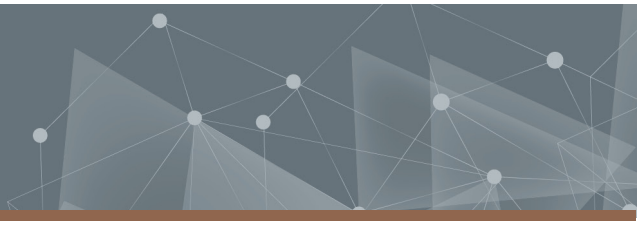
CHALMERS
UNIVERSITY OF TECHNOLOGY



Investigating abelian categories in univalent type theory

With a focus on the category of Rmodules

Master's thesis in Engineering mathematics and computational science

DAVID ELINDER

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Investigating abelian categories in univalent type theory

With a focus on the category of Rmodules

David Elinder

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2021

Investigating abelian categories in univalent type theory
With a focus on the category of Rmodules
David Elinder

Supervisor: Thierry Coquand, Department of Computer Science and Engineering
Examiner: Martin Raum, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone: +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Investigating abelian categories in univalent type theory
With a focus on the category of Rmodules
David Elinder
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In 1976 the first computer checked proof was made to prove the four color theorem. This raised questions about how these computations should be interpreted to create a formal proof, as well as the validity of the proof given. In 2005 the proof was recreated in the proof assistant Coq, both showing the structure of the proof and proving its correctness as long as one trusts the correctness of the Coq kernel.

Other examples of formalized proofs are Feit-Thompson (in finite group theory) and Kepler's conjecture. One area of mathematics that has however been difficult to formalize is category theory because of its high level of abstraction. With the development of homotopy type theory and research on the consequences of the univalence axiom formalizations of concepts in category theory are now possible. The goal of this thesis is to formalize the concept of abelian categories and to prove that, for a commutative ring R, the category of R modules is abelian. These concepts will also be formalized in the proof checker Cubical Agda.

Since we represent these concepts in the setting of a univalent foundation, this will automatically ensure that all our definitions made in this framework are invariant under algebraic isomorphisms and category theoretical equivalences. This will enable us to create formulations that act naturally both in the context of category theory and in the context of algebra without sacrificing either correctness or clarity in order to achieve these results. It will also enable us to use proofs that closely mirror those we are used to from these fields.

Keywords: Category theory, Abelian category, Univalent type theory, Cubical Agda.

Acknowledgements

I would like to thank my supervisor Thierry Coquand for his guidance during the development of this thesis. I would also like to thank my examiner Martin Raum for helpful insight into the realm of abelian categories. I would like to thank Anders Mortberg for insight into the SIP representation and for inviting me to a meeting with the Agda development team. I would also like to thank the members of the agda development team for their work on the cubical agda standard library, and for helpful insight about algebraic structures in cubical agda. Lastly, I would like to thank my many friends at Chalmers who have made the time during the pandemic bearable and who have come to digital meetups when no other were allowed. I would especially want to thank Erik von Brömssen who has been a great help in discussing both the thesis work and life during the pandemic in general.

David Elinder, Gothenburg, June 2021

Table of Contents

1. Introduction

In 1976 the first computer checked proof was made to prove the four color theorem. Some criticized the proof and claimed that it was unclear how the computations relate to the proof. Some also claimed that the correctness of the proof relies on the absence of coding mistakes. In 2005 the proof was recreated in the proof assistant Coq, both showing the structure of the proof and proving its correctness as long as one trusts the correctness of the Coq kernel [1].

There have also been formalizations of proofs from many other fields of mathematics, such as the Feit-Thompson (in finite group theory) and Kepler's conjecture. One area of mathematics that has however been difficult to formalize is category theory because of its high level of abstraction. Category theory also has the concept of isomorphism. In category theory objects that are isomorphic are considered similar and can often be used interchangeably, despite the fact that they may, for instance, be different objects as sets. Since this concept of isomorphism is separate from our usual definition of equality it was for a long time unclear how this should be represented in type theory.

In 2013 the book "Homotopy Type Theory - Univalent Foundations of Mathematics" [2] was released which aimed to provide a framework in type theory that could formalize abstract fields of mathematics in a manner that would be intuitively understood by human beings, not just the proof checking programs. Among the fields discussed in the book are homotopy theory and category theory. In homotopy type theory the concept of equality is defined as abstract paths between elements of types and two types are equivalent if there is a function between the types that is invertible. To unify these concepts the powerful univalence axiom was added, which states that the concept of equality of types is equivalent to the concept of equivalence of types. For more information see chapter 2 of this thesis.

Using this foundation of homotopy type theory this thesis aims to investigate how well this framework can represent abelian categories with a focus on the category of Rmodules. It will also investigate what challenges arise when we try to formalize category theory in univalent type theory, and is a small step on the way to finding how well higher category theory can be encoded in univalent type theory.

All the results of this thesis have been implemented in the cubical type theory libraries of the Agda proof assistant, which we will refer to as "cubical Agda". Cubical type theory is a branch of homotopy type theory where the abstract paths are functions from the interval type $\mathbf{I}$ to the relevant types, similar to how we define paths in topology. For more information on cubical type theory see section

2.2. All the code created during the development of this thesis can be found at `https://github.com/ByteBucket123/ThesisWorkGitCopy` [3].

2. Homotopy type theory

In the following sections we will briefly go over the basic concepts of homotopy type theory. Many of these concepts are described in more detail in the book "Homotopy Type Theory - Univalent Foundations of Mathematics" [2] and references will be given where appropriate.

2.1 Type theory

This section will serve as a quick refresher on type theory, and will only go over the most basic concepts. For a more detailed overview the author suggests "Types and Programming Languages"[4] by Benjamin C. Pierce or the introductory chapter of [2].

Some standard notation to keep track of is that $t : T$ represents an element t of type T. Also we let $A \to B$ denote the type of functions with input of type A and output of type B. We will also write $A := B$ to denote "define A as B".

2.1.1 Propositions as types

One of the basic ideas of type theory is the idea of propositions as types. Here we view types as propositions and their elements as proofs of their corresponding proposition. These elements are sometimes also called witnesses. Thus a type T may be viewed as a true proposition if it has an element t. In the same spirit we can consider a function $f : A \to B$. This function takes a proof, also called witness, of A and gives a proof of B. Thus functions can be viewed as implications. Here A implies B since for all $a : A$ we get an element $f(a) : B$. Shortly, if A is a true proposition then B is also a true proposition. Alternatively we may read $f : (a : A) \to B$ as "For all a in A we can prove B".

From this perspective, we soon find two trivial types. The type $Unit$ with the only element $unit$, and the type $Empty$ which has no elements. Here it is clear that the type $Empty$ represents the trivially false proposition, since it has no elements and thus can not be proven. We also realize that $Unit$ represents the trivially true proposition, since we always know we can prove it with $unit$.

2.1.2 Universe levels

When considering two types A and B it is convenient to have a type that contains both of them. The naive definition for a type of all types inherits similar paradoxes

to what originally plagued basic set theory. Similarly to how these can be solved in set theory by set theory universes we will introduce type theoretic universes. The first is the universe U_0 and for any U_ℓ there is a universe $U_{\ell+1}$. This forms a hierarchy $U_0, U_1, U_2, \ldots$.

To represent that A is a type in the universe U_ℓ we will write $A : \text{Type } U_\ell$. We will not go into the construction of these but the reader should note that $U_\ell : \text{Type } U_{\ell+1}$ and that if a dependent type, such as a record, contains an arbitrary type of the universe U_ℓ then it must be in $U_{\ell+1}$ or above. For more information see section 1.3 in [2]. We will use the shortened notation Type to represent Type U_ℓ in cases where the universe U_ℓ is arbitrary or can be easily inferred from the context.

Note that since our universes form a hierarchy we may define $U_{\max(\ell,\ell')}$ as the largest of U_ℓ and $U_{\ell'}$. Also note that this means that $U_{\max(\ell',\ell)}$ is the same universe as $U_{\max(\ell,\ell')}$.

2.1.3 Type theory and set theory

When considering type theory it is also possible to take a more set-theoretic viewpoint. Here we may consider the type $\mathbb{N}$ which has two constructors, the first one returns an element of $\mathbb{N}$ denoted 0, and the other constructor, denoted succ, takes an element $n : \mathbb{N}$ and returns another element of $\mathbb{N}$. Since the Martin-Löf type theory is based on the concept of "no junk" all elements given by different constructors, or for different inputs to the same constructor are themselves different. Thus the elements 0 and succ n are different for all $n : \mathbb{N}$. For more information on Martin-Löf's type theory see [4].

We can now define some basic arithmetic with this type. We define a function $\_ + \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ by pattern matching on the inputs $n : \mathbb{N}$ and $m : \mathbb{N}$ where

$$\begin{aligned} &\_ + \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ &n + 0 = n \\ &n + (\text{succ } m) = \text{succ } (n + m). \end{aligned}$$

Note here that we have used simple recursion here to define $\_ + \_$. Now that we have a basic type for the set $\mathbb{N}$ we may want to start proving properties of $\mathbb{N}$. Here it is useful to merge our perspectives of types as sets and types as propositions.

An important notion to capture when talking about propositions is the notion of equality. In standard type theory we define this by the type $\_ \equiv \_$ which has only one constructor, namely refl. This constructor takes an element $a : A$ and returns an element of $a \equiv a$. Here it is important to note that for two elements $a : A$ and $a' : A$ we may have $a \equiv a'$ if a and a' are *judgmentally* equal. I will not go into the

exact definition of this, but this may be that they are equal except for some simple reductions, for example: $f(a) \equiv (\lambda a \to f(a))a$ or $f \equiv (\lambda a \to f(a))$.

Using this it is easy to prove that $\_ \equiv \_$ is an equivalence relation. First we see that $(a : A) \to a \equiv a$ is just the refl constructor. Then we note that to prove $a \equiv b \to b \equiv c \to a \equiv c$ we pattern match on the first element and realize that since refl is the only constructor of $a \equiv b$, and it gives an element $a \equiv a$ we have that b must be *judgmentally* equal to a. Thus the function is actually on the form $a \equiv a \to a \equiv c \to a \equiv c$, and we only need to return the second argument. Finally we prove $a \equiv b \to b \equiv a$ by again pattern matching on the first argument and realizing that the function is $a \equiv a \to a \equiv a$ and we can use refl to prove the output.

Before we leave equality there is one last concept we want to consider. We will want to prove that $(f : A \to B) \to a \equiv b \to f(a) \equiv f(b)$. This is a simple concept, but it is also one of the most important for proving propositions. The proof is once again pattern matching on $a \equiv b$, realizing b is *judgmentally equal* to a and then proving $fa \equiv fa$ by refl. We denote this proof as cong.

Now that we have the basics for equality we can do some simple arithmetic. We can prove $(n : \mathbb{N}) \to n \equiv n + 0$ by refl since $n + 0$ is defined as n. The proof of $(n : \mathbb{N}) \to n \equiv 0 + n$ is more tricky however, and it requires us to pattern match on n. For the 0 case we see that $0 + 0$ is defined as 0 and $0 \equiv 0$ by refl. For the succ n case we will have to prove succ $n \equiv 0 +$ succ n. Since $0 +$ succ n is defined as succ $(0 + n)$ we will have to prove succ $c \equiv$ succ $(0 + n)$ and if we here use cong on succ we will only need to prove that $n = 0 + n$, but this is what our function is proving! So we can use induction and we are done. We may write the proof as

$$
\begin{aligned}
&\text{AddZeroLeft} \ : (n : \mathbb{N}) \to n \equiv 0 + n \\
&\text{AddZeroLeft } 0 = \text{refl } 0 \\
&\text{AddZeroLeft } (\text{succ } n) = \text{cong } (\text{AddZeroLeft } n).
\end{aligned}
$$

A problem one might run into is that we might have two different definitions that we intuitively want to use interchangeably. If we defined the type *OneElem*, which has the only element $*$ then, since their definitions are very similar, it might be reasonable to want it to be equal to *Unit*. But since their types are different they are not equal. However, for any property we prove for *OneElem* there is a corresponding property, and a corresponding proof for *Unit*, which we receive by exchanging *OneElem* for *Unit* and $*$ for *unit* everywhere in the proof and in the definition of the property.

As an example, if we define HasOneElem $: (A : \text{Type}) \to (x\ y : A) \to x \equiv y$ then we could prove HasOneElem *OneElem* by pattern matching on the elements, seeing that both are $*$ and then proving $* \equiv *$ by refl $*$. It does not follow directly however

that HasOneElem *Unit* despite the fact that they have a very similar definition. To solve these and similar problems it will help to have a more general concept of equality. This leads us nicely into the main concept of homotopy type theory.

2.1.4 Concrete dependent types

Before we leave this brief look into type theory we will describe dependent types and $\Sigma-$Types. A dependent type is something whose type depends on some other type, or an element of a type. The most simple example is a list. A list of integers for example contains integers, but it is not the same type as a list of strings, despite the fact that they are both lists. Another example is a vector, which is defined as a list of a certain length. Here a vector of integers with 4 elements is not the same type as a vector of integers with 5 elements, since here the type depends also on the number of elements. This may be clearer if we look at some Agda pseudocode for their definitions. Now, consider

$$\begin{aligned}&\text{data List } (A : \text{Type}) : \text{Type where}\\ &\quad [] : \text{List } A\\ &\quad \_ :: \_ : A \to \text{List } A \to \text{List } A\end{aligned}$$

and

$$\begin{aligned}&\text{data Vec } (A : \text{Type}) \ (n : \mathbb{N}) : \text{Type where}\\ &\quad [] : \text{Vec } A\ 0\\ &\quad \_ :: \_ : \{n : \mathbb{N}\} \to A \to \text{Vec } A\ n \to \text{Vec } A\ (\text{succ } n).\end{aligned}$$

Here data is just the keyword to define a new type and we can clearly see that the type List depends only on the type A, which is the type of its elements, while Vec depends both on A and on an integer $n : \mathbb{N}$, which is the length if the vector. Thus when we take in a vector as an argument to a function we must either know the length of the vector beforehand, or be able to infer it. Here $\{n : \mathbb{N}\}$ means that Agda should be able to infer our n in question, since it is the length of the vector it will be given. Thus there is no need in demanding that the value n is given to the function. The type Vec might seem complicated, but there are occasions where it is useful to know the length of the vector. For example the function fst $: \text{List } A \to A$ that returns the first element of a list of type A will have a problem when the list is empty. Here we might have unexpected behavior or we might need to send an error message. This can all be avoided by instead having the function fst $: \{n : \mathbb{N}\} \to \text{Vec } A\ (\text{succ } n) \to A$ where we know that the list is not empty since 0 is not succ n for any $n : \mathbb{N}$.

2.1.5 Σ-types and fibers

For $a : A$ and a $P : A \to \text{Type}$ we define $\Sigma\ A\ (\lambda a \to P\ a)$ or simply $\Sigma\ A\ P$ as a pair of an element $a : A$ and an element of $P(a)$ for our particular a. We will allow ourselves to use the syntax $\Sigma\ (a : A)\ (P\ a)$ to represent $\Sigma\ A\ P$. Here it is easy to define the functions fst $: \Sigma\ A\ P \to A$ and snd $: (\sigma : \Sigma\ A\ P) \to P(\text{fst } \sigma)$. A pair of two types A and B is a special case of a Σ-type where our P is always the type

B and does not depend on a. Thus we can define $A \times B$ as $\Sigma\ A\ (\lambda a \to B)$. We will also let (a, pa) denote an element in $\Sigma\ A\ P$, meaning that $(a, pa) : \Sigma\ A\ P$ is notation for $a : A$ and $pa : P\ a$.

If we consider Σ-types from the point of view of types as proposition we may think of $\Sigma\ (a : A)\ (P\ a)$ as "There exists a in A such that there is a proof of $P\ a$". Similarly we may think of $A \times B$ as "There exists proofs of both A and B". This intuition is useful for understanding how we use these types, but there is an issue with this intuition. There may be several proofs of $\Sigma\ (a : A)\ (P\ a)$ and thus this is not a well behaved type for such propositions. We will solve this in section 2.2.3 where we define propositions in homotopy type theory and propositional truncation.

Lastly we are briefly going to talk about fibers. For a function $f : A \to B$ and an element $b : B$ we say that fiber $f\ b := \Sigma\ (a : A)\ (f(a) \equiv b)$ which is the type of elements from A such that f maps them to b. These have several applications, but we will mostly use them for equivalences in homotopy type theory, see section 5.1 in this thesis. For more information on fibers, see section 4.2 in [2].

2.2 Homotopy type theory

This section will be a brief explanation of the main concepts of homotopy type theory and is inspired by chapter 2 in [2]. Relevant references will be given for the interested reader throughout this section.

In homotopy type theory the main idea is not to view types as sets, but rather as topological spaces. Here we want our notion of equality to not be as restrictive as our $\_ \equiv \_$ above, and we instead let the proof that $a \equiv b$ be a continuous function from a to b that stays within the space. We will call these equality proofs "paths". One may think of this equality proof as a function $f : [0, 1] \to A$, for the relevant type A, where $f(0) = a$ and $f(1) = b$. With this insight it is natural to define equality of such equality proofs as homotopies, which in turn can be regarded as functions $H : [0, 1] \times [0, 1] \to A$ where $H(s, 0) = p(s)$ and $H(s, 1) = q(s)$ for two paths $p\ q : a \equiv b$. We also put the restriction on H that $H(0, t) = a$ and $H(1, t) = b$ to make $H(-, t)$ a path from a to b for every fix value t. We also let the function $f(t) = a$, which is the stationary path at a, represent our previous equality refl a.

We now have our main idea of paths and paths between paths. Now comes the question of how we define this in our type theory, as we do not have a set of real numbers from 0 to 1. In cubical type theory the solution is to define a type that represents the set $[0, 1]$ and to define paths as the functions given above. For a more detailed walkthrough of this see [5] and [6].

More generally in homotopy type theory we do not define these functions directly, but rather we use their properties, and the induction rule on refl to create these paths. A detailed construction of these are found in chapter 2 of [2].

We will see that this idea leads to a useful equivalence relation. For those readers who have studied topology it should be intuitive that for any path $p : a \equiv b$ we can define $p^{-1} : b \equiv a$ where the idea is to follow the path in the opposite direction, which in set theoretic terms mean $p^{-1}(t) = p(1 - t)$. We could also define the composition of paths as for any $p : a \equiv b$ and $q : b \equiv c$ we let

$$\begin{aligned} &\text{compPath } : [0,1] \to A \\ &\text{compPath } t = p(2t) \qquad \text{for all } t < \frac{1}{2} \\ &\text{compPath } t = q(2t-1) \quad \text{otherwise .} \end{aligned}$$

and we should see that we have compPath : $a \equiv c$. This is thus an equivalence relation, but it is not obvious how to define this in the type theory we are working with. For the sake of brevity we will leave their construction to chapter 2 in [2] and move on to another of the main concepts in homotopy type theory, the univalence axiom.

2.2.1 Equivalence and the univalence axiom

To define the univalence axiom we will first define the concept of equivalence. The idea behind the univalence axiom is that types are equivalent iff they are equal. That relates back to our example with the types *Unit* and *OneElem*. In standard type theory they had very similar definitions, but were not equal. This will now be solved and will give a more intuitive view of equivalence. For the readers that are familiar with category theory this will also help in making categories that behave naturally, where two of the objects are equal iff they are isomorphic, something that does not hold automatically. For more information see 3.2.

This concept of equivalence will not be the definition of $\_ \equiv \_$ but a different definition relating to functions and their inverses. Chapter 4 in [2] goes over the equivalence type, as well as a lot of equivalent formulations that in practice are very useful for different applications. This section will only present the basic definition, see 5.1 for more information on different definitions of equivalence.

