Category Theory in Coq 8.5

Amin Timany[1] **and Bart Jacobs**[2]

1 **iMinds-DistriNet – KU Leuven**
Dept. of Computer Science, B-3001, Heverlee, Belgium
`amin.timany@cs.kuleuven.be`

2 **iMinds-DistriNet – KU Leuven**
Dept. of Computer Science, B-3001, Heverlee, Belgium
`bart.jacobs@cs.kuleuven.be`

Abstract

We report on our experience implementing category theory in Coq 8.5. Our work formalizes most of basic category theory, including concepts not covered by existing formalizations, in a library that is fit to be used as a general-purpose category-theoretical foundation.

Our development particularly takes advantage of two features new to Coq 8.5: primitive projections for records and universe polymorphism. Primitive projections allow for well-behaved dualities while universe polymorphism provides a relative notion of largeness and smallness. The latter is one of the main contributions of this paper. It pushes the limits of the new universe polymorphism and constraint inference algorithm of Coq 8.5.

In this paper we present in detail smallness and largeness in categories and the foundation they are built on top of. We furthermore explain how we have used the universe polymorphism of Coq 8.5 to represent smallness and largeness arguments by simply ignoring them and entrusting them to the universe inference algorithm of Coq 8.5. We also briefly discuss our experience throughout this implementation, discuss concepts formalized in this development and give a comparison with a few other developments of similar extent.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving, F.4.1 Mathematical Logic

Keywords and phrases Category Theory, Coq 8.5, Universe Polymorphism, Homotopy Type Theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.<article-no>

1 Introduction

A category [11, 2] consists of a collection of objects and for each pair of objects A and B a collection of morphisms (aka arrows or homomorphisms) from A to B. Moreover, for each object A we have a distinguished morphism $id_A : A \to A$. Morphisms are composable, i.e., given two morphisms $f : A \to B$ and $g : B \to C$, we can compose them to form: $g \circ f : A \to C$. Composition must satisfy the following additional conditions: $\forall f : A \to B.\ f \circ id_A = f = id_B \circ f$ and $\forall f, g, h.\ (h \circ g) \circ f = h \circ (g \circ f)$.

The notion of a category can be seen as a generalization of sets. In fact sets as objects together with functions as morphisms form the important category **Set**. On the other hand, it can be seen as a generalization of the mathematical concept of a preorder. In this regard, a category can be thought of as a preorder where objects form the elements of the preorder and morphisms from A to B can be thought of as "witnesses" of the fact that $A \preceq B$. Thus, identity morphisms are witnesses of reflexivity whereas composition of morphisms forms witnesses for transitivity and the additional axioms simply spell out coherence conditions for witnesses. Put concisely, categories are preorders where the quality and nature of the relation holding between two elements is important. In this light, categories are to preorders what intuitionistic logic is to classical logic. A combination of these two interpretations of categories can provide an essential and useful intuition for understanding most, if not all, of the theory.

This generality and abstractness is what led some mathematicians to call this mathematical theory "general abstract nonsense" in its early days. However category theory, starting from this simple yet vastly abstract and general definition, encompasses most mathematical concepts and has found applications not only in mathematics but also in other disciplines, e.g, computer science.

In computer science it has been used extensively, especially in the study of semantics of programming languages [15], in particular constructing the first (non-trivial) model of the untyped lambda calculus by Dana Scott [17], type systems [10], and program verification [5, 3, 4].

Given the applications of category theory and its fundamentality on the one hand and the arising trend of formalizing mathematics in proof assistants on the other, it is natural to have category theory formalized in one; in particular, a formalization that is practically useful as a category-theoretical foundation for other works. This paper is a report of our experience developing such a library. There already exist a relatively large number of formalizations of category theory in proof assistants [14, 16, 9, 1, 7]. However, most of these implementations are not general purpose and rather focus on parts of the theory which are relevant to the specific application of the authors. See the bibliography of Gross et al. [8] for an extensive list of such developments.

Features of Coq 8.5 used: η for records and universe polymorphism

This development makes use of two features new to Coq 8.5. Namely, primitive projection for records (i.e., the η rule for records) and universe polymorphism.

Following Gross et al. [7], we use primitive projections for records which allow for well behaved-dualities in category theory. The dual (aka opposite) of a category $\mathcal{C}$ is a category $\mathcal{C}^{op}$ which has the same objects as $\mathcal{C}$ where the collection of morphisms from A to B is swapped with that from B to A. Drawing intuition from the similarity of categories and preorders, the opposite of a category (seen as a preorder) is simply a category where the order of objects is reversed. Use of duality arguments in proofs and definitions in category theory are plentiful, e.g., sums and products, limits and co-limits, etc. One particular property of duality is that it is involutive. That is, for any category $\mathcal{C}$, $(\mathcal{C}^{op})^{op} = \mathcal{C}$. The primitive projection for records simply states that two instances of a record type are definitionally equal if and only if their projections are. In terms of categories, two categories are definitionally equal if and only if their object collections are, morphism collections are and so forth. This means that we get that the equality $(\mathcal{C}^{op})^{op} = \mathcal{C}$ is definitional. Similar results hold for the duality and composition of functors, for natural transformations, etc. That is we get definitional equalities such as $(\mathcal{F}^{op})^{op} = \mathcal{F}$, $(\mathcal{N}^{op})^{op} = \mathcal{N}$ and $(\mathcal{F} \circ \mathcal{G})^{op} = \mathcal{F}^{op} \circ \mathcal{G}^{op}$ where $\mathcal{F}$ and $\mathcal{G}$ are functors and $\mathcal{N}$ is a natural transformation.

To achieve well behaved dualities, in addition to primitive projections one needs to slightly adjust the definition of a category itself. More precisely, the definition of the category must carry a symmetric form of associativity of composition. The reason being the fact that for the dual category we can simply swap the proof of associativity with its symmetric form and thus after taking the opposite twice get back the proof we started with.

In this development we have used universe polymorphism, a feature new to Coq 8.5, to represent relative smallness/largeness. In short, universe polymorphism allows for a definition to be polymorphic in its universe variables. This allows us, for instance, to construct the category of (relatively small) categories directly. That is, the category constructed is at a universe level (again polymorphic) while its objects are categories at a lower universe level. We will elaborate the use of universe polymorphism to represent relative largeness and smallness below in Section 2.

