Category Theory in Coq 8.5

Amin Timany Bart Jacobs

iMinds-DistriNet – KU Leuven
firstname.lastname@cs.kuleuven.be

We report on our experience implementing category theory in Coq 8.5 [1]. The repository of this development can be found at `https://bitbucket.org/amintimany/categories/`. This implementation most notably makes use of features *primitive projections* for records and *universe polymorphism* that are new to Coq 8.5. The latter provides relative smallness and largeness in the development. This will be elaborated below. The former allows for specification of well-behaved dualities in the category theoretical sense. That is, we get definitional equalities such as[2]:

$$(C^{op})^{op} = C \qquad (F^{op})^{op} = F \qquad (N^{op})^{op} = N$$

$$(F \circ F')^{op} = F^{op} \circ F'^{op}$$

Where C is a category, F and F' are functors and N is a natural transformation.

In this development the category `Type_Cat` plays the role of category of sets **Set**. This is the category of types in Coq as objects and functions among them as arrows. In the sequel, we will simply use **Set** to denote this category.

The following is the list of the most important notions and features of this development.

- basic constructions:
  - terminal/initial object
  - products/sums
  - equalizers/coequalizers
  - pullbacks/pushouts
  - exponentials
  - $+ \dashv \Delta \dashv \times$ and $(- \times a) \dashv a^{-}$
- external constructions:
  - comma categories
  - product category
- for **Cat**: (`Obj` := `Category`, `Hom` := `Functor`)
  - cartesian closure
  - initial object
- for **Set**: (`Obj` := `Type`, `Hom` := `fun A B ⇒ A → B`)
  - initial object
  - sums
  - equalizers
  - coequalizers†
  - pullbacks
  - cartesian closure
  - local cartesian closure†
  - completeness
  - co-completeness†
  - sub-object classifier (`Prop : Type`)†
  - topos†
- the Yoneda lemma
- adjunction
  - hom-functor adjunction, unit-counit adjunction, universal morphism adjunction and their conversions
  - duality : $F \dashv G \Rightarrow G^{op} \dashv F^{op}$
  - uniqueness up to natural isomorphism
- kan extensions
  - global definition
  - local definition with both hom-functor and cones (along a functor)
  - uniqueness
  - preservation by adjoint functors
  - pointwise kan extensions (preserved by representable functors)
- (co)limits
  - as (left)right local kan extensions along the unique functor to the terminal category
  - (sum)product-(co)equalizer (co)limits
  - pointwise (as kan extensions)
- $T-(co)algebras$ (for an endofunctor T)

  †uses the axioms of propositional extensionality and constructive indefinite description (axiom of choice).

In addition, we have used the axiom of functional extensionality and proof-irrelevance. The axiom of proof-irrelevance is mostly used in proofs of equality of arrows, e.g., to prove two functors are equal, one just needs to prove the object- and arrow-maps are equal.

Even though (co)limits are defined in general, we have defined most important and useful (co)limits separately: terminal object, products, equalizers and pullbacks. The duals of these notions, i.e., initial object, sums, coequalizers and pushouts, respectively, are simpliy defined as their counterparts in the opposite category. Similarly, only the right local kan extensions (in both versions) are defined directly and local left can extensions are simply assumed as local right kan extensions with opposite functors.

Although dualities behave nicely (in the aforemen-

[1] The last tested version is Coq 8.5-beta1 but we intend to keep compatibility at least up to the official release of Coq 8.5

[2] This was achieved similarly to the implementation of category theory on top of Coq/HoTT [2]

tioned sense), working with dual definitions is not always as smooth. This is especially evident in rewriting equalities. In some cases one has to add the equality to the proof context (usually applied to the arguments that are difficult to match) and perform a simplification on them before they can be used with the `rewrite` tactic. In some rare extreme cases, simplifications with tactics like `cbn` and `simpl` are not enough and one has to change the goal in such a way that those lemmas can be used with, e.g., the `apply` tactic, instead of `rewrite`.

