Formalising Yoneda Ext in Univalent Foundations

Jarl G. Taxerås Flaten ✉
University of Western Ontario, London, Ontario, Canada

Abstract

Ext groups are fundamental objects from homological algebra which underlie important computations in homotopy theory. We formalise the theory of Yoneda Ext groups in homotopy type theory (HoTT) using the Coq-HoTT library. This is a constructive approach to Ext which does not require projective or injective resolutions. Using univalence, we show how these Ext groups can be naturally represented in HoTT. We give a novel proof and formalisation of the usual six-term exact sequence via a fibre sequence of 1-types (or groupoids), along with an application. In addition, we discuss our formalisation of the contravariant long exact sequence of Ext, an important computational tool. Along the way we implement and explain the Baer sum of extensions and how Ext is a bifunctor.

2012 ACM Subject Classification Mathematics of computing → Algebraic topology; Theory of computation → Logic and verification, Constructive Mathematics, Type theory

Keywords and phrases homotopy type theory, homological algebra, Yoneda Ext, formalisation, Coq

Supplementary Material github.com/jarlg/Yoneda-Ext

Funding We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2022-04739.

Acknowledgements The theory of Yoneda Ext in HoTT is joint work with Dan Christensen, to whom I am also grateful for discussions and contributions related to the formalisation. I also thank Jacob Ender for contributions to the formalisation of the Baer sum. Finally, I thank the collaborators of the Coq-HoTT library—and Ali Caglayan in particular—for their careful review of, and contributions to, the various pull requests related to this project.

1 Introduction

The field of homotopy type theory lies at the intersection of type theory and algebraic topology, and serves as a bridge to transfer tools and insights from one domain to the other. In one direction, the formalism of type theory has proven to be a powerful language for reasoning about some of the highly coherent structures occurring in branches of modern algebraic topology. Several of these structures are "natively supported" by HoTT, and we can reason about them much more directly than in classical set-based approaches. This makes HoTT an ideal language in which to formalise results and structures from algebraic topology. Moreover, theorems in HoTT are valid in any ∞-*topos*, not just for ordinary spaces.

We present a formalisation of Ext groups in HoTT following the approach of Yoneda [9]. Ext groups are fundamental objects in homological algebra, and they permeate computations in homotopy theory. For example, the universal coefficient theorem relates Ext groups and cohomology, and features in the classical proof that $\pi_5(S^3) \simeq \mathbb{Z}/2$. Much of our formalisation has already been accepted into the Coq-HoTT library under the `Algebra.AbSES` namespace, though we have also contributed to other parts of the library throughout this project. The long exact sequence is currently in a separate repository named `Yoneda-Ext`. We supply links to formalised statements using a trailing $^\diamond$-sign throughout.

In ordinary mathematics, Ext groups of modules over a ring are usually defined using projective (or injective) resolutions. This is possible because the axiom of choice implies the existence of such projective resolutions, and Ext groups are independent of any particular choice of resolution. In our setting, however, even abelian groups may fail to admit projective resolutions. This stems from the fact that sets themselves may fail to be projective, which may be familiar to those working constructively or internally to a topos. Accordingly, to define Ext groups in homotopy type theory we cannot rely on resolutions. Fortunately, Yoneda [9] gave such a general approach, whose theory is detailed in [4], our main reference.

Our formalisation build upon the Coq-HoTT library, which contains sophisticated homotopy-theoretic results, but which is presently lacking in terms of "basic" algebra. For this reason, we have opted to simply develop Ext groups of abelian groups, instead of for modules over a ring or in a more general setup. Nevertheless, it is clear that everything we have done could have been over an arbitrary ring, given a well-developed library of module theory. Moreover, we emphasise that higher Ext groups in HoTT are interesting even for abelian groups. While in classical mathematics such Ext groups of abelian groups are trivial in dimension 2 and up, in HoTT they may be nontrivial in all dimensions! This is because there are models of HoTT in which these Ext groups are nontrivial—details about this are forthcoming in a joint paper with Dan Christensen.

In Section 3 we explain how univalence lets us naturally represent Yoneda's approach to Ext in HoTT. We construct the type $\texttt{AbSES}(B, A)$ of short exact sequences between two abelian groups A and B, and define $\texttt{Ext}^1(B, A)$ to be the set of path-components of $\texttt{AbSES}(B, A)$. This definition is justified by characterising the paths in $\texttt{AbSES}(B, A)$, which crucially uses univalence. We also show that the loop space of $\texttt{AbSES}(B, A)$ is isomorphic to the group $\texttt{Hom}(B, A)$ of group homomorphisms, and that $\texttt{Ext}^1(P, A)$ vanishes whenever P is projective, in a sense we define. These results all play a role in the subsequent sections.

The main content of Section 4 is a proof and formalisation of the following:

▶ **Theorem 1.** *Let $A \xrightarrow{i} E \xrightarrow{p} B$ be a short exact sequence of abelian groups. For any abelian group G, pullback yields a fibre sequence:* $\texttt{AbSES}(B, G) \xrightarrow{p^*} \texttt{AbSES}(E, G) \xrightarrow{i^*} \texttt{AbSES}(A, G)$.$^\diamondsuit$

This is a novel mathematical result whose proof requires managing a considerable amount of coherence. Its formalisation benefited greatly from the `WildCat` library of Coq-HoTT, which makes it convenient to work with types equipped with an imposed notion of paths (see Section 2.2). This allows us to work with *path data* in $\texttt{AbSES}(B, A)$ with better computational properties than actual paths, but which correspond to paths via the aforementioned characterisation. From the fibre sequence of the theorem we deduce the usual six-term exact sequence of Ext groups (Proposition 21), which we then use to show Corollary 23:

$$\texttt{Ext}^1(\mathbb{Z}/n, A) \;\cong\; A/n$$

for any nonzero $n : \mathbb{N}$ and abelian group A [4, Proposition III.1.1].$^\diamondsuit$ The six-term exact sequence, along with this corollary, have already been applied in [1]. We also discuss how $\texttt{Ext}^1$ becomes a bifunctor into abelian groups using the Baer sum.

Finally, in Section 5 we define $\texttt{Ext}^n$ for any $n : \mathbb{N}$ and discuss our formalisation of the long exact sequence:$^\diamondsuit$

▶ **Theorem 2.** *Let $A \xrightarrow{i} E \xrightarrow{p} B$ be a short exact sequence of abelian groups. For any abelian group G, there is a long exact sequence by pulling back:*

$$\cdots \xrightarrow{i^*} \texttt{Ext}^n(A, G) \xrightarrow{-\circledcirc E} \texttt{Ext}^{n+1}(B, G) \xrightarrow{p^*} \texttt{Ext}^{n+1}(E, G) \xrightarrow{i^*} \cdots$$

Our proof follows that of Theorem 5.1 in [4], which is originally due to Stephen Schanuel.

Notation and conventions We use typewriter font for concepts which are defined in the code, such as `AbSES` and `Ext`. When we use normal mathematical font, such as $\mathrm{Ext}^n(B, A)$, we mean the classical notion (i.e., not the one we define in HoTT). For mathematical statements we prefer to stay close to mathematical notation while trusting the reader to make the appropriate translation to code, or including appropriate references. Thus for example $\mathtt{Ext}^n(B, A)$ means `Ext n B A` in Coq.