Firstly, we define equivalence for functions as two functions being equivalent if they are equal for all arguments. This can be seen as

$$f \sim g := (a : A) \to f(a) \equiv g(a).$$

Secondly, a function is then considered to be an equivalence of types if it has a right and a left inverse. That is for $f : A \to B$ we define

$$\text{isEquiv } (f) := (\Sigma(g : B \to A)\ (g \circ f \sim \text{Id } A)) \times (\Sigma\ (h : B \to A)\ (f \circ h \sim \text{Id } B))$$

where Id A is the identity function on A defined by $\lambda a \to a$, and $\_ \circ \_$ is normal function composition.

This definition is called bi-invertible and represents that f has a left and a right inverse. It is natural to want the definition to be that there exists a function g :

$B \to A$ which is both a right and a left inverse. This definition is called a quasi-inverse and it has proven problematic for proof relevant mathematics. For more information see section 2.4 in [2]. Also note that if A and B are 0-types, see section 2.2.3, then quasi-inverses are well behaved.

Finally we define our wanted equivalence of types as

$$A \simeq B := \Sigma\ (f : A \to B)\ (\text{isEquiv}\ f)$$

that is, we prove that A is equivalent to B by giving a pair of a function $f : A \to B$ and a proof that f is an equivalence. This in turn means that to prove that two types are equivalent we need to supply a function $f : A \to B$, two functions $g\ h : B \to A$ as well as proofs that g is a left inverse of f and that h is a right inverse of f.

This is a very powerful definition, and we will illustrate this with an example where we prove that $Unit \simeq OneElem$ as we wanted earlier. For the function $f : Unit \to OneElem$ we will have to chose the function $\lambda unit \to *$ and for g and h we will chose $\lambda * \to unit$, since these are the only elements that we can map to. It should now be clear that $(a : Unit) \to g(f(a)) \equiv a$ and $(b : OneElem) \to f(h(b)) \equiv b$ by the definition of g, h and the fact that $Unit$ and $OneElem$ only has one element.

It is now high time to introduce the univalence axiom. The univalence axiom can be stated concisely by

$$(A\ B : \text{Type}\ U_\ell) \to (A \equiv B) \simeq (A \simeq B).$$

From a propositional view point this can be read as "for all types A and B the concepts of equivalence and equality are themselves equivalent". As noted in section 2.10 in [2] this is technically not an axiom, only a property of our universe U_ℓ. The axiom part comes from the fact that we now assume that every universe we will work in has this property.

2.2.2 Useful tools

In this section we will introduce some tools that are natural from the view of propositions as types. First, assume we have a function $f : A \to B$ and a proof that for $a\ a' : A$ we have that $a \equiv a'$. Then it is natural to want a proof that $f(a) \equiv f(a')$, since $\equiv$ is supposed to represent equality. The book [2] refers to this function as $ap_f : (b \equiv b') \to (fa \equiv fa')$.

In this paper we will use the more general

$$\begin{aligned} cong :&(A : \text{Type}) \to (B : A \to \text{Type}) \to (f : (a : A) \to B\ a) \to \\ &(a\ a' : A) \to (a \equiv a') \to f(a) \equiv f(a'). \end{aligned}$$

Another useful function is *transport*. The type of *transport* is

$$transport : (A : \text{Type}) \to (P : A \to \text{Type}) \to (a\ a' : A) \to (P\ a \equiv P\ a') \to P\ a \to P\ a'.$$

To understand this, for any type A we let $P\ a$ be a type for any $a : A$. Then *transport* says that if we have any $a\ a' : A$ such that $P\ a \equiv P\ a'$ then if we have an element of $P\ a$ we also have an element of $P\ a'$. For more information on *transport* and its definition see section 2.3 in [2].

The last tool in this section is functional extensionality. It says that for any type A, $P : (a : A) \to \text{Type}$ and functions $f\ g : (a : A) \to P\ a$ then $f \equiv g$ if for all $a : A$ we have that $f(a) \equiv g(a)$. This simply means that two functions are equal if they are equal for all inputs. We will denote the proof of this as funExt, similarly to how it is done in cubical agda. For more information on funExt see section 2.9 in [2].

2.2.3 Homotopy levels

Before we end our walkthrough of basic homotopy type theory we will touch on the important concept on homotopy levels.

2.2.3.1 (-2)-types

The first homotopy level is (-2)-types. A type is a (-2)-type if it has an element that is equal to every other element of the same type. This can be stated as

$$\text{isContr}\ A := \Sigma(a : A)((b : A) \to a \equiv b)$$

for a specific type A. These types are also called contractions, since from a topological view this means that there exists a point in the space that has paths to every other point, meaning that the space is contractible. It should also be clear that contractible types are equivalent, since if we have $contA : \text{isContr}\ A$ and $contB : \text{isContr}\ B$ we can let $f := \lambda a \to (\text{fst}\ contB)$ and $g, h := \lambda b \to (\text{fst}\ contA)$ and then $g(f(a)) \equiv a$ by $(\text{snd}\ contA)\, a$ and similarly $f(h(b)) \equiv b$ by $(\text{snd}\ contB)\, b$. It should be clear from this that any contractible type is equivalent to $Unit$, or the true proposition. This also gives that $Unit \simeq OneElem$.

2.2.3.2 (-1)-types

Secondly we consider the (-1)-types. A type is a (-1)-type if its elements are pairwise equal. This can be formulated as

$$\text{isProp}\ A := (a\ b : A) \to a \equiv b.$$

for our type A. The (-1)-types are called propositions. One may naively think that this is the same as being a (-2)-type but note that we here said that they should be equal for any two elements, and not that we have to give a specific element to start with. This means that the type *Empty* satisfies isProp *Empty* but not isContr *Empty* since it has no element. Where one may think of (-2)-types as representing the true proposition, the (-1)-types may be thought of as representing either the true proposition, if it has an element, or the false proposition, if it does not have any elements. This makes them ideal to represent the behavior we expect from propositions in logic, where we usually are not concerned with what proof was given to prove them, hence all its proofs are equal.

2.2.3.3 0-types

We define the 0-types as those types where all equalities on the same elements are equal. That is, for any two objects that have paths between them there is a homotopy between any two paths. Stated more compactly we define

$$\text{isSet } A := (a\ b : A) \to (p\ q : a \equiv b) \to (p \equiv q))$$

for our given type A. These types are called sets. Note here that these are not the normal sets from set-theory. We have not defined the union or cardinality of them. They are just types that capture the basic property of equality for elements in sets. Note that we have not demanded any equality between the elements, so sets can have different elements, we have just demanded that their equalities are the same, which in turn reflect the idea that we in set-theory do not trouble ourselves with what proof was used to prove that two elements are the same.

2.2.3.4 n-types for $n \geq 1$

For $n \geq 1$ we have a similar expression to the one for isSet. A type is a n-type if pairwise equality on its elements is a $(n-1)$-type. This can be written as

$$\begin{aligned}
&\text{isNType } : (A : \text{Type}) \to (n : \mathbb{N}) \to \text{Type} \\
&\text{isNType } A\ 0 = \text{isSet } A \\
&\text{isNType } A\ (\text{succ } n) = (a\ b : A) \to \text{isNType } (a \equiv b)\ n.
\end{aligned}$$

It might be interesting for those with a mathematics background to know that the 1-types are called groupoids. This is because the pairwise equality of these types form groups, since the equalities are paths. Another interesting property is that for any n-type, including (-2)-types and (-1)-types, it is possible to prove that they are also $(n+1)$-types. The reader should note that to be a n-type is itself a proposition, see section 3.3 in [2].

It is also possible to truncate any type A to an n-type. We leave the explanation of truncation to section 3.7 in [2], but the reader may think of this as a version of A that can be proven by an element of A and is an n-type but where we may not use the elements of A unless we are proving another n-type. We will denote propositional truncation by $||\_||$.

3. Category Theory

This chapter will give some of the basic definitions from category theory and how these can be viewed in homotopy type theory. For more information on category theory see [7].

3.1 Definition of a category

It is natural to start with the definition of a precategory. A precategory contains two things, objects and morphisms between objects. These also must satisfy the following properties:

1. For any object there must exist a morphism from the object to itself called the identity morphism.

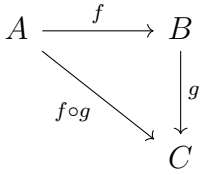
2. There must exist a composition operator on morphisms.

3. Composition with the identity morphism returns the original morphism.

4. The composition operator on morphisms is associative.

In other words we have that morphisms have an origin and a destination, both of which are objects in the category. These morphisms may be thought of as pointed arrows between objects. We also have that each object has its own identity morphism, which is an arrow pointing from the object to itself.

There is also the composition of morphisms, which can be thought of as first following the first arrow and then the second arrow. Note that the destination of the first morphism must be the origin of the second morphism. If two morphisms have that the destination of the first is the origin of the second they are called "composable". We will denote the composition of morphisms by $\circ$, so the composition of a morphism f, from A to B, and a morphism g, from B to C, will be denoted $f \circ g$. This is illustrated in the following diagram:



It is standard practice to denote a morphism from A to B by $f : A \to B$, but since this notation clashes with our notation for functions we will instead use $f : A \Rightarrow B$.

We will in the future not state specifically that the morphisms we are composing are composable. We will also let $\hom(A, B)$ denote all morphisms from A to B, so $f \in \hom(A, B)$ is the same thing as $f : A \Rightarrow B$. We will also use the notation hom A B for $\hom(A, B)$, and hom C A B for $\hom(A, B)$ in the category C.

Then we have the identity morphisms. These can be thought of as an arrow that goes back to the same point it started at, similarly to how we thought of the identity path as a stationary path. From this way of thinking about morphisms it should be intuitive that composing with the identity arrow gives back the original arrow. The reader should note that this can be written more compactly as id $A \circ f = f$ and $f \circ \text{id } B = f$, where id A is the identity morphism on the object A.

In practice categories can be implemented as a record containing the following:

Object : Type U_ℓ

hom : $(A\ B : \text{Object}) \to \text{Type } U_{\ell'}$

id : $(A : \text{Object}) \to \text{hom } A\ A$

leftComp : $\{A\ B : \text{Object}\} \to (f : \text{hom } A\ B) \to \text{id } A \circ f \equiv f$

rightComp : $\{A\ B : \text{Object}\} \to (f : \text{hom } A\ B) \to f \circ \text{id } B \equiv f$

Assoc : $\{A\ B\ C\ D : \text{Object}\} \to (f : \text{hom } A\ B) \to (g : \text{hom } B\ C) \to (h : \text{hom } C\ D) \to f \circ (g \circ h) \equiv (f \circ g) \circ h$

Note here that U_ℓ and $U_{\ell'}$ may be different universes. We say that a precategory is a category if $\hom(A, B)$ is a set for all objects A and B.

3.1.1 Some examples of categories

Before we move on we will look at some basic examples.

One of the most basic examples of a category is the category of small sets. Here the word small just refers to the fact that we only consider the sets within our universe. Here the objects are sets and the morphisms are the functions between these sets. We also have that the identity morphisms are the identity functions and our composition operation is functions composition. The reader should be able to verify that this fulfills the requirements. We will denote this category as **SET**.

One important thing to note is that morphisms between objects need not be unique. For any function $f : A \to \mathbb{N}$ from a nonempty set A we have that $f \not\equiv f \circ \text{succ}$ since $\text{succ}(n)$ is not equal to n for any $(n : \mathbb{N})$. This also shows us that morphisms from an object to itself does not have to be the identity, since $succ : \mathbb{N} \to \mathbb{N}$ but it does not equal the identity function.

Another example are categories called posets. For any binary relation $\leq$ on some type A, which is both transitive and reflexive, we can define a category by the following: The objects are the elements of A and for any objects a and b we let there be a unique morphism from a to b if and only if $a \leq b$. By definition of reflexive we have that for any $(a : A)$ we have $a \leq a$ and thus there is a unique arrow from a to itself. We define this as our identity arrow. Similarly if their exists morphisms $f : a \Rightarrow b$ and $g : b \Rightarrow c$ then $a \leq b$ and $b \leq c$ and thus by transitivity $a \leq c$ and we have a unique morphism from a to c, which we define as $f \circ g$. Checking the remaining requirements is now trivial, since all morphism between objects are unique.

We leave it to the reader to verify that these precategories are categories.

3.1.2 Commutative diagrams

To represent categories, or parts of categories we will use diagrams. Here we will denote the objects with capital letters, and the morphisms by arrows between these letters. To avoid clutter in these diagrams we will usually not draw identity morphisms and compositions of morphisms, since these can be inferred from the diagram. We also say that a diagram commutes if for two compositions of morphisms in the diagram we have that if they have the same origin and destination then they are equal. For example the diagram

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[k]{} & D \end{array}$$

is intended to represent the same category as

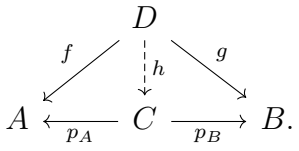
$$\begin{array}{ccc} \text{id } A \circlearrowleft A & \xrightarrow{f} & B \circlearrowright \text{id } B \\ & {\scriptstyle h}\downarrow \quad \searrow^{f\circ g} \quad \downarrow{\scriptstyle g} & \\ & {\scriptstyle h\circ k} & \\ \text{id } C \circlearrowleft C & \xrightarrow[k]{} & D. \circlearrowright \text{id } D \end{array}$$

We will use commutative diagrams in the rest of the paper, unless otherwise stated. To clarify one more thing. For a diagram to commute we require that the compositions of any morphisms commute, not just pairwise. So for the following diagram to commute

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{g} & C \\ {\scriptstyle r}\downarrow & & & & \downarrow{\scriptstyle h} \\ D & \xrightarrow[s]{} & E & \xrightarrow[t]{} & F \end{array}$$

we require that $f \circ g \circ h \equiv r \circ s \circ t$. Also note that we do not explicitly write down the parenthesis for the composition, since it is associative.

We will also use dashed lines when a morphism is the unique morphism that makes the diagram commute. If we have a diagram where a morphism is the unique morphism from an object to another that makes the diagram commute, this does not mean that it is the only morphism between these objects in general. As an example consider the product diagram below



Here the morphism h is the unique morphism such that $f \equiv h \circ p_A$ and $g \equiv h \circ p_B$, but there may still be other morphisms from D to C that does not satisfy these condition. We will explain the product diagram in the section 3.3.3.

3.2 Equality and categories

Category theory has its own concept of equality between objects in a category. In category theory two objects are considered equal if they are isomorphic. In turn, two objects, A and B, are isomorphic if there exists a morphism $f : A \Rightarrow B$ and a morphism $f^{-1} : A \Rightarrow B$ such that $f^{-1} \circ f = \mathrm{id}\ B$ and $f \circ f^{-1} = \mathrm{id}\ A$. Here both f and f^{-1} are called isomorphisms.

This is not our standard definition of equality that we defined in homotopy type theory. Thus we may define categories where two objects are isomorphic, but they are not the same object. We will thus define that a category is a univalent category if any isomorphic objects in the category are also equal. These can be thought of as well behaved categories, and we will prove that the categories we want to study are univalent.

The attentive reader may notice the similarity in the names of univalent category and univalent universe. This is intentional, and is well illustrated by the example of **SET**. Here if we have that if A and B are isomorphic, then there exists isomorphisms f and f^{-1} as defined above. Since these are functions and the composition operation in **SET** is functions composition we have that f^{-1} is both a right inverse and a left inverse to f as functions. Thus it is clear that f is an equivalence of A and B, as types. If we now use the univalence axiom we realize that $A \equiv B$ and thus **SET** is a univalent category.

While the example of **SET** shows the power of the univalence axiom, we should note that not all categories have that their morphisms are functions, and that their composition operator is functions composition. The reader should note that this does not stop a category from being univalent. If we recall our example of a poset category from above we may add an additional constraint on our relation $\leq$. If $\leq$ is an anti-symmetric relation, meaning that if $a \leq b$ and $b \leq a$ then $a \equiv b$, then we clearly have that the poset category of that relation is a univalent category.

If we decide to use univalent categories we will get a lot of important tools automatically. For instance, let us consider two objects A and B in a category C, a proof that A is isomorphic to B, as well as $P : \text{Object } C \to \text{Type}$. If P represents some property from category theory then it should respect isomorphisms, which means that PA should imply PB. In a univalent category our isomorphism is equivalent to an equality, and transporting over that equality gives us what we want. If we are not in a univalent category we will have to prove that all our definitions of categorical properties respect isomorphisms for every object in the definition. Aside from requiring a lot of extra proofs, this might also add compilation time to the implementation if these proofs are complex and thus take a long time to reduce.

We will denote isomorphism of objects, A and B, in a category by $A \cong B$. While there is a straightforward way to represent isomorphisms in categories as a record that includes:

f **:** $A \Rightarrow B$

f^{-1} **:** $A \Rightarrow B$

leftInv : $f^{-1} \circ f \equiv \text{id } A$

rightInv : $f \circ f^{-1} \equiv \text{id } B$,

there is more than one way to represent that a category is univalent. We will discuss some of these later in section 5.3.1.1.

Another concept based on isomorphism is that an object may be "unique up to isomorphism". We say that an object A which satisfies some condition is unique up to isomorphism if for any other object B satisfying the same condition we have that $A \cong B$. If these objects are in a univalent category C then we have that $A \cong B$ implies $A \equiv B$ and so A is the unique object in C satisfying the condition. For any example of this see initial objects, proposition 3.3.2.

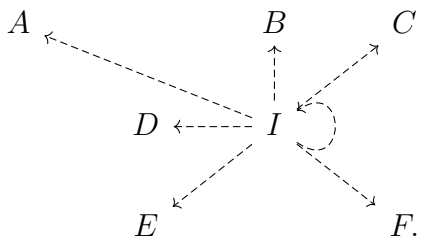
3.3 Limits

One important concept in category theory is limits. These are important constructions that often include both objects and morphisms that have some important properties. We will not go into the general definition of a limit here, but will instead look at some specific limits that we need for our application later.

3.3.1 Initial and Terminal objects

Two of the most basic limits are the initial and terminal object. An object I is initial if for any object A in the category there is a unique morphism from I to A.

An illustration of this is given in the following diagram



The property that an object A is an initial object has a simple representation in type theory. This representation for a given category C is given by

$$\begin{aligned}&\text{isInitial} \ : (I : \text{Object } C) \to \text{Type}\\&\text{isInitial } I = (A : \text{Object } C) \to \text{isContr } (\text{hom } C \ I \ A)\end{aligned}$$

Thus a proof that an object I is initial is a function that takes any object A and returns a proof that the homset from I to A is contractible, meaning it contains a unique element.

The definition for terminal objects is similar. Here an object T is terminal if for any object A there is a unique morphism from A to T. Note that this is the same definition as for initial objects, where we have just flipped the morphisms. Similarly this can be represented in homotopy type theory as

$$\begin{aligned}&\text{isTerminal} \ : \ (T : \text{Object } C) \to \text{Type}\\&\text{isTerminal } T = (A : \text{Object } C) \to \text{isContr } (\text{hom } C \ A \ T).\end{aligned}$$

Recall from the introduction that we want to prove when important concepts in category theory are propositions and sets and so on. This is our first example of this. We will want to prove that isInitial A and isTerminal A are propositions for all objects A. In this case the proof is very straightforward.