Contributions

The main contributions of this development are its extent of coverage of basic concepts in category theory and its use of the universe polymorphism of Coq 8.5 and its universe inference algorithm to represent relative smallness/largeness. The latter, as explained below, allows us to represent smallness and largeness using universe levels by simply forgetting about them and letting Coq's universe inference algorithm take care of smallness and largeness requirements as necessary.

The structure of the rest of this paper

The rest of this paper is organized as follows. Section 2 gives an explanation of smallness and largeness in category theory based on the foundation used. This is followed by a detailed explanation of our use of the new universe polymorphism and universe constraint inference algorithm of Coq 8.5 to represent relative smallness/largeness of categories. There, we also give a short comparison of the way other developments represent (relatively) large concepts.

In Section 3, we give a high-level explanation of the concepts formalized and some notable features in this work. We furthermore provide a comparison of our work with a number of other works of similar extent. We also briefly discuss the axioms that we have used throughout this development.

Section 4 describes the work that we have done or plan to do which is based on the current work as category-theoretical foundation. Finally, in Section 5 we conclude with a short summary of the paper.

Development source code The repository of our development can be found in GitHub [21].

2 Universes, Smallness and Largeness

A category is usually called small if its objects and morphisms form sets and large otherwise. It is called locally small if the morphisms between any two objects form a set but objects fail to. For instance, the category **Set** of sets and functions is a locally small category as the collection of all sets does not form a set while for any two sets, there is a set of functions between them. These distinctions are important when working with categories. For instance, a category is said to be complete if it has the limit of all *small* diagrams ($\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is a small diagram if $\mathcal{C}$ is a small category). For instance, **Set** is complete but does not have the cartesian product of all large families of sets.

These terminology and considerations are due to the fact that the original foundations of category theory by Eilenberg and Mac Lane were laid on top of NGB (von Neumann-Gödel-Bernays) set theory. In NBG, in addition to sets, the notion of a class (a collection of sets which itself is not *necessarily* a set) is also formalized. For any property φ, there is a class C_φ of all sets that have property φ. If the collection of sets satisfying φ forms a set then

C_φ is just that set. Otherwise, C_φ is said to be a proper class. In this formalism, one can formalize large categories but cannot use them. For instance, the functor category $\mathbf{Set}^{\mathbf{Set}}$ is not defined as its objects are already proper classes and there is no class of proper classes in NBG.

The other foundation that is probably the most popular among mathematicians is that of ZF with Grothendieck's axiom of universe. Roughly speaking, a Grothendieck universe V is a set that satisfies ZF axioms, e.g., if $A \in V$ and $B \in V$ then $\{A, B\} \in V$ (axiom of pairing), if $A \in V$ then $2^A \in V$ (axiom of power set), etc. We also have if $A \in B$ and $B \in V$ then $A \in V$. *Grothendieck's axiom* says that for any set x there is a Grothendieck universe V such that $x \in V$. This also implies that for any Grothendieck universe V, there is a Grothendieck universe V' such that $V \in V'$.

Working on top of this foundation, one can talk about V-small categories and use all the set-theoretic power of ZF. The notion of completeness for a V-small category can be defined as having all V-small limits. The category of all V-small sets will be a V'-small category where $V \in V'$. It is also a V-locally-small category as its set of morphisms are V-small but its set of objects fail to be. For more details on foundations for category theory see chapter 12 of McLarty's book [13].

The type hierarchy of Coq (also known as universes), as explained below, bears a striking resemblance to Grothendieck universes just explained. In the rest of this section we discuss how Coq's new universe polymorphism feature allows us to use Coq universes instead of Grothendieck universes in a completely transparent way. That is, we never mention any universes in the whole of the development and Coq's universe inference algorithm (part of the universe polymorphism feature) infers them for us.

2.1 Coq's Universes

In higher-order dependent type theories such as that of Coq, types are also terms and themselves have types. As expected, allowing existence of a type of all types results in self-referential paradoxes, such as Girard's paradox [6]. Thus, to avoid such paradoxes type theories like Coq use a countably infinite hierarchy of types of types (also known as universes): $\texttt{Type}_0 : \texttt{Type}_1 : \texttt{Type}_2 : \ldots$ The type system of Coq additionally has the cumulativity property, i.e., for any term $\texttt{T} : \texttt{Type}_n$ we also have $\texttt{T} : \texttt{Type}_{n+1}$.

The type system of Coq has the property of *typical ambiguity*. That is, in writing definitions, we don't have to specify universe levels and/or constraints on them. The system automatically infers the constraints necessary for the definitions to be valid. In case, the definition is such that no (consistent) set of constraints can be inferred, the system rejects it issuing a "universe inconsistency" error. It is due to this feature that throughout this development we have not had the need to specify any universe levels and/or constraints by hand.

To better understand typical ambiguity in Coq, let's consider the following definition.

```
Definition Tp := Type.
```

In this case, Coq introduces a new universe variable for the level of the type `Type`. That is, internally, the definition looks like[1]:

```
Definition Tp : Type@{i+1} := Type@{i}.
```

[1] `Type@{i}` is Coq's syntax for $\texttt{Type}_i$.

Note that in older version of Coq and when universe polymorphism is not enabled in Coq 8.5 the universe level `i` above is a global universe level, i.e., it is fixed. Hence, the following definition is rejected with a universe inconsistency error.

```
Definition TpTp : Tp := Tp.
```

The problem here is that this definition requires (`Type@{i} : Type@{i}`) which requires the system to add the constraint `i` < `i` which makes the set of constraints inconsistent. Without universe polymorphism, one way to solve this problem would be to duplicate the definition of `Tp` as `Tp'` which would be internally represented as:

```
Definition Tp' : Type@{j+1} := Type@{j}.
```

Now we can define `TpTp'`:

```
Definition TpTp' : Tp' := Tp.
```

which Coq accepts and consequently adds the constraint `i` < `j` to the global set of universe constraints. As these constraints are global however, after defining `TpTp'` we can't define `Tp'Tp`

```
Definition Tp'Tp : Tp := Tp'.
```

This is rejected with a universe inconsistency error as it requires `j` < `i` to be added to the global set of constraints which makes it inconsistent as it already contains `i` < `j` from `TpTp'`.