Our terminology mirrors that of [7]; in particular we say 'path types' for what are also called 'identity types' or 'equality types.' We write `pType` for the universe of pointed types, and `pt` for the base point of a pointed type. The $\equiv$-symbol refers to definitional equality.

The symbol $^\diamond$ is used to refer to relevant parts of the formalisation.

2 Preliminaries

2.1 Homotopy type theory

We briefly explain the formal setup of homotopy type theory along with some basic notions that we need. For a thorough introduction to HoTT, the reader reader may consult [7, 6].

Homotopy type theory (HoTT) extends Martin–Löf type theory (MLTT) with the *univalence axiom* and often various higher inductive types (HITs). We simply need propositional truncation and set truncation, which we explain in more detail below.

The univalence axiom characterises the identity types of universes. In ordinary MLTT, there is always a function

$$\mathtt{idtoequiv} : \prod_{X,Y:\mathtt{Type}} (X = Y) \to (X \simeq Y)$$

defined by sending the reflexivity (or constant) path on a type X to the identity self-equivalence on X, using the induction principle of path types. The univalence axiom asserts that `idtoequiv` is an equivalence for all X and Y. In HoTT, the first thing we do after defining a new type is to characterise its path types. The univalence axiom does this for the universe.

From univalence, a general *structure identity principle* [7, Chapter 9.8] follows which characterises paths between structured types, such as groups and other algebraic structures. In the case of groups, univalence implies that paths between groups correspond to group isomorphisms.

Propositions, sets, and groupoids

In HoTT there is a hierarchy of ***n*-truncated types** (or *n-types*, for short) for any integer $n \geq -2$. In general, a type X is an $(n+1)$-type just when all the path types $x_0 =_X x_1$ are n-types. The recursion starts at -2, when the condition is just that the map $X \to 1$ is an equivalence.

We only deal with the bottom four levels of this hierarchy: **contractible types**, **propositions** ((-1)-types), **sets** (0-types) and 1-types. A type X is a proposition just when any two points in X are equal (but there may not be any points). A type X is a set just when the the path types $x_0 =_X x_1$ are all propositions—this amounts to there being "at most" one path between x_0 and x_1. Lastly, a type X is a groupoid whenever its path types are sets—in particular, for any $x : X$, the **loop space** $\Omega X :\equiv (x =_X x)$ is an ordinary group under path composition.

There are truncation operations which create a proposition or a set from a given type X. In HoTT, we often denote $\|X\|$ and $\pi_0 X$ for propositional truncation and set truncation (or *set of connected components*), respectively. In Coq, the corresponding notation is `merely X` and `Tr 0 X`. The map $\texttt{tr} : X \to \pi_0 X$ sends a point to its connected component.

2.2 The Coq-HoTT library

The Coq-HoTT library is an open-source repository of formalised mathematics in homotopy type theory using Coq. It is particularly aimed at developing synthetic homotopy theory, and includes theory about spheres, loop spaces, classifying spaces, modalities, "wild ∞-categories," and basic results about abelian groups, to mention a few things. The library is part of the *Coq Platform* and is available through the standard `opam` package repositories.

Below we explain some of the main features of this library, and of Coq itself, which are important for the present work.

Universes and cumulativity

We assume basic familiarity with universes and universe levels in Coq, and in particular that they are *cumulative*: a type X : `Type@{u}` can be resized to live in `Type@{v}` under the constraint $\texttt{u} \leq \texttt{v}$. Resizing is done implicitly by Coq.

In the Coq-HoTT library, we additionally make most of our structures cumulative. This essentially means that resizing commutes with forming a data structure—i.e., it does not matter whether your resize the inputs to the data structure or whether you resize the resulting data structure. As an example, consider the data structure `prod` which forms the product of two types in a common (for simplicity) universe `u`. Suppose we have two universes `u` and `v` with the constraint `u < v`. Given `X Y : Type@{u}`, we can form the product at level `u` and then resize, or first resize and then form the product. By making `prod` a cumulative data structure, the two results agree (with implicit resizing):

$$\texttt{prod@\{u\} X Y} \;\equiv\; \texttt{prod@\{v\} X Y}.$$

Cumulativity of data structures is an essential Coq feature which facilitates the kind of formalisation we do in this paper. For example, it lets us resize groups and homomorphisms. It also lets us reduce the number of universes in some of our definitions via the following trick: instead of having separate universes for different inputs, we can often have single universe (which represents the maximum) and leverage cumulativity.

We also make use universe constraints since our constructions move between various universe levels. The constraints both document and verify the mathematical intent.

The `WildCat` library

The `WildCat` namespace contains the development of "wild ∞-categories," functors between such, and related things. This library was spearheaded by Ali Caglayan, tslil clingman, Floris van Doorn, Morgan Opie, Mike Shulman, and Emily Riehl. The concepts generalise those appearing in [8, Section 4.3.1], and are not currently present in the literature. We explain the basics of this library which are especially relevant for our formalisation.

Starting from the notion of *graph*$^\diamond$—a type A with a binary operation `Hom`—the notion of a 0-**functor**$^\diamond$ is that of a homomorphism of graphs:

```
Class IsGraph (A : Type) := { Hom : A -> A -> Type }.
Class Is0Functor {A B : Type} `{IsGraph A} `{IsGraph B} (F : A -> B)
```

```
  := { fmap : forall {a b : A} (f : Hom a b), Hom (F a) (F b) }.
```

We will often use the notation `Hom` in this text, leaving the graph structure implicit.

From here one could go ahead and define categories by defining a composition operation and using the identity types of the type `Hom`(a, b) to express the various laws the a category needs to satisfy, such as associativity of composition. A more flexible approach is to instead allow `Hom`(a, b) to itself be a graph, making A into a **2-graph**.$^{\diamond}$ This is the approach taken by `WildCat`, and this flexibility is important for our formalisation.

```
Class Is2Graph (A : Type) `{IsGraph A}
  := isgraph_hom : forall (a b : A), IsGraph (Hom a b).
```

For a 2-graph A, a category structure can then be defined in a straightforward manner using `isgraph_hom` to express the various laws that need to hold. This structure is bundled into a class called `Is1Cat`.$^{\diamond}$ For example, associativity is expressed as follows, using the notation `$==` as a shorthand for the 2-graph structure and `$o` for composition:

```
  cat_assoc : forall (a b c d : A)
    (f : Hom a b) (g : Hom b c) (h : Hom c d),
      (h $o g) $o f $== h $o (g $o f);
```

Finally, we have the notion of a **1-functor** between categories, for which the laws are also expressed using the 2-graph structure.$^{\diamond}$

```
Class Is1Functor {A B : Type} `{Is1Cat A} `{Is1Cat B}
  (F : A -> B) `{!Is0Functor F} := {
    fmap_id : forall a, fmap F (Id a) $== Id (F a);
    fmap_comp : forall a b c (f : Hom a b) (g : Hom b c),
      fmap F (g $o f) $== fmap F g $o fmap F f;
    fmap2 : forall a b (f g : Hom a b),
      (f $== g) -> (fmap F f $== fmap F g) }.
```

The terms `fmap_id` and `fmap_comp` express that the functor `F` respects identities and composition, as usual. If we had used identity types instead of a 2-graph structure, so that `f $== g` simply meant `f = g`, then `F` would automatically respect equality between morphisms, making `fmap2` redundant. However, in the more general 2-graph setup, this needs to be included as a law. If all the morphisms in A are invertible, then A is a **groupoid**.$^{\diamond}$

The adjective "wild" is used for the sort of categories just defined to indicate that they do not capture all the coherence needed to represent ∞-categories, only the 1-categorical structure. However, in our usage we will only encounter genuine 1-categories and groupoids. In particular, any type X defines a groupoid$^{\diamond}$ via its identity types, and if X is a 1-type then this groupoid structure captures everything about X. The utility of the approach above is that we can impose our own notion of paths (which we call "path data") for certain groupoids.

