FinSet computacionalmente

Coproducto en FinSet:

```
Definition finSetCoProduct {X: Type} (a b: @finSet(@Tag X)):=
pair ( trip (disjointU a b)
            (setArr a setA (disjointU a b))
            (setArr b setB (disjointU a b)) )
     (univ a b).
```

Consideraciones

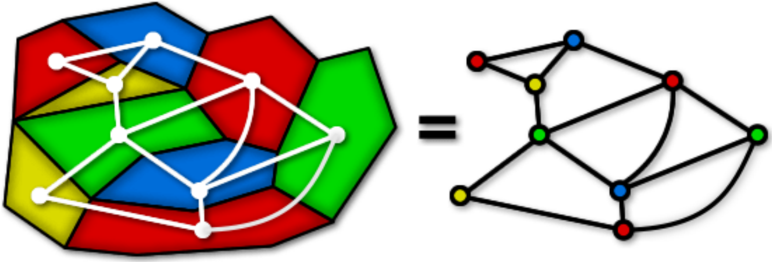
Algunas consideraciones sobre el proyecto realizado:

- En este proyecto solo se implementaron las estructuras para representar algunas construcciones categóricas. Sin embargo las pruebas sobre los axiomas de estas estructuras no están establecidos.
- Alguna información es perdida al hacer la traducción de matemáticas a los diferentes tipos, como lo son algunos de los axiomas relacionados con funciones (axiomas para asociatividad, identidad o composición). Sin embargo, existen muchas otras implementaciones de categorías en Coq que si contienen estos axiomas dentro de las definiciones de los tipos. En el caso de este proyecto, es trabajo del programador valorar si las estructuras utilizadas cumplen los axiomas.

Conclusiones y reflexiones

Bajo estas estructuras categóricas en Coq se puede pensar en futuros usos que ahondan en otras áreas como:

- Pruebas matemáticas por computador
- Razonamiento automatizado y "automated theorem proving"
- Teoría de pruebas



Bibliografía

1. D.E. Rydeheard & R.M. Burstall. (1988). *Computational Category Theory*. Prentice Hall
2. T. Leinster. (2014). *Basic Category Theory*. Cambridge University Press
3. `https://github.com/frafle/Categorias-en-Coq`