Proposition 3.3.1. *In a category C the type* isInitial A *is a proposition for any object A in C.*

Proof. We need to prove that for any $p \ q$: isInitial A we have that $p \equiv q$. Since p and q are functions, by functional extensionality we have that if we can prove $p(B) \equiv q(B)$ for any object B in C we have that $p \equiv q$. The last step is simple, since isContr is a proposition for any type we have that both $p(B)$ and $q(B)$ are proofs of isContr (hom $C \ I \ A$) and thus $p(B) \equiv q(B)$ and by functional extensionality $p \equiv q$. □

The reader should note that the proof for isTerminal is completely analogous. As an exercise to see the relation between the math and homotopy type theory we will prove that an initial object is unique in a category theoretic sense. That is we will prove that any two initial objects are isomorphism.

Proposition 3.3.2. *In any category C any two initial objects are isomorphic.*

Proof. We start by giving the proof from a mathematical viewpoint. Here we let I and I' be two initial objects, and we want to prove that $I \cong I'$. First we note that I is initial so there must exist a unique morphism $f : I \Rightarrow I'$. Similarly I' is initial so there must exist a unique morphism $g : I' \Rightarrow I$. Now we note that $f \circ g : I \Rightarrow I$ and since I is initial this must be a unique morphism. But id $I : I \Rightarrow I$ and it must also be a unique morphism since I is initial. By uniqueness we now have that $f \circ g \equiv \text{id } I$. The same argument gives that $g \circ f \equiv \text{id } I'$ and thus I and I' are isomorphic by f and g.

The proof in homotopy type theory is completely analogous, except that we need to prove that if we have two unique objects, then they are equal. This in homotopy type theory becomes a function isContr $A \to$ isProp A for some type A. Recall that isContr is to have an element that is equal to every other element, which is what it means to be unique, and isProp means that all elements are equal. A proof of isContr A is an element a and a function $eq\ A : (b : A) \to a \equiv b$. If $p\ q : A$ then $eq\ A\ p : a \equiv p$ and $eq\ A\ q : a \equiv q$ and thus but transitivity of $\_ \equiv \_$ we have that $p \equiv a \equiv q$ and thus isProp A. □

We could do the homotopy type theory step more abstractly by referring to the fact that we know from basic homotopy type theory that if A is a n-type then it is also a $(n+1)$-type. So if A is contractible, which is to be a (-2)-type, then it is also a proposition, since this is to be a (-1)-type.

By an analogous proof we see that terminal objects are also isomorphic.

3.3.2 The zero object and zero morphisms

Now that we have defined what it means for an object to be initial and terminal it is time to define what it means to be a zero object. A zero object is simply an object that is both initial and terminal. The representation of this in homotopy type theory is straightforward and it is given by

$$\begin{aligned} &\text{isZeroObject} \ : \ (Z : \text{Object}) \to \text{Type} \\ &\text{isZeroObject}\ Z = (\text{isInitial}\ Z) \times (\text{isTerminal}\ Z) \end{aligned}$$

which is the pair of a proof of isInitial Z and isTerminal Z. We see that isZeroObject is clearly a proposition, since by functional extensionality we only need to prove it for all Z and we know from basic homotopy type theory that pairs are equal if each of their elements are equal. Thus we only need to prove that isInitial Z and isTerminal Z are propositions, which we saw earlier.

Also note that any zero objects are unique up to isomorphism. This follows directly from the fact that a zero object is an initial object. The reader should note that we could also have used that a zero object is a terminal object.

For a univalent category C we can define hasZeroObject C as a pair of an object Z and a proof of isZeroObject Z. This is a proposition since Z is unique up to isomorphism and we are in a univalent category, which means that Z is unique.

Related to the concept of a zero object, Z, is the concept of a zero morphism. A zero morphism from an object A to an object B is defined as a morphism that is equal to the composition of a morphism from A to Z and a morphism from Z to B. Note that any morphism $toZero : A \Rightarrow Z$ is unique since Z is a zero object, and thus terminal. Similarly, any morphism $fromZero : Z \Rightarrow A$ is unique since Z is an initial object. Thus if $f : A \Rightarrow B$ is a zero morphism then $f \equiv toZero \circ fromZero$. It is also unique since if $g : A \Rightarrow B$ is a zero morphism then $f \equiv toZero \circ fromZero \equiv g$. Knowing that the zero morphism is unique and that it is equal to $toZero \circ fromZero$, where both $toZero$ and $fromZero$ are themselves unique we can say that $toZero \circ fromZero$ is the unique zero morphism.

Thus we can define the function

$$\begin{aligned} &\text{getZeroMorphism } : (A\ B : \text{Object } C) \to \text{hom } C\ A\ B \\ &\text{getZeroMorphism } A\ B = (\text{getToZero } A) \circ (\text{getFromZero } B) \end{aligned}$$

where getToZero A returns the unique morphism from A to Z and getFromZero B returns the unique morphism from Z to B.

The reader may verify that **SET** has no zero object. This follows since the only initial object is the empty type, where the outgoing functions are the empty functions, and the terminal objects are the one element sets, since the incoming functions have to map to the same object. Thus when discussing properties of categories with a zero object we will have to take another category as an example.

3.3.3 Products

Products are one of the first objects that one typically gets introduced to in category theory. This is because their definition is fairly simple, but they provide a lot of structure that is useful in both proofs and applications.

We say that an object AXB is the product of objects A and B if

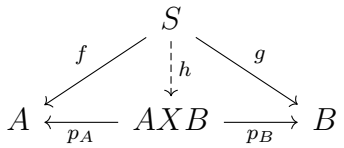
p_A **:** There exists a morphism $p_A : AXB \Rightarrow A$.

p_B **:** There exists a morphism $p_B : AXB \Rightarrow B$.

morphismProd : For any object S and any morphisms $f : S \Rightarrow A$ and $g : S \Rightarrow B$ there exists a morphism $h : S \Rightarrow AXB$ such that $f \equiv h \circ p_A$ and $g \equiv h \circ p_B$.

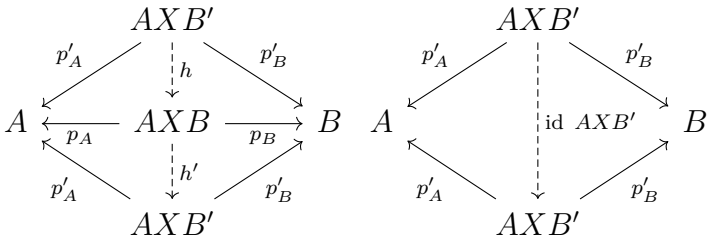
uniqueness : Given S, f and g above, for any two morphisms $h\ h' : S \Rightarrow AXB$ that satisfy condition "morpshismProd", we have that $h \equiv h'$.

This definition is usually accompanied by the following diagram



that tries to illustrate the above definition. The morphisms p_A and p_B are called projections, to A and B respectively. The morphism h is sometimes denoted $< f, g >$ since it can be thought of as a product of the morphisms f and g. The triple of AXB, p_A and p_B as defined above is called a product diagram.

It is possible to show that the object AXB is unique up to isomorphism, just as we did for initial objects. We will not give the detailed proof here, but the interested reader should be able to deduce from the following diagrams



that by uniqueness we have $h \circ h' \equiv \text{id } AXB'$ and by a similar argument $h' \circ h \equiv \text{id } AXB$.

One problem still remains. We would like to show that our above definition of a product is a property. Sadly, if we define isProduct AXB as the definition above we will find that it is not a property. We will illustrate this with an example in **SET**.

To start our example, let us show the intuitive products in **SET**. We define AXB to be $A \times B$, where A and B are two objects in **SET**. Then we let $p_A = \text{fst}$ and $p_B = \text{snd}$. Now if we want to find our $h : S \to AXB$ we first let $h(s) = (a, b)$. We then see that if $f \equiv h \circ p_A$ we have by functional extensionality that $f(s) \equiv p_A(h(s)) \equiv \text{fst } (a, b) = a$ by the definition of $h(s)$ and p_A. Similarly we find that $g(s) \equiv p_B(h(s)) \equiv b$. Thus h is uniquely determined by the function $\lambda s \to (f(s), g(s))$.

One might think that this should be the only proof that A and B has the product AXB. Though AXB is unique up to isomorphism, and "morphismProd" and the uniqueness of h are properties by functional extensionality we still have to choose our p_A and p_B. To see that there are other options, let A be a set with at least two distinct elements. That is there exists $s\ t : A$ such that $s \not\equiv t$. Then let $P'_A : A \to A$ be a permutation on A that is not the identity function. A permutation is a function from a type to itself that is invertible. We say that a permutation is of order 1 if it is its own inverse, meaning $P'_A \circ P'_A \equiv \text{id } A$. An example of a permutation on A

that is of order 1 is the function

$$e(x) = \begin{cases} s & \text{if } x \equiv t \\ t & \text{if } x \equiv s \\ x & \text{otherwise} \end{cases}$$

which clearly has the property $e(e(x)) \equiv x$. Now we let $p_A := \text{fst} \circ P'_A$ and $p_B := \text{snd}$. It remains to show that if $f : S \to A$ and $g : S \to B$ then there exists a unique $h : AXB$ such that $f \equiv h \circ p_A$ and $g \equiv h \circ p_B$. Once again, let $h(s) = (a, b)$. Then if $f \equiv h \circ p_A$ we must have that $f(s) \equiv p_A(h(s)) \equiv p_A(a, b) \equiv P'_A(a)$. To find a from this we simply apply P'_A to each side of the equation and get $P'_A(f(s)) \equiv P'_A(P'_A(a)) \equiv a$ since P'_A is a permutation of order 1. Similarly to what we did above, we must have that $b \equiv g(s)$. Thus h is uniquely defined as $h(s) := (P'_A(f(s)), g(s))$.

3.3.3.1 Implementation

Let us now consider how to define types representing these propositions we have seen above. First, for objects A, B, AXB and projections $p_A : AXB \Rightarrow A$, $p_B : AXB \Rightarrow B$ we may define the type isProductDiagram $A\ B\ AXB\ p_A\ p_B$ as a record containing the two conditions "morphismProd" and "uniqueness". Now, since these conditions end in equalities on morphisms we get that isProductDiagram $A\ B\ AXB\ p_A\ p_B$ is a proposition for any category C. If we want to consider a general precategory then we would have to truncate the equalities to make it a proposition, since the homsets may no longer be sets.

Next, we may define isProduct $A\ B\ AXB$ to be a record containing two morphisms $p_A : AXB \Rightarrow A$, $p_B : AXB \Rightarrow B$ as well as a proof of isProductDiagram $A\ B\ AXB$ $p_A\ p_B$. Since these projections are not unique we will need to propositionally truncate the record to make isProduct $A\ B\ AXB$ a proposition.

We will also define hasProduct $A\ B$ as a record containing an object AXB and a proof of isProduct $A\ B\ AXB$. Note that products are unique up to isomorphism, which means that if we are in a univalent category then this is already a proposition. If we want this in a general precategory however we will need to propositionally truncate this record since the object AXB may not be unique. For a precategory C we may also define hasAllProducts as a function that takes any two objects A and B in C as input and give a proof of hasProduct $A\ B$. This is a proposition by functional extensionality since hasProduct $A\ B$ is a proposition. In our implementation we chose to give the definition from the point of view of a univalent category so most of the truncations could be removed.

3.3.4 Kernels

The structure of a kernel object is more advanced than what we have seen so far. It also requires the category to contain a zero object. The definition of a kernel can be stated as follows: For a given morphism $f : A \Rightarrow B$, the object K is a kernel object of f if

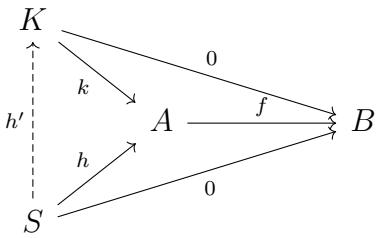
kernelMorphism There exists a morphism $k : K \Rightarrow A$.

KerComp The composition $k \circ f$ is equal to the zero morphism from K to B.

kerFactors For any morphism $h : S \Rightarrow A$ where $h \circ f$ equals the zero morphism, there exists $h' : S \Rightarrow K$ such that $h' \circ k \equiv h$.

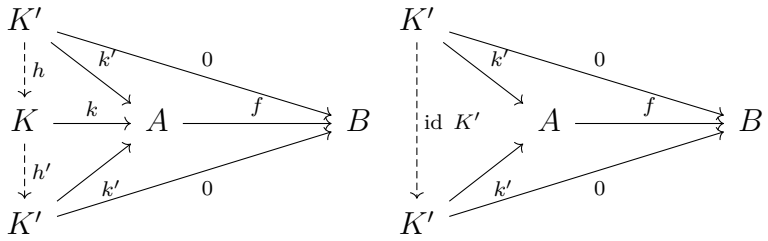
uniqueness For h given above, if two morphisms h' and h'' satisfy the above condition then $h' \equiv h''$.

This is illustrated by the diagram



.

Similarly to how we proved the uniqueness of the product, the reader should be able to deduce the proof that the kernel object is unique from the following diagrams



. Note that we did not draw the zero morphism from K to B for readability. Also note that there is no ambiguity in which zero morphisms we mean, since the zero morphism between any objects are unique.

Again, similarly to our approach for the product, we denote the pair of a Kernel object and its kernel morphism as a kernel diagram. It is possible to prove that to be a kernel diagram is a property, but to be a kernel object is not. For brevity the proof will be omitted, but a reader with knowledge of abstract algebra, might be interested to know that in the category of monoids, where morphisms are homomorphisms, the kernel object is the algebraic kernel of f and the kernel morphism is the inclusion $\mathrm{Ker} f \to A$. Here it is clear that the kernel morphism is not unique, since adding an appropriate permutation gives a different, but still valid, kernel morphism. The category of monoids where the morphisms are homomorphisms is called **MON** and will be used as an example for objects that do not exist in **SET**.

The implementation of kernels is similar to how we did for products in section 3.3.3.1. First we can define a type isKernelDiagram′ *zero* f K k containing the

conditions "KerComp", "kerFactors" and "uniqueness" for f, K and k defined as above, and *zero* : hasZeroObject C. As we said earlier, these conditions end in equalities, so they are propositions if we are in a category. However, if we are in a general precategory we will have to propositionally truncate the equalities. Note that we could define this as isKernelDiagram *zero* f k where we can infer K since it is the origin of k.

Then we define isKernel *zero* f K as a propositional truncation of the record containing a morphism $k : K \Rightarrow A$ and a proof of isKernelDiagram′ *zero* f K k. Note that the propositional truncation is necessary since k does not have to be unique.

Finally we define hasKernel *zero* f as a record containing an object K and a proof of isKernel *zero* f K. Note that if we are in a univalent category we will not need to propositionally truncate this type since K is unique. We also define hasAllKernels C for a category C as a function that takes a morphism f, a proof that C has a zero object and returns a proof of hasKernel *zero* f. Note that this is a proposition by functional extensionality since hasKernel *zero* f is a proposition.

3.3.5 Cocategory

Before we go over the coming objects, it is natural that we talk about cocategories. A cocategory is simply a category where we have flipped all morphisms. The objects are the same and there exists a morphism $f : A \Rightarrow B$ in the cocategory if and only if $f : B \Rightarrow A$ in the original category. The composition operator is defined as $f \circ g$ in the cocategory if $g \circ f$ in the original category. The reader can verify that this fulfills the definition of a category. A cocategory is also called a dual category or an opposite category. We denote the cocategory of a category C as C^{op}. For any property P in C we say that Q is the dual of P if (P holds in C) implies (Q holds in C^{op}). We should also note that $(C^{op})^{op} = C$, which also means that if P is the dual of Q then Q is the dual of P.

3.3.6 Initial, terminal and zero objects

If we look at the definition it is clear that being initial and terminal are dual properties, which means that if an object is initial in the category, then it is terminal is the cocategory, and vice versa. This is since properties on morphisms going from an object in the category are transferred to properties of morphisms going to an object in the cocategory. We also note that this means that the zero object is its own dual.

3.3.7 Coproducts

A more interesting case is the coproduct. The dual of the definition of a product is

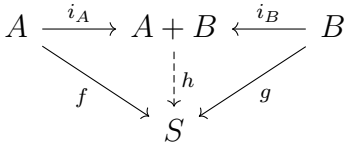
i_A **:** There exists a morphism $i_A : A \Rightarrow A + B$.

i_B **:** There exists a morphism $i_B : B \Rightarrow A + B$.

morphismProd For any object S and any morphisms $f : A \Rightarrow S$ and $g : B \Rightarrow S$ there exists a morphism $h : A + B \Rightarrow S$ such that $f \equiv i_A \circ h$ and $g \equiv i_b \circ h$.

uniqueness Given S, f and g above, for any two morphisms $h\ h' : S \Rightarrow A + B$ that satisfy condition "morpshismProd", we have that $h \equiv h'$.

where $A + B$ is the coproduct. This is clearly illustrated by the coproduct diagram:



Similarly to how we called p_A and p_B projections since they in some sense project the product into each part we call i_A and i_B injections since they, in some sense, inject the objects A and B into the coproduct. We also call a triple of a coproduct and its injections a coproduct diagram, and to be a coproduct diagram is a property, but not to be a coproduct, unless we use propositional truncation on the definition, just as for products.

Lets once again return to **SET** and show that it has coproducts. The coproduct of A and B in **SET** is the disjoint union of A and B. The disjoint union differs from the regular union in that we also remember which set the element came from. This can be implemented in type theory as the type

$$\begin{aligned}&\text{data DisjointUnion } (A\ B : \text{Type}) : \text{Type where}\\ &\qquad \text{ElemA} \ : A \to \text{DisjointUnion } A\ B\\ &\qquad \text{ElemB} \ : B \to \text{DisjointUnion } A\ B\end{aligned}$$

Here the two constructors ElemA and ElemB can be used to get an element in the disjoint union from any element in A or B respectively. For any element in the disjoint union we can also case split on the constructors to see which set it originally came from. The natural injections are then $i_A :=$ ElemA and $i_B :=$ ElemB. From these definitions we find that $h(\text{ElemA } a) \equiv f(a)$ and $h(\text{ElemB } b) \equiv g(b)$ and thus it is uniquely determined on all of $A + B$.

The implementation of coproducts is analogous to the implementation of products, see section 3.3.3.1.

3.3.8 Cokernels

Similarly to how we defined the coproduct as the dual of a product, we will define a cokernel that is the dual of a kernel. We say that an object E is a cokernel of a morphism $f : A \Rightarrow B$ if

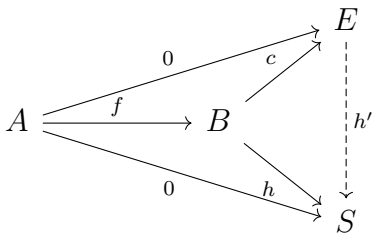
coKernelMorphism There exists a morphism $c : B \Rightarrow E$.

coKerComp The composition $f \circ c$ is equal to the zero morphism from A to E.

coKerFactors For any morphism $h : B \Rightarrow S$ where $f \circ h$ equals the zero morphism, there exists $h' : E \Rightarrow S$ such that $c \circ h' \equiv h$.

uniqueness For h given above if two morphisms h' and h'' satisfy the above condition then $h' \equiv h''$.

The diagram for this definition is



which we note is just the dual of the diagram for the kernel. Similarly to kernels, the category **SET** has no cokernels since it has no zero object, but the reader with a background in abstract algebra may verify that in **MON** the cokernel object is $B/\mathrm{Im}f$, which is the quotient monoid of B and the image of f. Here the cokernel morphism is just the injection into the cokernel object, which need not be unique.

The implementation of cokernel is analogous to the implementation of kernels, see section 3.3.4.

3.4 Morphism properties

Thus far we have talked about structures in category theory, but not about properties on morphisms themselves. In this section we will discuss the concept of monics and epics, and how they relate to the structures we have seen thus far.

