Starting Category Theory In Idris

Richard Southwell

December 26, 2020

The aim of this document is to explain the very basics of Statebox's Idris code for doing category theory. The code here belongs to Statebox [Statebox]. The required category theory can be learned by reading [Milewski]. The required type theory can be learned by reading the first chapter of [HoTT]. The Idris code discussed in this document can be found here [github], although it was copied from [Statebox].

I am not part of Statebox. Misunderstandings are my own.

1 Encoding Categories

The Idris code

```
record Category where
    constructor MkCategory
    obj           : Type
    mor           : obj -> obj -> Type
    identity      : (a : obj) -> mor a a
    compose       : (a, b, c : obj)
                 -> (f : mor a b)
                 -> (g : mor b c)
                 -> mor a c
    leftIdentity  : (a, b : obj)
                 -> (f : mor a b)
                 -> compose a a b (identity a) f = f
    rightIdentity : (a, b : obj)
                 -> (f : mor a b)
                 -> compose a b b f (identity b) = f
    associativity : (a, b, c, d : obj)
                 -> (f : mor a b)
                 -> (g : mor b c)
                 -> (h : mor c d)
                 -> compose a b d f (compose b c d g h) = compose a c d (compose a b c f g) h
```

defines the type of categories.

.

The type of categories can be expressed mathematically as

$$
\begin{aligned}
\text{Category} \quad = \quad & \sum_{\text{obj}:\mathbb{U}} \quad \sum_{\text{mor}:\text{obj}\to\text{obj}\to\,\mathbb{U}} \quad \sum_{\text{identity}:\Pi_{a:\text{obj}}\text{mor(a,a)}} \\
& \sum_{\text{compose}:\Pi_{a,b,c:\text{obj}}\Pi_{f:\text{mor}(a,b)}\Pi_{g:\text{mor}(b,c)}\text{mor}(a,c)} \\
& \sum_{\text{leftIdentity}:\Pi_{a,b:\text{obj}}\Pi_{f:\text{mor}(a,b)}(\text{compose}(a,a,b,\text{identity}(a),f)=f)} \\
& \sum_{\text{rightIdentity}:\Pi_{a,b:\text{obj}}\Pi_{f:\text{mor}(a,b)}(\text{compose}(a,b,b,f,\text{identity}(b))=f)} \\
& \prod_{a,b,c,d:\text{obj}} \quad \prod_{f:\text{mor(a,b)}} \quad \prod_{g:\text{mor(b,c)}} \\
& \prod_{h:\text{mor(c,d)}} \text{compose}(a,b,d,f,\text{compose}(b,c,d,g,h)) \\
& = \text{compose}(a,c,d(\text{compose}(a,b,c,f,g),h)
\end{aligned}
$$

In other words, a category consists of:

1. A type obj of objects.
2. A type $\text{mor}(a,b)$ of morphisms/arrows from a to b, for each pair of objects a,b.
3. For each object a an identity arrow $\text{identity}(a)$.
4. For each triple a,b,c of objects and any morphism f from a to b, and any morphism g from b to c we have a morphism $[f \quad \text{before} \quad g]$ from a to c.
5. For each pair of objects a,b and each morphism f from a to b we have a proof that composing the identity arrow of a before f equals f.
6. For each pair of objects a,b and each morphism f from a to b we have a proof that composing f before the identity arrow of b equals f.
7. For any arrows $a \xrightarrow{f} b$ and $b \xrightarrow{g} c$ and $c \xrightarrow{h} d$ we have a proof that $[f \quad \text{before} \quad [g \quad \text{before} \quad h]]$ equals $[[f \quad \text{before} \quad g] \quad \text{before} \quad h]$.

2 Encoding Discrete Categories

To illustraite the above encoding of a category in Idris, we wish to define a function called discreteCategory which sends a type a to the discrete category (the category which has elements of a as objects, and which has no arrows except identity arrows). To do this, one can start with the code:

```
DiscreteMorphism : (x, y : a) -> Type
DiscreteMorphism x y = (x = y)
```

where it is implicit that a is a type. In fact, in our implementation, a will be the type obj of objects of our discrete category, and DiscreteMorphism will define the arrows. Here

DiscreteMorphism : $a \to a \to \mathbb{U}$ and for $x, y : a$ we have that DiscreteMorphism(x, y) is the identity type $x =_a y$ (which is occupied (with a single occupant) iff x is identical to y).

The identity arrows of our discrete category are described by

```
discreteIdentity : (x : a) -> DiscreteMorphism x x
discreteIdentity _ = Refl
```

which sends each object x in the type a (of objects) to the identity arrow $\text{Refl}_x : (x =_a x)$.

Arrow composition is described by the code:

```
discreteCompose : (x, y, z : a) -> DiscreteMorphism x y -> DiscreteMorphism y z -> DiscreteMorphism x z
discreteCompose _ _ _ Refl Refl = Refl
```

which says that identity arrows composed with identity arrows always give identity arrows (there are no other cases to consider in discrete categories).

In a similar way left and right identities, and the associativity proof are defined on identity arrows by referring to Refl, in the following code:

```
discreteLeftIdentity : (x, y : a) -> (f : DiscreteMorphism x y) -> discreteCompose x x y (discreteIdentity x) f = f
discreteLeftIdentity _ _ Refl = Refl

discreteRightIdentity : (x, y : a) -> (f : DiscreteMorphism x y) -> discreteCompose x y y f (discreteIdentity y) = f
discreteRightIdentity _ _ Refl = Refl

discreteAssociativity : (w, x, y, z : a)
                      -> (f : DiscreteMorphism w x)
                      -> (g : DiscreteMorphism x y)
                      -> (h : DiscreteMorphism y z)
                      -> discreteCompose w x z f (discreteCompose x y z g h)
                       = discreteCompose w y z (discreteCompose w x y f g) h
discreteAssociativity _ _ _ _ Refl Refl Refl = Refl
```

This is all put together by the code:

```
discreteCategory : (a : Type) -> Category
discreteCategory a = MkCategory
  a
  DiscreteMorphism
  discreteIdentity
  discreteCompose
  discreteLeftIdentity
  discreteRightIdentity
  discreteAssociativity
```

Essentially discreteCategory sends a type a to the discrete category with objects corresponding to members of a. In particular, type a is sent to

$$
\begin{aligned}
&(a, \text{DiscreteMorphism}, \text{discreteIdentity}, \text{discreteCompose}, \\
&(\text{discreteLeftIdentity}, \text{discreteRightIdentity}, \text{discreteAssociativity}) \quad : \text{Category}
\end{aligned}
$$

For example, the following code generates the discrete category on the type of booleans, and points out the identity arrow of the object corresponding to True:

```
EndomorphismsOfTrue : Type
EndomorphismsOfTrue = mor MyFirstCategory MyTrue MyTrue

MyFirstArrow : EndomorphismsOfTrue
MyFirstArrow = Refl
```

References

[Statebox] statebox idris ct

[Milewski] Milewski, Bartosz. Category theory for programmers. Blurb, 2018.

[HoTT] Program, The Univalent Foundations. "Homotopy Type Theory: Univalent Foundations of Mathematics." arXiv preprint arXiv:1308.0729 (2013).

[github] my idris ct code