2.2 Universe Polymorphism

Coq has recently been extended [18] to support universe polymorphism. This feature is now included in the recently released Coq 8.5. When enabled, universe levels of a definition are bound at the level of that definition. Also, any universe constraints needed for the definition to be well-defined are local to that definition. That is the definition of `Tp` defined above is represented internally as:

```
Definition Tp@{i} : Type@{i+1} := Type@{i}. (* Constraints: *)
```

Note that the universe level `i` here is local to the definition. Hence, `Tp` can be instantiated at different universe levels. As a result, the definition of `TpTp` above is no longer rejected and is represented internally as:

```
Definition TpTp@{i j} : Tp@{j} := Tp@{i}. (* Constraints: i < j *)
```

That is, the two times `Tp` is mentioned, two different instances of it are considered at two different universe levels `i` and `j` resulting in the constraint `i` < `j` for the definition to be well-defined.

Note the resemblance between universes in Coq and Grothendieck universes. E.g., the fact that if `A : Type@{i}` and `B : Type@{i}` then `{x : Type@{i} | x = A ∨ x = B} : Type@{i}`, cumulativity, etc.

In the sequel, in some cases, we only show the internal representation of concepts formalized in Coq.

2.3 Smallness and Largeness

In this implementation, we use universe levels as the underlying notion of smallness/largeness. In other words, we simply ignore smallness and largeness of constructions and simply allow

Coq to infer the necessary conditions for definitions to be well-defined. We define categories without mentioning universe levels. They are internally represented as:

```
Record Category@{i j} :=
  {
    Obj : Type@{i};
    Hom : Obj → Obj → Type@{j};
    ...
  } : Type@{max(i+1, j+1)} (* Constraints: *)
```

The category of (small) categories is internally represented as:

```
Definition Cat@{i j k l} :=
  {|
    Obj := Category@{k l};
    Hom := fun (𝒞 𝒟 : Category@{k l}) ⇒ Functor@{k l k l} 𝒞 𝒟;
    ...
  |} : Category@{i j} (* Constraints: k < i, l < i, k ≤ j, l ≤ j *)
```

That is, **Cat** has as objects categories that are small compared to itself.

Having a universe-polymorphic **Cat** means for any category $\mathcal{C}$ there is a version of **Cat** that has $\mathcal{C}$ as an object. Therefore, for example, to express the fact that two categories are isomorphic, we simply use the general definition of isomorphism in the specific category **Cat**. This means we can use all facts and lemmas proven for isomorphisms, for isomorphisms of categories with no further effort required.

The category of types (representation of **Set** in Coq) is internally represented as:

```
Definition Set@{i j} :=
  {|
    Obj := Type@{j};
    Hom := fun (A B : Type@{j}) ⇒ A → B;
    ...
  |} : Category@{i j} (* Constraints: j < i *)
```

The constraint `j` < `i` above is exactly what we expect as **Set** is locally small. The reason that Coq's universe inference algorithm produces this constraint is that the type of objects of **Set** is `Type@{j}` which itself has type `Type@{i}`. But, the homomorphisms of this category are functions between two types whose type is `Type@{j}`. Thus, the type of homomorphisms themselves is `Type@{j}`. For details of typing rules for function types see the manual of Coq [12].

Complete Small Categories are Mere Preorder Relations! Perhaps the best showcase of using the new universe polymorphism of Coq to represent smallness/largeness can be seen in the theorem below which simply implies that any complete category is a preorder category, i.e., there is at most one morphism between any two objects.

```
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 **Discr**(`Arrow` $\mathcal{C}$) that maps every object to `y`, `(Arrow` $\mathcal{C}$`)` is the type of all homomorphisms of category $\mathcal{C}$ and $\simeq$ denotes isomorphism. In other words, for any pair of objects `x` and `y` the set of functions from the set of all morphisms in $\mathcal{C}$ to the set of morphisms from `x` to `y` is isomorphic to the set of morphisms from `x` to some constant object `y'`. This though, would result in a contradiction as soon as we have two objects A and B in $\mathcal{C}$ for which the collection of morphisms from A to B has more than one element. Hence, we have effectively shown that *any* complete category is a preorder category.

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 (in the conventional sense) categories. That is, any *small and complete* category is a preorder category[2].

As expected, the constraints on the universe levels of this theorem that are inferred by Coq do indeed confirm this fact. That is, this theorem is in fact only applicable to a category $\mathcal{C}$ for which the level of the type of objects is less than or equal to the level of the type of arrows. This is in direct conflict with the constraints inferred for **Set** as explained above. Hence, Coq will refuse to apply this theorem to the category **Set** with a universe inconsistency error.

2.4 Limitations Imposed by Using Universe Levels for Smallness and Largeness

The universe polymorphism of Coq, as explained in Sozeau et al. [18], treats inductive types by considering copies of them at different levels. Furthermore, if a term of a universe polymorphic inductive type is assumed to be of two instances of that inductive type with two different sets of universe level variables, additional constraints are imposed so that the corresponding universe level variables in those two sets are required to be equal. As records are a special kind of inductive types, the same holds for them. For us, this implies that if we have $\mathcal{C}$: `Category@{i j}` and we additionally have that $\mathcal{C}$: `Category@{i' j'}`, Coq enforces `i` = `i'` and `j` = `j'`. This means, **Cat**@{`i j k l`} is in fact *not* the category of *all* smaller categories. Rather it is the category of smaller 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 constraints 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 constraints that `j` = `k` = `l`. Consequently, only those copies have exponentials for which this constraints holds. Looking back at **Set**, we had the constraint that the level of the type of morphisms 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.

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

$$\left(a^b\right)^c \simeq a^{b\times c} \tag{1}$$

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

It is worth noting that although the category **Cat**@{`i j k l`} is the category of all categories `Category@{k l}` and not lower, for any lower category it contains an "isomorphic copy" of that category. That is any category $\mathcal{C}$: `Category@{k' l'}` such that `k'` ≤ `k` and `l'` ≤ `l` can be "lifted" to `Category@{k l}`. Such a lifting function can be simply written as:

```
Definition Lift (𝒞 : Category@{k' l'}) : Category@{k l} :=
  {| Obj := Obj 𝒞; Hom := Hom 𝒞; . . . |}.
```

[2] This theorem and its proof are taken from Awodey's book [2].

and the appropriate constraints, i.e., $\mathtt{k}' \leq \mathtt{k}$ and $\mathtt{l}' \leq \mathtt{l}$ are inferred by Coq. However, working with such liftings is in practice cumbersome as in interesting cases where $\mathtt{k}' < \mathtt{k}$ and/or $\mathtt{l}' < \mathtt{l}$, we can't prove or even specify `Lift` $\mathcal{C} = \mathcal{C}$ as it is ill-typed. This means, any statement regarding $\mathcal{C}$ must be proven separately for `Lift` $\mathcal{C}$ in order for them to be useful for the lifted version.

It is possible to alleviate these problems if we have support for cumulative inductive types in Coq, as proposed in Timany et al. [24]. In such a system, any category $\mathcal{C}$: `Category@{i j}` will also have type `Category@{k l}` so long as the constraints $\mathtt{i} \leq \mathtt{k}$ and $\mathtt{j} \leq \mathtt{l}$ are satisfied.

However, these limitations are not much more than a small inconvenience and in practice we can work in their presence with very little extra effort. At least as far as basic category theory goes. Our development is an attestation to that.

2.5 Smallness and Largeness in Other Developments

In homotopy type theory (HoTT) [20] a category $\mathcal{C}$ has a further constraint that for any two objects A and B the set of morphisms from A to B must form an hSet (a homotopy type-theoretical concept). On the other hand, for two categories $\mathcal{C}$ and $\mathcal{D}$, the set of functors from $\mathcal{C}$ to $\mathcal{D}$ does not necessarily form an hSet. It does however when the set of objects of $\mathcal{D}$ forms an hSet. Therefore, in HoTT settings one can construct the category of small *strict* categories, i.e., small categories whose type of objects forms an hSet, and not the category of all small categories. However, the category of small strict categories itself is not strict. Hence, contrary to the category **Cat** in our development, there is no category (in the HoTT sense, i.e., one whose objects form an hSet) that has the category of small strict categories as one of its objects. In this regard, working in HoTT is similar to working in NBG rather than ZF with Grothendieck universes.

The situation regarding the category of small strict categories discussed above is due to the fact that homotopy type-theoretical levels for types (e.g., hSet) concern a notion of (homotopy theoretical) *complexity* rather than cardinality. In fact, in other situations, e.g., in defining limits of functors, where cardinality is concerned universe levels can be used to express smallness and largeness. In other words, in HoTT settings, when defining limits, one can simply not mention universe levels and let Coq infer that the definition of limit for a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is well-defined whenever, $\mathcal{C}$ is relatively small compared to $\mathcal{D}$. This also means that the restrictions mentioned above are also present in HoTT settings when universe levels are used to represent smallness and largeness. For instance isomorphism 1 above can't be proven in **Cat** using the Yoneda lemma even if a, b and c are strict categories.

This is how smallness and largeness works in both Gross et al. [7] and Ahrens et al. [1]. This is also the case for our development when ported on top of the HoTT library [19]. As one consequence, contrary to what was explained above, in migrating to the HoTT library settings we can't simply consider the isomorphism of categories as the general notion of isomorphism in the specific case of **Cat**.

In Huet et al. [9], working in Coq 8.4, the authors define a duplicate definition of categories, `Category`', tailored to represent large categories. This way, they form the `Category`' of categories (`Category`) – much like we used `Tp'` above.

Peebles et al. [16] however use universe levels to represent smallness and largeness. But working in Agda which provides no typical ambiguity or cumulativity, they have to hand code all universe levels everywhere; whereas we rely on Coq's inference of constraints to do the hard work. Noteworthy is also the fact that their categories have three universe variables instead of our two. One for the level of the type of objects, one for the level of the type of morphisms and one for the level of the type of the setoid equality for their setoids of morphisms.

3 Concepts Formalized, Features and Comparison

In this development we have formalize most of the basic category theory. Here, by basic we mean not involving higher (e.g., 2-categories), monoidal or enriched categories. This spans over the simple yet important and useful basic concepts like terminal/initial objects, products/sums, equalizers/coequalizers, pullbacks/pushouts and exponentials on the one hand and adjunctions, Kan extensions, (co)limits (as (left)right local Kan extensions) and toposes on the other.

The well-behaved dualities (in the sense discussed above) allow us to simply define dual notions, just as duals of their counterparts, e.g., initial objects as terminal objects of the dual category or the local left Kan extension of $\mathcal{F}$ along $\mathcal{G}$ as the local right Kan extension of $\mathcal{F}^{op}$ along $\mathcal{G}^{op}$.

3.1 Concepts Formalized: Generality and Diversity

Throughout this development we have tried to formalize concepts in as general a way as possible so long as they are comfortably usable. For instance, we define (co)limits as (left)right local Kan extensions along the unique functor to the terminal category. By doing so, we can extend facts about them to (co)limits. As an example, consider (left)right adjoints preserving (co)limits and (co)limit functors being adjoint to Δ explained below.

Different versions of adjunction and Kan extensions In this formalization, we have multiple versions of the definition of adjunctions and Kan extensions. In particular, we define unit-universal morphism property adjunction, unit-co-unit adjunction, universal morphism adjunction and hom-functor adjunction. For these different versions, we provide conversions to and from the unit-universal morphism property definition which is taken to be the main definition. This definition is also taken to be the main definition of adjunction in Awodey's book [2]. For local Kan extensions, we define them as (initial)terminal (co)cones along a functor as well as through the hom-functor. Global Kan extensions are simply defined through adjunctions.

The main reason for this diversity, aside from providing a versatile category theory library, is the fact that each of these definitions is most suitable for some specific purpose.

For instance, using the hom-functor definition of adjunctions makes it very easy to prove that isomorphic functors have the same adjoints: $\mathcal{F} \simeq \mathcal{F}' \Rightarrow \mathcal{F} \dashv \mathcal{G} \Rightarrow \mathcal{F}' \dashv \mathcal{G}$, duality of adjunction: $\mathcal{F} \dashv \mathcal{G} \Rightarrow \mathcal{G}^{op} \dashv \mathcal{F}^{op}$, and uniqueness of adjoint functors: $\mathcal{F} \dashv \mathcal{G} \Rightarrow \mathcal{F}' \dashv \mathcal{G} \Rightarrow \mathcal{F} \simeq \mathcal{F}'$. The last case simply follows from the Yoneda lemma. On the other hand, the unit-universal morphism property definition of adjunctions together with the definition of Kan extensions as cones along a functor provide an easy way to convert from local to global Kan extensions.

Universal morphism adjoints in practice express sufficient conditions for a functor to have a (left)right adjoint. That is, a functor $\mathcal{G} : \mathcal{C} \to \mathcal{D}$ is a right adjoint (has a left adjoint functor) if the comma category $(x \downarrow \mathcal{G})$ has a terminal object for any $x : \mathcal{D}$. As we will briefly discuss below, (left)right adjoint functors preserve (co)limits. Freyd's adjoint functor theorem gives an answer to the question "when is a functor that preserves all limits a right adjoint (has a left adjoint functor)". Universal morphism adjoints appear in this theorem and that's why we have included them in our formalization.

(Left)right adjoints preserve (co)limits Awodey [2] devotes a whole section to this fact with the title "RAPL" (Right Adjoints Preserve Limits). For a better understanding of this fact and perhaps the concept of adjunctions, let us draw intuition from categorical interpretations of logic. In categorical interpretations of logic, the existential and universal quantifiers are interpreted as left and right adjoints to some functor while conjunctions and disjunctions are defined as products and sums respectively which respectively are in turn limits and co-limits (see Jacobs' book [10] for details). In this particular case, RAPL and its dual boil down to: $\forall x.\ P(x) \wedge Q(x) \Leftrightarrow \forall x.\ P(x) \wedge \forall x.\ Q(x)$ and $\exists x.\ P(x) \vee Q(x) \Leftrightarrow \exists x.\ P(x) \vee \exists x.\ Q(x)$. We prove this fact in general for (left)right local Kan extensions. To this end, the unit-co-unit definition of adjunctions is the easiest to use to prove the main lemma which along with hom-functor definition of Kan extensions proves that (left)right adjunctions preserve (left)right Kan extensions. That is for an adjunction $\mathcal{L} \dashv \mathcal{R}$ where $\mathcal{R} : \mathcal{D} \to \mathcal{E}$ and $\mathcal{L} : \mathcal{E} \to \mathcal{D}$ if in the diagram on the left $\mathcal{H}$ is the local right Kan extension of $\mathcal{F}$ along $\mathcal{P}$ then in the right diagram $\mathcal{R} \circ \mathcal{H}$ is the local right Kan extension of $\mathcal{R} \circ \mathcal{F}$ along $\mathcal{P}$:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "\mathcal{F}"] \arrow[d, "\mathcal{P}"'] & \mathcal{D} \\
\mathcal{C}' \arrow[ur, "\mathcal{H}"'] &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathcal{C} \arrow[r, "\mathcal{R}\circ\mathcal{F}"] \arrow[d, "\mathcal{P}"'] & \mathcal{E} \\
\mathcal{C}' \arrow[ur, "\mathcal{R}\circ\mathcal{H}"'] &
\end{tikzcd}$$

The case of (co)limits follows immediately. In Coq we show this by constructing a local right Kan extension (using the hom-functor definition) of $\mathcal{R} \circ \mathcal{F}$ along $\mathcal{P}$ where the Kan extension functor (HLRKE) is $\mathcal{R}$ composed with the Kan extension functor of $\mathcal{F}$ along $\mathcal{P}$:

```
Definition Right_Adjoint_Preserves_Hom_Local_Right_KanExt
   {𝒞 𝒞′ : Category} (𝒫 : Functor 𝒞 𝒞′) {𝒟 : Category} (ℱ : Functor 𝒞 𝒟)
   (hlrke : Hom_Local_Right_KanExt 𝒫 ℱ)
   {ℰ : Category} {ℒ : Functor ℰ 𝒟} {ℛ : Functor 𝒟 ℰ} (adj : UCU_Adjunct ℒ ℛ)
: Hom_Local_Right_KanExt 𝒫 (ℛ ∘ ℱ) :=
    {|
      HLRKE := (ℛ ∘ (HLRKE hlrke));
      HLRKE_Iso := ...
    |}.
```

(Co)limit functors are adjoint to Δ In order to show that (co)limits are adjoint to the diagonal functor (Δ) we simply use the fact that local (left)right Kan extensions assemble together to form (left)right global Kan extensions. As global Kan extensions are defined as (left)right adjoints to the pre-composition functor, putting these two facts together, we effortlessly obtain that (co)limits form functors which are (left)right adjoint to Δ.

Cardinality restrictions We introduce the notion of cardinality restriction in the category **Set**. A cardinality restriction is a property over types (objects of **Set**) such that if it holds for some type, it must hold for any other type isomorphic (in **Set**) to it. That is, if a cardinality restriction holds for a type, it must hold for any other type with the same cardinality.

```
Record Card_Restriction : Type :=
  { Card_Rest : Type → Prop;
    Card_Rest_Respect : forall (A B : Type),
      (A ≃≃ B ::> Set) → Card_Rest A → Card_Rest B }.
```

The type (A ≃≃ B ::> **Set**) is the type of isomorphisms $A \simeq B$ in **Set**. As an example, the cardinality restriction corresponding to finiteness is defined as follows.

```
Definition Finite : Card_Restriction :=
  {| Card_Rest := fun A ⇒ inhabited {n : nat & (A ≃≃ {x : nat | x < n} ::> Set)}; ... |}.
```

The definition above basically says that a type A is finite if there exists some n such that A is isomorphic to the type $\{x : \texttt{nat} \mid x < n\}$ of natural numbers less than n.

(Co)limits restricted by cardinality We use the notion of cardinality restrictions above to define (co)limits restricted by cardinality. For a cardinality restriction P, we say a category $\mathcal{C}$ has (co)limits of cardinality P ($\mathcal{C}$ is P-(co)complete) if for all functors $\mathcal{F} : \mathcal{D} \to \mathcal{C}$ such that $P(Obj_{\mathcal{D}})$ and $\forall AB \in Obj_{\mathcal{D}}, P(Hom(A, B))$, $\mathcal{C}$ has the (co)limit of $\mathcal{F}$.