3.4.1 Monics and Epics

A morphism $f : A \Rightarrow B$ is called monic if for any morphisms $g\ h : S \Rightarrow A$ we have that $h \circ f \equiv g \circ f$ implies $h \equiv g$. This can be seen in the diagram

$$S \underset{h}{\overset{g}{\rightrightarrows}} A \xhookrightarrow{f} B$$

. Monic morphisms are also called monomorphisms. This concept has a dual called epics. A morphism $e : A \Rightarrow B$ is epic if for any $g\ h : B \Rightarrow S$ we have that $e \circ g \equiv e \circ h$ implies $g \equiv h$. This is shown in the diagram

$$A \xrightarrow{e} B \underset{h}{\overset{g}{\rightrightarrows}} S$$

which clearly is the dual of the diagram for monics. An epic morphism is also called an epimorphism. Some authors also say that the morphism is epi. Also note that

we denote epimorphism with a double arrow and monomorphisms with a hook at the start of the arrow.

If we look at the definition for kernels, we see that the fourth condition about uniqueness is equivalent to requiring that the kernel morphism is monic. This is shown in the following proposition:

Proposition 3.4.1. *Let C be a category with a zero object. Given a morphism $f : A \Rightarrow B$, for an object K with a morphism $k : K \to A$ satisfying $k \circ f \equiv 0$, for any object S in C and any $h : S \Rightarrow A$ satisfying $h \circ f \equiv 0$ we have a $h' : S \Rightarrow K$ such that $h' \circ k \equiv h$. Then we have that the condition that h' is unique is equivalent to the condition that k is monic.*

Proof. The simple direction is to prove that k monic implies h' unique. Assume that there are two h' and h'' that satisfy the above conditions. Then $h' \circ k \equiv h \equiv h'' \circ k$ and by definition of k being monic we get $h' \equiv h''$.

For the other direction let h' be unique for each h satisfying the conditions above and for $g\ l : S \Rightarrow K$ let $g \circ k \equiv l \circ k$. We want to show that $g \equiv l$. Now define $h := g \circ k$. We see that $h \circ f \equiv g \circ k \circ f \equiv g \circ 0 \equiv 0$ and thus we get our h' from above. But since g satisfies the same condition by our assumption we get that $h' \equiv g$ by uniqueness of h'. But similarly $l \circ k \equiv g \circ k \equiv h$ and so l satisfy the same conditions, so $h' \equiv l$. Finally we see that $g \equiv h' \equiv l$ and we are done. □

Now since the cokernel is the dual of the kernel, the cokernel morphism is the dual of the kernel morphism and being epic is the dual of being monic it should be clear from the definitions that the fourth condition of the cokernel is equivalent to the cokernel morphism being epic. The interested reader may find a proof for this by taking the dual of each statement in the above proof.

For our application we will find proving that the kernel morphism is monic to be simple, and thus will prefer this over proving the last condition for the kernel. Similarly we will prove that the cokernel morphism is epic.

3.4.2 Monic and injective

In **SET** a morphism is a function and it is monic if it is injective. We show this by a quick proof.

Proposition 3.4.2. *A morphism in* **SET** *is monic if and only if it is injective.*

Proof. Let $f : A \to B$. We want to prove that for all $a\ b : A$ where $f(a) \equiv f(b)$ implies $a \equiv b$ is equivalent to for all $g\ h : S \to S$ where $g \circ f \equiv h \circ f$ implies $g \equiv h$.

We start with the easy direction, and prove that if f is injective then it is monic. If

$g \circ f \equiv h \circ f$ then for all $x : S$ we have $f(g(x)) \equiv f(h(x))$ by functional extensionality. But then $g(x)\ h(x) : A$ and by injectivity of f we have that $g(x) \equiv h(x)$ for all $x : S$ and thus $g \equiv h$.

For the other direction let $a\ b : A$ where $f(a) \equiv f(b)$. We define two functions $g := \lambda x \to a$ and $h := \lambda x \to b$. Then for all $x : S$ we have that $f(g(x)) \equiv f(a) \equiv f(b) \equiv f(h(x))$ and thus $g \circ f \equiv h \circ f$. Now we use the fact that f is monic and get $g \equiv h$. This in turn gives us that $a \equiv g(x) \equiv h(x) \equiv b$ for all $x : S$ and we are done. □

The reader with a background in proof checkers or constructive logic might appreciate the fact that this proof is constructive. The more common proof is the same for the first direction, but for the second one assumes there exists $a\ b : A$ such that $a \not\equiv b$ and instead take two functions $g\ h : S \to A$ where $g(x) \equiv h(x)$ except for some $x : A$ where $g(x) \equiv a$ and $h(x) \equiv b$. Then $g \circ f \equiv h \circ f$ but $g(x) \not\equiv h(x)$ which leads to a contradiction.

We should note that while the proof that a surjective function is epic in **SET** is simple, we can not prove that every epic function in **SET** is surjective in a constructive manner. This is since the proof requires the axiom of choice, and thus is not constructive. We will show that this holds in RMod but we will need to use the fact that we have some more structure in this category.

We should note that while it often holds that monics are injective and epics are surjective in applications, these concepts are more general since they apply in any category, not just the ones where morphisms are functions. As an example, consider a poset category. Here it is clear that every morphism is both monic and epic, since morphisms between the same objects are unique. They are not functions however and thus they are not injective or surjective.

4. Abelian categories

Abelian categories are categories where the homsets are abelian groups themselves and that have the necessary structure to do abstract algebra in. For more information on abstract algebra and abelian groups see appendix A. Instead of giving the definition directly we will build it up from more simple definitions. For more details see chapter 8 in "Categories for the working mathematician" [7].

4.1 Preadditive

We say that a category C has Ab-homsets if every homset is an abelian group. We then say that a category is preadditive if

1. it has Ab-homsets.

2. composition is bilinear.

Note that the condition of composition being bilinear means it satisfies the two following conditions:

- For any morphisms $f : A \Rightarrow B$ and $g\ h : B \Rightarrow D$ we have $f \circ (g + h) \equiv (f \circ g) + (f \circ h)$.

- For any morphisms $f\ g : A \Rightarrow B$ and $h : B \Rightarrow D$ we have $(f + g) \circ h \equiv (f \circ h) + (g \circ h)$.

where $+$ is the operators of the abelian groups given by the relevant homsets. It should be clear from the context which abelian groups are intended.

4.2 Additive

Before we define an additive category we need to define a structure called a binary direct sum. In a preadditive category an object D is a binary direct sum of two objects A and B if

1. There are morphisms $i_A : A \Rightarrow D$ and $i_B : B \Rightarrow D$.

2. There are morphisms $p_A : D \Rightarrow A$ and $p_B : D \Rightarrow B$.

3. The above morphisms satisfy $i_A \circ p_A \equiv \mathrm{id}\ A$, $i_B \circ p_B \equiv \mathrm{id}\ B$ and $(p_A \circ i_A) +$

$(p_B \circ i_B) \equiv \text{id } D$.

The morphisms in the above definition form the following diagram:

$$A \underset{i_A}{\overset{p_A}{\leftrightarrows}} D \underset{i_B}{\overset{p_B}{\leftrightarrows}} B$$

Note here that we do not require that $p_A \circ i_A$ and $p_B \circ i_B$ are id D, and in our case we will see that this is not the case.

Now that we have binary direct products defined we say that a category C is an additive category if

- It is a preadditive category.
- It has a zero object.
- For all objects A and B in C it also has a binary direct sum of A and B.

Here there are many basic properties to prove. Some of these are

- The zero object in the abelian groups for each homset is the zero morphism in that homset.
- For any binary direct product we have that $i_A \circ p_B \equiv 0$ and $i_B \circ p_A \equiv 0$.

Note here that 0 is the zero morphism between the relevant objects, which by the first property is also the zero object in the abelian group for that homset.

4.3 Abelian category

For abelian categories there are many equivalent definitions, but we will start by giving the following: A category C is an abelian category if

1. It is a preadditive category.
2. It has a zero object.
3. It has all products and coproducts.
4. It has all kernels and cokernels.
5. Every monomorphism is a kernel morphism and every epimorphism is a cokernel morphism.

Here there are many equivalent definitions since many of the conditions can be derived from the others. Given condition 1, by proposition 1 in [7] we could replace

condition 2 by C having an initial object or by C having a terminal object. Given condition 1, by theorem 8.2.2 in [7] we see that condition 3 is equivalent to just having all products, having all coproducts or having all binary direct sums.

By the remark above it is clear that an abelian category is also an additive category. Some authors also divide this further, and say that a category satisfying conditions $1-4$ is a preabelian category.

There is also a proof that given conditions $2-5$ then condition 1 can be derived. This is useful in many categories where we have found these limits, but where the abelian structure on the homsets might be complex. There is a proof of this fact implemented in the proof checker UniMath at UniMath/CategoryTheory/AbelianToAdditive but to implement the proof in Agda proved to be outside the scope of this thesis. The proof is quite complex and requires us to infer a lot of structure and limits for abelian categories before attempting it.

That conditions $2-5$ implies 1 was also shown in section 2.3 of "Abelian Categories - An introduction to the theory of functors" [8] by Peter Freyd.

4.4 Implementation

Similarly to how we have done before, the definitions that contain several conditions are implemented as records containing corresponding conditions. Note that if we regard univalent categories we will not need to propositionally truncate our equalities.

For the definition of abelian category we have several choices, as mentioned above. Our implementation uses the definition of conditions $2-4$. Explicitly, this results in the definition

record isAbelian C *where*
$zero$: hasZeroObject C
$prod$: hasAllProducts C
$coprod$: hasAllCoProducts C
ker : hasAllKernels C
$coker$: hasAllCoKernels C
$monicsAreKernels$: $\{A\ S : \text{Object } C\} \to (k : \text{hom } C\ S\ A) \to \text{isMonic } C\ k \to$
$||\Sigma\ (B : \text{Object } C)\ (\Sigma\ (f : \text{hom } C\ A\ B)\ (\text{isKernel } C\ zero\ f\ k))||$
$epicsAreCoKernels$: $\{B\ S : \text{Object } C\} \to (k : \text{hom } C\ B\ S) \to \text{isEpic } C\ k \to$
$||\Sigma\ (A : \text{Object } C)\ (\Sigma\ (f : \text{hom } C\ A\ B)\ (\text{isCoKernel } C\ zero\ f\ k))||$

where C is the category in question. Note here that we need the propositional truncations since k may act as a kernel resp. cokernel to several f.

5. Implementation and challenges

5.1 Equivalent ways to represent equivalence

There are several ways to represent equivalence in homotopy type theory. The one we saw in the earlier chapter on homotopy type theory was the version used by [2] but there are other equivalent ways to choose from, as described in chapter 4 in [2].

Some of these are appropriate for different applications, and cubical Agda has chosen to use the one of contractible fibers. This says that two types A and B are equivalent if there exists a function $f : A \to B$ such that we can prove isContr (fiber f b) for all $b : B$.

From a mathematical perspective, if there is an $a : A$ for all $b : B$ such that $f(a) \equiv b$ then f is surjective. We also have that since a is unique for every $b : B$ we must also have that f is injective. Now, since it is both injective and surjective it is a bijection and we know that this implies that f is invertible.

The last type of equivalence we are going to consider is fiberwise equivalence. Let A be a type and let $P\ Q : A \to \text{Type}$. Also let $f : (a : A) \to P\ a \to Q\ a$. Then we may define the function total(f) by

$$\begin{aligned}&\text{total}(f) : \Sigma\ A\ P \to \Sigma\ A\ Q\\&\text{total}(f)\ w = (\text{fst}\ w, f(\text{fst}\ w, \text{snd}\ w)).\end{aligned}$$

Now Theorem 4.7.7 in [2] states that $f(a)$ is an equivalence for all $a : A$ if and only if total(f) is an equivalence. We can use this to prove that for any a type A, a relation $\sim$: $A \to A \to$ Type, a function $f : (a\ b : A) \to (a \equiv b) \to (a \sim b)$ and a proof of $(a : A) \to$ isContr $(\Sigma\ (b : A)\ (a \sim b))$ then for all $a\ b : A$ we have that $f\ a\ b$ is an equivalence between $a \equiv b$ and $a \sim b$. The proof of this is implemented in Cubical/Foundations/Equiv/Fiberwise.agda.

5.2 Implementing modules and module homomorphisms

Using our knowledge from abstract algebra, see appendix A, it is natural to define a module as a record with the following fields

$$
\begin{aligned}
&A : \text{Type}\\
&0m : A\\
&\_ + \_ : A \to A \to A\\
&-\_ : A \to A\\
&\_ \cdot \_ : \langle R \rangle \to A \to A\\
&isMod : isModule\ A\ 0M\ \_ + \_\ -\_\ \_ \cdot \_.
\end{aligned}
$$

Here A is the underlying type of the module, $\langle R \rangle$ is the underlying type of the commutative ring we are defining the module over, $0m$ is the zero object, $\_ + \_$ and $\_ \cdot \_$ are defined as in appendix A. The operator $-\_$ is the operator that takes an element to its additive inverse, as we define in appendix A. For a module M we will denote the underlying type as $\langle M \rangle$. This notation and the added $-\_$ operator was chosen to bring the thesis in line with the algebra libraries in the cubical agda standard library.

This definition is equivalent to the Σ-type

$$
\begin{aligned}
&\Sigma\ (A : \text{Type})\\
&\quad \Sigma\ (0m : A)\\
&\qquad \Sigma(\_ + \_ : A \to A \to A) \ldots .
\end{aligned}
$$

This representation is called the SIP representation (Structure Identity Principle) and is used in cubical agda to represent algebraic structures. There is also a library that facilitates equivalence proofs between these types. For more information see [9] [10]. Cubical agda is in the process of moving to the more general representation of DURGs (Displayed Univalent Reflexive Graphs)[11] but these are not fully implemented at the time of writing.

The reader should note that since the concept of left modules over a ring R is already defined in the standard library we have chosen to define $isModule$ as $isLeftModule$, but over a commutative ring.

Here it is important to consider the universe levels used. For a commutative ring R with an underlying type in U_ℓ we have decided to define the underlying type of a module over R to also be in U_ℓ. This was done to follow the definition of left modules given in the standard library. Translating our proofs to this more general case, where $\langle M \rangle : U_{\ell'}$ for another universe $U_{\ell'}$, should not prove to much of a challenge however, and we have provided the script ThesisWorkGitCopy/RModulesAlt/Liftings.agda with liftings for all the relevant algebraic structures in order to facilitate this process.

In a similar manner we implement module homomorphisms between modules M and N as a record with the following fields

$$\begin{aligned} &func : \langle M \rangle \to \langle N \rangle \\ &additive : (x\ y : \langle M \rangle) \to f(x +_M y) \equiv f(x) +_N f(y) \\ &presScalar : (r : \langle R \rangle) \to (x : \langle M \rangle) \to f(r \cdot x) \equiv r \cdot f(x) \end{aligned}$$

where we call $func$ the underlying function of the module homomorphism. For a module homomorphism f' we will denote its underlying function by f. We denote the type of module homomorphisms from M to N by ModuleHomo M N and the constructor moduleHomo.

Since $\langle N \rangle$ is a set by definition we get that both *additive* and *presScalar* are properties, since they end in equalities. Thus we can prove that two module homomorphisms are equal if and only if their underlying functions are equal. This means that if we have two module homomorphisms f' g' : ModuleHomo M N then $f' \equiv g'$ is equivalent to $f \equiv g$ and by functional extensionality we may show that this is equivalent to $f(x) \equiv g(x)$ for all $x : \langle M \rangle$.

5.3 The category RMod

Now that we have defined modules and module homomorphisms it is time to define the category RMod. For a given commutative ring R we define RMod as the precategory with objects being modules over R and where the morphisms are module homomorphisms. The identity morphisms are the identity functions. The composition is function composition on the underlying functions. The reader may verify that the identity function is linear, the compositions of module homomorphisms are themselves module homomorphisms and that this defines a precategory.

5.3.1 RMod is a univalent category

Before we show that RMod is abelian we first show that it is both a category and is univalent.

Proposition 5.3.1. RMod *is a category.*

Proof. To show this we need to show that for any modules M and N we have that isSet (ModuleHomo M N). This is simpler to do with a Σ-type so we construct the corresponding type

$$\text{ModuleHomo}\Sigma = \Sigma\ (h : \langle M \rangle \to \langle N \rangle)\ (\Sigma\ (add : \text{isAdditive}\ h)\ (scal : \text{presScalar}\ h))$$

and the functions

$$\text{ModuleHomoΣToModuleHomo} : \text{ModuleHomoΣ} \to \text{ModuleHomo}$$
$$\text{ModuleHomoΣToModuleHomo}\ (h\ ,\ add\ ,\ scal) = \text{moduleHomo}\ h\ add\ scal$$
$$\text{ModuleHomoToModuleHomoΣ} : \text{ModuleHomo} \to \text{ModuleHomoΣ}$$
$$\text{ModuleHomoToModuleHomoΣ}\ (\text{moduleHomo}\ h\ add\ scal) = h\ ,\ add\ ,\ scal.$$

These clearly form an isomorphism between ModuleHomoΣ and ModuleHomo. Thus since isomorphisms are equivalences we may use the univalence axiom to see that they are equal. Thus if we prove isSet(ModuleHomoΣ M N) then we can use transport to prove isSet(ModuleHomo M N).

Here we will use the function isSetΣ : isSet $A \to ((a : A) \to \text{isSet}\ (P\ a)) \to$ isSet $(\Sigma\ A\ P)$ for A : Type and $P : A \to$ Type. This proof is implemented in ThesisWork/SetSigmaType.agda. Intuitively, isSet says that if both parts of a Σ-type are sets, then the Σ-type itself is a set. There is a similar proof isPropΣ for propositions. Now, the additive and scalar parts are both propositions as we saw earlier. Thus by isPropΣ and since propositions are sets it is sufficient to show that the type of $\langle M \rangle \to \langle N \rangle$ is a set.

Since the underlying type of modules are sets we may instead prove the more general statement that if B is a set then the type of functions from A to B is a set for any type A. To clarify, we will prove that for any functions $h\ k : A \to B$ and $p\ q : h \equiv k$ where isSet B we have that $p \equiv q$. Here we will use cubical type theory to prove the statement. What we want is to create a homotopy H between p and q. To this end for any $i : I$ we want to define $H(i) : h \equiv k$ such that $H(0) = p$ and $H(1) = q$. Recall here that I in cubical type theory is the type of the unit interval. Since $H(i)$ is an equality between functions we may use functional extensionality to get an element of $h \equiv k$. Thus for any $x : A$ we need to give a proof of $h(x) \equiv k(x)$. To this end let $p'(x) = \lambda j \to (p(j))(x)$ and $q'(x) = \lambda j \to (q(j))(x)$. Thus if we can define $H' : (x : A) \to p'(x) \equiv q'(x)$ where $H'(x, 0) \equiv p'(x)$ and $H'(x, 1) \equiv q'(x)$ then $H := \lambda i \to \text{funExt}\ (\lambda x \to H'(x, i))$ will prove our theorem. To define H' is simple, since p' and q' are equalities on the set B, which is a set.

The full proof can be stated compactly as the function

$$\text{isSetFunc}\ setB\ h\ k\ p\ q = \lambda i \to \text{funExt}\ (\lambda x \to setB\ (hx)\ (kx)\ (\lambda j \to (p\ j)x)\ (\lambda j \to (q\ j)x)\ i)$$

where $setB : (y\ z : B) \to (p\ q : y \equiv z) \to p \equiv q$ is the proof that B is a set.

Now that we have proven that both parts of the sigma expression are sets we use isSetΣ to show $isSet$(ModuleHomoΣ M N) which is what we wanted. □

5.3.1.1 Univalence

To prove that RMod in univalent is going to be more complicated. This is since there are several equivalent definitions of being univalent.