3 Yoneda Ext

As mentioned in the introduction, we will follow the approach of [9] to Ext groups, which does not require projective (or injective) resolutions. At present, the Coq-HoTT library—with which this work has been formalised—does not contain much theory related to modules over a general ring (nor the theory of abelian categories, or anything of the sort). We therefore only formalise and state our results for abelian groups. It is clear, however, that everything we say could be done for modules over a general ring.

For the classically-minded reader, let us also emphasise that in homotopy type theory the category of abelian groups does *not* have global dimension 1, so that the higher Ext groups we define in Section 5 do not necessarily vanish.

3.1 The type of short exact sequences

Given two abelian groups A and B, Yoneda defines a group $\mathrm{Ext}^1(B, A)$ by considering the set[1] of all short exact sequences $A \xrightarrow{i} E \xrightarrow{p} B$ and taking a quotient by a certain equivalence relation. The sequence being exact means that i is injective, p is surjective, that $p \circ i = 0$, and that the image of i is equal to the kernel of p. We usually simply write E for the short exact sequence $A \to E \to B$ when no confusion can arise. The equivalence relation which Yoneda quotients out by is defined as "$E \sim F$ if and only if there exists a morphism[2] $E \to F$ which respects the maps from A and to B." Equivalently, but more topologically, one can consider the *groupoid* of short exact sequences $A \to E \to B$ and define $\mathrm{Ext}^1(B, A)$ to be the set of path components of this groupoid—see, e.g., [4, Chapter III] for details about both of these descriptions.

In homotopy type theory, given two abelian groups A and B we form the **type of short exact sequences** from A to B as the Σ-type over all abelian groups E equipped with an injection $\texttt{inclusion}_E : A \to E$, a surjection $\texttt{projection}_E : E \to B$, and a witness that these two maps form an exact complex. We represent this data as the following record-type:$^\diamond$

```
Record AbSES@{u v | u < v} (B A : AbGroup@{u}) : Type@{v} := {
    middle : AbGroup@{u};
    inclusion : Hom A middle;
    projection : Hom middle B;
    isembedding_inclusion : IsEmbedding inclusion;
    issurjection_projection : IsSurjection projection;
    isexact_inclusion_projection
      : IsExact (Tr (-1)) inclusion projection;
  }.
```

The abelian group `middle` plays the role of E in the prose above. Here, the condition that $\texttt{projection}_E \circ \texttt{inclusion}_E = 0$ is baked into the `IsExact` field, which also expresses exactness.[3] We have included universe annotations which express that E lives in the same universe `u` as the abelian groups A and B. Accordingly, the resulting type $\texttt{AbSES}(B, A)$ lives in a universe `v` which is strictly greater than `u`, as in Yoneda's construction above. The type $\texttt{AbSES}(B, A)$ is pointed by the **trivial short exact sequence**$^\diamond$ $A \to A \oplus B \to B$.

We now define $\texttt{Ext}^1(B, A)$ as the set-truncation of the type of short exact sequences.$^\diamond$

```
Definition Ext (B A : AbGroup) := Tr 0 (AbSES B A).
```

The notation `Tr 0` constructs the set of *path components* of a type, also called the set-truncation. Thus $\texttt{Ext}^1(B, A)$ is defined as the set of path components of the type of short exact sequences from A to B. In Section 3.3 we make this set into an abelian group using the so-called *Baer sum*. These abelian groups, and their higher variants defined in Section 5, are our main objects of study.

[1] Since we quantify over all abelian groups E the result is a *large* set (or a *class*).
[2] Any morphism that respects the stated maps is automatically an isomorphism, by the "short five lemma." This implies that $\sim$ is an equivalence relation.
[3] The term `Tr (-1)` can safely be ignored; it expresses that the induced map from A to the kernel of $\texttt{projection}_E$ is (-1)-connected, i.e., a surjection.

Whenever we define a new type in homotopy type theory, the first thing we often do is to characterise its path types. Theorem 7.3.12 of [7] characterises paths in truncations, yielding

$$|E|_0 =_{\mathrm{Ext}^1} |F|_0 \iff \|E = F\|_{-1}$$

for any E, F : `AbSES`(B, A). As such, it suffices to understand paths in `AbSES`(B, A). These are in turn characterised by Theorem 2.7.2 of loc. cit., which characterises paths in general Σ-types, combined with the fact that paths in `AbGroup` are isomorphisms. In our case, the result is that paths between short exact sequences correspond to isomorphisms between the `middle`s making the appropriate triangles commute. We refer to this data as *path data*, and bundle it into a separate type (where `*` denotes products of types):$^\diamondsuit$

```
Definition abses_path_data_iso {B A : AbGroup} (E F : AbSES B A)
  := {phi : E $<~> F & (phi $o inclusion E == inclusion F)
                      * (projection E == projection F $o phi)}.
```

The characterisation of paths in Σ-types then gives an equivalence$^\diamondsuit$

$$(E = F) \;\simeq\; \texttt{abses\_path\_data\_iso}(E, F)$$

for any E, F : `AbSES`(B, A). However, a bit more can be said: the *short five lemma*$^\diamondsuit$ implies that if we replace `phi` by a homomorphism above, then `phi` is automatically an isomorphism. We define `abses_path_data`$^\diamondsuit$ exactly as `abses_path_data_iso` above, except that `phi` is only a homomorphism, not an isomorphism. It is convenient to have both types around: it is easier to construct an element of `abses_path_data` since you don't have to supply a proof that the underlying map is an equivalence; however we will see situations later on where it is convenient to keep track of a *specific* inverse to the underlying map, which `abses_path_data_iso` lets us do.

▶ **Definition 3.** *The type* `AbSES`(B, A) *is a groupoid whose graph structure is given by* `abses_path_data_iso` *and a corresponding category structure. For the 2-graph structure, we assert that two path data are equal just when their underlying maps are homotopic.*$^\diamondsuit$

This definition is justified by the preceding discussion, which yields:

▶ **Lemma 4.** *For any* E, F : `AbSES`(B, A)*, there are equivalences of types*$^\diamondsuit$

$$(E = F) \;\simeq\; \texttt{abses\_path\_data\_iso}(E, F) \;\simeq\; \texttt{abses\_path\_data}(E, F).$$

Though elementary, this lemma has an interesting consequence. This statement appears as the $n, i = 1$ case of [5, Theorem 1].