```
Definition Has_Restr_Limits (𝒞 : Category) (P : Card_Restriction) :=
  forall {𝒥 : Category} (ℱ : Functor 𝒥 𝒞), P 𝒥 → P (Arrow 𝒥) → Limit ℱ.
```

We state several lemmas about cardinality restricted (co)completeness, e.g., if a category has all limits of a specific cardinality its dual has all co-limits of that cardinality.

```
Definition Has_Restr_Limits_to_Has_Restr_CoLimits_Op
        {𝒞 : Category} {P : Card_Restriction}
        (HRL : Has_Restr_Limits 𝒞 P) : Has_Restr_CoLimits (𝒞^op) P := ...
```

This also allows us to define a topos, simply as a category that is cartesian closed, has all finite limits and a subobject classifier where finiteness is represented as a cardinality restriction.

```
Class Topos : Type :=
  { Topos_Cat : Category;
    Topos_Cat_CCC : CCC Topos_Cat;
    Topos_Cat_Fin_Limit : Has_Restr_Limits Topos_Cat Finite;
    Topos_Cat_SOC : SubObject_Classifier Topos_Cat }.
```

(Co)Limits by (Sums)Products and (Co)Equalizers A discrete category is a category where the only morphisms are identities. That is, any set can induce a discrete category by simply considering the category which has as objects members of that set and the only morphisms are identity morphisms. We define the discrete category of a type A as a category, **Discr**(A) with terms of type A as objects and the collection of morphisms from an object x to an object y are proofs of equality of $x = y$.

```
Definition Discr_Cat (A : Type) : Category := {|Obj := A; Hom := fun a b ⇒ a = b; ... |}.
```

Similarly, a discrete functor is a functor that is induced from a mapping f from a type A to objects of a category $\mathcal{C}$:

```
Definition Discr_Func {𝒞 : Category} {A : Type} (f : A → 𝒞) : Functor (Discr_Cat A) 𝒞 :=
   {| FO := f; ... |}.
```

We define the notion of generalized (sums)products to be that of (co)limits of functors from a discrete category.

```
Definition GenProd {A : Type} {𝒞 : Category} (f : A → 𝒞) := Limit (Discr_Func f).
```

We use these generalized (sums)products to show that any category that has all generalized (sums)products and (co)equalizers has all (co)limits. We also prove the special case of cardinality restricted (co)limits. Using the notions explained above, we show that given a cardinality restriction P if a category has (co)equalizers as well as all generalized (sums)products that satisfy P, then that category is P-(co)complete.

```
Definition Restr_GenProd_Eq_Restr_Limits
   {C : Category} (P : Card_Restriction)
   {CHRP : forall (A : Type) (f : A → C), (P A) → (GenProd f)}
   {HE : Has_Equalizers C}
: Has_Restr_Limits C P := ...
```

Categories of Presheaves To the best of our knowledge, ours is the only category theory development featuring facts about categories of presheaves such as their (co)completeness, and being a topos. The category of presheaves on $\mathcal{C}$, $(\mathbf{PSh}(\mathcal{C}))$, is a category whose objects are functors of the form $\mathcal{C}^{op} \to \mathbf{Set}$ and whose morphisms are natural transformations. In other words, a presheaf $P : \mathcal{C}^{op} \to \mathbf{Set}$ on $\mathcal{C}$ is a collection of sets indexed by objects of $\mathcal{C}$ such that for a morphism $f : A \to B$ in $\mathcal{C}$, there is a function (a conversion if you will) $P(f) : P(B) \to P(A)$ in $\mathbf{Set}$. Presheaves being toposes, each come with their own logic. As an example, Birkedal et al. [4] show that the logic of the category of presheaves on ω (the preorder of natural numbers considered as a category) corresponds to the step-indexing technique used in the field of programming languages and program verification. For more details about elementary properties of categories of presheaves see Awodey's book [2]. There categories of presheaves are called categories of diagrams.

3.2 Comparison

Figures 1 and 2 give an overall comparison of our development with select other implementations of category theory of comparable extent. These figures mention only the most notable features and concepts formalized and do not contain many notions and lemmas in these developments. Notice also that the list of concepts and features appearing in these tables is by no means exhaustive and is not the union of all formalized concepts and features of these developments. In these figures, our development is the first column.

3.3 Axioms

One axiom that is used ubiquitously throughout the development is the uniqueness of proofs of equality.

```
forall (A : Type) (x y : A) (p q : x = y), p = q
```

We in practice enforce this axiom using proof-irrelevance (as `p` and `q` are proofs). To facilitate the use of this axiom, we prove a number of lemmas, e.g.:[3]