Before we give the standard definition, consider the function pathToIso that takes two object A and B in our precategory, an equality $A \equiv B$ and returns $A \cong B$, which we recall is a proof that A and B are isomorphic as objects. Since we know that $A \cong A$ by the identity morphism on A we can use the equality $A \equiv B$ and then transport on the second A in $A \cong A$ to get $A \cong B$.

Given pathToIso the standard definition of a category being univalent is that for all object A and B in the category there is a proof of isEquiv (pathToIso A B). Given this it follows that $(A \equiv B) \simeq (A \cong B)$ since pathToIso is an equivalence between $(A \equiv B)$ and $(A \cong B)$.

For our application a direct proof of this proved more complicated than to prove $(A \equiv B) \simeq (A \cong B)$ and then proving that this implies isEquiv (pathToIso A B). The proof of $(A \equiv B) \simeq (A \cong B)$ follows from the SIP representation and the similar proof for left modules in Cubical/Algebra/Module. The translation from isomorphism of left modules to isomorphism in the category RMod will be omitted since it is mostly mechanical.

To prove that $(A \equiv B) \simeq (A \cong B)$ implies isEquiv (pathToIso A B) we will first prove another equivalent condition for a precategory to be univalent. An alternative definition for a precategory C to be univalent is that for all object A in C we must prove isContr (Σ (B : Object C) ($A \cong B$)). That is for any object A there is only one pair of an object B and a proof that A is isomorphic to B in C. We denote this definition $UnivalentAlt$.

Now consider the similar expression isContr (Σ (B : Object C) ($A \equiv B$)). Here the only part that is different is $A \equiv B$. Thus if we knew $(A \equiv B) \equiv (A \cong B)$ and could prove isContr (Σ (B : Object C) ($A \equiv B$)) for any A we could transport $(A \equiv B)$ to $(A \cong B)$ and get isContr (Σ (B : Object C) ($A \cong B$)). Luckily there is a known proof for isContr (Σ (B : Object C) ($A \equiv B$)) and since we have $(A \equiv B) \simeq (A \cong B)$ one may think that we are done, by using the univalence axiom to get $(A \equiv B) \equiv (A \cong B)$.

The problem here is that the univalence axiom is a property of a universe. Similarly paths must also stay within the same universe. An intuition for this is that if we had a path that did not end in the same universe it started in, then such a path would at some point have to jump from one universe to another, and would thus not be continuous. Functions however may return elements in another universe, and we see from the definition that such a function can be an equivalence. Thus equivalences of types can go between universes, but paths can not, and thus the univalence axiom is not applicable for such equivalences.

One should note that this is not just a technical detail, this is exactly the problem we are currently facing. In order to simplify the notation we have thus far not written down the universe levels. We will however need to do that now to clarify the problem. Let M be a module over R and let U_ℓ be the universe containing

the underlying type of M, which is $\langle M \rangle$. Then since M itself contains $\langle M \rangle$ it can not be within the same universe U_ℓ, but instead the next universe $U_{\ell+1}$. A module homomorphism from M to N, where both $\langle M \rangle$ and $\langle N \rangle$ are in U_ℓ, only deal with mapping the elements of $\langle M \rangle$ and $\langle N \rangle$ and thus the module homomorphism belongs to U_ℓ.

Knowing this, we recall that paths stay within the same universe, and we see that $M \equiv N$ is in the universe $U_{\ell+1}$ while $M \cong N$ only contains elements of the module homomorphism types and thus stay in U_ℓ. From this we see that $(A \equiv B) \simeq (A \cong B)$ goes between universes and we can not apply the univalence axiom. In section 5.3.1.2 we will prove the equivalence of these definitions for univalence in the general case. To do this we will however use the proof in the specific case where the objects and the morphisms of the precategory is in the same universe. We will here assume this is the case, for now, and later show that this assumption is unnecessary.

By our assumption we can then use univalence to prove the alternative definition and we only need to prove that if for every object A in our precategory we have isContr $(\Sigma\ (B : \text{Object } C)(A \cong B))$ then isEquiv (pathToIso A B). Here we realize that these are two equivalent ways to represent equivalence in homotopy type theory, see section 5.1, and we are done. We now realize that these three expressions,

1. $(A\ B : \text{Object } C) \to (A \equiv B) \simeq (A \cong B)$
2. $(A : \text{Object } C) \to \text{isContr } (\Sigma(B : \text{Object } C)(A \cong B))$
3. isEquiv (pathToIso A B)

are equivalent ways to represent that a precategory is univalent. Note here that we have proven that 3 implies 1, 1 implies 2 and 2 implies 3. Also note that the implication between 1 and 2 was only done for precategories where the objects and the morphisms were in the same universe. We will prove the general case in the following section.

5.3.1.2 Liftings and univalence

To do the proof for the general case we will use liftings. A lifting is an element of the lifting type, which is defined as a dependent type that takes two universes U_ℓ and $U_{\ell'}$ as well as a type A in U_ℓ. It then returns a record in the universe $U_{\max(\ell,\ell')}$. In pseudo Agda syntax it can be defined as

$$
\begin{aligned}
&\text{record Lift } \{U_\ell\}\ \{U_{\ell'}\}\ (A : \text{Type } U_\ell) : \text{Type } U_{\max(\ell,\ell')}\ \textit{where}\\
&\qquad \text{constructor lift}\\
&\qquad \text{field}\\
&\qquad\quad \text{lower } : A
\end{aligned}
$$

where the standard library definition is given in Cubical/Foundations/Prelude.agda.

Note here that the universes U_ℓ and $U_{\ell'}$ can be inferred from the context. If we realize that the constructor lift lifts a specific element of A and lower returns the specific element that was originally lifted, then it should be clear that these are inverses. In cubical type theory it is also easy to see that liftings are equal if and only if their underlying elements are equal. It is proven by the theorems

$$\begin{aligned}&\text{liftExt} \ : \{a\ b : Lift\{U_\ell\}\{U_{\ell'}\}A\} \to \text{lower}\ a \equiv \text{lower}\ b \to a \equiv b\\&\text{liftExt}\ p\ i = \text{lift}\ (p\ i)\end{aligned}$$

and the similar proof

$$\begin{aligned}&\text{lowerExt} \ : \{a\ b : Lift\{U_\ell\}\{U_{\ell'}\}A\} \to a \equiv b \to \text{lower}\ a \equiv \text{lower}\ b\\&\text{lowerExt}\ p\ i = \text{lower}\ (p\ i).\end{aligned}$$

These are intuitive if we think of lift and lower as continuous functions and recall that the image of a path under a continuous function is also a path.

Let C be a category with objects in U_ℓ and morphisms in $U_{\ell'}$, and where for all objects A and B we have $(A \equiv B) \simeq (A \cong B)$. We want to show that isContr $(\Sigma\ (B :$ Object $C)\ (A \cong B))$. The idea of our proof will be to lift both the objects and the morphisms to $U_{\max(\ell,\ell')}$, use the proof we already have for when the objects and the morphisms are in the same universe, and finally to use lowerExt to bring it all back down to the relevant universes. To this end we start by defining the dependent type PreCatLift. PreCatLift takes a category C and returns a new category C' by mapping lift on the objects and morphisms in C. We thus lift the objects to $U_{\max(\ell,\ell')}$ and the morphisms to $U_{\max(\ell',\ell)}$ which is equal to $U_{\max(\ell,\ell')}$. We can define composition by

$$\_ \circ_{C'} \_ := \lambda(\text{lift}\ f)(\text{lift}\ g) \to \text{lift}\ (f \circ_C g).$$

Note here that since we are dealing with both the categories C and C' we have used the notation $\_ \circ_C \_$ to refer to the composition in C. Similarly we will use $\_ \cong_C \_$ as notation for isomorphism in C. The other conditions of being a precategory are all paths, which can be lifted by liftExt.

For the sake of notation we let $C' =$ PreCatLift C. For A and B that are objects in C we let A' and B' be the corresponding objects in C'. We first want to show $(A' \equiv B') \simeq (A' \cong_{C'} B')$. We will show that this follows from transitivity of $\simeq$, and a few lemmas.

Lemma 5.3.1. *For any category C and objects A, B in C we let $C' =$ PreCatLift C and let A', B' be the corresponding liftings of A and B respectively. Then we have that $(A' \equiv B') \simeq (A \equiv B)$.*

Proof. We simply note that LowerExt and liftExt are inverses since they only apply lower and lift respectively, which are themselves inverses. □

Lemma 5.3.2. *Let $A\ B$: Object C, $C' =$ PreCatLift C and let A', B' be the corresponding liftings of A and B respectively. Then $(A \cong_C B) \simeq (A' \cong_{C'} B')$.*

Proof. First we define a function toLift : $(A \cong_C B) \to (A' \cong_{C'} B')$ by using lift on the morphism and liftExt on there equalities. For the other direction we define fromLift : $(A' \cong_{C'} B') \to (A \cong_C B)$ by using lower and lowerExt. Since the functions we applied on each morphism and path are each others inverses we see that toLift and fromLift are inverses by reflexivity of paths. Thus we have proven that $(A \cong_C B) \simeq (A' \cong_{C'} B')$. □

Now we are ready to prove the proposition we were after.

Proposition 5.3.2. *Let* $C' =$ PreCatLift C *and for two objects* $A\ B :$ Object C *let* A', B' *be the corresponding liftings of* A *and* B *respectively. Then the proposition* $(A \equiv B) \simeq (A \cong_C B)$ *implies* $(A' \equiv B') \simeq (A' \cong_{C'} B')$.

Proof. This follows directly from the above lemmas and transitivity of $\simeq$ since $(A' \equiv B') \simeq (A \equiv B) \simeq (A \cong_C B) \simeq (A' \cong_{C'} B')$. □

Now since we have that the objects and the morphism in C' are in the same universe and for all objects A' and B' we have that $(A' \equiv B') \simeq (A' \cong_{C'} B')$ we can use our theorem above to get a proof for $(A' :$ Object $C') \to$ isContr $(\Sigma\ (B :$ Object $C')\ (A' \cong_{C'} B'))$. Before we can lower this back down we will need one last lemma.

Lemma 5.3.3. *For any object* A *in* C *let* $C' :=$ PreCatLift C *and* A' *be the corresponding lifting of* A. *Then*

$$\begin{aligned}&\text{Lift}\ \ (\Sigma\ (B : \text{Object}\ C)\ (A \cong_C B)) \simeq \\ &\qquad \left(\Sigma\ (B' : \text{Object}\ C')\ (A' \cong_{C'} B')\right).\end{aligned}$$

Proof. Here we once again define a function toLiftCat that first takes (isContr $(\Sigma\ (B :$ Object $C)\ (A \cong_C B))$) out of the lifting and then lifts B to B' and $A \cong_C B$ to $A' \cong_{C'} B'$ as we have done before. And similarly toLowerCat maps lift on (isContr $(\Sigma\ (B :$ Object $C)\ (A \cong_C B))$) which we get from lowering B' and $A' \cong_{C'} B'$, as above.

Thus we see that these expressions are equal and we have proven the lemma. □

Finally we can prove the theorem we wanted.

Theorem 5.3.4. *Given a category* C *and that for all* A *and* B *we have* $(A \equiv B) \simeq (A \cong_C B)$ *we can prove* $(A :$ Object $C) \to$ isContr $(\Sigma\ (B :$ Object $C)\ (A \cong_C B))$.

Proof. We will need to prove that for any A in C we have isContr $(\Sigma\ (B :$ Object $C)\ (A \cong_C B))$. We can prove this by giving an element of $\Sigma\ (B :$ Object $C)\ (A \cong_C B))$ and proving that it is equal to every other element.

For our initial element let B be A and let $A \cong_C B$ be $A \cong_C A$ by the identity morphisms on A. What remains to show is that this element is equal to every other element.

Let our element with the identity morphism be denoted x. Let z be any other element. We need to show $x \equiv z$. Here we use lowerExt and thus only need to show lift $x \equiv$ lift z. Now note that these are both of the type Lift $(\Sigma\ (B : \text{Object } C)\ (A \cong_C B))$. Also note that this type is equivalent to $(\Sigma\ (B : \text{Object } C')\ (A' \cong_{C'} B'))$ by lemma 5.3.3. The lemma claimed that they were equivalent, but since they are in the same universe we can use univalence to prove that they are equal. Also since we have proven that $(\Sigma\ (B : \text{Object } C')\ (A' \cong_{C'} B'))$ is a contraction we can also prove that Lift $(\Sigma\ (B : \text{Object } C)\ (A \cong_C B))$ is a contraction by transporting along their equality and using cong on isContr.

Thus lift x and lift z are elements in a contraction, and since contractions are propositions they must be equal. As stated earlier, lift $x \equiv$ lift z gives $x \equiv z$ by lowerExt and we are done. □

5.3.2 Proving RMod is abelian

To prove that RMod is abelian we will start by constructing the necessary structures. A large part of these constructions are purely mechanical and follow from simple algebraic properties. Thus we will omit some of the mechanical parts and instead focus on the key points in the construction.

5.3.2.1 Zero Object

First we start by showing that RMod has a zero object. We know from abstract algebra that this object should be a module where the zero element is the only element. From this it should be clear that the one element type, $OneElem$, is a good choice for the underlying type of the zero object. To make it a module we will equip it with the following structure

$$\begin{aligned} 0_{Zero} &:= * \\ * +_{Zero} * &:= * \\ -_{Zero} * &:= * \\ r \cdot_{Zero} * &:= * \end{aligned}$$

where we recall that $*$ is the only element of $OneElem$. The proof that this forms a module is simple, and most conditions are derived by case splitting on the elements of $OneElem$. We denote this module as $ZeroModule$.

To see that $ZeroModule$ is a zero object in RMod we will show that it is both initial and terminal. Before we can prove these properties we will have a short lemma.

Lemma 5.3.5. *For any modules M and N the function from $\langle M \rangle$ to $\langle N \rangle$ defined by $\lambda x \to 0_N$ is a homomorphism.*

Proof. Let us denote this function f. We will show that f is additive and preserves scalars. The first condition follows directly by

$$f(a +_M b) \equiv 0_N \equiv 0_N +_N 0_N \equiv f(a) +_N f(b).$$

Similarly the condition for scalars is shown by

$$f(r \cdot_M a) \equiv 0_N \equiv r \cdot_N 0_N \equiv r \cdot f(a).$$

Note here that $r \cdot_N 0_N \equiv 0_N$ is one of the module properties proven in appendix A. □

Proposition 5.3.3. *ZeroModule is a terminal object in R*Mod.

Proof. We need to show that for any module M the homset from M to *ZeroModule* is a contraction. We start by defining a homomorphism from M to *ZeroModule*. The underlying function of this is simply $h := \lambda x \to *$. By lemma 5.3.5 this is a homomorphism h'. It remains to show that any homomorphism k' we have that $h' \equiv k'$. As we discussed in section 5.2 it is sufficient to show that for all $x : \langle M \rangle$ we have $h(x) \equiv k(x)$. This follows directly from the fact that they are both elements of *OneElem* and *OneElem* is a contraction, and thus a proposition. □

Proposition 5.3.4. *ZeroModule is an initial object in R*Mod.

Proof. Let M be a module over R. We will show that the homset from *ZeroModule* to M is a contraction. We again define the function between underlying types by $h := \lambda x \to 0_M$ and by lemma 5.3.5 we get a homomorphism h'. Let k' be a homomorphism from *ZeroModule* to M. It remains to show $h' \equiv k'$ which is equivalent to for all $x : \mathit{OneElem}$ we have $h(x) \equiv k(x)$. Now this follows by

$$h(x) = 0_M \equiv k(0_{\mathit{OneElem}}) \equiv k(x)$$

where $k(0_{\mathit{OneElem}}) \equiv 0_M$ is a property we have shown for any homomorphism in appendix A and $x \equiv 0_{\mathit{OneElem}}$ since *OneElem* is a contraction and thus a proposition. □

Thus, since *ZeroModule* is both initial and terminal it is a zero object. Before we move on we should note that the zero morphisms from any module M to any module N are the morphisms where their underlying functions maps every element of $\langle M \rangle$ to 0_N.

5.3.2.2 Product

We start by constructing the product object in RMod, which we denote MXN for two modules M and N. The underlying type of MXN is $\langle M \rangle \times \langle N \rangle$. To make this a module we equip this type with the following structure

$$\begin{aligned} 0_{MXN} &:= (0_M, 0_N) \\ (a, b) +_{MXN} (c, d) &:= (a +_M c, b +_N d) \\ -_{MxN}(a, b) &:= (-_M a, -_N b) \\ r \cdot_{MXN} (a, b) &:= (r \cdot_M a, r \cdot_N b) \end{aligned}$$

It is simple to show that this is a module using that each condition for being a module holds for both M and N as well as the property for Σ-types that if $a\ b : A \times B$, fst $a \equiv$ fst b and snd $a \equiv$ snd b then $a \equiv b$.

We now define the projections as $p_M := fst$ and $p_N := snd$. To verify that p_M and p_N are homomorphisms is left as a simple exercise to the reader. The proof that this forms a product diagram is identical to the one in **SET** and we thus refer the interested reader to that proof, section 3.3.3.

5.3.2.3 Coproduct

We will now see that the coproduct is the same object as the product. We know this must be the case since in an abelian category products are binary direct products, which in turn are both products and coproducts. Also recall that coproducts are unique up to isomorphism.

To construct the injections we define $i_A := \lambda a \to (a, 0_N)$ and $i_B := \lambda b \to (0_M, b)$. The reader may verify that these are linear. We prove that this is a coproduct diagram in the following proposition.

Proposition 5.3.5. *The module MXN is a coproduct in R*Mod *where the injections may have the underlying functions $i_A := \lambda a \to (a, 0_N)$ and $i_B := \lambda b \to (0_M, b)$.*

Proof. We will need to show MXN, i'_A and i'_B form a coproduct diagram. For a module Z and for any two homomorphism $f' : M \Rightarrow Z$ and $g' : N \Rightarrow Z$ we define h' as the homomorphism with the underlying function $h(a, b) := f(a) +_Z g(b)$. The linearity of h follows from the linearity of f and g, as well as the commutativity and associativity of $\_ +_N \_$. The proof is purely mechanical and is left to the reader.

It is clear that $i'_A \circ h' \equiv f'$ since

$$h(i_A(a)) \equiv h(a, 0_N) \equiv f(a) +_Z g(0_N) \equiv f(a) +_Z 0_Z \equiv f(a)$$

by the arithmetic properties of modules that we have shown in appendix A. A similar proof shows $i'_B \circ h' \equiv g'$.

We want to show that if there is a homomorphism k' satisfying $i'_A \circ k' \equiv f'$ and $i'_B \circ k' \equiv g'$ then $k' \equiv h'$. Since they are homomorphism and by functional extensionality it is sufficient to prove $h(a,b) \equiv k(a,b)$. The proof is given by

$$h(a,b) \equiv f(a) +_Z g(b) \equiv k(i_A(a)) +_Z k(i_B(b)) \equiv k(a,0_N) +_Z k(0_M,b) \equiv k(a,b)$$

where the last step follows from linearity of k and properties of additions with 0 in a module. □

5.3.2.4 Kernels

Let f' be a module homomorphism between modules A and B. We will show that there is a kernel of f' in RMod.