▶ **Proposition 5.** *The loop space of* `AbSES`(B, A) *is naturally isomorphic to* `Hom`(B, A).$^\diamondsuit$

Proof. It suffices, by the previous lemma, to give an isomorphism between `Hom`(B, A) and `abses_path_data`$(A \oplus B, A \oplus B)$. One can easily check that a map $\phi : A \oplus B \to A \oplus B$ subject to the constraints of path data, is uniquely determined by the composite$^\diamondsuit$

$$B \to A \oplus B \xrightarrow{\phi} A \oplus B \to A.$$

Moreover, this association defines a group isomorphism—details are in the formalisation.$^\diamondsuit$ ◀

To formalise the previous proposition, we first developed basic theory about biproducts of abelian groups which now live in `Algebra.AbGroups.Biproduct`.

In ordinary homological algebra, an abelian group P is *projective* if for any homomorphism $f : P \to B$ and epimorphism $p : A \to B$, there exists a *lift* $l : P \to A$ such that $f = e \circ l$. It is well-known that $\mathrm{Ext}^1(P, A)$ always vanishes whenever P is projective, and that this property characterises projectivity. In our setting, we define an abelian group P to be **projective** if for any homomorphism f and epimorphism p as above, there *merely* exists a lift l such that $f = l \circ l$. The propositional truncation makes this into a *property* of an abelian group, and not a structure. In Coq, we express this as a type-class:$^\diamond$

```
Class IsAbProjective (P : AbGroup) : Type :=
  isabprojective : forall (A B : AbGroup),
    forall (f : Hom P B), forall (e : Hom A B),
    IsSurjection e -> merely (exists l : P $-> A, f == e $o l).
```

It is then not too hard to show that this notion of projectivity makes our Ext groups vanish:

▶ **Proposition 6.** *An abelian group P is projective if and only if* $\texttt{Ext}^1(P, A) = 0$ *for all A.*$^\diamond$

From the induction principle of $\mathbb{Z}$ it follows that $\mathbb{Z}$ is projective$^\diamond$ in the sense we defined above. Consequently $\texttt{Ext}^1(\mathbb{Z}, A) = 0$ for any abelian group A, and we will use this later on.

▶ Remark 7. There is a subtle point related to projectivity that merits discussion. Our definition of projectivity only requires the lift l to *merely* exist, but one could have asked for actual existence. In ordinary mathematics, there is no concept of "mere existence," and so when translating concepts into HoTT we have to carefully analyse whether something should be a structure or a property. In this case, our definition of projectivity is justified by Proposition 6. Moreover, if we had made projectivity a structure (i.e., asked for actual existence of l) then not even $\mathbb{Z}$ would be projective, which we need it to be.

3.2 Ext as a bifunctor

Some of the important structure of Ext^1 is captured by the fact that it defines a *bifunctor* $\mathrm{Ext}^1(-,-) : \mathrm{Ab}^{op} \times \mathrm{Ab} \to \mathrm{Ab}$. This means that $\mathrm{Ext}^1(-,-)$ is a functor in each variable and that the following "bifunctor law" holds:

$$\mathrm{Ext}^1(f,-) \circ \mathrm{Ext}^1(-,g) = \mathrm{Ext}^1(-,g) \circ \mathrm{Ext}^1(f,-). \tag{1}$$

We added a basic implementation of bifunctors to the `WildCat` library for our purposes, asserting the bifunctor law using the 2-graph structure:$^\diamond$

```
Class IsBifunctor {A B C : Type} `{IsGraph A, IsGraph B, Is1Cat C}
  (F : A -> B -> C) := {
    bifunctor_isfunctor_10 : forall a, Is0Functor (F a);
    bifunctor_isfunctor_01 : forall b, Is0Functor (fun a => F a b);
    bifunctor_isbifunctor :
      forall a0 a1 (f : Hom a0 a1), forall b0 b1 (g : Hom b0 b1),
        fmap (F _) g $o fmap (flip F _) f
          $== fmap (flip F _) f $o fmap (F _) g }.
```

Here `flip` is the map which reverses the order of arguments of a binary function. We note that in order to state the bifunctor law, we only require `F` to be a 0-functor in each variable. As such we only include those instances in this class.

The bifunctor instance of $\texttt{Ext}^1$ will come from a bifunctor instance of `AbSES`, so we work with the latter. First of all, $\texttt{AbSES} : \texttt{AbGroup}^{op} \to \texttt{AbGroup} \to \texttt{Type}$ becomes a 0-functor in each variable by pulling back and pushing out, respectively.

▶ **Lemma 8.** *Let $g : B' \to B$ be a homomorphism of abelian groups. For any short exact sequence $A \to E \to B$, we have a short exact sequence $A \to g^*(E) \to B'$.*$^\diamondsuit$ *Moreover, if E is trivial, then so is the short exact sequence$g^*(E)$.*$^\diamondsuit$

Dually, one can push out a short exact sequence $A \to E \to B$ along a map $f : A \to A'$ to get a short exact sequence $A' \to f_*(A) \to B$.$^\diamondsuit$

We supply careful proofs that pushout and pullback respect composition of pointed maps$^\diamondsuit$ and homotopies between maps,$^\diamondsuit$ and that pushing out along the identity map gives the pointed identity map.$^\diamondsuit$ These identities could be shown with shorter proofs, however in Section 4 we will have to prove coherences involving the paths constructed here, and these coherences are simpler to solve when phrased in terms of path data. In any case, these proofs make `AbSES` into a 1-functor in each variable.$^{\diamondsuit\diamondsuit}$

For the bifunctor law we make use of the following proposition, which is remarkably useful for showing that a given extension is a pullback of another one.

▶ **Proposition 9.** *Suppose given the following diagram with short exact rows:*

$$\begin{tikzcd}
A \arrow[r] \arrow[d, "\alpha"] & E' \arrow[r] \arrow[d] & B' \arrow[d, "g"] \\
A \arrow[r] & E \arrow[r] & B
\end{tikzcd}$$

If $\alpha = \mathrm{id}$ then the top row is equal to the pullback of the bottom row along g.$^\diamondsuit$

Proof. Since the right square commutes, we get a map $E' \to g^*(E)$ by the universal property of the pullback. This map respects the inclusions and projections, and therefore defines a path by Lemma 4. ◀

There is a dual statement for pushouts in which the rightmost map must be the identity.$^\diamondsuit$

▶ **Corollary 10.** *Any diagram with short exact rows as follows yields a path $f_*(E) = g^*(F)$.*$^\diamondsuit$

$$\begin{tikzcd}
A \arrow[r] \arrow[d, "f"] & E \arrow[r] \arrow[d] & B' \arrow[d, "g"] \\
A' \arrow[r] & F \arrow[r] & B
\end{tikzcd}$$

The corollary lets us swiftly show bifunctoriality:

▶ **Proposition 11.** *The binary map* `AbSES` : `AbGroup`^{op} $\to$ `AbGroup` $\to$ `Type` *is a bifunctor.*$^\diamondsuit$

Proof. Consider a short exact sequence $A \to E \to B$ along with two homomorphisms $f : A \to A'$ and $g : B' \to B$. There is an obvious diagram with short exact rows:

$$\begin{tikzcd}
A \arrow[r] \arrow[d, "f"] & g^*(E) \arrow[r] \arrow[d] & B' \arrow[d, "g"] \\
A' \arrow[r] & f_*(E) \arrow[r] & B
\end{tikzcd}$$

which by the previous corollary yields a path $f_*(g^*(E)) = g^*(f_*(E))$, as required. ◀

▶ Remark 12. The results from Section 3.3 will show that `AbSES` is an *H-space.*$^\diamondsuit$ Combining this with [1, Lemma 2.6]$^\diamondsuit$, we deduce that `AbSES` is a bifunctor into pointed types. This does not play a role in the rest of the paper, however.