```
Lemma Functor_eq_simplify (𝒞 𝒟 : Category) (ℱ 𝒢 : Functor 𝒞 𝒟) :
      (FO ℱ = FO 𝒢) → (FA ℱ = FA 𝒢) → ℱ = 𝒢
```

[3] This is an over-simplification: in practice types of `FA` $\mathcal{F}$ and `FA` $\mathcal{G}$ don't match and therefore their equality as stated here is ill-typed. In practice, we adjust the type of `FA` $\mathcal{F}$ using the equality of object maps.

Concept / Feature	[21]	[7]	[9]	[1]	[16]
Automation	partial	✓			
Based on HoTT	in [22]$^\sharp$	✓		✓	
Setoid for Morphisms			✓		✓
Assumes UIP or equivalent	few restricted cases				✓
Basic constructions:					
Terminal/Initial object	✓	✓	✓	✓	✓
Products/Sums	✓		✓	✓	✓
Equalizers/Coequalizers	✓		✓		
Pullbacks/Pushouts	✓		✓	✓	✓
Basic constructions	✓		✓		
above are (co)limits					
exponentials	✓		✓		✓
Subobject classifier	✓			✓	✓
External constructions:					
Comma categories	✓	✓	✓	✓	✓
Product category	✓	✓	✓	✓	✓
Sum category		✓			
Cat. of categories (**Cat**):	✓	✓	✓		✓
Cartesian closure	✓		✓		
Initial/terminal object	✓	✓	✓		✓
Category of sets (**Set**):	✓	✓	✓	✓	✓
Basic (co)limits	✓	init./term.	partial		
(Local†)Cartesian closure	✓		CCC		
(Co†)Completeness	✓		comp.		✓
Sub-object classifier	(`Prop : Type`)†				
Topos	✓†				
Hom functor	✓	✓	✓	✓	✓
Fully-faithful functors	✓	✓		✓	✓
Essentially (inj)sur-jective functors	✓	✓		✓	✓
The Yoneda lemma	✓	✓	✓	✓	✓
Monoidal Categories		partial			✓
Enriched Categories		partial			partial
2-categories					✓
Pseudo-functors		✓			✓
(Co)monads and algebras :					
(Co)Monad				✓	✓
T-(co)algebras	✓			✓	✓
(T: an endofunctor)					
Eilenberg Moore cat.					✓
Kleisli cat.					✓

†Uses the axioms: propositional extensionality and constructive indefinite description (choice).
$^\sharp$The version of our development we are migrating to HoTT settings, on top of HoTT library.

Figure 1 Comparison of features and concepts formalized with a few other implementations of comparable extent.

Concept / Feature	[21]	[7]	[9]	[1]	[16]
Adjunction	✓	✓	✓		✓
Unit-universal morphism adjunction	✓	✓			
Hom-functor adjunction	✓	✓	✓		
Unit-counit adjunction	✓	✓	✓	✓	✓
Universal morphism adjunction	✓	✓	✓		
Uniqueness up to natural isomorphism	✓				
Naturally isomorphic functors have the same left/right adjoints	✓				
Adjoint composition laws	✓	✓			✓
Category of adjunctions (objects: categories; morphisms: adjunctions)	✓				
Partial adjunctions		✓			
Adjoint Functor Theorem	✓				✓
Kan extensions	✓	✓			✓
Global definition	✓	✓		✓	
Local definition	✓	✓			
Through hom-functor	✓				
Through cones (along a functor)	✓				✓
Through partial adjoints		✓			
Uniqueness	✓				
Preservation by adjoint functors	✓				
Naturally isomorphic functors form the same left/right Kan extension	✓				
Pointwise kan extensions (preserved by representable functors)	✓	✓			
(Co)Limits	✓	✓	✓	✓	✓
As (left)right kan extensions	✓	✓			
As (initial)terminal (co)cones			✓	✓	✓
(Sum)Product-(co)equalizer (co)limits	✓				
(Co)Limit functor	✓	✓			
(Co)Limits functor adjoint to Δ	✓	✓			
(Co)limits restricted by cardinality	✓				
Pointwise (as kan extensions), i.e., preserved by Hom functor	✓		✓		
Category of presheaves over $\mathcal{C}$ ($\mathbf{PSh}_{\mathcal{C}}$):	✓				✓
Terminal/Initial object	✓				
Products/Sums	✓				
Equalizers/Coequalizers	✓†				
Pullbacks	✓				
Cartesian closure	✓				
Completeness/Co-completeness	✓†				
Sub-object classifier (Sieves)	✓†				
Topos	✓†				

†Uses the axioms: propositional extensionality and constructive indefinite description (choice).

Figure 2 Comparison of features and concepts formalized with a few other implementations of comparable extent (cont.).

which says two functors are equal if their object and arrow maps are. If so, the proofs that the arrow maps preserve identity and composition are just assumed equal using proof-irrelevance (uniqueness of equality proofs).

Using uniqueness of equality proofs in the definition of categories is an essential necessity. As otherwise, as explained in the HoTT book [20], the category defined is not a category but a form of higher category. That's why in any formalization of category theory this axiom is assumed or enforced in one way or another.

In homotopy type theory (HoTT) settings, assuming uniqueness of proofs of equality in general is in direct contradiction with the univalence axiom which sits at the heart of HoTT. Therefore in developments of category theory on top of HoTT, e.g., Gross et al. [7] and Ahrens et al. [1], they include the fact that proofs of equalities of morphisms are unique as part of the definition of a category. This is precisely the requirement that collections of morphisms should form hSets discussed above.

In developments using setoids, e.g. Huet et al. [9] and Peebles et al. [16], the authors customize the setoid equalities so that proofs are never considered. For instance, they *define* the setoid equality for functors so that two functors are equal whenever their object and morphism maps are.

We are currently in the process of porting a version of our development on top of the HoTT library[4]. There we also stop using this axiom and change the definition of categories. As expected almost all of the cases where we use uniqueness of proofs of equality (in a direct or indirect way) are not problematic in HoTT settings, i.e., they are applied to equality of morphisms. However, there are a few limited cases were they are not. Some of these cases are no longer relevant in the HoTT settings and some others are very easily surmountable. For more details of our ongoing effort of porting this development on top of the HoTT library see the extended version of this paper [25].

Apart from the axiom of uniqueness of proofs of equality, we have made frequent use of the axiom of functional extensionality. However, this axiom is a consequence of the univalence axiom and is in fact provided in the HoTT library and frequently used therein.

We have in particular taken advantage of two other axioms, propositional extensionality and axiom of choice (constructive indefinite description in the library of Coq) which we have used, e.g., to construct co-limits in **Set** and presheaf categories. Along with using setoids, using these axioms to represent quotient types in type theory is standard practice. We plan to use higher inductive types, as explained in the HoTT book [20], to construct such co-limits in the version ported on top of the HoTT library.

4 Future Work: Building on Categories

We believe that this development is one that provides a foundation for other works based on category-theoretical foundations. We have plans to make use of the foundation of category theory that has been laid in this work. In particular, we plan to make use of this foundation for mechanization of categorical logic (see Jacobs' book [10]) and higher order separation logic (see Biering et al. [3]) for the purpose of using them as foundations for mechanization of program verification. In particular, the theory of presheaves developed provides a basis for formalization of the internal logic of presheaf categories with a particular interest in the topos of trees [4].

[4] The version being ported on top of the HoTT library can be found at GitHub [22].