Universe levels, smallness and largeness

In this implementation, we use universe levels as the underlying notion of smallness/largeness. In other words, each category has universe levels that indicate its relative smallness/largeness. In practice, the type of categories has two universe level parameters, `Category@{i,j} : Type@{max(i+1, j+1)}`, where `i` is the level of the type of objects and `j` is the level of the type of arrows. This relative notion of smallness/largeness works so well, in fact, that we can prove the following theorem in our implementation:

```
Theorem Complete_Preorder (C : Category) (CC : Complete C) :
  forall x y : (Obj C), Hom x y' ≃ ((Arrow C) → Hom x y)
```

where `y'` is the limit of the constant functor from the discrete category `(Arrow C)` that maps every object to `y` and `(Arrow C)` is the type of all homomorphisms of category `C`. This though, would result in a contradiction as soon as we have two objects `c` and `d` in `C` for which `Hom c d` has more than one element. That is, we have effectively shown that *any* complete category is a preorder category, i.e., between any two objects there is at most one arrow. This is indeed absurd as the category **Set** is complete and there are types in Coq that have more than one function between them! However, this theorem holds for small categories. That is, any *small* and complete category is a preorder category[3]. Expectedly, the restrictions on the universe levels of this theorem do indeed confirm this fact. That is, this theorem is in fact only applicable to categories in which the level of the type of objects is less than or equal to the level of the type of arrows. Hence, Coq will refuse to apply this theorem to the category **Set** with a universe inconsistency error as for it the level of the type of arrows is strictly less than the level of the type of objects.

On the other hand, the universe polymorphism of Coq treats inductive types by considering copies of them at different levels. See [3]. That implies that if we have `C : Category@{i, j}` and we additionally have that `C : Category@{i', j'}`, Coq enforces `i = i'` and `j = j'`. In this setting, the category of (relatively small) categories **Cat**, which in the implementation has type

```
Cat@{i, j, k, l} : Category@{i, j}
```

is *not* the category of all smaller categories. Rather it is the category of categories that are at level `k` and `l` and not any lower level.

Apart from the fact that **Cat** defined this way is not the category of all relatively small categories, these restrictions on universe levels impose practical restrictions as well. For instance, looking at the fact that `Cat@{i, j, k, l}` has exponentials (functor categories), we can see the restriction that `j = k = l`. That is only those copies have exponentials for which this restriction holds. Looking back at the category of types, **Set**, we had the restriction that the level of the type of arrows is strictly less than that of objects. This means, there is no version of **Cat** that both has exponentials and a version of **Set** in its objects.

This means that we can't simply assume that **Cat** has exponentials and get the exponential transpose of the hom-functor to be the Yoneda embedding[4].

Moreover, we can use the Yoneda lemma to show that in any cartesian closed category, for any objects a, b and c:

$$\left(a^b\right)^c \simeq a^{b\times c}$$

Yet, this theorem can't be applied to **Cat**, even though it holds for **Cat**.

On the other hand, if we show that **Set** has the type `unit` as its terminal object, we, strangely, get the restriction that the level of the type of arrows of **Set** is universe `Set` but, expectedly, not for objects. A similar problem happens with showing that the category whose object type and arrow type are `unit` is the terminal object of **Cat**. It is not clear to the authors wether this is intensional or the result of a bug. In any case, we have elected to go around this problem by postulating existence of a universe polymorphic type that has a single inhabitant:

```
Parameter UNIT : Type.
Parameter TT : UNIT.
Axiom UNIT_SINGLETON : forall x y : UNIT, x = y.
```

References

[1] Steve Awodey. *Category theory.* Oxford University Press, 2010.

[2] Jason Gross, Adam Chlipala, and David I. Spivak. Experience implementing a performant category-theory library in coq. In *Interactive Theorem Proving - 5th International Conference, ITP 2014. Proceedings*, pages 275–291, July 2014.

[3] Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in coq. In *Interactive Theorem Proving, ITP 2014, Proceedings*, pages 499–514, July 2014.

[3] This theorem and its proof are taken from [1].

[4] However, using the definition of currying defined *independently of the notion of exponentials* for functors can be used to this end and it is precisely how we have defined the Yoneda embedding.