3.3 The Baer sum

The *Baer sum* is a binary operation on $\mathrm{Ext}^1(B, A)$ which makes it into an abelian group. Given two extensions $E, F : \mathrm{Ext}^1(B, A)$ their Baer sum is defined as

$$E + F :\equiv \Delta^*\nabla_*(E \oplus F)$$

where $E \oplus F$ is the point-wise direct sum, $\nabla(a, b) :\equiv a_0 + a_1 : A \oplus A \to A$ is the codiagonal map, and $\Delta(b) :\equiv (b, b) : B \to B \oplus B$ is the diagonal map.

Together with Dan Christensen and Jacob Ender, we have implemented the Baer sum in `Algebra.AbSES.BaerSum`. We define this operation on the level of short exact sequences and then descend the operation to the set $\texttt{Ext}^1$ by truncation-recursion.$^\diamondsuit$

```
Definition abses_baer_sum `{Univalence} {B A : AbGroup}
  : AbSES B A -> ABSES B A AbSES B A
  := fun E F => abses_pullback ab_diagonal
                  (abses_pushout ab_codiagonal (abses_direct_sum E F)).

Definition baer_sum `{Univalence} {B A : AbGroup}
  : Ext B A -> Ext B A -> Ext B A.
Proof.
  intros E F; strip_truncations.
  exact (tr (abses_baer_sum E F)).
Defined.
```

Above, the `strip_truncations` tactic is a helper for doing truncation-recursion; it lets us assume that both E and F are elements of $\texttt{AbSES}(B, A)$ in order to map into the set $\texttt{Ext}^1(B, A)$. We then simply form the Baer sum of E and F on the level of short exact sequences before applying `tr` to the result.

The formalisation that the Baer sum makes $\texttt{Ext}^1(B, A)$ into an abelian group closely follows the "second proof" of [4, Theorem III.2.1].

▶ **Theorem 13.** *The set $\texttt{Ext}^1(B, A)$ is an abelian group under the Baer sum operation.*$^\diamondsuit$

The proof can be done entirely by chaining together equations once the bifunctoriality of $\texttt{Ext}^1$ has been established along with its interaction with direct sums. To illustrate this, we prove that pushouts respect the Baer sum:

▶ **Proposition 14.** *Let $\alpha : A \to A'$ be a homomorphism of abelian groups. For any abelian group B, pushout defines a group homomorphism $\alpha_* : \texttt{Ext}^1(B, A) \to \texttt{Ext}^1(B, A')$.*$^\diamondsuit$

Proof. Using bifunctoriality of $\texttt{Ext}^1$ and naturality of $\oplus$, we have:

$$\begin{aligned}\alpha_*(E + F) &= \Delta^*(\alpha_*\nabla_*(E \oplus F)) = \Delta^*(\nabla_*(\alpha_* \oplus \alpha_*)_*(E \oplus F))\\ &= \Delta^*(\nabla_*(\alpha_*E \oplus \alpha_*F)) \equiv \alpha_*E + \alpha_*F\end{aligned}$$ ◀

Similarly, pullback defines a group homomorphism as well.$^\diamondsuit$ These results make $\texttt{Ext}^1$ into a bifunctor valued in abelian groups.$^\diamondsuit$

4 The pullback fibre sequence

The main goal of this section is to explain and prove the following mathematical result, and to discuss its formalisation$^\diamondsuit$ along with some applications.

▶ **Theorem 15.** *Let $A \xrightarrow{i} E \xrightarrow{p} B$ be a short exact sequence of abelian groups. For any abelian group G, pullback yields a fibre sequence: $\texttt{AbSES}(B, G) \xrightarrow{p^*} \texttt{AbSES}(E, G) \xrightarrow{i^*} \texttt{AbSES}(A, G)$.*$^\diamondsuit$

A sequence of pointed maps $F \xrightarrow{i} E \xrightarrow{p} B$ is a **fibre sequence** if $p \circ i$ is pointed-homotopic to the constant map, and the induced map $F \to \texttt{fib}_p$ is an equivalence. Any fibre sequence induces a long exact sequence of homotopy groups [7, Theorem 8.4.6]:

$$\cdots \to \pi_n(F) \to \pi_n(E) \to \pi_n(B) \to \cdots \to \pi_0(F) \to \pi_0(E) \to \pi_0(B).$$

As such, fibre sequences capture a lot of data, which will be apparent from our applications below.

In the situation of our theorem, it is immediate from functoriality and exactness of E that $i^* \circ p^*$ is constant. Therefore our goal is to show that the induced map $c : \texttt{AbSES}(B, G) \to \texttt{fib}_{i^*}$ is an equivalence.[4] We will do this by showing that the fibre of c is contractible, and we proceed in steps. A key part of the formalisation is to work with path data instead of actual paths, since the former has better computational properties. We will simply use $E = F$ to denote path data, and refer to it as such, in this section.

▶ **Lemma 16.** *Let $G \to F \to E$ be a short exact sequence, and assume path data $p : i^*(F) =$ $\texttt{pt}$. Then we construct a short exact sequence $G \to F/A \to B$.*$^\diamondsuit$

Proof. The path data p means that the sequence $i^*(F)$ splits. Thus we can form the cokernel F/A as in the diagram:

$$\begin{tikzcd}
 & i^*(F) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "i"] \arrow[dl, dashed] \\
G \arrow[ur, bend left] \arrow[r, "j"] \arrow[dr, dotted, bend right] & F \arrow[r] \arrow[d, dashed] & E \arrow[d, "p"] \\
 & F/A \arrow[r, dotted] & B
\end{tikzcd}$$

The two maps $G \to F/A \to B$ are given by composition and the universal property of the cokernel, respectively. It is clear that this forms a complex and that the second map is an epimorphism, since it factors one. To see that the map $G \to F/A$ is an injection, suppose $g : G$ is sent to $0 : F/A$. Then $j(g)$ is in the image of some $a : A$ by $A \to F$. But the map $i^*(F) \to F$ is an injection, being the pullback of one, and so using the path data we get an equality $(g, 0) = (0, a)$ in $G \oplus A$. Of course, this implies that $g = 0$, as required.