In this regard, we have already used this development as a foundation to formalize the theory of Birkedal et al. [5] to solve category theoretical recursive ultra-metric space equations [23]. In Birkedal et al. [5], the authors use the theory of ultra-metric spaces to build unique (up to isomorphism) fixed-points of particular category-theoretical recursive domain-theoretic equations. More precisely, they construct fixed-points of a particular class of mixed variance functors, i.e., functors of the form $\mathcal{F} : (\mathcal{C}^{op} \times \mathcal{C}) \to \mathcal{C}$. Solutions to such mixed-variance functors can for example be used to construct models for imperative programming languages. Successful implementation of this theory [23] on top of our general foundation of categories, although arguably not huge, is evidence that this development is fit for being used as a general-purpose foundation.

In Birkedal et al. [5], the authors define the notion of an M-category to be a category in which the set of morphisms between any two objects form a non-empty ultra-metric space. In our formalization, based on a general theory of ultra-metric spaces, we define M-categories as categories in which the type of morphisms between any two objects forms an ultra-metric space, dropping the rather strong non-emptiness requirement. We instead require some weaker conditions which still allow us to form fixed-points.

An interesting instance of M-categories is the presheaf topos of the preorder category of natural numbers, i.e., the topos of trees. In our development, just showing that this category qualifies as an M-category is sufficient to immediately be able to construct desired fixed-points. This is due to the fact that in the foundations provided, all necessary conditions for an M-category to allow formation of solutions, e.g., existence of limits of a particular class of functors is already established.

5 Conclusion

The most important conclusion of this paper is that Coq 8.5 with its new features: η for records and universe polymorphism, is next to ideal for formalization of category theory and related parts of mathematics. We believe that Coq 8.5 is the first version of Coq that makes it possible to lay a truly useful and versatile general purpose category theoretical foundation as we have demonstrated.

In summary, we surveyed our development of the foundations of category theory. This development features most of the category-theoretical concepts that are formalized in most other such developments and some more. We pushed the limits of the new feature of universe polymorphism and the constraint inference algorithm of Coq 8.5 by using them to represent relative smallness/largeness. As discussed, it gives very encouraging results despite the restrictions imposed by not having cumulative inductive types.

We have successfully used this implementation as the categorical foundation to build categorical ultra-metric space theoretic fixed-points of recursive domain equations. This seems an encouraging initial indication that this work is fit to perform the important role of a general purpose category theoretical foundation for other developments to build upon.

Acknowledgements

We would like to thank Jason Gross, Matthieu Sozeau and Georges Gonthier for helpful discussions. We are also grateful to the anonymous reviewers for their invaluable comments and suggestions.

This work was funded by EU FP7 FET-Open project ADVENT under grant number 308830.

References

1 Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the rezk completion. *Mathematical Structures in Computer Science*, 25(05):1010–1039, jan 2015. URL: `https://github.com/benediktahrens/rezk_completion`, `doi:10.1017/s0960129514000486`.

2 Steve Awodey. *Category theory*. Oxford University Press, 2010.

3 Bodil Biering, Lars Birkedal, and Noah Torp-Smith. Bi-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Trans. Program. Lang. Syst.*, 29(5), August 2007. URL: `http://doi.acm.org/10.1145/1275497.1275499`, `doi:10.1145/1275497.1275499`.

4 Lars Birkedal, Rasmus Ejlers Mogelberg, Jan Schwinghammer, and Kristian Stovring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*. Institute of Electrical & Electronics Engineers (IEEE), jun 2011. URL: `http://dx.doi.org/10.1109/LICS.2011.16`, `doi:10.1109/lics.2011.16`.

5 Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. The category-theoretic solution of recursive metric-space equations. *Theoretical Computer Science*, 411(47):4102–4122, oct 2010. URL: `http://dx.doi.org/10.1016/j.tcs.2010.07.010`, `doi:10.1016/j.tcs.2010.07.010`.

6 T. Coquand. An analysis of Girard's paradox. Technical Report RR-0531, INRIA, May 1986. URL: `https://hal.inria.fr/inria-00076023`.

7 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. `doi:10.1007/978-3-319-08970-6_18`.

8 Jason Gross, Adam Chlipala, and David I. Spivak. Experience implementing a performant category-theory library in coq, 2014. `arXiv:arXiv:1401.7694`.

9 Gérard P. Huet and Amokrane Saïbi. Constructive category theory. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 239–276, 2000. URL: `http://www.lix.polytechnique.fr/coq/pylons/coq/pylons/contribs/view/ConCaT/v8.4`.

10 Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

11 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 1978.

12 The Coq development team. *Coq 8.5 Reference Manual*. Inria, 2015.

13 Colin McLarty. *Elementary Categories, Elementary Toposes*. Oxford University Press, Oxford, UK, 1996.

14 Adam Megacz. Category Theory in Coq. URL: `http://www.megacz.com/berkeley/coq-categories/`.

15 John C. Mitchell. *Foundations of Programming Languages*. MIT Press, Cambridge, MA, USA, 1996.

16 Daniel Peebles, James Deikun, Ulf Norell, Dan Doel, Andrea Vezzosi, Darius Jahandarie, and James Cook. copumpkin/categories. URL: `https://github.com/copumpkin/categories`.

17 Antonino Salibra. Scott is always simple. In *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, MFCS'12, pages 31–45, Berlin, Heidelberg, 2012. Springer-Verlag. URL: `http://dx.doi.org/10.1007/978-3-642-32589-2_3`, `doi:10.1007/978-3-642-32589-2_3`.

18 Matthieu Sozeau and Nicolas Tabareau. Universe Polymorphism in Coq. In *Interactive Theorem Proving, ITP 2014, Proceedings*, pages 499–514, July 2014.

19 The Univalent Foundations Program. HoTT Version of Coq and Library. URL: `https://github.com/HoTT/HoTT`.

20 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

21 Amin Timany. Categories. URL: `https://github.com/amintimany/Categories/tree/FSCD16`, `doi:10.5281/zenodo.50689`.

22 Amin Timany. Categories-HoTT. URL: `https://github.com/amintimany/Categories-HoTT`.

23 Amin Timany and Bart Jacobs. The Category-theoretic Solution of Recursive Ultra-metric Space Equations. Presented at CoqPL'16: The Second International Workshop on Coq for PL. URL: `https://github.com/amintimany/CTDT`.

24 Amin Timany and Bart Jacobs. First Steps Towards Cumulative Inductive Types in CIC. In *12th International Colloquium on Theoretical Aspects of Computing (ICTAC 2015) 2015. Proceedings*, pages 608–617, 2015. URL: `http://dx.doi.org/10.1007/978-3-319-25150-9_36`, `doi:10.1007/978-3-319-25150-9_36`.

25 Amin Timany and Bart Jacobs. Category Theory in Coq 8.5: Extended Version. Technical Report CW697, iMinds-Distrinet, KU Leuven, April 2016. URL: `http://www2.cs.kuleuven.be/publicaties/rapporten/cw/CW697.abs.html`.