The underlying type of the kernel object is $\Sigma\ (a : A)\ (f(a) \equiv 0_B)$. These are the objects in A such that f' maps them to the 0 element in B. To simplify the notation, let us first consider how the structure will affect the first element of our Σ-type.

$$\begin{aligned} \text{fst}\ (0_{Kerf}) &:= 0_A \\ \text{fst}\ ((a, fa = 0) +_{Kerf} (b, fb = 0)) &:= a +_A b \\ \text{fst}\ (-_{Kerf}(a, fa = 0)) &:= -_A a \\ \text{fst}\ (r \cdot_{Kerf} (a, fa = 0)) &:= r \cdot_A a \end{aligned}$$

For the second part we use following proofs

$$\begin{aligned} f(0_A) &\equiv 0_B \\ f(a +_A b) \equiv f(a) +_B f(b) &\equiv 0_B +_B 0_B \equiv 0_B \\ f(-_A a) \equiv -_B f(a) &\equiv -_B 0_B \equiv 0_B \\ f(r \cdot_A a) \equiv r \cdot_B f(a) &\equiv r \cdot_B 0_B \equiv 0_B \end{aligned}$$

where each step follows from simple properties we have shown for module homomorphisms. To show that this is a module we first note that $\langle B \rangle$ is a set, so $f(a) \equiv 0_B$ is a property. Thus to show that two elements $(a, fa = 0)\ (b, fb = 0) : \Sigma\ (a : A)\ (f(a) \equiv 0_B)$ are equal it is sufficient to prove $a \equiv b$.

Note that every condition except that a module is a set ends in showing an equality. By the previous remarks to show these equalities it is sufficient to prove them for the first elements of the Σ-type. If we note that the structure on these is the same as for A we can use that we know that A is a module to prove that this also is a module. We denote this module $Kerf$. It remains to show that this is the kernel of f.

Proposition 5.3.6. *$Kerf$ is the kernel of f'.*

Proof. We start by defining the kernel morphism. Let $kerf :=$ fst be the underlying function of the module homomorphism $kerf'$ from $Kerf$ to A. Linearity follows

directly by the definitions of the operators for $Kerf$. Note that this is simply the inclusion of $Kerf$ into A.

For any element $(a, fa = 0) : \langle Kerf \rangle$ it is clear that $f(kerf(a, fa = 0)) \equiv f(a) \equiv 0_B$ where the first equality is by definition of $kerf$ and the second by $fa = 0 : f(a) \equiv 0_B$. Thus $kerf' \circ f' \equiv 0$ since the zero morphism maps every element to 0.

Now we have that if $h' : D \Rightarrow A$ and for all $d : D$ we have $hf = 0 : f(h(d)) \equiv 0_B$ then we need to show there exists a homomorphism k' from D to $Kerf$ such that $k' \circ ker' \equiv h'$. We define k' by the underlying function $k := \lambda d \to (h(d), hf = 0)$. Again, since $\langle B \rangle$ is a set it is sufficient to prove that $\lambda d \to h(d)$ is linear, which holds since h' is a homomorphism. Now to see that $k' \circ ker' \equiv h'$ we simply note that $ker(k(d)) \equiv \text{fst } (h(d), hf = 0) \equiv h(d)$.

Finally we will want to prove that k' is unique. To do this it is simple to prove the more general condition that k' is monic. Let $g'\ f' : D \Rightarrow Kerf$ and $f' \circ kerf' \equiv g' \circ kerf'$. Then we must prove that $f' \equiv g'$. By functional extensionality on the underlying functions and since B is a set it is sufficient to prove that the first elements of the Σ-type are equal, thus fst $f(d) \equiv$ fst $g(d)$ for all $d : D$. But this is just the underlying function of $f' \circ kerf' \equiv g' \circ kerf'$ which we were given a proof of by assumption. Since the underlying functions are equal if and only if the homomorphisms are equal we are done. □

5.3.2.5 Cokernels

For any module homomorphism $f' : A \Rightarrow B$, we know from abstract algebra that the cokernel of f' should be the quotient space $\langle B \rangle / \langle \text{Im } f \rangle$. This is a space of equivalence classes where two elements from $\langle B \rangle$ are in the same equivalence class if they are equal modulo an element of $\langle \text{Im } f' \rangle$. To this end we start by defining our equivalence relation.

We define the relation $\sim$ as $b \sim b' := \Sigma\ (a : \langle A \rangle)\ (b' \equiv f(a) +_B b)$. For this relation $b \sim b'$ if there exists an element of $\langle A \rangle$ such that $b' \equiv f(a) +_B b$. Note that we could also have used the equality $b' +_B (-_B b) \equiv f(a)$ as the second condition. If f is not injective then the proof of $b \sim b'$ might not be unique, since f might map two different elements of $\langle A \rangle$ to $b' +_B (-_B b)$. Thus we will need to use propositional truncation on the relation to make it propositionally valued. This is also called prop-valued and a relation $\sim$ is defined as propositionally valued if $b \sim b'$ is a proposition for all $b\ b' : \langle B \rangle$.

We will start by showing that this is an equivalence relation. Since the truncation of an equivalence relation is an equivalence relation it is sufficient to show this for the untruncated relation.

Proposition 5.3.7. *The relation $\_ \sim \_$ is an equivalence relation.*

Proof. First we prove reflexivity. For any $b : \langle B \rangle$ it is clear that $b \sim b$ by 0_A since

$$b +_B (-_B b) \equiv 0_B \equiv f(0_A)$$

.

For symmetry we see that if $(a, b' = fab) : b \sim b'$ then $-_A a$ satisfies that $b \equiv f(-_A a) +_B b'$ since

$$b \equiv b' +_B (-_B f(a)) \equiv (-_B f(a)) +_B b' \equiv f(-_A a) +_B b'$$

by known properties of module homomorphisms.

Finally we see that the relation is transitive since if $(a, b' = fa + b) : b \sim b'$ and $(a', b'' = fa' + b') : b' \sim b''$ then $a' +_A a$ satisfy $b'' \equiv f(a' +_A a) +_B b$ since

$$b'' \equiv f(a') +_B b' \equiv f(a') +_B f(a) +_B b \equiv f(a' +_A a) +_B .$$

by linearity of f. Since we have now shown that the relation is reflexive, transitive and symmetric we know that this is an equivalence relation.

□

Now that we have our equivalence relation we can define our underlying set $\langle B \rangle / \sim$ which is the set of equivalence classes from $\langle B \rangle$ generated by the relation $\_ \sim \_$. In pseudo Agda code the definition can be given as

$$\begin{aligned}
&\text{data } _/_ \ (A : \text{Type }) \ (\sim\!: A \to A \to \text{Type }) \ \textit{where} \\
&\quad [_] : (a : A) \to A/\sim \\
&\quad \textit{eq}/ : (a\ b : A) \to (r : a \sim b) \to [a] \equiv [b] \\
&\quad \textit{squash}/ : (x\ y : A/\sim) \to (p\ q : x \equiv y) \to p \equiv q.
\end{aligned}$$

Note here that [_] is the constructor which takes an element of A to its equivalence class under $\sim$, $eq/$ proves that if $a \sim b$ then $[a] \equiv [b]$ and $squash/$ ensures that $A/\sim$ is a set. This definition can be found in the standard library at Cubical/HITs/SetQuotients/Base.agda.

Now that we have our underlying set we will need to define our structure and show that this is a module, denoted $CoKf$. To do this we will need to use elimination rules, but for these the notation might become quite complicated. For the implementation of this we have provided helper functions in the script ThesisWork/SetQuotientHelp.agda. This script contains versions of the elimination rule for function with 2 or 3 inputs.

The original elimination rule says that for any $P : (x : \langle B \rangle / \sim) \to \text{Type}$ such that $P\ x$ is a set for all $x : \langle B \rangle / \sim$ and a function $g : (b : \langle B \rangle) \to P\ [b]$ where for any $b\ b' : \langle B \rangle$ where $b \sim b'$ we have that $g(b) \equiv g(b')$. Then there exists a function

$g' : (x : \langle B \rangle / \sim) \to P\ x$. In set theory this rule states that we may use a function on the original set to induce a function on our equivalence classes if such a function would be well defined. Note that such a function is well defined if g maps every underlying element of an equivalence class x to the same element in $P\ x$.

To simplify the notation we will prove the necessary conditions to use the elimination rule and leave the reader the task of applying the elimination rule. Note that since $\langle B \rangle$ is a set we will only need to construct g and prove that for $b\ b' : \langle B \rangle$ we have $g(b) \equiv g(b')$.

First we define $\_ +' \_ : (b\ b' : \langle B \rangle) \to \langle B \rangle / \sim$ as $\lambda b\ b' \to [b +_B b']$, where $[b]$ denotes the equivalence class of b under $\sim$. We need to show that for $b\ b'\ b''$ where $(a, b' = b + fa) : b \sim b'$ we have $[b +_B b''] \equiv [b' +_B b'']$ and $[b'' +_B b] \equiv [b'' +_B b']$. To prove $[b +_B b''] \equiv [b' +_B b'']$ it is sufficient to show that $b +_B b'' \sim b' +_B b''$ but this is clear since

$$b' +_B b'' \equiv f(a) +_B b +_B b''$$

by cong $(\lambda x \to x +_B b'')$ and $b' = b + fa : b' \equiv b +_B f(a)$. The proof is analogous for $[b'' +_B b] \equiv [b'' +_B b']$. Thus we can use the elimination rule to induce a function $\_ +_{CoKf} \_$.

Using the elimination rule we make similar constructions to define $-_{CoKf}[b]$ as $[-_B b]$ and $r \cdot [b]$ as $[r \cdot b]$.

To show that this forms a module we will use that we know B is a module along with the elimination rule. Since the quotient set is a set by definition our equalities are propositions. It is thus trivial to prove the set requirement for the elimination rule. Going from the proof that B is a module to the proof that $\langle B \rangle / \sim$ is a module follows a very similar structure. We will show how to prove that $\_ +_{CoKf} \_$ is associative and the reader should be able to construct similar proofs for the rest of the conditions.

Lemma 5.3.6. $\_ +_{CoKf} \_$ *is associative.*

Proof. We will need to show that for any $x\ y\ z : \langle B \rangle / \sim$ we have that $x +_{CoKf} (y +_{CoKf} z) \equiv (x +_{CoKf} y) +_{CoKf} z$. To do this we let $P := \lambda x\ y\ z \to x +_{CoKf} (y +_{CoKf} z) \equiv (x +_{CoKf} y) +_{CoKf} z$ and $P\ x\ y\ z$ is a proposition since the quotient set is a set by definition. Thus we can use out elimination rule for 3 arguments to prove the lemma given that we can prove $(b\ b'\ b'' : \langle B \rangle) \to [b] +_{CoKf} ([b'] +_{CoKf} [b'']) \equiv ([b] +_{CoKf} [b']) +_{CoKf} [b'']$ and prove that the induced function is well defined.

To prove $(b\ b'\ b'' : \langle B \rangle) \to [b] +_{CoKf} ([b'] +_{CoKf} [b'']) \equiv ([b] +_{CoKf} [b']) +_{CoKf} [b'']$ we let $b\ b'\ b'' : \langle B \rangle$. Then we realize that by definition of $+_{CoKf}$ we have that

$$[b] +_{CoKf} ([b'] +_{CoKf} [b'']) \equiv [b] +_{CoKf} [b' +_B b''] \equiv [b +_B (b' +_B b'')]$$

and

$$([b] +_{CoKf} [b']) +_{CoKf} [b''] \equiv [b +_B b'] +_{CoKf} [b''] \equiv [(b +_B b') +_B b''].$$

So what remains to be proven is that $[b +_B (b' +_B b'')] \equiv [(b +_B b') +_B b'']$. But by associativity of $+_B$ we have $b +_B (b' +_B b'') \equiv (b +_B b') +_B b''$ and thus

$$(b +_B b') +_B b'' \equiv f(0_A) + b +_B (b' +_B b'')$$

which gives us $b +_B (b' +_B b'') \sim (b +_B b') +_B b''$. Now since $b +_B (b' +_B b'') \sim (b +_B b') +_B b''$ we also have that $[b +_B (b' +_B b'')] \equiv [(b +_B b') +_B b'']$.

The final step is to show that we give the same proof for any underlying elements in our equivalence classes, but this is trivial since we have shown an equality of elements in a set. So since $\langle B \rangle / \sim$ is a set, our proof is unique and thus the induced function is well defined. □

The proofs for the other conditions follow the same line of reasoning. We use the elimination rule to only consider the underlying elements of the equivalence classes. Then we reduce the expressions and realize that we only need to show equality of two equivalence classes where the underlying elements are equal since B is a module. Then we use that $b \equiv b'$ implies $b \sim b'$ and that these are equalities on $\langle B \rangle / \sim$, which is a set.

Now that we have proven that $CoKf$ is a module we need to show that it is a cokernel of f'.

Proposition 5.3.8. *$CoKf$ a cokernel of f'.*

Proof. First let us define the cokernel morphism $coKf' : B \Rightarrow CoKf$ by the underlying function $cokf := \lambda x \to [x]$. It is clear that this function is linear since

$$[b +_B b'] \equiv [b] +_{CoKf} [b']$$

and

$$r \cdot_{CoKf} [b] \equiv [r \cdot_B b]$$

by definition.

Then to see that $f' \circ coKf' \equiv 0$ we note that for all $a : A$ we have that

$$0_B \equiv f(0_A) \equiv f((-_A a) +_A a) \equiv f(-_A a) +_B f(a)$$

so $0_B \sim f(a)$ by $-_A a$ and the proof above, and thus $[0_B] \equiv [f(a)]$.

Now we must show that for any module homomorphism $h' : B \Rightarrow D$ where $f' \circ h' \equiv 0$ we have that there exists $k' : CoKf \Rightarrow D$ such that $coKf' \circ k' \equiv h'$. To this end we use the elimination rule to define $k : CoKf \to \langle D \rangle$. Since $\langle D \rangle$ is a set we only need to define a function on the underlying elements of our equivalence classes. To this end we choose this to be $\lambda b \to h(b)$, which is just h. It remains to show that if

$b \sim b'$ then $h(b) \equiv h(b')$. Let $a : A$ such that $b' \equiv f(a) + b$ and consider the following equation

$$h(b') \equiv h(b +_B f(a)) \equiv h(b) +_D h(f(a)) \equiv h(b) +_D 0_D \equiv h(b)$$

since $f' \circ h' \equiv 0$ which means that their underlying functions are equal, and that the zero morphism from A to D maps every element of A to 0_D.

To show that the induced function k is linear we simply use elimination and it is linear on the underlying elements of each equivalence class since h is linear. To show that this is well defined we simply note that $\langle D \rangle$ is a set so any two equivalence proofs are equal.

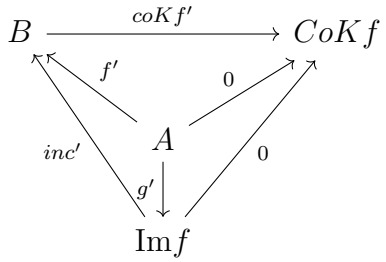
To see that $coKf' \circ k' \equiv h'$ it is sufficient to show that the underlying functions are equal. By functional extensionality we take any $b : B$ and prove $k(coKf(b)) \equiv h(b)$ which by definition of $coKf$ is $k([b]) \equiv h(b)$ which is the definition of k.

It remains to show the uniqueness of k'. This follows if we prove that $coKf'$ is epic. To do this let D be a module and $g'\ h' : CoKf \Rightarrow D$ where $coKf' \circ g' \equiv coKf' \circ h'$. We need to prove $g' \equiv h'$ and as above it holds if we have $g(x) \equiv h(x)$ for any $x : \langle B \rangle / \sim$. Now we use the elimination rule on x and need to show that for any $a : A$ we have $g([a]) \equiv h([a])$. This is just the underlying functions of $coKf' \circ g' \equiv coKf' \circ h'$ evaluated at a. We now note that the elimination is an equality and that $\langle D \rangle$ is a set and thus we are done. □

5.3.2.6 Monics are kernals

In order to prove that monics are kernels and that epics are cokernels we will follow a similar proof to that used in "Additive, abelian, and exact categories" [12].

We start by showing the idea of the proof. For a monic module homomorphism $f' : A \Rightarrow B$ consider the following diagram



Here $CoKf$ is the cokernel of f', $coKf'$ is the cokernel morphism, Imf is the image of f and inc is the inclusion map. To make the diagram commute we note that $g' \circ inc' \equiv f'$ and thus we chose g to be the function $\lambda a \to fa$. We want to show that f' is a kernel of $cokf'$. To do this we will first show a lemma which proves it is sufficient to show that inc' is a kernel of $cokf'$ and that g' is an isomorphism. The paper [12] claims that it is clear that inc' is a kernel to $coKf'$. It also claims that

since f' is monic f is also injective, and thus g is injective, and since g is surjective by construction it is an isomorphism.

For our proof we will start by proving the lemma mentioned above. This is stated in the paper [12] but the proof is left to the reader.

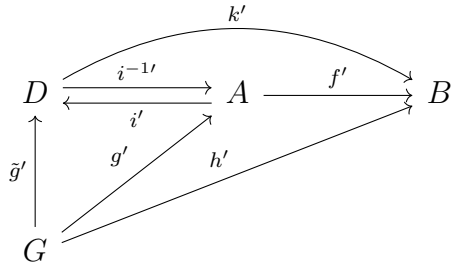
Lemma 5.3.7. *Let $f' : A \Rightarrow B$ be a homomorphism. If $k' : D \Rightarrow B$ is the kernel morphism of $m' : B \Rightarrow E$ and we have an isomorphism $i' : A \Rightarrow D$ such that $i' \circ k' \equiv f'$ then f' is a kernel to m'.*

Proof. As stated in the paper, the proof is simple. First we see that

$$f' \circ m' \equiv i' \circ k' \circ m' \equiv i' \circ 0 \equiv 0$$

since any composition with the zero morphism is itself the zero morphism.

Next we need to show that for any morphism $h' : G \Rightarrow B$ where $h' \circ m' \equiv 0$ there is a morphism $g' : G \Rightarrow A$ such that $g' \circ f' \equiv h'$. To justify our coming definitions consider the following diagram



We note that since k' is a kernel of m' it follows that there exists $\tilde{g}' : G \Rightarrow D$ such that $\tilde{g}' \circ k' \equiv h'$. Now we define $g' := \tilde{g}' \circ i^{-1\prime}$. We now see that

$$g' \circ f' \equiv \tilde{g}' \circ i^{-1\prime} \circ i' \circ k' \equiv \tilde{g}' \circ \mathrm{id}\ D \circ k' \equiv \tilde{g}' \circ k' \equiv h'$$

by the definition of $\tilde{g}'$ and i being an isomorphism.

Finally we need to show that g' is unique. It is sufficient to show that f' is monic. To show that f' is monic, assume r' $s' : H \Rightarrow A$ such that $r' \circ f' \equiv s' \circ f'$. Then $r' \circ i' \circ k' \equiv s' \circ i' \circ k'$ and since k' is monic we have $r' \circ i' \equiv s' \circ i'$. This gives us that $r' \circ i' \circ i^{-1\prime} \equiv s' \circ i' \circ i^{-1\prime}$ and thus $r' \equiv s'$ and we have shown that f' is monic. Note that the last step is just the proof that i' is monic. □

An important part of the proof is that if a module homomorphism is monic then its underlying function is injective. Unfortunately most of the proofs we found for this were either not constructive, not suited for our homotopy type theory framework or required more complex proofs for properties of categories. Thus we have constructed the following proof.

Proposition 5.3.9. *A module homomorphism is monic in* RMod *if and only if its underlying function is injective.*

Proof. The first part is similar to the proof we gave for monics in **SET**. We assume $f' : A \Rightarrow B$ is a module homomorphism and that f is injective. Then if $g'\ h' : D \Rightarrow A$ and $g' \circ f' \equiv h' \circ f'$ we have by functional extensionality that for $x : A$ we have $g(x)\ h(x) : A$ and $f(g(x)) \equiv f(h(x))$ and thus $g(x) \equiv h(x)$ since f is injective. Now since the underlying functions of g' and h' are equal we have that $g' \equiv h'$.