Exactness of $G \to F/A \to B$ follows from a straightforward diagram chase. ◀

The diagram above exhibits F as the pullback of F/A along p^*, yielding:

▶ **Lemma 17.** *We have path data $q : p^*(F/A) = F$.*$^\diamondsuit$

Thus we have given a preimage F/A of F under p^*. To show that the fibre of c is inhabited we will show that $c(F/A) = (F, p)$, which is a path in $\texttt{fib}_{i^*}$. We express all of this in terms of path data, and such a path in $\texttt{fib}_{i^*}$ then corresponds to path data $q : p^*(F/A) = F$ which makes the following triangle commute:$^\diamondsuit$

$$\begin{tikzcd}
i^*p^*(F/A) \arrow[rr, "i^*(q)"] \arrow[dr] & & i^*(F) \arrow[dl, "p"] \\
 & G \oplus A &
\end{tikzcd} \qquad (2)$$

[4] The map c is called $\texttt{cxfib}$ in the code.

where the rightmost map comes from i^*p^* being trivial. The key reason we have formulated things in terms of path data is so that the maps in the triangle above simply compute, because they have all been concretely constructed.

In the following, c refers to the map which lands in $\texttt{fib}_{i^*}$ expressed in terms of path data.$^\diamond$

▶ **Lemma 18.** *We have $q : c(F/A) = (F, p)$ in $\texttt{fib}_{i^*}$.*$^\diamond$

Proof. The previous lemma already yields path data $q : p^*(F/A) = F$, thus it remains to show that the triangle in Equation (2) commutes. The way the maps have been constructed, it's easiest to show this after flipping the triangle so that it starts at $G \oplus A$ and ends at $i^*p^*(F/A)$. (This is fine since all the maps are isomorphisms.) Thus we are comparing two maps out of a biproduct into a pullback. To check whether they are equal, we can check it on each inclusion of the biproduct and after projecting out of the pullback. In each of these cases one obtains diagrams which commute, but checking this is somewhat involved. Fortunately, by our having carefully crafted the path data involved, the maps simply compute and Coq is able to reduce the goal to a simple computation. ◀

Putting together the three previous lemmas, we have constructed a section of $c : \texttt{AbSES}(B, G) \to \texttt{fib}_{i^*}$. To conclude that c is an equivalence, we show that any other element in the fibre of (F, p) is equal to F/A.

▶ **Lemma 19.** *Suppose $G \to Y \to B$ is a short exact sequence, and let $q' : c(Y) = (F, p)$ in $\texttt{fib}_{i^*}$. Then $(F/A, q) = (Y, q')$ in the fibre of c over (F, p).*$^\diamond$

Proof. Under our assumptions, we have the composite map $\phi : G \oplus A \to i^*p^*(Y) \to p^*(Y)^\diamond$ which by a diagram chase can be seen to be the inclusion $G \to p^*(Y)$ on one component, and $(0, p) : A \to p^*(Y)$ on the other.$^\diamond$. Consequently, the composite $\texttt{pr}_1 \circ \phi \circ \texttt{in}_A : A \to Y$ is trivial. By the universal property of the cokernel, we get an induced map $F/A \to Y$. Once again, by our careful construction of all the maps involved, it is straightforward to simply compute that this map defines path data $F/A = Y$ and moreover that this path lifts to a path in the fibre of c. There is a coherence between three paths in $\texttt{AbSES}(A, G)$ which is trivially satisfied, since $\texttt{AbSES}(A, G)$ is a 1-type. ◀

The final lemma implies that the fibres of c are contractible, which means that c is an equivalence, concluding the proof of Theorem 15. We now turn our attention to two applications of this theorem. The first application requires a lemma.

▶ **Lemma 20.** *Let $g : B' \to B'$ be a homomorphism of abelian groups. For any A, the following diagram commutes, where the vertical isomorphisms are all given by Proposition 5:*$^\diamond$

$$\begin{tikzcd}
\Omega\,\texttt{AbSES}(B, A) \arrow[r, "\Omega(g^*)"] \arrow[d, "\sim"] & \Omega\,\texttt{AbSES}(B', A) \arrow[d, "\sim"] \\
\texttt{Hom}(B, A) \arrow[r, "\phi \mapsto \phi \circ g"] & \texttt{Hom}(B', A).
\end{tikzcd}$$

Proof. Let $p : A \oplus B = A \oplus B$ be an element of the upper left corner, seen as path data. By path induction, one can easily show that the action of $\Omega(g^*)$ on paths is given by pulling back the path data. (Formally, one first proves this for paths with free endpoints, then you can specialise to loops.) This means that the following diagram commutes

$$\begin{tikzcd}
B' \arrow[r] \arrow[dr, bend right, "(0,g)"'] & A \oplus B' \arrow[r, "\Omega(g^*)(p)"] \arrow[d, "\mathrm{id} \oplus g"] & A \oplus B' \arrow[d, "\mathrm{id} \oplus g"] \arrow[dr, bend left] & \\
& A \oplus B \arrow[r, "p"] & A \oplus B \arrow[r] & A
\end{tikzcd}$$

where we have used the functions underlying the path data p and $\Omega(g^*)(p)$. The composite around the top right corner of this diagram is the result of sending p around the top right corner of the diagram in the statement, and vice-versa for the bottom left corner. In other words, the diagram in the statement commutes. ◀

▶ **Proposition 21** ([4, Theorem III.3.4]). *We have an exact sequence of abelian groups:*◇

$$\begin{tikzcd}
0 \arrow[r] & \mathtt{Hom}(B,G) \arrow[r, "p^*"] & \mathtt{Hom}(E,G) \arrow[r, "i^*"] & \mathtt{Hom}(A,G) \arrow[dll, rounded corners] \\
& \mathtt{Ext}^1(B,G) \arrow[r, "p^*"] & \mathtt{Ext}^1(E,G) \arrow[r, "i^*"] & \mathtt{Ext}^1(A,G).
\end{tikzcd}$$

Proof. This sequence comes from the long exact sequence of homotopy groups [7, Theorem 8.4.6] associated to the fibre sequence of Theorem 15, using Proposition 5 and the previous lemma to identify $\Omega\,\mathtt{AbSES}(-,G)$ with $\mathtt{Hom}(-,G)$. ◀

▶ Remark 22. The connecting map $\mathtt{Hom}(A,G) \to \mathtt{Ext}^1(B,G)$ in the sequence above is given by $\phi \mapsto \phi_* E$. Showing this from the fibre sequence is somewhat tedious; we have a proof on paper, but not yet a formalisation. Instead, we have formalised a direct proof that the map just stated yields exactness of the sequence.◇◇

We apply the six-term exact sequence to compute Ext groups of cyclic groups:

▶ **Corollary 23** ([4, Proposition III.1.1]). *For any $n > 0$ and abelian group A, we have*◇

$$\mathtt{Ext}^1(\mathbb{Z}/n, A) \cong A/n.$$

Proof. The short exact sequence $\mathbb{Z} \xrightarrow{n} \mathbb{Z} \to \mathbb{Z}/n$ yields a six-term exact sequence

$$\cdots \to \mathtt{Hom}(\mathbb{Z},A) \xrightarrow{n^*} \mathtt{Hom}(\mathbb{Z},A) \to \mathtt{Ext}^1(\mathbb{Z}/n,A) \to \mathtt{Ext}^1(\mathbb{Z},A) \to \cdots$$

in which the term $\mathtt{Ext}^1(\mathbb{Z},A)$ vanishes since $\mathbb{Z}$ is projective.◇◇ This means that the map $\mathtt{Hom}(\mathbb{Z},A) \to \mathtt{Ext}^1(\mathbb{Z}/n,A)$ is the cokernel of the preceding map. By identifying $\mathtt{Hom}(\mathbb{Z},A)$ with A, the claim follows. ◀

5 The long exact sequence

In this section we describe our formalisation of the higher Ext groups $\mathtt{Ext}^n(B,A)$ and the contravariant long exact sequence associated to them. We begin by discussing the definition of $\mathtt{Ext}^n(B,A)$, which follows [4, Chapter III.5] with minor discrepancies.

The formalisation of this section is in the separate repository `Yoneda-Ext` whose contents will be contributed to the Coq-HoTT library. The `README` file of that repository explains the connection between the formalisation and the results we describe below.

5.1 The type of length-n exact sequences

We start by defining a type $\mathtt{ES}^n$ which we will equip with an equivalence relation by which $\mathtt{Ext}^n$ will be the quotient. These constructions will yield functors, which we explain.

The type $\mathtt{ES}^n(B,A)$ of **length-n exact sequences** is recursively defined as

```
Fixpoint ES (n : nat) : AbGroup^op -> AbGroup -> Type
  := match n with
    | 0%nat => fun B A => Hom B A
    | 1%nat => fun B A => AbSES B A
    | S n => fun B A => exists M, (ES n M A) * (AbSES B M)
    end.
```

Thus $\texttt{ES}^0(B, A)$ is definitionally $\texttt{Hom}(B, A)$, and $\texttt{ES}^1(B, A)$ is definitionally $\texttt{AbSES}(B, A)$. One could also have started the induction at $n = 1$ instead of $n = 2$, but it is convenient to have this definitional equality at level $n = 1$. The functoriality of $\texttt{ES}^n$ is inherited from AbSES and defined in the obvious way by pulling back and pushing out. For $n > 0$, an element of $\texttt{ES}^{n+1}(B, A)$ is denoted by $(F, E)_M$, with the obvious meaning.

We make $\texttt{ES}^n(B, A)$ into a pointed type by recursion, using the trivial abelian group in the place of M in the inductive step.

▶ **Definition 24.** *The **splice** operation is defined as*

$$F \circledcirc E :\equiv (F, E)_B : \texttt{ES}^n(B, A) \to \texttt{AbSES}(C, B) \to \texttt{ES}^{n+1}(C, A).$$

By induction it's easy to define a general splicing operation in which the second parameter can have arbitrary length as well, but we only need the restricted version above.

Now we equip $\texttt{ES}^n(B, A)$ with a relation.

▶ **Definition 25.** *We define a relation* es_zig $: \texttt{ES}^n(B, A) \to \texttt{ES}^n(B, A) \to \texttt{Type}$ *recursively as follows. For $n = 0, 1$,* es_zig *is simply the identity type. For the recursive step, two elements $(F, E)_M$ and $(Y, X)_N$ are related whenever there exists a homomorphism $f : \texttt{Hom}(M, N)$ such that $f_*(E) = X$ (using functoriality of* AbSES*) and* es_zig$(F, f^*(Y))$ *holds (using functoriality of* $\texttt{ES}^n$*).*

The relation es_zig generates an equivalence relation es_eqrel whose propositional truncation is es_meqrel. The functoriality of $\texttt{ES}^n$ respects these relations.

▶ **Definition 26.** *The pointed set* $\texttt{Ext}^n(B, A)$ *is the quotient of* $\texttt{ES}^n(B, A)$ *by the equivalence relation* es_meqrel*.*

By pushing out and pulling back extensions, Ext^n becomes a functor in each variable as well. Moreover, we have equalities $f^*(F) \circledcirc E = F \circledcirc f_*(E)$ whenever this expression makes sense, by the definition of es_zig.

▶ Remark 27. The definition of $\texttt{Ext}^{n+1}(B, A)$ may be conceptually understood as the $(n+1)$-fold tensor product of functors $\texttt{Ext}^{n+1}(B, A) = \texttt{Ext}^n(-, A) \otimes \texttt{Ext}^1(B, -)$ [3, Theorem 9.20]. In our setup, this is a tensor product of Set-valued functors, which can be made into an abelian group by a construction similar to the Baer sum of Section 3.3 (though we have not yet formalised this). Alternatively, one could define $\texttt{Ext}^{n+1}(B, A)$ as the $(n + 1)$-fold tensor product of functors *into abelian groups.* [2, Lemma 2.1] implies that these two definitions coincide. We have chosen the present approach because we do not know of a direct construction of the long exact sequence for the latter approach.

5.2 The long exact sequence

We now begin working towards the long exact sequence, following the proof of [4, Theorem XII.5.1]. Let us first recall the statement:

▶ **Theorem 28.** *Let $A \xrightarrow{i} E \xrightarrow{p} B$ be a short exact sequence of abelian groups. For any abelian group G, there is a long exact sequence by pulling back:*

$$\cdots \xrightarrow{i^*} \texttt{Ext}^n(A, G) \xrightarrow{- \circledcirc E} \texttt{Ext}^{n+1}(B, G) \xrightarrow{p^*} \texttt{Ext}^{n+1}(E, G) \xrightarrow{i^*} \cdots$$

The proof in [4] first discusses the six-term exact sequence, which we proved as Proposition 21. It then reduces the question to exactness at the domain of the connecting map (Lemma XII.5.2, loc.cit.), and proves exactness at that spot using Lemma XII.5.3, XII.5.4, and XII.5.5. We will show the three latter lemmas, then directly prove exactness at the other spots, essentially "in-lining" Lemma XII.5.2.

The various constructions we need to do are simpler to carry out on the level of $\mathtt{ES}^n$ as opposed to $\mathtt{Ext}^n$. For this reason we work and formulate things in terms of the former, and then deduce the desired statement for the latter.

Before attacking Lemma XII.5.3, we show the following:

▶ **Lemma 29.** *Consider two pairs of short exact sequences which can be spliced:*

$$(A \xrightarrow{l} Y \xrightarrow{s} B',\ B' \xrightarrow{k} X \xrightarrow{r} C) \qquad (A \xrightarrow{j} F \xrightarrow{q} B,\ B \xrightarrow{i} E \xrightarrow{p} C)$$

Given $\mathtt{es\_zig}(Y \circledcirc X, F \circledcirc E)$*, we have maps* $\mathtt{fib}_{s_*}(X) \to \mathtt{fib}_{q_*}(E)$ *and* $\mathtt{fib}_{i^*}(F) \to \mathtt{fib}_{k^*}(Y)$*.*

Proof. We only describe the first map since the second is analogous. The zig from $Y \circledcirc X$ to $F \circledcirc E$ gives a homomorphism $f : B' \to B$ along with two paths $f^*(F) = Y$ and $f_*(X) = E$. Let $G : \mathtt{fib}_{s_*}(X)$; by path induction we may assume $q_*(G) \equiv X$. The path $f^*(F) = Y$ means we have a commuting diagram:

$$\begin{tikzcd} A \arrow[r, "l"] \arrow[d, equal] & Y \arrow[r, "s"] \arrow[d, "\phi"] & B' \arrow[d, "f"] \\ A \arrow[r, "j"] & F \arrow[r, "q"] & B \end{tikzcd}$$

Thus $\phi_*(G))$ defines an element of $\mathtt{fib}_{q_*}(E)$ by $q_*(\phi_*(G)) = f_*(s_*(G)) \equiv f_*(X) = E$. ◀

▶ **Lemma 30** ([4, Lemma XII.5.3])**.** *Given two short exact sequences* $A \xrightarrow{j} F \xrightarrow{q} B$ *and* $B \xrightarrow{i} E \xrightarrow{p} C$*, the following types are logically equivalent:*

1. $\mathtt{fib}_{i^*}(F)$*;*
2. $\mathtt{fib}_{q_*}(E)$*;*
3. $\mathtt{es\_eqrel}(\mathtt{pt}, F \circledcirc E)$*.*

Proof. The logical equivalence of between (1) and (2) is as described in Mac Lane. Moreover, the implication (2) to (3) is clear by the definition of `es_zig`. We need to show that (3) implies (1), and we proceed by induction on the length of the zig-zag.