Here we might want to give a similar proof to the one for **SET**, but naive constructions such as $\lambda x \to a$ and $\lambda x \to x + a$ are not linear for a unless $a \equiv 0$. Instead, we will give the following proof.

Assume $f' : A \Rightarrow B$ is monic and that $a\ b : A$ such that $f(a) \equiv f(b)$. We need to show that $a \equiv b$. We first construct the module $GenOneElem(a)$ which is the free module generated by an element a. This is the module of elements on the form $r \cdot a$. This means that the underlying type is $\Sigma\ (g : A)\ (\Sigma\ (r : \langle R \rangle)\ (g \equiv r \cdot a))$. Here the operators on the first element are the same as for A and the operators on $\langle R \rangle$ are the corresponding operators for R. To prove the equalities for the operators is simple and left as an exercise to the reader. To prove that this is a module is also simple. All the conditions follow from the fact that corresponding rules hold for A and R and that $g \equiv r \cdot a$ is a proposition. The details can be found in ThesisWork/RModules/MonicToInjective.agda.

Here one might want to use the fact that $f(r \cdot_A a) \equiv r \cdot_B f(a) \equiv r \cdot_B f(b) \equiv f(r \cdot_A b)$ to prove that the injections from $GenOneElem(a)$ and $GenOneElem(b)$ are equal and then use that f is monic to prove that $a \equiv b$. There are two problems with this approach. Firstly, to use the fact that f is monic we need to have that the injections come from the same module. This can be solved by taking the product module of $GenOneElem(a)$ and $GenOneElem(b)$, which we have shown exists for any two modules. The second issue is that any two elements of $GenOneElem(a)$ and $GenOneElem(b)$ are on the form $r \cdot a$ and $r' \cdot b$ and we can only use the proof above if $r \equiv r'$. We will solve these problems by choosing the following approach.

Firstly, we define a module $GenProd(a, b)$ which has the underlying type

$$\Sigma\ (r : \langle R \rangle)\ (\Sigma\ (g_1 : A)\ (\Sigma\ (g_2 : A)\ ((g_1 \equiv r \cdot a) \times (g_2 \equiv r \cdot b)))).$$

Similarly to the construction of $GenOneElem(a)$, the operators are defined by the corresponding operators for A and R and it is simple to prove that this forms a module. One may realize that this is the same module as $GenOneElem((a, b))$ for (a, b) in the product module of A and A.

Now we define $g := \lambda x \to \text{fst}\ (\text{snd}\ x)$ and $h := \lambda x \to \text{fst}\ (\text{snd}\ (\text{snd}\ x))$ and get that $g(x) \equiv r \cdot a$ and $h(x) \equiv r \cdot b$ for $r \equiv \text{fst}\ x$ where $x : GenProd(a, b)$. That these are linear follows directly from the definition of $+$ and $\cdot$ for $GenProd(a, b)$. Thus we

can now use our proof above to see that $f(g(x)) \equiv f(r \cdot_A a) \equiv f(r \cdot_A b) \equiv f(h(x))$ and thus $g' \circ f' \equiv h' \circ f'$ and since f' is monic we get $g' \equiv h'$.

Finally, since $g' \equiv h'$ we have that $g \equiv h$ and by functional extensionality $g(x) \equiv h(x)$ for all $x : GenProd(a, b)$. Let $x := (1_r, a, b, \dots)$ where $1_R \cdot_A a \equiv a$ and $1_R \cdot_A b \equiv b$ is part of the definition for A being a module. Then $a \equiv g(x) \equiv h(x) \equiv b$ and we are done.

□

We will also need to define the module $\mathrm{Im}f$. We do this by giving it the underlying type $\Sigma\ (b : B)\ (\Sigma\ (a : A)\ (b \equiv f(a)))$. Since we want elements in $\mathrm{Im}f$ to be equal if they are equal in B we will need to use propositional truncation on $\Sigma\ (a : A)\ (b \equiv f(a))$ to make it a proposition in the general case. In our case however f is injective, so this is already a proposition, so we will omit this step. This clearly forms a module by defining the operators by their corresponding operators from A and B.

To define g' we let $g := \lambda a \to (f(a), a, refl)$ and it follows that this is linear since f is linear. Then we define $g^{-1} := \lambda(b, a, p) \to a$ where $p : b \equiv f(a)$. It is clear that g^{-1} is linear from the definition of $+$ and $\cdot$ on $\mathrm{Im}f$. We see that $g^{-1}(g(a)) \equiv a$ by $refl$ and $g(g^{-1}(b, a, p)) \equiv (f(a), a, refl)$ since $b \equiv f(a)$ by $p : b \equiv f(a)$, $a \equiv a$ by $refl$ and $p \equiv refl$ since they are equalities in the set B. Since the compositions underlying functions are equal to the identity function they are the identity morphisms and thus g is an isomorphism.

We define $inc' : \mathrm{Im}f \Rightarrow B$ by the underlying function $inc := fst$ which clearly is linear. It remains to show that inc' is a kernel of $coKf'$.

Proposition 5.3.10. *The module homomorphism inc' is a kernel morphism to $coKf'$.*

Proof. First we need to show that $inc' \circ coKf' \equiv 0$ but by considering the underlying functions and functional extensionality it is sufficient to prove $[b] \equiv [0]$ for $(b, a, p) : \mathrm{Im}f$ where $p : b \equiv f(a)$. By equality on $CoKf$ it is sufficient to show $b \sim 0$ which is given by $b \equiv f(a) \equiv f(a) + 0$ and this step is done.

Now we must show that for any $h' : E \Rightarrow B$ where $h' \circ coKf' \equiv 0$ we have that there exists $u' : E \Rightarrow B$ such that $u' \circ inc' \equiv h'$. As above, from $h' \circ coKf' \equiv 0$ we can get $[h(e)] \equiv [0]$ for any $e : E$. Since $\sim$ is an equivalence relation and $b \sim b'$ is a proposition for all $b\ b' : B$ we can show that it is effective and thus get a proof of $h(e) \sim 0$ from $[h(e)] \equiv [0]$, see Cubical/HITs/SetQuotients/Properties.agda.

Now assume that $(a, p)\ (a', q) : b \sim b'$ for some $b\ b' : B$. Then $p : b' \equiv b +_B f(a)$ and $q : b' \equiv b +_B f(a')$ and we can show $f(a) \equiv b' -_B b \equiv f(a')$. Now since f is injective we get that $a \equiv a'$ and thus $(a, p) \equiv (a', q)$ since $p \equiv q$ from the fact

that $\langle B \rangle$ is a set. Thus $b \sim b'$ is a proposition before the propositional truncation. From this we can show that the truncation of $b \sim b'$ is equivalent to $b \sim b'$ itself. Thus we may remove the truncation in this case, and from $[h(e)] \equiv [0]$ we get $a : A$, $p : h(e) \equiv 0 +_B f(a)$ and that such an a must be unique.

Thus we get $q : h(e) \equiv 0 +_B f(a) \equiv f(a)$ from p and the definition of B being a module. Now we define $u := \lambda e \to (h(e), a, q)$ and $inc(u(e))) \equiv fst(h(e), a, q) \equiv h(e)$ and thus $u' \circ inc' \equiv h'$.

Finally we need to show that u' is unique. It is sufficient to prove that inc' is monic. This holds since inc is an injection, injections are injective and thus inc' is monic by proposition 5.3.9. The proof is analogous to how we showed that $kerf$ is monic. □

Now we have all the prerequisites to apply lemma 5.3.7 and we are done.

5.3.2.7 Shortened proof

In this case it is possible to give a shorter, more direct proof.

Proposition 5.3.11. *Let $f' : A \Rightarrow B$ be a monic module homomorphism. Then f' is a kernel morphism to $coKf'$.*

Proof. Here we will only give the proof idea here since the proof borrows a lot of steps from the previous proof.

First we note that $f' \circ coKf' \equiv 0$ by the definition of $cokf'$. We need to show that for any module homomorphism $h' : E \Rightarrow A$ where $h' \circ coKf' \equiv 0$ we have that there exists $u' : E \Rightarrow A$ such that $u' \circ f' \equiv h'$. As in proposition 5.3.10 we find that for every $e : E$ we have $h(e) \equiv f(a)$ for some $a : A$. We then define $u := \lambda e \to a$. It remains to show that u is linear, since we know $f(u(e)) \equiv f(a) \equiv h(e)$.

We will show that u is additive and leave the analogous proof that it preserves scalars to the reader. To prove this we get $e\ e' : E$ and we need to prove $u(e + e') \equiv u(e) + u(e')$. Since these are elements in A and f is injective it is sufficient to prove $f(u(e + e')) \equiv f(u(e) + u(e'))$. This follows from the calculations

$$f(u(e +_E e')) \equiv h(e +_E e') \equiv h(e) +_B h(e') \equiv f(u(e)) +_B f(u(e')) \equiv f(u(e) + u(e'))$$

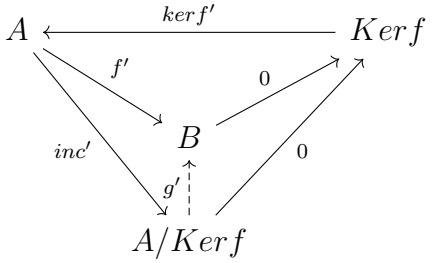
where we use that $f(u(x)) \equiv h(x)$ for all $x : E$ and that both h and f are linear.

Finally we note that since f' is monic u' must be unique. □

5.3.2.8 Epics are cokernels

We note that this is the dual statement for of monics are kernels. Thus the paper [12] presents a similar solution. For an epic module homomorphism f' consider the

following diagram



.

The proof idea is that we define g', $A/Kerf$ and inc' such that g' is an isomorphism, inc' is the cokernel morphism of $kerf$ and $inc' \circ g' \equiv f'$. Then we use the following lemma.

Lemma 5.3.8. *Let $f' : A \Rightarrow B$ be a morphism. If $c' : A \Rightarrow D$ is the cokernel morphism of $m' : E \Rightarrow A$ and we have an isomorphism $i' : D \Rightarrow B$ such that $c' \circ i' \equiv f'$ then f' is a cokernel morphism of m'.*

Proof. We see that this is just the dual statement of lemma 5.3.7 and thus the proof follows by exchanging the origin and destination of each morphism as well as the order of composition. □

The paper [12] then claims that $A/Kerf \cong A/\mathrm{Im}kerf$ and that $c := \lambda x \to [x]$ incuses a module homomorphism c' which is the cokernel of $kerf'$. Then since f' is epic we have that $B \cong \mathrm{im}f$ and then the paper uses a theorem to prove the existence of g'.

We will start by defining a relation on A. We define $a \sim a' := f(a) \equiv f(a')$. We note that this is an equivalence relation since $\equiv$ is an equivalence relation and that $a \sim a'$ is a proposition for all $a\ a' : A$ since $\langle B \rangle$ is a set.

Now we define $A/Kerf$ as having the underlying type $A/\sim$ and define the operators by using the corresponding operators on A and the elimination rule on set quotients. Now we define $inc := \lambda x \to [x]$ which is linear by the definitions of the operators and the elimination rule. This construction is analogous to how we defined $CoKf$, see section 5.3.2.5.

Now we show that this is a cokernel morphism of $kerf$.

Proposition 5.3.12. *The morphism inc' is a cokernel morphism of $kerf'$.*

Proof. First we need to show that $kerf' \circ inc' \equiv 0$, but by functional extensionality it is sufficient to prove $inc(kerf(a, p)) \equiv [0]$ for $(a, p) : Kerf$. Here $inc(kerf(a, p)) \equiv [a]$ and $p : f(a) \equiv 0$ so $f(a) \equiv 0 \equiv f(0)$ and thus $a \sim 0$. Since $A/\sim$ is a set this is sufficient.

Next we need to show that for any morphism $h' : A \Rightarrow E$ such that $kerf' \circ h' \equiv f'$ we have that there exists $u' : A/Kerf \Rightarrow E$ such that $inc' \circ u' \equiv h'$. We use the elimination rule to define $u([a]) := h(a)$. We need to show that $u([a]) \equiv u([b])$ if $a \sim b$ to see that it is well defined. By definition $u([a]) \equiv h(a)$ so it is sufficient to prove $h(a) \equiv h(b)$. We note that this holds if and only if $h(a) +_B (-_B h(b)) \equiv 0$ and thus $h(a +_A (-_A b)) \equiv 0$. Now since $h(kerf(x)) \equiv 0$ for all $x : Kerf$, by functional extensionality, we only need to prove some $p : f(a +_A (-_A b)) \equiv 0$ since then $(a +_A (-_A b), p) : Kerf$. Now we use that $a \sim b$ and get $f(a) \equiv f(b)$ and thus $f(a +_A (-_A b)) \equiv 0$, similarly to how we did for h'.

Thus u is well defined and that it is linear follows from the fact that h' is linear, similarly to how we did for $coKf$, see section 5.3.2.5.

Finally we need to show that inc' is epic. To this end we get $g\ h : A/Kerf \Rightarrow F$ and need to show $inc' \circ g \equiv inc' \circ h$ implies $g' \equiv h'$. It is sufficient to show that $g(x) \equiv h(x)$ for all $x : A/\sim$. By the elimination rule it is sufficient to show that $g([a]) \equiv h([a])$ for all $a : A$ and this follows from functional extensionality on the underlying functions of $inc' \circ g \equiv inc' \circ h$. Note that this is automatically well defined since $\langle F \rangle$ is a set and $g([a]) \equiv h([a])$ is thus a property. □

To show that there exists an isomorphism g' we will need the following proposition. Note that we here define that a function $f : A \to B$ is surjective if there is a function that takes $b : B$ to the propositional truncation of $\Sigma\ (a : A)\ (f(a) \equiv b)$.

Proposition 5.3.13. *A module homomorphism $f' : A \Rightarrow B$ is epic if and only if f is surjective.*

Proof. First we show that if f is surjective then f' is epic. We thus get $g'\ h' : B \Rightarrow D$ where $f' \circ g' \equiv f' \circ g'$ and need to show $g' \equiv h'$. By functional extensionality we need to show $g(b) \equiv h(b)$ for all $b : B$. But since $\langle B \rangle$ is a set this is a proposition. We can thus use the elimination rule for propositional truncation. This states that the specific element of the truncated type, which was originally used to prove the truncation, may be used in order to prove a proposition, see section 6.9 in [2]. Thus for our b we get an element $(a, p) : \Sigma\ (a : A)\ (f(a) \equiv b)$.

We now see that

$$g(b) \equiv g(f(a)) \equiv h(f(a)) \equiv h(b)$$

where $g(f(a)) \equiv h(f(a))$ is just functional extensionality on $f' \circ g' \equiv f' \circ h'$. Now we assume that f' is epic and will show that f is surjective. This follows the standard category theory proof where we use that our category has cokernels for every morphism.

First we note that $f' \circ coKf' \equiv 0 \equiv f' \circ 0$ by definition of $coKf'$ and properties of composition with the 0 morphism. Now since f' is epic we get that $g \equiv 0$ and thus for all $b : B$ we get $[b] \equiv [0]$. As we have seen earlier, the propositional truncation

of our relation $\sim$ for the cokernel is an equivalence relation and thus it is effective. This means that since we have $[b] \equiv [0]$ we get an element of $b \sim 0$, but under a propositional truncation.

By the elimination rule for propositional truncation we may remove the truncation when proving propositions, since being surjective is a proposition. Thus when we remove the truncations we get that for our $b : B$ we have an $a : A$ such that $b \equiv 0 + f(a)$. Thus since $b \equiv 0 + f(a) \equiv f(a)$ we have a proof of $\Sigma\ (a : A)(f(a) \equiv b)$ and we are done. □

Using this proposition we now prove the last step.

Proposition 5.3.14. *There exists an isomorphism $g' : A/Kerf \to B$ such that $inc' \circ g' \equiv f'$.*

Proof. We define g' by using the elimination rule on the underlying function $g([a]) := f(a)$. To prove that this is well defined we will need to show that if $a\ b : A$ and $a \sim b$ where $\sim$ is the relation of $A/Kerf$ then we have $f(a) \equiv f(b)$. But this is just the definition of $a \sim b$, and this step is done. The reader may verify that linearity of g follows from linearity of f.

Then we will show that for any $b : B$ there exists a proof of $\Sigma\ (x : \langle A/Kerf \rangle)\ (b \equiv g(x))$, which can be written as fiber $g\ b$. Now we first show that fiber $g\ b$ is a proposition. Since the second element of the Σ-type is an equality and $\langle B \rangle$ is a set we get that it is sufficient to show that the first element is a proposition. Thus given $(x, p)\ (y, q) :$ fiber $g\ b$ we need to show $x \equiv y$.

Here we use a lemma which claims that two equivalence classes are equal if their underlying elements are related. Since this is a known fact from algebra the proof will be omitted, but a curious reader may read the proof in ThesisWork/SetQuotientHelp.agda. Thus if we have $x \equiv [a]$ and $y \equiv [b]$ it is sufficient to prove $a \sim b$. Thus what remains to prove is that $f(a) \equiv f(b)$. Here we note that

$$f(a) \equiv g([a]) \equiv g(x) \equiv b \equiv g(y) \equiv g([b]) \equiv f(b)$$

since $g(x) \equiv b$ and $g(y) \equiv b$ by assumption and $g([a]) \equiv f(a)$ by definition.

Now that we have proven that fiber $g\ b$ is a proposition, we may use the elimination rule for truncation on the proof that f is surjective to get an element in $\Sigma\ (a : A)\ (f(a) \equiv b)$. Thus we want to prove fiber $g\ b$ from an element (a, p) where $a : A$ and $p : f(a) \equiv b$. We realize that $([a], p) :$ fiber $g\ b$ since $g([a]) \equiv f(a)$ by definition. Thus for any $b : B$ we can get an element $fib :$ fiber $g\ b$ and we define $g^{-1} :=$ fst (fib).

We now prove that $g(g^{-1}(b)) \equiv b$ for all $b : B$. If we apply the definitions of g^{-1} and g we get $f(a) \equiv b$ where $b \equiv f(a)$, and this is just the same equality, but the other way around, so we are done with this step. We also want to show

that $g^{-1}(g(x)) \equiv x$ for $x : A/\sim$. Again, we use that equivalence classes are equal if their underlying elements are related. Thus it is sufficient to show that if $[a] \equiv g^{-1}(g(x))$ and $x \equiv [c]$ then $a \sim c$, which means $f(a) \equiv f(c)$. Here we note that $f(a) \equiv g([a]) \equiv g(g^{-1}(g(x)))$ since $[a] \equiv g^{-1}(g(x))$. Now $g(g^{-1}(y)) \equiv y$ for all $y : B$, which means $g(g^{-1}(g(x))) \equiv g(x)$. Finally $g(x) \equiv g([c]) \equiv f(c)$ and we are done.

Since g and g^{-1} are inverses for each element in B and $A/\sim$ respectively we have that they are inverses as functions by functional extensionality. We know from abstract algebra that the inverse of a linear function is a linear function, and the interested reader may read the proof in ThesisWork/RModules/RModuleHomomorphismProperties.agda.

Thus this induces a module homomorphism $g^{-1\prime}$ which is an inverse to g, and thus g is an isomorphism and we are done. □

Here if we combine our lemma with our proposition we get what we wanted and have proven the last requirement for showing that RMod is abelian.