In the base case we have an actual equality $\mathtt{pt} = F \circledcirc E$, in which case (1) clearly holds. For the inductive step, suppose we have two short exact sequences $A \xrightarrow{l} Y \xrightarrow{s} B'$ and $B' \xrightarrow{k} X \xrightarrow{r} C$ such that $Y \circledcirc X$ is related to `pt` by a length n zig-zag, and we have either zig or a zag relating $Y \circledcirc X$ to $F \circledcirc E$. If we have a zig, then we use the induction hypothesis to get an element of $\mathtt{fib}_{s_*}(X)$ to which we apply the map $\mathtt{fib}_{s_*}(X) \to \mathtt{fib}_{q_*}(E)$ from the previous lemma. This suffices since (1) and (2) are logically equivalent.

If we have a zag, then the previous lemma gives a map $\mathtt{fib}_{k^*}(Y) \to \mathtt{fib}_{i^*}(F)$, so we are done by the induction hypothesis. ◀

We reformulate condition (2) in a manner that generalises to $\mathtt{ES}^n$.

```
Definition es_ii_family `{Univalence} {n : nat} {C B A : AbGroup}
  : ES n.+1 B A -> ES 1 C B -> Type
  := fun E F => { alpha : { B' : AbGroup & B' $-> B }
                          & (es_eqrel pt (es_pullback alpha.2 E))
                            * (hfiber (abses_pushout alpha.2) F) }.
```

▶ **Lemma 31** ([4, Lemma XII.5.4]). *In the situation of the previous lemma, the types* $\texttt{fib}_{q_*}(E)$ *and* $\texttt{es\_ii\_family}(F, E)$ *are logically equivalent.*

Whereas Mac Lane appeals to the six-term exact sequence to prove this lemma, we give a direct construction—see the formalisation.

In order to show Lemma XII.5.3, we prove a higher analogue of Lemma 29. This analogue is phrased in terms of the "relation fibre" `rfiber`, which takes the fibre of a point with respect to a relation.

▶ **Lemma 32.** *Let* $n > 0$ *and consider* $Y : \texttt{ES}^n(B', A)$*,* $F : \texttt{ES}^n(B, A)$*, and two short exact sequences* $B' \xrightarrow{k} X \to C$ *and* $B \xrightarrow{i} E \to C$*. Given* $\texttt{es\_zig}(Y \circledcirc X, F \circledcirc E)$*, we have maps* $\texttt{rfiber}_{i^*}(F) \to \texttt{rfiber}_{k^*}(Y)$ *and* $\texttt{es\_ii\_family}(Y, X) \to \texttt{es\_ii\_family}(F, E)$*.*

▶ **Lemma 33** ([4, Lemma XII.5.5]). *Let* $n > 0$*,* $F : \texttt{ES}^n(B, A)$*, and* $E : \texttt{ES}^1(C, B)$*. The following types are equivalent:*

1. $\texttt{fib}_{i^*}(E)$*;*
2. $\texttt{es\_ii\_family}(F, E)$*;*
3. $\texttt{es\_eqrel}(\texttt{pt}, F \circledcirc E)$

Proof. We first prove an auxiliary lemma which shows that if the three statements are equivalent for a given n, then (1) and (2) are equivalent for $n + 1$. The base case for this lemma is simply Lemma 30. For the inductive step, our auxiliary lemma gives us that (1) and (2) are equivalent. It is easy to show that (2) always implies (3), so it remains to show that (3) implies either (1) or (2). For this we induct on the length of a zig-zag, and use the equivalence of (1) and (2) along with the previous lemma, similarly (at least in structure) to the proof of Lemma 30. ◀

With this lemma at hand, and using similar methods to the ones presented here, we follow the proof of [4, Lemma 5.2] to deduce exactness of the long sequence of Theorem 28.

6 Conclusion

We have presented a formalisation of the theory of Yoneda Ext in the novel setting of homotopy type theory, starting from the basic definition of a short exact sequence and arriving at the long exact sequence, with various related results along the way. Many results have already been contributed to the Coq-HoTT library under the `Algebra.AbSES` namespace which currently weighs in at about 2900 lines of code (whitespace and comments included). This excludes the various contributions made to other parts of the library. In addition, the code for the long exact sequence is currently at about 1350 lines.

The formalisation covers a substantial part of chapters III.1-3, III.5, and XII.5 of [4], but also extends beyond the classical theory. In particular, Theorem 15 is new mathematical result even for classical Yoneda Ext, to which our proof also applies. This theorem presents one of the most challenging parts of this formalisation, as it requires managing considerable amounts of coherence. The other challenging part was the long exact sequence, whose proof involves an intricate induction and numerous constructions. By formalising these theorems we have not only established their correctness but also contributed evidence of the feasibility of dealing with sophisticated mathematical structures in a proof assistant like Coq.

References

1 Ulrik Buchholtz, J. Daniel Christensen, Jarl G. Taxerås Flaten, and Egbert Rijke. Central H-spaces and banded types, 2023. arXiv:2301.02636, doi:10.48550/ARXIV.2301.02636.

2 René Guitart and Luc Van den Bril. Calcul des satellites et présentations des bimodules à l'aide des carrés exacts. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 24(3):299–330, 1983. URL: http://archive.numdam.org/item/CTGDC_1983__24_3_299_0/.

3 René Guitart and Luc Van den Bril. Calcul des satellites et présentations des bimodules à l'aide des carrés exacts (2e partie). *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 24(4):333–369, 1983. URL: http://archive.numdam.org/item/CTGDC_1983__24_4_333_0/.

4 Saunders Mac Lane. *Homology*. Springer, 1963.

5 Vladimir S. Retakh. Homotopic properties of categories of extensions. *Russian Mathematical Surveys*, 41(6):217–218, 12 1986. doi:10.1070/rm1986v041n06abeh004237.

6 Egbert Rijke. Introduction to homotopy type theory, 2022. URL: https://arxiv.org/abs/2212.11082, doi:10.48550/ARXIV.2212.11082.

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

8 Floris van Doorn. *On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory*. PhD thesis, Carnegie Mellon University, 2018. arXiv:1808.10690.

9 Nobuo Yoneda. On Ext and exact sequences. *Journal of the Faculty of Science, the University of Tokyo Section I*, 8:507–576, 1960.