One may want to do a more direct proof that epics are cokernels, similarly to what we did for the proof that monics are kernels, but this is difficult. The difficulty comes from the fact that if we try to prove that f' is a cokernel morphism directly we will have trouble eliminating the propositional truncation for surjectivity. We would want a proof that for any type B and $P : B \to \text{Type}$ we have that $(b : B) \to ||P\ b||$ gives a proof of $||(b : B) \to Pb||$. This is however the definition for the axiom of choice in homotopy type theory, see section 3.8 in [2]. One may think of this as saying that if for every element of B there is a proof for $P\ b$ then there is a function which for every element of B chooses a proof of $P\ b$. These are in mathematics called choice functions, and since we want to give a constructive proof we will avoid using them.

Unfortunately Agda needs a lot of time to reduce the statement that the inverse of a linear function is linear for this case. Thus a more direct proof for linearity of g^{-1} can be found in ThesisWork/RModules/DirectProofKernels.agda.

5.3.3 RMod is additive

Technically we have proven all the necessary conditions for RMod to be abelian, since the additive structure can be inferred from the properties above. As mentioned in section 4.3 the proof that this is the case proved outside the scope of this thesis, thus we will here give a short proof that RMod is additive.

Theorem 5.3.9. *For a commutative ring R the category RMod is additive.*

Proof. First we note that for modules A and B the homset from A to B is an abelian

group under the following operators

$$f +_{\text{Hom A B}} g := \lambda x \to f(x) + g(x)$$
$$-_{\text{Hom A B}} f := \lambda x \to -_B f(x).$$

Here linearity of these operators follows from the linearity of f and g, and the proof that this is abelian follows from that B is abelian with respect to $+_B$. We also see that linearity of f and g gives that this forms a preadditive category.

Since we have already proven that RMod has a zero object it remains to show that RMod have binary direct products. Thus for any A and B we need to show there is an object AXB, module homomorphisms $p'_A : AXB \Rightarrow A$, $p'_B : AXB \Rightarrow B$, $i'_A : A \Rightarrow AXB$ and $i'_B : B \Rightarrow AXB$ such that $i'_A \circ p'_A \equiv \text{id } A$, $i'_B \circ p'_B \equiv \text{id } B$ and $(p'_A \circ i'_A) + (p'_B \circ i'_B) \equiv \text{id } AXB$.

Recall that we defined the product and the coproducts in RMod as the same object. We now choose AXB as the product of A and B and let p_A, p_B be the projections that made it a product, while i_A and i_B are the injections that makes it a coproduct. By this definition we see that $p_A(i_A(a)) \equiv \text{fst } (a, 0) \equiv a$ and thus by functional extensionality $i'_A \circ p'_A \equiv \text{id } A$. The same argument gives us that $i'_B \circ p'_B \equiv \text{id } B$. Similarly

$$i_A(p_A(a,b)) +_{AXB} i_B(p_B(a,b)) \equiv (a, 0_B) +_{AXB} (0_A, b) \equiv (a +_A 0_A, 0_B +_B b) \equiv (a, b)$$

and thus by functional extensionality $(p'_A \circ i'_A) + (p'_B \circ i'_B) \equiv \text{id } AXB$ and we are done.

□

6. Conclusion

6.1 Cubical agda version and disclaimer

All code provided in this thesis was developed for cubical agda version 2.6.2 from the development branch. During the work on the thesis the standard library has developed quickly. In an attempt to make the code more compatible with later versions, some of the parts that have undergone the most changes in the standard library have been moved to a script named "Compatibility code". This script also includes code snippets that were required for the proofs, but that was not in the original version. Each part in "Compatibility code" should state clearly which script it originally belonged to. I take no credit for the code in "Compatibility code" and all credits go to the original creators of the cubical agda development team [13]. ThesisWork/CompatibilityCode.agda

6.2 Reflections

From the proofs given in this thesis it should be clear that homotopy type theory is sufficiently powerful to represent abstract concepts of category theory. The reader might also notice that the proofs given are in many cases similar, if not identical, to how these concepts are proven in regular category theory and abstract algebra. The reader may also appreciate how naturally these definitions of category theoretical concepts behave in homotopy type theory, and how well they capture the concepts in question.

With respect to the implementation it should also be clear that cubical agda is well suited to formalize these concepts, and aside from the last proof that epic module homomorphisms are cokernels every script compiles quickly. This might be since the proof uses a lot of elimination rules on objects with many nestled definitions which results in very long expressions that need to be reduced. The compilation is still sufficiently fast, and the more direct proof reduces much faster.

6.3 Future work

With the current development of the standard library in cubical agda a more extensive library for category theory is currently being added, which will enable more advanced category theory to be proof checked in this environment in the future.

Using the proofs we have presented as a base line there is a lot of further work that

can be done.

6.3.1 Other abelian categories

There is a known theorem [14, Theorem 16.2.4] which states that if a category is categorically equivalent to an abelian category then it is also abelian. This can be used to show many interesting properties, for instance that the cocategory of an abelian category is abelian.

We could also define a type called $ModuleWithProperty$ which for a commutative ring R and $P : (M : ModuleR) \to$ Type is on the form $\Sigma\ (ModuleR)\ P$. With this we could define the type of finite modules, finitely generated modules, finite dimensional modules, projective modules, etc, where P in each case ensures this property. We could then define the category of modules, where for fix R and P the objects are of type $ModuleWithProperty\ R\ P$ and the morphisms are module homomorphisms on the underlying module. Then in order to show that this category is abelian it would be sufficient to prove that the objects we defined in the proof that RMod is abelian also satisfy the property P. This means that for $f : A \Rightarrow B$ it is sufficient to prove that the zero module, AXB, $Kerf$, $CoKf$, etc. all satisfy P given proofs of $P\ A$, $P\ B$.

In the folder ThesisWork/RProj I have given potential definitions for finite dimensional modules and projective modules. I have also translated the first half of the proof that these form abelian categories.

6.3.2 K-Theory

Another potential continuation of this work is to take the definition of projective modules and move on to define some of the basic concepts in K-theory. One might first want to finish the proof that the category RProj is abelian, which is started in ThesisWork/RProj/AbelianRProj.agda.

6.3.3 Vector spaces

It is also possible to define a field and use modules over these to define vector spaces and move on to the field of linear algebra.

6.3.4 Alternative definitions of abelian

One may also want to prove that all the different definitions we presented in 4.3 are in fact equivalent. Some of the proof can be found in [7, Chapter 8] and would not present too much challenge to implement. However, for the proof that the abelian structure of the homsets can be inferred from the other definitions one will need to prove a lot of properties and infer many standard limits from the definition. The groundwork for this has been done in ThesisWork/AbelianCategory/AbelianToAdditive.agda.

6.3.5 More properties of RMod

As discussed in section 5.2, for a general definition of a module we let the underlying set of the module be from a different universe than the commutative ring R. Thus the category of modules $R\text{Mod}(U_{\ell'})$ depends on a universe $U_{\ell'}$ which is the universe that the underlying sets of the module are from. An interesting statement to prove is that for two commutative rings R and S we have that $R\text{Mod}(U_{\ell'}) \simeq S\text{Mod}(U_{\ell'})$ if and only if $R \simeq S$.

We may also denote the subcategory of finitely presented modules in $R\text{Mod}(U_{\ell'})$ as $fpR\text{Mod}(U_{\ell'})$. Here it would be interesting to show that $fpR\text{Mod}(U_{\ell'})$ is independent of the universe $U_{\ell'}$. With this we mean to prove that for any $U_{\ell'}$ and $U_{\ell''}$ we have that $fpR\text{Mod}(U_{\ell'}) \simeq fpR\text{Mod}(U_{\ell''})$.

References

[1] G. Gonthier, "Formal proof—the fourcolor theorem," *Notices of the American Mathematical Society*, vol. 55, no. 11, pp. 1382–1393, 2008. [Online]. Available: http://www.ams.org/notices/200811/tx081101382p.pdf.

[2] T. Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[3] *Code developed during the thesis*. [Online]. Available: https://github.com/ByteBucket123/ThesisWorkGitCopy.

[4] B. C. Pierce, *Types and Programming Languages*. Massachusetts Institute of Technology Cambridge, Massachusetts 02142: The MIT Press, 2002, ISBN: 0262162091.

[5] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg, *Cubical type theory: A constructive interpretation of the univalence axiom*, 2016. arXiv: 1611.02108 [cs.LO].

[6] T. Coquand, S. Huber, and A. Mörtberg, *On higher inductive types in cubical type theory*, 2018. arXiv: 1802.01170 [cs.LO].

[7] S. M. Lane, *Categories for the working mathematician*. Springer-Verlag, 1978, ISBN: 9781475747218.

[8] P. Freyd, *Abelian Categories - An introduction to the theory of functors*. Harper & Row, 1964.

[9] A. Mörtberg, *The structure identity principle in cubical agda*, 2020-11. [Online]. Available: https://staff.math.su.se/anders.mortberg/slides/PalmgrenMemorial2020.pdf.

[10] *Introduction to univalent foundations of mathematics with agda*. [Online]. Available: https://www.cs.bham.ac.uk/~mhe/HoTT-UF-in-Agda-Lecture-Notes/HoTT-UF-Agda.html#sns.

[11] J. P. M. S. von Branitz, "Higher groups via displayed univalent reflexive graphs in cubical type theory," 2020-10.

[12] S. Pettersson, "Additive, abelian, and exact categories," MSc thesis, Uppsala University, 2016.

[13] *Cubical agda development team*. [Online]. Available: https://github.com/agda/cubical/graphs/contributors.

[14] H. Schubert, *Kategorien II*. Springer, 1970.

[15] R. M. F. David S Dummit, *Abstract Algebra*. Massachusetts Institute of Technology Cambridge, Massachusetts 02142: Wiley, 2003, ISBN: 9780471433347.

A. Modules And Algebra

A.1 Basic structures

For the sake of brevity we will assume the reader has some basic knowledge of abstract algebra. We will however go through some of the basic definitions as a reminder as we work our way towards a definition for the category RMod. For a more detailed look see "Abstract Algebra" by David S Dummit and Richard M Foote [15]. The first definition we will go over is a semigroup.

A.1.1 Semigroups

A semigroup is a type $G : Type$ and a binary operator $\_ + \_ : G \to G \to G$ that satisfies the following:

- isSet G
- $\_ + \_$ is associative, meaning there is a function $(abc : G) \to a + (b + c) \equiv (a + b) + c$

Note that we may denote the operation on G by $\_ +_G \_$ where such a distinction is important.

A.1.2 Monoids

The semigroup is only the most basic structure we will see, and a more commonly used structure is the monoid. A monoid has the same basic structure as a semigroup but adds the following

- There exists an identity element $e : G$ such that $(g : G) \to g + e \equiv g$ and $(g : G) \to e + g \equiv g$.

It is trivial to see that the identity element is unique since if there were two e, e' then $e \equiv e + e' \equiv e'$ by definition. Note that we may use e_G to denote the identity element of G when it is important to clarify which monoid the element belongs to. In many cases monoids are both simple and useful, having the right amount of constraints to make them easy to prove properties for.

A.1.3 Monoid homomorphisms

Before we move on we will have to know what a monoid homomorphism is. For two monoids M and N a monoid homomorphism between them is a function $f : M \to N$ that satisfy the following properties:

- The function f preserves the identity element, meaning $f(e_M) = e_N$.
- The function f preserves the group structure, meaning $(a\ b : M) \to f(a +_M b) \equiv f(a) +_N f(b)$.

These are then the morphisms in the category of monoids, called **MON**. We leave it to the reader to verify that compositions of monoid homomorphism are themselves monoid homomorphisms, and that these form a category under composition.

A.1.4 Groups

The most well known algebraic structure is the group. A Group G is a monoid that adds the following structure:

- For any element $g : G$ there exists an element in G, denoted $(-g)$, such that $g + (-g) \equiv e$ and $(-g) + g \equiv e$.

Note here that $(-g) : G$ is just an element, and that we have not defined an operator $-\_ : G \to G$. However it is clear that such an operator can be inferred as $-\_ = \lambda g \to (-g)$.

For group homomorphisms we would like them to preserve our inverses, meaning $f(-g) \equiv -f(g)$ but this follows from the fact that monoid homomorphism preserve the monoid structure, and some simple manipulations. The proof is left as an exercise to the reader.

A.1.5 Abelian Group

An abelian group is as the name suggests a group with one bit of added structure. We say that a group is abelian if

- The operator $\_ + \_$ is commutative, meaning that we have a function $(a\ b : G) \to a + b \equiv b + a$.

Here there is no reason to add extra constraints to the definition of group homomorphisms, since the added structure clearly is preserved by the original definition.

A.1.6 Rings

The last of our basic structures is the rings. Until now we have only concerned ourselves with one operator per structure, but for rings we have two. We say that a

type $R : \text{Type}$ with operators $\_ + \_ : R \to R \to R$ and $\_ * \_ : R \to R \to R$ is a ring if

- We have that R is an abelian group with respect to $\_ + \_$.
- We have that R is a monoid with respect to $\_ * \_$.
- The operator $\_*\_$ is distributive over $\_+\_$, meaning $(r\ a\ b : G) \to r*(a+b) \equiv (r * a) + (r * b)$.

Some authors replace the second condition in the definition of a ring with associativity of $\_*\_$. They say that our definition is "a ring with unity element", namely the identity element of our monoid. For rings the standard notation for the identity element for $\_+\_$ is 0. This is since many of its properties are similar to how 0 functions under addition and multiplication for real numbers. Some important properties to prove are $0 * a \equiv 0 \equiv a * 0$ for all $a : R$ and $(-a) * b \equiv -(a * b) \equiv a * (-b)$.

For ring homomorphism we will have to preserve the ring structure of $\_*\_$ as well as the group structure of $\_+\_$. Thus we add the constraint to monoid homomorphisms that $f(r) * f(a) \equiv f(r * a)$ for all $r\ a : R$.

The last structure we will look at is a commutative ring. This is simply a ring with the added condition that

- For any $a\ b : R$ we have that $a * b \equiv b * a$. Shortly, $\_ * \_$ commutes.

We could also define more structures such as integral domains and fields, but these will be left out since they are not relevant to our goal, which is to get to modules!

A.2 Modules and vector spaces

The reader with a background in physics or mathematical analysis will be familiar with the concept of a vector space. The most common example is $\mathbb{R}^n$ where $n : \mathbb{N}$. Here the elements are vectors of length n, addition is element wise and so is multiplication by a scalar.

A particularly useful generalization of a vector space is a module, which is defined as a type $M : \text{Type}$ along with a commutative ring R, operators $\_ +_M \_ : M \to M \to M$ and $\_ \cdot \_ : R \to M \to M$ which satisfy

- M with $\_ +_M \_$ is an abelian group.
- Multiplication with scalars is associative, which means $(r\ s : R) \to (a : M) \to (r * s) \cdot a \equiv (r \cdot (s \cdot a))$.
- Both $\_ +_M \_$ and $\_ \cdot \_$ distribute over each other, which means $(r : R) \to$

$(a\ b : M) \to r \cdot (a +_M b) \equiv (r \cdot a) +_M (r \cdot b)$ and $(r\ s : R) \to (a : M) \to (r +_R s) \cdot a \equiv (r \cdot a) +_M (s \cdot a)$.

- Scaling an element a of M with the identity of R, called 1_R preserves the element a, $1_R \cdot a \equiv a$.

Note here that $\_ +_M \_$ is a binary operator on M, while $\_ +_R \_$, is the operator on R. We will also refer to the identity element of the ring under $\_ *_R \_$ as 1_R and the identity element of the module as 0_M. The elements of R are also referred to as scalars. Here there are many useful properties that has to be proven, and many of them can be found in the code, see ThesisWork/RModules.

We should note that the definition of a vector space is very similar, it only adds that the commutative ring R must be a field, which adds a division operator that is the inverse of multiplication. There are also more general definitions, such as left modules, with the operator $\_ \cdot_{Left} \_ : R \to M \to M$ and right modules where $\_ \cdot_{Right} \_ : M \to R \to M$. For these we do not require R to be commutative, and thus get different structures depending on if the operator acts on the left or the right. We see that if $r \cdot_{Left} a \equiv a \cdot_{Right} r$ for all $r : R$ and $a : M$ then $(r * s) \cdot_{Left} a \equiv r \cdot_{Left} (s \cdot_{Left} a) \equiv (a \cdot_{Right} s) \cdot_{Right} r \equiv a \cdot_{Right} (s * r)$. If R is commutative then $r * s \equiv s * r$ and we can prove that the left and right modules are the same, but if R is not commutative this need not hold. Since the left and right modules are the same for commutative rings we will choose to represent modules as left modules.

A.2.1 Module homomorphisms

Module homomorphisms are similar to ring homomorphisms in that we want to preserve both the structure given by $\_ + \_$ and $\_ \cdot \_$. Thus we way that for two modules M and N, over the same commutative ring R, a function $f : M \to N$ is a module homomorphism if

- f preserves $\_ + \_$, which means $(a\ b : M) \to f(a +_M b) \equiv f(a) +_N f(b)$.
- f preserves $\_ \cdot \_$, meaning $(r : R) \to (a : M) \to f(r \cdot_M a) \equiv r \cdot_N f(a)$.

We sometimes say that a function is R-linear if it is a module homomorphism between modules over R. Note here that we did not demand that $f(0_M) \equiv 0_N$ since this can be inferred by a simple proof. This and many other properties are derived in ThesisWorkGitCopy/RModules/RModuleHomomorphismProperties.agda .

Since the proof is so simple we will show the proof idea here, but will skip some of the trivial details.

Proposition A.2.1. *For any homomorphism f between groups G and H we need not assume $f(0_G) \equiv 0_H$*

Proof. We want to show $f(0_G) \equiv 0_H$ from the basic group axioms and $f(a+b) \equiv f(a) + f(b)$. The calculations are simple and goes as follows:

$$\begin{aligned} f(0_G) &\equiv f(0_G) + (f(0_G) + (-f(0_G))) \equiv f(0_G + 0_G) + (-f(0_G)) \\ &\equiv f(0_G) + (-f(0_G)) \equiv 0_H \end{aligned}$$

where the steps in between, and which axiom is used where is left as an exercise to the reader. □

Now since a module is an abelian group over $\_ + \_$ this also holds for module homomorphisms. The properties in ThesisWorkGitCopy/RModules/RModuleHomomorphismProperties.agda include

- $(-0_M) \equiv 0_M$
- $r \cdot 0_M \equiv 0_M$ and $0_R \cdot a \equiv 0_M$
- For a and b in a module M we have $-(a+b) \equiv (-a) + (-b)$
- $-(r \cdot a) \equiv r \cdot (-a)$.
- for a module homomorphism f we have that $f(-a) \equiv -(f(a))$.

among others.

A.3 Categories of algebraic structures

We have seen that there is a category of all small monoids, namely **MON**. Many algebraic structures form categories in a similar manner, such as **GRP** for groups, **Ab** for abelian groups and **RNG** for rings. The category we will be interested in however is the category RMod of all small modules over R.

In RMod the objects are modules over R and the morphisms are module homomorphisms.

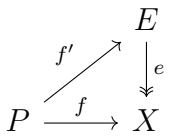
A.4 Modules with properties

A.4.1 Projective modules

Projective modules are important in applications since they have many well behaved properties. We say that a module P over a commutative ring R is protective if

- For any module homomorphism $f : P \Rightarrow X$ and surjective module homomorphism $e : E \Rightarrow X$ where is a module homomorphism $f' : P \Rightarrow E$ such that $f' \circ e \equiv f$.

The reader with a background in category theory may realize that if we replace module by object, homomorphism by morphism and surjective with epic then this is the definition of a projective object in a category. The diagram for a projective object P is the following



. We will show later that epimorphisms in RMod are surjective and we can then realize that the projective objects in RMod are the projective modules.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY