ENS - PSL

MATHEMATICS, LOGIC AND TYPES - M1 INTERNSHIP

Cohomology in Homotopy Type Theory

Louise LECLERC

Under the supervision of :
Thierry COQUAND

Hosted by Chalmers Logic and Types team.
7/02/2022 - 7/06/2022

1 Introduction

As part of my first year of master's degree in mathematics, I decided to explore a fairly recent field, between logic, category and theoretical Computer Science. To that end, I did an internship under the supervision of Thierry COQUAND, in the logic and types team at Chalmers University, in Sweden.

Before introducing the concepts involved, I would like to sincerely thank my supervisor for his help and patience, and the time he took to talk to me about complicated concepts such as Topos or Higher Category Theory. On many occasions, I was able to participate in moments of mathematical exchange, in conferences or in discussions with other members of the team such as Felix CHERUBINI or David WÄRN. These opportunities allowed me to see the organization of the team, and even to some extent to take part in the research work. What I would also like to underline is the complexity of the subject to be described in a short internship report. For my part, understanding the concepts necessary for my internship this year was the result of a journey of more than a year. Beginning with the discovery of proof assistants during a semantics course given at the ENS by Xavier RIVAL and Jérôme FERET. I then had the chance to do a one and a half month internship supervised by Sylvie BOLDO, at INRIA Paris-Saclay, where I used the COQ proof assistant to formalize the Bochner integral. And finally I studied Homotopy Type Theory (HoTT) in my free time for about 1 year.

From now on, (Homotopy) Type Theory has become an extremely practical synthetic tool for formulating questions of higher algebra and homotopy theory. It is a language of mathematical discussion, conducive to formalizing questions or statements similarly to Set Theory. However, two major differences stand out between these two languages. The first notable difference is the constructivist flavor of type theory, whereas the Mathematics as usually formalized in ZFC are classical and non-constructive. While one can obtain the existence of functions (typically of choice) in Set Theory by arguments that do not explain their construction or their computational behavior, in Type Theory functions can be used to calculate return values, in the same way as a computer program does. The second difference between the two theories is that the first allows to naturally formulate questions of higher category theory, where one can think of the objects described as being simplicial complexes, or topological spaces up to homotopy. This stems from the fact that types naturally carry a structure of *weak ω-groupoid* [1]. (For a historical introduction to this geometric understanding of type theory, see [5])

In algebraic topology, there is a family of spaces which make it possible to establish a link between cohomology and homotopy. These are spaces of the EILENBERG-MACLANE type. These are spaces defined up to homopy equivalence by their homology groups, which can be introduced into the formalism of HoTT. The first objective of my internship will have been the formalization of these spaces, with the help of the proof assistant AGDA precisely developed by the team of Mathematics and Theoretical Computer Science of Chalmers in which I was received. I was then led to work on the properties of cohomology. I had to study in the HoTT formalism the existence of results such as fibration sequences or the values of the first cohomology groups of spheres. I also spent some time working on the models of Homotopy Type Theory. It was a very interesting experience, also very complex. I tried to obtain applications of the fibration sequence in algebra, within the framework of the cohomology of groups. Unfortunately, I did not manage to complete this objective. Finally, I was led to compare different notions of cohomology. And I tried to show that two possible definitions of group cohomology coincide in type theory. Now I doubt the validity of this result to which I did not manage to obtain a complete answer, but it was also a subject of very enriching reflection, and usefull for my understanding of the involved concepts.

This immersive situation allowed me to familiarize myself with research in sciences. Its rhythm really different from that of studies. Sometimes involving weeks focused on the same issues, where it is necessary to change points of view and learn on the job. Sometimes by reading or watching conferences. I sincerely thank the ENS for allowing me to carry out this mathematically and humanly rewarding experience abroad.

2 Types and Logic

2.1 What is a type theory ?

Historically, the story of types theory begins in the 1900s. After the discovery of his paradox, RUSSELL, with the help of FREGE, formulate the first ideas of type theory. In order to avoid self reference in a formal logic system, the key idea is to hierarchise the objects, by giving a tag to each one: its type. In logic, this gives an alternative to set theory where instead of having every object being a set, every object is now a point a of some type A, A being uniquely determined. This is to be related (but not confused) with the mebership relation $\in$ of set theory: in a *first approach*, we may understand $a : A$ as $a \in A$. Since in such systems it is not possible to have $A : A$, this prevents Russell-type paradoxes from appearing. But since its advent, types theory have become a much wider field, used in Computer Science or as a logical framework suitable for constructive mathematics.

In computer science, the notion of *type* or *typing system* is closely related to functionnal programming and more generally soundness of programs. In a large class of language, as C++ or JAVA, there is a rudimentary form of typing. It consists in flags attributed to variables, as `int`, `float` or `char`, which allows to distinguish between datas that should represent an integer, a

decimal or a letter even if in the computer they all turns out to be represented as 0s and 1s. This class of language is called *weakly typed* but it is only the basic idea of type theory. There are also so-called *strongly typed* languages, and they usually corresponds to fonctionnal programming ones, such as OCaml or Haskell. In functional programming, everything is a function. You can define a function which takes in argument a function and returns a function, and even an entire program may be considered as a huge function. In those languages, when defining a function you need to specify the type of its argument and the type of its return value. If a function f eats elements of type A and spits out elements of type B, it will be given the type $A \to B$ and in this way, every object will come with a well defined type. The strong typing avoid problems of non-terminating functions. For exemple, it is not possible to define a function which maps a function f to $f(f)$ because there is no way to attribute a type to such a function.

The notion of *typing system* that interests us is of this last form. It consists of a family of rules to derive the type of a variable. This is really close (and actually closely related) to the natural deduction. In order to obtain a *typing judgment* such as $a : A$, you should build a whole tree where the rules are nodes and the judgment appears as the root of the tree.

2.2 The Intuitionistic Theory of Type

I will describe now the *Intuitionistic Theory of Type*, originally formulated by MARTIN-LÖF in 1972 [7]. In what follows, conversely to Set Theory, we will not rely on the first order logic. Indeed, as we will further explain the operations on types will also act as logical operators. First, there is a list of "basic types", that we will use as building blocks to build more complicated ones. They all go with **constructors** or **native elements**, which are *the only way* to introduce elements of this type.

- There is an **empty** type, which we will denote $\bot$ or $\mathbb{0}$. There is no native element or constructor of this type.
- There is a **unit** type, which we will denote $\top$ or $\mathbb{1}$. There is one native element of this type, which we denote $*$. Hence, we may write $* : \mathbb{1}$.
- There is a type of **booleans**, which we will denote Bool or $\mathbb{2}$. There is two native elements of this type, which will be called false and true or 0 and 1. Hence, we may write $0 : \mathbb{2}$ or $1 : \mathbb{2}$.
- There is a type of **natural numbers**, which we will denote $\mathbb{N}$. There is one native element 0, so we have $0 : \mathbb{N}$. And there is one constructor succ. Given an element $n : \mathbb{N}$, we also have $\mathsf{succ}(n) : \mathbb{N}$. This is to be related with PEANO's formulation of Arithmetic. For exemple, we have $\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(0))) : \mathbb{N}$ and we should also write the term $\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(0)))$ as "3".

Now we will also give types operations. There are really similar to the operation that we may form with sets, such as the cartesian product or the disjoint union.

- When both A and B are types, we may form their **arrow type** $A \to B$. It is the type of maps from A to B. Their is one constructor for this type, which will be denoted λ. Given an expression $\Phi(a) : B$ which may depend on the free variable $a : A$, we may form the element $\lambda(a : A).\Phi(a) : A \to B$. It is akin to the standard notation $a \mapsto \Phi(a)$, which mathematicians are more familiar with. As a first exemple, we may define the function $\mathsf{plus\_two} : \mathbb{N} \to \mathbb{N}$, which adds two to a given natural number, as follows : $\lambda(n : \mathbb{N}).\mathsf{succ}(\mathsf{succ}(n))$. When $f : A \to B$, and $a' : A$, we may always form the expression $f(a')$, and we have $f(a') : B$. When f is $\lambda(a : A).\Phi(a)$, then this expression *reduces* to $\Phi(a')$. In our previous exemple, this means that our function $\mathsf{plus\_two}$ *computes* as expected. For instance: $\mathsf{plus\_two}(1)$ is the same thing as $\mathsf{succ}(\mathsf{succ}(1))$, which is, by definition, 3. We have a native element $\mathbb{N} \to \mathbb{N}$, which is the constructor succ. More generally, constructors (exepted λ) may be seen as native elements of arrows types.
- When A and B are a type, we may form their **cartesian product** type $A \times B$. The only constructor for this type is the *pairing operation* $\mathsf{pair} : A \to B \to A \times B$. This means that the only way to build an element in $A \times B$ is to give its first and second component. Hence, if $a : A$ and $b : B$, then $\mathsf{pair}(a)(b) : A \times B$. We will also more suggestively denote this element by (a, b). For instance, $(\mathsf{succ}(\mathsf{succ}(2)), *) : \mathbb{N} \times \mathbb{1}$, and $(\mathsf{plus\_two}, \mathsf{false}) : (\mathbb{N} \to \mathbb{N}) \times \mathsf{Bool}$.
- When A and B are a type, we may form their **sum** type $A + B$. This is to be related with the usual disjoint union. Their are two constrctors for this type, which are $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$. This means that to produce an element $c : A + B$, we may use an element from $a : A$ and define c as $\mathsf{inl}(a)$ or an element $b : B$ and define c as $\mathsf{inr}(b)$. For instance $\mathsf{inl}(*) : \mathbb{1} + \mathbb{N}$.

At this point, I need to introduce the concept of **Universes**. It is akin to the concept of universes in Set Theory, as introduced by TARSKI or GROTHENDIECK. It seems natural to ask whether there is a "type of types". However, stated like this, it will

eventually leads to RUSSELL or BURALI-FORTI style paradoxes. Hence, here again we need to hierarchise. We will not assume *one* universe of "all types", but a whole "cumulative" hierarchy of such.

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \dots$$

And we assume $\mathbb{0}$, $\mathbb{1}$, $\mathbb{2}$, $\mathbb{N}$ to be of type $\mathcal{U}_0$. Moreover, we ask for each universe to be closed under product, sum, and exponentiation (arrow types). Hence our previous rules stated as "When A and B are types, then so is $A \times B$" should be read as if $A, B : \mathcal{U}_n$, then $A \times B : \mathcal{U}_n$. We may also now consider $+$, $\times$, $\to$ to be maps $\mathcal{U}_n \to \mathcal{U}_n \to \mathcal{U}_n$. Since we think of $\mathcal{U}_{n+1}$ as beeing bigger than $\mathcal{U}_n$, then we will suppose a cumulativity. There is two ways to handle this. The first one is to allow elements to have several types in this specific case. Precisely, if $A : \mathcal{U}_n$, then $A : \mathcal{U}_{n+1}$. But this removes the desirable feature that everything gets one and only one type. A "cleaner" way of doing this is to postulate the existence of *lifting maps*, $\uparrow : \mathcal{U}_n \to \mathcal{U}_{n+1}$. preserving all the properties that we need in order that $\uparrow A$ behaves exactly as A does. But we will not worry so much about this subtlety, since it may becomes a bit technical sometimes. However, note that when working with a proof assistant as I did in my internship, you need to care about those universes !

Now, I will describe how to define functions out of some type. In type theory, every function is made from small bricks, namely the constructors and the recursors. Roughly, a constructor explains how to map *into* certain type, and the recursor allows to map *out* of a certain type. So for the 7 points introduced above, I explain now what are the recursors.

- For the empty type, since there is no way to introduce an element, you can define a map out of $\mathbb{0}$ without having to specify anything. This is really similar to what happens in set theory with the empty set. Hence the recursor say the following : For each universe $\mathcal{U}$ and type $A : \mathcal{U}$, there is a map $\mathsf{rec}_{\mathbb{0}}(A) : \mathbb{0} \to A$.
- For the unit type, since there is only one native element $* : \mathbb{1}$, specifying a map out of unit is the same as picking a value for $* : \mathbb{1}$. Hence, the recursor say the following. If A is a type and $a : A$, then there is a map $\mathsf{rec}_{\mathbb{1}}(A, a) : \mathbb{1} \to A$. Moreover, we have the *reduction* (also called *computation*) rule $\mathsf{rec}_{\mathbb{1}}(A, a)(*) \equiv a$.
- For the type of booleans, there is two native elements, hence we need to specify the value of both. If A is a type, and a_0, $a_1 : A$, then $\mathsf{rec}_{\mathbb{2}}(A, a_0, a_1) : \mathbb{2} \to A$. And we have the computation rules $\mathsf{rec}_{\mathbb{2}}(A, a_0, a_1)(0) \equiv a_0$ and $\mathsf{rec}_{\mathbb{2}}(A, a_0, a_1)(1) \equiv a_1$.
- For the type of natural numbers, there is one native element and one constructor $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$. Hence, in order to define a map $\mathbb{N} \to A$ for a certain type A, we should specify the image of 0, and how succ acts through the map we are defining. The recursor for $\mathbb{N}$ works as follows : Given a type A, an element $a : A$ and a map $\phi : \mathbb{N} \to A \to A$, we have a map $\mathsf{rec}_{\mathbb{N}}(A, a, \phi) : \mathbb{N} \to A$, which compute as follows:
  - $\mathsf{rec}_{\mathbb{N}}(A, a, \phi)(0) \equiv a$
  - $\mathsf{rec}_{\mathbb{N}}(A, a, \phi)(\mathsf{succ}(n)) \equiv \phi(n, \mathsf{rec}_{\mathbb{N}}(A, a, \phi)(n))$

  This is really as defining by induction a function when programming. Indeed, recursors allows us to "pattern match" as in some languages (OCAML, HASKELL, RUST...).
- For the arrow type there is not really a recursor. The way you use function is by evaluating them. But to my knowledge there is no recursor for this type.
- In order to define a map $A \times B \to C$ out of a cartesian product, what you need is a map $f : A \to B \to C$. Since the only way to introduce an element $c : A \times B$ is by pairing together two elements $a : A$ and $b : B$, then f should suffice to define the map $A \times B \to C$. If C is a type and $f : A \to B \to C$, then we have a map $\mathsf{rec}_{\times, A, B}(C, f) : A \times B \to C$ with the computation rule $\mathsf{rec}_{\times, A, B}(C, f)(pair(a)(b)) \equiv f(a)(b)$.
- if A and B are types, the elements of $A + B$ are introduced by $\mathsf{inl} : A \to A + B$ or $\mathsf{inr} : B \to A + B$. Hence when C is a type, in order to define a map $A + B \to C$ it suffice to deal with those two cases. If $f : A \to C$ and $g : B \to C$, then we have $\mathsf{rec}_{+, A, B}(C, f, g) : A + B \to C$ with the computation rules
  - $\mathsf{rec}_{+, A, B}(C, f, g)(\mathsf{inl}(a)) \equiv f(a)$
  - $\mathsf{rec}_{+, A, B}(C, f, g)(\mathsf{inr}(b)) \equiv g(b)$

Now, notice that the intriduction of universes allows us to define **types families**. That is maps $A \to \mathcal{U}$ for A a type and $\mathcal{U}$ a certain universe. For exemple, we may define by recursion on $\mathbb{N}$ the finite types as

$$f(0) :\equiv \mathbb{0} \qquad \text{and} \qquad f(\mathsf{succ}(n)) :\equiv \mathbb{1} + f(n)$$

That is :

$$f :\equiv \mathsf{rec}_{\mathbb{N}}(\mathcal{U}_0,\, \mathbb{0},\, \lambda(n:\mathbb{N}).\lambda(A:\mathcal{U}_0).(\mathbb{1}+A))$$

There are 3 more type operations that we may introduce now : the **dependant sum** Σ, the **dependant product** Π and the **identity type** Id. I describe the first two of them, which are respectively generalizations of the cartesian product and the arrow type. And we will come back to the latter in the section "Types and Geometry".

- Given a type A and a type family $B : A \to \mathcal{U}$ (we say that B is indexed by A or a type family *over* A), we may define their **dependant product**: $\prod_{(a:A)} B(a)$. It is the type of **dependant functions** f such that, for every $a : A$, $f(a) : B(a)$. The adjective "dependant" here comes from the fact that the type of $f(a)$ *depends* on the element $a : A$. It is a key feature of our type theory, and the reason why this theory is called a **dependant type theory**. This is a generalization of the type of functions, hence we will stick to the notation λ for the constructor. If $\Phi(a)$ is an expression potentially depending on the free variable $a : A$, such that, for every $a : A$, $\Phi(a) : B(a)$ then we may form the term $\lambda(a:A).\Phi(a) : \prod_{(a:A)} B(a)$. We have the expected computation rule : $(\lambda(a:A).\Phi(a))\,(a') \equiv \Phi(a')$. Notice that, for any type C (which does not depends on $a : A$), $A \to C$ is the same type as $\prod_{(a:A)} C$.
- Given a type A and a type family $B : A \to \mathcal{U}$, we may define their **dependant sum**: $\sum_{(a:A)} B(a)$. It is the type of **dependant pairs** (a, b) with $a : A$ and $b : B(a)$. Here again, the element b lives in a type which may depends on the first argument. To construct an element of this type, we need an element $a : A$ and an other one $b : B(a)$, then we may form the element $\mathsf{pair}(a)(b) : \sum_{(a:A)} B(a)$. Hence pair has the type $\prod_{(a:A)} \left(B(a) \to \sum_{(a':A)} B(a')\right)$. Notice that here again, the dependant sum and the constructor pair generalize the product in the case B constant. As with the product, we will write more suggestively (a, b) for the element $\mathsf{pair}(a)(b)$. The recursor for this type has the following form: Given a type C,

  $$\mathsf{rec}_{\Pi,A,B}(C) : \prod_{a:A} (B(a) \to C) \to \left(\sum_{a:A} B(a)\right) \to C$$

  . Which means that given, for each $a : A$, a map $\phi(a) : B(a) \to C$, we have a map $\mathsf{rec}_{\Pi,A,B}(C, \phi) : \left(\sum_{(a:A)} B(a)\right) \to C$. It satisfies the computation rule, for $a : A$ and $b : B(a)$ $\mathsf{rec}_{\Pi,A,B}(C, \phi)((a, b)) \equiv \phi(a)(b)$. This is a generalized form of *curryfication*.

Before going further, there is some conventions that I will describe. Firstly, the arrow are right associative, which means that $A \to B \to C \to D$ should be parsed as $A \to (B \to (C \to D))$. Secondly, every expression located after a binder Σ and Π is by default considered as being inside its scope. Which means that $\prod_{(a:A)} B(a) \to C$ must be parsed as $\prod_{(a:A)} (B(a) \to C)$, and not $\left(\prod_{(a:A)} B(a)\right) \to C$. Finally, we implicitely currify functions : We we work with $f : A \to B \to C$, we write often $f(a, b)$ instead of $f(a)(b)$. Even if it do not work out the details (the interested reader can find the proof in [8]), there is as expected an isomorphism between $A \times B \to C$ and $A \to B \to C$, or more generally between $\left(\sum_{(a:A)} B(a)\right) \to C$ and $\prod_{(a:A)} B(a) \to C$.

2.3 The CURRY-HOWARD correpondance

Instead of giving a precise theorem or isomorphism, I will try to give the flavour of what is the CURRY-HOWARD correpondance. This is about types and logic. More precisely, there is a correspondance between writing a proof of a given *logical statement*, and defining a type theoretic expression of a certain *type*. In this philosophy, elements of a certain type will correspond to proofs of the associated logical statement.

Let A and B be two type. And think of it as being logical propositions. We also think of any element $a : A$ as a certain proof of A. Then what does it mean to proove the logical implication $A \to B$? It means than we have a way to turn an evidence of A into an evidence of B. Hence, whenever A, then we should have B to. This is exactly what we are doing when writting down a proof of $A \to B$: We suppose A, then we try to produce an evidence of B. In the same fashion, building an evidence of the proposition $A \wedge B$ will be the same as giving an element of the type $A \times B$. When *constructively* proving that $A \vee B$ holds, we give either a proof $a : A$ that A holds, or a proof $b : B$ that B holds, hence $A \vee B$ should corresponds to $A + B$. Actually, $A + B$ is not exactly the type corresponding to $A + B$ but we will not dive in those details for now. You may also interprete the truth value as $\top$ and the false value as $\bot$. Hence, the negation of the proposition associated with the type A will be associated to the type $\neg A :\equiv (A \to \bot)$. Notice that the constructors and recursors gives us the logical laws of introduction and elimination. For exemple, the explosion principles is given by the recursor of $\bot$, and the *modus ponens* is given by the evaluation map $f \mapsto a \mapsto f(a) : (A \to B) \to A \to B$.

What about the first order logic then ? Suppose that A is a type and $P(a)$ is another type, depending on $a : A$. We think of $P(a)$ as proposition depending on $a : A$. What does it mean to prove $\forall(a : A)P(a)$? In order to derive such a statement,

we assume $a : A$ is an arbitrary element, and we try to find a proof $\pi(a)$ that $P(a)$ holds. Hence, it is the same as defining a dependant map π of type $\prod_{(a:A)} P(a)$.
For the existential, it works in a similar way with the dependant sum : when *constructively* proving $\exists(a : A)P(a)$, we pick a certain $a : A$ and we show that $P(a)$ holds. Hence, it correponds to giving an element $(a, p) : \sum_{(a:A)} P(a)$. (Here we have the same issue as before, but lets keep to this interpretation for now.)

And this is how the CURRY-HOWARD correpondance works in practice. Indeed, this is the conceptual base of proofs assistants such as COQ, LEAN or AGDA. When working with a proof assistant, you translate and formalize the logic as types, translate theorems as complicated types, and try to define elements of those types. Here is an exemple of proof that the double negation of the excluded middle holds: Given a type A, the excluded middle for A is $A + \neg A$, its double negation is $\neg\neg(A + \neg A)$ Hence to give an evidence of this, we need to exhibite a map of type $((A + \neg A) \to \bot) \to \bot$. So lets pick a map $\phi : (A + \neg A) \to \bot$. Then, by composing with inl, we have a map $\phi \circ \mathsf{inl} : A \to \bot$. Hence an element $\phi \circ \mathsf{inl} : \neg A$. We may now apply our function ϕ to the argument $\mathsf{inr}(\phi \circ \mathsf{inl})$ to get an element $\phi(\mathsf{inr}(\phi \circ \mathsf{inl})) : \bot$. Finally, this means that

$$\lambda(\phi : (A + \neg A) \to \bot).\, \phi(\mathsf{inr}(\phi \circ \mathsf{inl})) \ : \ \neg\neg(A + \neg A)$$

And this is our proof that the double negation of the excluded middle holds !

3 Types and Geometry

3.1 Paths, paths between paths and the higher homotopy interpretation

3.1.1 The identity type

Pursuing in the interpretation of *types as propositions*, the logical statement "$x = y$" for x and y two terms of a given type A may also be internalised as a type. This idea come from the intuitionist type theory, as first formulated by Per MARTIN-LÖF in [7]. We should stick to this idea by defining an identity type over any type A, or more precisely a family of identy types indexed by the elements of A as follow:

$$\mathsf{Id}_A : A \to A \to \mathcal{U}$$
$$\text{for all } a : A,\ \mathsf{refl}_a : \mathsf{Id}_A(a, a)$$

So for any elements a, b of type A, we have a type of equalities in A, namely $\mathsf{Id}_A(a, b)$, and we only know one way to construct an element for the identity type: when $a \equiv b$, with the constructor $\mathsf{refl}_a : \mathsf{Id}_A(a, a)$. And this should capture the notion of equality.

for readability, we will now write the equality type with the usual symbol "=", so $a = b$ will be a notation for the type $\mathrm{Id}\, a\, b$, and we may also write "$a =_A b$" to specify the type A.

Now, as with every other type construction in type theory, one should define a recursor for this type, in order to know how to use an equality to produce other statements. We may give two recursors, and refer the reader to the HoTT reference for the proof that they are equivalent.

Path induction:

$$\mathsf{ind}_{=_A} : \prod_{(C:\prod_{(x,y:A)}(x=_A y)\to\mathcal{U})} \left(\prod_{(x:A)} C(x, x, \mathsf{refl}_x)\right) \to \prod_{(x,y:A)} \prod_{(p:x=_A y)} C(x, y, p)$$

with the computation rule, for every $a : A$, $\mathsf{ind}_{=_A}(C, f, a, a, \mathsf{refl}_a) :\equiv f(a, \mathsf{refl}_a)$

Based path induction:

$$\mathsf{ind}'_{=_A} : \prod_{(a:A)} \prod_{(C:\prod_{(x:A)}(a=_A x)\to\mathcal{U})} C(a, \mathsf{refl}_a) \to \prod_{(x:A)} \prod_{(p:a=_A x)} C(x, p)$$

with the computation rule, for every $a : A$, $\mathsf{ind}'_{=_A}(a, C, d, a, \mathsf{refl}_a) :\equiv d$

What is worth noticing from those recursor is that they tells us the equality behave as the LEIBNIZ equality : if $a =_A b$ then every property which holds for a will hold for b and inversely. But has every other types, the identity type could carry more structure that the mere propositionnal one, and should be thought as an algebraic structure. Indeed that is the fundamental consideration of Homotopy Type Theory, which give it its higher algebraic flavor as we will describe it later.

One can recover the usual properties of equality and the fact that it is an equivalence relation. This may be stated as the existence of the following functions (Those functions may be defined by path induction, and the reader should refer to [8] for this construction and the proofs of the following statement):

Definition 3.1 : concatenation, inversion

Given a type A and $x, y, z : A$, one may form the **concatenation** of any two elements $p : x =_A y$ and $q : y =_A z$. This concatenation is denoted $p \cdot q$ and has type $x =_A z$
Given a type A and $x, y : A$, one may form the **inverse** any $p : x =_A y$. This inverse is denoted p^{-1} and has type $y =_A x$

Moreover, given a type A, those operations respects the following properties :

unit for $\cdot$:
For every $x_0, x_1 : A$ and for every $p : x_0 = x_1$,

$$\mathsf{refl}_{x_0} \cdot p = p \quad \text{and} \quad p \cdot \mathsf{refl}_{x_1} = p$$

associativity of $\cdot$:
For every $x_0, x_1, x_2, x_3 : A$ and for every $p : x_0 = x_1, q : x_1 = x_2, r : x_2 = x_3$,

$$p \cdot (q \cdot r) = (p \cdot q) \cdot r$$

inversion :
For every $x_0, x_1 : A$ and for every $p : x_0 = x_1$,

$$p \cdot p^{-1} = \mathsf{refl}_{x_0} \quad \text{and} \quad p^{-1} \cdot p = \mathsf{refl}_{x_1}$$

And these properties may be resumed as the following :

Theorem 3.1

the families of identity types $(x =_A y)_{x\,y:A}$ is equiped with a structure of groupoid.

3.1.2 The homotopical point of view

As previously mentionned, one should consider that two elements of a given type may be equal in differents ways. Thinking as equality beeing paths, and types as spaces, we may imagine that two points could be linked along distincts paths. In fact, this interpretation is really accurate in our context, and provides us a convenient way to think about identity types. For example, the groupoid structure explicited in the previous section may be interpreted as the path-groupoid structure of a space. Following this interpretation, the concatenation of identity proofs now become the concatenation of paths, and the inversion become the reversal.

We may take this *space-as-types* interpretation further. Assume we have a type A, two elements $a, b : A$ and two paths $p, q : a =_A b$ between them. Then in the geometric setting, we may have a whole diversity of homotopies between p and q. This may be considered to in type theory, and it will go by the name

$$p =_{a =_A b} q$$

In fact, we already mentionned the existence of elements in this kind of type in the previous section, by claiming the unitarity, associativity and inversibility properties for identity proofs operations. And as in homotopy theory, there is no reason to stop at level two, and one would even consider types of the form $isprop(P)$

$$\alpha =_{p =_{a =_A b} q} \beta$$

Which we should write more readibly as $\alpha = \beta$. As before, inhabitants of this kind of type could be properties about the proof of associativity of types, or the proof of inversibility... The point is that more than beeing a convenient way to think about identity type, this perspective guide us towards defining objects akin to classic homotopy theory ones. And beautiful constructions of usual geometry may be reproduced in Homotopy Type Theory, such as loopspaces, n-spheres, homotopy groups, suspensions, universal covers, Hopf fibrations, and so on. One may also prove type theoretic formulations of standard theorems such as the WHITEHEAD principle, or the long exact sequence induced from a fibration. This last point will be at the center of the following developement.

3.2 Types dimension and connectedness

3.2.1 dimensionality

As sketched in the previous sections, the type of equality in a given type may host a rich diversity of elements. In order to measure, given a type A, how complex this higher structure is, we may define a notion of "dimension" for types as the maximal depth of a non-trivial identity type in A. That is the maximal height of an identity type tower (as in the previous section) one may form and still having a non trivial type. The first notion to define here is the meaning of "trivial type" used above, and it will be the starting point for the hierarchy.

Definition 3.2 : Contractible type

A type A is **contractible**, or a **singleton**, if there is $a : A$, called the **center of contraction**, such that $a = x$ for all $x : A$.

$$\mathsf{isContr}(A) :\equiv \sum_{(a:A)} \prod_{(x:A)} (a = x).$$

Then the hierarchy is defined by recursion :

Definition 3.3 : Level of a type

Define the predicate $\mathsf{is}\text{-}n\text{-}\mathsf{type} : \mathcal{U} \to \mathcal{U}$ for $n \geq -2$ by recursion as follows:

$$\mathsf{is}\text{-}n\text{-}\mathsf{type}(X) :\equiv \begin{cases} \mathsf{isContr}(X) & \text{if } n = -2, \\ \prod_{(x,y:X)} \mathsf{is}\text{-}n'\text{-}\mathsf{type}(x =_X y) & \text{if } n = n' + 1. \end{cases}$$

We say that X is an **n-type**, or that it is *n-truncated*, if $\mathsf{is}\text{-}n\text{-}\mathsf{type}(X)$ is inhabited.

Notice that the "trivial type" case correspond to $n = -2$. The firsts levels got specific names; given a type A, for

- $\mathsf{is}\text{-}(-1)\text{-}\mathsf{type}(A)$, one say $\mathsf{isProp}(A)$, and that "A is a **(mere) proposition**"
- $\mathsf{is}\text{-}0\text{-}\mathsf{type}(A)$, one say $\mathsf{isSet}(A)$, and that "A is a **set**"
- $\mathsf{is}\text{-}1\text{-}\mathsf{type}(A)$, one say that "$A$ is a **groupoid**"

When doing mathematics in a usual context, one do not matter about the algebraic structure of a proposition, and in how many ways one can prove it. In the *type-as-proposition* interpreatation, this correspond to assume that given a type P, thinking of it as being a proposition, every two elements $p, q : P$ (*i.e.* two proofs of P) are equals. one can show that this assertion is actually equivalent to $\mathsf{isProp}(P)$. So the types P such that $\mathsf{isProp}(P)$ behave as usual propositions, and this explains the terminology.

Now, given a type A such that $\mathit{isset}(A)$, we know that for every two elements a, $b : A$, the identity type $a =_A b$ is a mere proposition. And we recover in this way types whose identity types behave as in usual set theory: there is at most one way to be equals as elements of a set. This is the reason why the level hierarchy begins at -2; because that way one fix the set-like behaved types at level 0.

Finally, when unfolding the definition, on get that given two points x, y of a given groupoid G, the identity type $x =_G y$ is a set. And being equals as paths between x and y in now a proposition, so the structure G behave as a usual groupoid, in which every $x : G$ represent an object of G, and every path $p : x =_G y$ between two elements x, $y : G$ represent an isomorphism $x \to y$ in the groupoid. The structure of groupoid is given by the one given in section 3.1.1.

For a deeper explanation of this concept, the reader should refer to chapters 3 and 7 of [8].

For any type A, one may think of "a best approximation of A by a mere proposition $\|A\|$". This type exists and is called the **propositional truncation of** A. This type is caracterised by the following attributes.

- There is a function $|\cdot| : A \to \|A\|$.
- $\|A\|$ is a mere proposition
- For each mere proposition B and map $f : A \to B$, f factors through $|\cdot|$ and yields a map $\|A\| \to B$.

3.2.2 connectedness

Another (in some sens "dual") way to study the higher homotopy structure of a type A, is to look at "how much" A is connected. This consideration takes its motivation straightly in the *types-as-spaces* interpretation.

Given a space X, one can ask whether X is connected or not, and this is what will correspond for a type to be 0-*connected*. But in homotopy theory, one is also interested in the property of being "simply-connected", which in fact correspond to the property of being connected, and having a connected path-space. this will correspond to our notion of 1-*connected* types. Once again, there is no reason to stop the definition at $n = 1$, and one may consider also the property of having k-fold loopspaces connected for every $k \leq n$. This is what correspond to the definition *n-connectedness* in homotopy theory, and what is reproducible in type theory. This leads naturally to a recursive definition, as follows :

Definition 3.4 : Connectedness

Define the predicate $\mathsf{is}\text{-}n\text{-}\mathsf{connected} : \mathcal{U} \to \mathcal{U}$ for $n \geq -1$ by recursion as follows:

$$\mathsf{is}\text{-}n\text{-}\mathsf{connected}(X) :\equiv \begin{cases} \|X\| & \text{if } n = -1, \\ \|X\| \times \prod_{(x,y:X)} \mathsf{is}\text{-}n'\text{-}\mathsf{connected}(x =_X y) & \text{if } n = n' + 1. \end{cases}$$

We say that X is an n**-connected** type if $\mathsf{is}\text{-}n\text{-}\mathsf{connected}(X)$ is inhabited.

Actually, in [8], the definition of connectedness differ from the presentation given here. It makes use of the n-th truncation of a type, while this definition only need the propositionnal truncation. The definition was suggested by Thierry COQUAND, and one of the objectives of my internship was to redemonstrate known results on connectivity from this alternative definition; more elementary and more conducive to inductive reasoning.

I started by proving, and formalising in Agda, the following lemmas about connectivity:

Lemma 3.1

Given A, B two types, and $n \geq -1$ such that $\mathsf{is}\text{-}n\text{-}\mathsf{connected}(A)$ and $\mathsf{is}\text{-}n\text{-}\mathsf{type}(B)$, the map

$$\lambda(a : A).\ (\lambda(\_ : B).\ a) : B \to (A \to B)$$

is an equivalence.

Lemma 3.2

Given $n \geq -1$, A a n-connected type, and $P : A \to \mathcal{U}$ a family of $(n-1)$-types over A, the projection map

$$\left(\prod_{x:A} P\,x\right) \to P\,a$$

is an equivalence.

Proof. This follows from the first lemma by noticing that the projection map is the composition of $\left(\prod_{(x:A)} P\,x\right) \to \left(\prod_{(x:A)} a = x \to P\,x\right)$, which is an equivalence by the first lemma, and the map $\left(\prod_{(x:A)} a = x \to P\,x\right) \to P\,a$ which is always an equivalence by path-induction. □

Lemma 3.3

Given $n \geq -1$, $k \geq -2$, A a n-connected type with a point $a : A$, and $P : A \to \mathcal{U}$ a family of $(n+k+1)$-types over A and $b : P\,a$, the type of pointed sections $\sum_{(s : \prod_{(x:A)} P\,a)} (s\,a = b)$ is a k-type.

4 Homotopy and cohomology

4.1 Looping, delooping

4.1.1 the loopspace functor

As suggested above, one should consider when working in the context of HoTT, identities type as a path space. We then define, for a given type A, and $a : A$, the type $\Omega(A, a) :\equiv a =_A a$, and give it the name **loopspace at** a **in** A. As it was seen in the section *the identity type*, we get a natural structure of group on $\Omega(A, a)$, and a distinguished element $\mathsf{refl}_a : \Omega(A, a)$, which we will also write 1_a while thinking of it as the unit for the group structure on the loopspace. This makes the loopspace a pointed type: $(\Omega(A, a), 1_a)$. This way one may iterate the loopspace, and more generally define iterated loopspaces in the following way:

Definition 4.1 : Loopspaces

the type family $\Omega^n(A, a)$ for $n : \mathbb{N}$ is inductively defined by:

$$\Omega^n(A, a) :\equiv \begin{cases} (A, a) & \text{if } n = 0, \\ \Omega(\Omega^{n'}(A, a), 1) & \text{if } n = n' + 1. \end{cases}$$

For any $n : \mathbb{N}$, $\Omega^n(A, a)$ is then a pointed type, but we will identify it with its underlying space when needed.

The construction Ω gives rise to an important family of groups in Homotopy Type Theory. Moreover, any pointed map give rise to a (pointed) map between the associated loopspaces:

Definition 4.2 : Looping of a function

Given any two pointed types (A, a), (B, b) and a pointed function $f_\bullet = (f, f_0) : (A, a) \to_\bullet (B, b)$

$$\Omega f_\bullet :\equiv \lambda\alpha.\, (f_0^{-1} \cdot f_* \alpha \cdot f_0) : \Omega(A, a) \to \Omega(B, b)$$

(One can also make this function pointed; this follows from the identity $f_0^{-1} \cdot f_0 = 1_b$)

And we call $\Omega f_\bullet$ the **looping of** $f_\bullet$ As for the loopspace case, we should overload the same notation for the looping of $f_\bullet$ and the pointed version of the looping of $f_\bullet$ when it is clear from the context. One can check that $\Omega f_\bullet$ preserves the composition, so it is even a *group morphism* between the loopspaces. By denoting $\mathcal{U}_\bullet$ the type $\sum_{(X:\mathcal{U})} X$ of pointed types in the universe $\mathcal{U}$, we have the following result:

Theorem 4.1 : Functoriality of Ω

Ω defines a *endofunctor* in the category $\mathcal{U}_\bullet$ of pointed types and pointed maps

For the following discussion, I will now introduce a more specific category $\mathcal{H}_n$ as follows.

Definition 4.3 : $\mathcal{H}_n$

Given $n > 0$, the (univalent) category $\mathcal{H}_n$ is defined by having:

- as objects the pointed types $(X, x_0) : \mathcal{U}_\bullet$ such that X is a $(n-1)$-connected n-type.
- as morphism from (X, x_0) to (Y, y_0) the pointed map $(X, x_0) \to_\bullet (Y, y_0)$

For $n = 1$, it is moreover asked for the objects to be *homogeneous*.
And for $n = 0$, $\mathcal{H}_0$ is defined to be the category of abelian groups which are set. The category $\mathcal{H}_0$ will also be denoted **Ab** in what follows.

Notice that the proofs of is-$(n-1)$-connected(X) and is-n-type(X) are not considered in the above definition. This is because those are *mere propositions*, so their proofs are *irrelevant*. And thus they are automatically preserved by the morphisms of the category $\mathcal{H}_n$. Thanks to this property, in what follows we should never worry about the proof terms of these propositions, and just consider them as properties as it is usually done in mathematics.

Now, directly from the definitions of connectivity and type level, one get the following corollary from the functoriality of Ω:

Theorem 4.2 : Functoriality of Ω

For any $n > 0$, Ω defines a *functor* from the category $\mathcal{H}_n$ to the category $\mathcal{H}_{n-1}$

(In the case $n = 2$, this follows from the fact that every loopspace is an homogeneous type, and in the case $n = 1$, this follows from the property that every homogeneous type has abelian lopspaces)

4.1.2 delooping

One of the main objectives of my internship was to formalize the following impressive result :

Theorem 4.3 : Equivalence of Ω

For any $n \geq 2$, the functor $\Omega : \mathcal{H}_n \to \mathcal{H}_{n-1}$ is as *equivalence of category*.
In particular, any pointed type (A, a) in $\mathcal{H}_n$ may be **delooped**, that is there exist a unique pointed type (K, k_0) up to isomorphism in $\mathcal{H}_{n-1}$ such that $\Omega(K, k_0) = (A, a)$. And moreover, any pointed map $(f, f_0) : \Omega(A, a) \to_\bullet \Omega(B, b)$ gives rise to a unique pointed map $(g, g_0) : (A, a) \to_\bullet (B, b)$ such that $\Omega(g, g_0) = (f, f_0)$

The proof has been fully formalised in Agda, by taking advantage of the existing *HoTT-Agda library*, and I will now explain some details of the proof, which involves the explicit definition of an inverse to Ω; which is the construction of torsors over a pointed (homogeneous) type. This proof follows the crucial ideas of David WÄRN.

First, fix an integer $n \geq 1$ and a type A in $\mathcal{H}_n$, we have a notion of torsor over A given by the following definition:

Definition 4.4 : A-torsors

Given a n-type B which is $(n-1)$-connected, a **structure of A-torsor** on B correspond to an element of type $\prod_{(b:B)}(B, b) =_{\mathcal{U}_\bullet} (A, a)$ Then we define a A-torsor to be such a type together with a structure of A-torsor. Formally, we may express the type A-torsor as:

$$A\text{-torsor} :\equiv \sum_{B:\mathcal{U}} \mathsf{is}\text{-}n\text{-}\mathsf{type}(B) \times \mathsf{is}\text{-}(n-1)\text{-}\mathsf{connected}(B) \times \prod_{b:B}(B, b) = (A, a)$$

Intuitively, a A-torsor is a type which is isomorphic to (A, a) from any point of view, that with any choose of a distinguished point. This is similar to the usual notion of G-torsor for a group G, which correspond to a set who inherit a group structure by identifying the neutral element of G with any point on the torsor. In particular, it implies that every torsor in automatically an homogeneous type.

Because it follows from the definition of $\mathcal{H}_n$ that A must be a homogeneous type, the type of A-torsors hosts a distinguished torsor: namely the trivial **torsor over** A.

Definition 4.5 : Trivial torsor

The **trivial torsor** $\mathcal{T}A$ over A is defined by the following data:

- the base space is given by A itself
- the properties of being a n-type, and being $(n-1)$-connected are satisfied by assumption on A.
- for every point a' in A, the identification $(A, a') = (A, a)$ is given by the homogeneity of A.

Given B a A-torsor, every choice of point b on B induces an identification between (B, b) and (A, a), and *a posteriori* an identification between the torsor B and the trivial torsor $\mathcal{T}A$. One may indeed show that this is an equivalence, and that every identification between the *torsor* B and the trivial torsor $\mathcal{T}A$ correspond to a unique point on B. I have formalised this result in my Agda developement, and this is a cornerstone for the definition of the delooping. From this result, follows the corollary:

Lemma 4.1

The type of identifications $\mathcal{T}A =_{A\text{-torsor}} \mathcal{T}A$ is isomorphic (and thus *equal*, according to the univalence axiom) to the type A. And under this identification, the distinguished point a of A correspond to the trivial path $1_{A\text{-torsor}}$. This may be

synthetically reformulated as:

$$\Omega\left(A\text{-torsor}, \mathcal{T}A\right) = (A, a)$$

This motivate the following definition:

Definition 4.6 : Delooping of (A, a)

In order to make abstract the precedent construction, the pointed type of torsors over (A, a) is also denoted $\mathbf{B}(A, a)$ or just $\mathbf{B}A$, and is called the **delooping** of (A, a). We thus have the relation

$$\Omega\mathbf{B}(A, a) = (A, a)$$

Now for the other way around, let's fix a pointed type (A, a) in $\mathcal{H}_{n+1}$. Notice that there is a special class of torsors over the loopspace $\Omega(A, a)$. For any $x : A$, one may define a ΩA-torsor structure on $x =_A a$ with the following isomorphism:

$$\text{For every } \alpha_0 : x =_A a,\ \Phi_{\alpha_0} :\equiv \left(\lambda\alpha.\, \alpha_0^{-1} \cdot \alpha\right) : x =_A a \to \Omega(A, a)$$

Indeed, every torsor on ΩA arise in this way, in order to see this, it suffices to show that the map

$$\lambda(x : A).\ (x = a) : A \to \Omega A\text{-torsor}$$

is an equivalence, *i.e* that for any ΩA-torsor TB, the fiber $\sum_{(x:A)}(TB =_{\Omega A\text{-torsor}} (x = a))$ is contractible. But since being contractible is a mere proposition, we may suppose to have a point b in the base space B of TB in order to show this. This point gives us an identification $TB = \mathcal{T}(\Omega A)$. So we are therefore reduced to showing that the type $\sum_{(x:A)}(\mathcal{T}(\Omega A) =_{\Omega A\text{-torsor}} (x = a))$. Now once more, this is equivalent, as previously mentionned, to choosing a path $\alpha_0 : x = a$. That is the fiber is equivalent to the type $\sum_{(x:A)}(x =_A a)$ which is known to be contractible. (see for example Lemma 3.11.8 in [8].)

Thus, we showed that the type $\mathbf{B}\Omega(A, a)$ is identical to (A, a) itself.

Now, we show that the functor Ω is fully faithful. In order to get this result, fix (A, a), (B, b) in $\mathcal{H}_{n+1}$ and a map $g_\bullet = (g, g_0) : \Omega(A, a) \to_\bullet \Omega(B, b)$. To begin with, one may show that giving a map $f_\bullet = (f, f_0) : (A, a) \to_\bullet (B, b)$ such that $\Omega f_\bullet = g_\bullet$ is the same as giving, for every $x : A$, a point $y : B$ and a map $h : x = a \to y = b$ such that for every $\alpha : x = a$ the following pointed map h^α

$$\lambda\omega.\ (h\,\alpha)^{-1} \cdot (h\,(\alpha \cdot \omega)) : \Omega(A, a) \to \Omega(B, b)$$

(with the proof of being pointed given by the inverse path property) is equal to $g_\bullet$ as a pointed map. So this show that the fiber of Ω at $g_\bullet$ is isomorphic to the type

$$\prod_{(x:A)} \sum_{(y:B)} \sum_{(h:x=a\to y=b)} \prod_{(\alpha:x=a)} h^\alpha = g_\bullet$$

Then by a clever use of the lemmas previously stated about connectivity, one can show that this type is contractible.

Finally, I also formalised a part of an extension of the equivalence in the case $n = 1$:

Theorem 4.4 : Equivalence of Ω in the case $n = 1$

In the case $n = 1$, one may refine the definition of torsor (this is the traditionnal notion of G-torsor for a group G) such that Ω and $(-)$-torsor are still inverses as functors. Given an abelian group G, the notation $\mathbf{B}G$ denotes the (pointed) type G-torsor; its underlying type is an homogeneous, connected groupoid.

The result we hand up with is a beautiful way to identify the category of abelian group with any $\mathcal{H}_n$.

4.1.3 EILENBERG-MACLANE space of a group

Fix an abelian group G (we implicitely assume that the underlying type of the group G is a *set* when making this assertion). By using the previous lemmas, it is possible to deloop G as many times as wanted. And after n iterations this construction results in an object $\mathsf{K}(G, n)$ which is a $(n-1)$-connected n-type.

Definition 4.7 : EILENBERG-MACLANE spaces

Given $n : \mathbb{N}$, the n-th EILENBERG-MACLANE space (EM-space) of an abelian group G is defined as

$$\mathsf{K}(G, n) :\equiv \mathbf{B}^n G$$

It is a pointed $(n-1)$-connected n-type, and satisfies the property

$$\Omega^n(\mathsf{K}(G,\,n)) = G$$

And, since $\Omega^n : \mathcal{H}_n \to \mathcal{H}_0$ is an equivalence, $\mathsf{K}(G,\,n)$ is caracterized among the objects of $\mathcal{H}_n$ by the following universal property:

$$\text{For every } (A,a) \text{ in } \mathcal{H}_n, \quad \mathsf{Hom}_{\mathcal{H}_n}(\mathsf{K}(G,\,n),\,(A,a)) \simeq \mathsf{Hom}_{\mathbf{Ab}}\,(G,\,\Omega^n(A,a))$$

or by the dual one:

$$\text{For every } (A,a) \text{ in } \mathcal{H}_n, \quad \mathsf{Hom}_{\mathcal{H}_n}((A,a),\,\mathsf{K}(G,\,n)) \simeq \mathsf{Hom}_{\mathbf{Ab}}\,(\Omega^n(A,a),\,G)$$

In Homotopy Type Theory, (or in the context of higher categories), there is a synthetic definition of what is the cohomology group of a given type (or space) which is expressed by using the notion of EILENBERG-MACLANE space.

Definition 4.8 : Cohomology group of a type

Given a type A and $n : \mathbb{N}$, the n-th **cohomology group** of A with coefficients in G is defined as

$$\mathrm{H}^n(A,G) :\equiv \pi_0\,(A \to \mathsf{K}(G,\,n))$$

The group structure is given by the one induced from G to $\mathsf{K}(G,\,n)$.

And it turns out that this definition of cohomology satisfies the EILENBERG-STEENROD axioms, thus defining the usual cohomology.

4.2 Homotopy of the n-sphere

A striking consequence of the existence of EM-spaces is that it makes it possible to compute synthetically the firsts (n) homotopy groups of the n-sphere. This is an idea spotted by David WÄRN and I managed to retrieve the detailed reasonment wich is now presented.

In order to keep synthetic the calculations, the explicit construction of the n-sphere will not be needed, and the only property that is required is the following universal property:

$$\text{There is a bijection, natural is the pointed type } (A,a), \quad \left(\mathbb{S}^n \to_\bullet (A,a)\right) \simeq \Omega^n(A,a)$$

where $\mathbb{S}^n$ denotes the n-sphere, which is here considered as pointed type.

The following universal property of $\mathbb{Z}$ is also used:

$$\text{There is a bijection, natural is the group } G, \quad \mathsf{Hom}_{\mathcal{H}_n}(\mathsf{K}(G,\,n),\,(A,a)) \simeq \mathsf{Hom}_{\mathbf{Gp}}\,(\mathbb{Z},\,G) \simeq G$$

Where **Gp** denotes the category of groups, and G is identified with its underlying type.

By the universal property of the n-sphere, choosing a map $\mathbb{S}^n \to_\bullet \mathsf{K}(\mathbb{Z},\,n)$ is the same as choosing a point in $\Omega^n(\mathsf{K}(\mathbb{Z},\,n))$. Knowing that $\mathbb{Z} = \Omega^n(\mathsf{K}(\mathbb{Z},\,n))$, we can pick the point 1, and define $[-] : \mathbb{S}^n \to_\bullet \mathsf{K}(\mathbb{Z},\,n)$ to be the associated pointed map.

Lemma 4.2

For every $n : \mathbb{N}$, the n-th EILENBERG-MACLANE space of the abelian group $\mathbb{Z}$ is a **n-th truncation** of the n-th sphere $\mathbb{S}^n$. That is, satisfies the following universal property:

$$\text{For every } n\text{-type } A, \text{ the precomposition by } [-] \text{ defines an equivalence} \quad \left(\mathsf{K}(\mathbb{Z},\,n) \to A\right) \simeq \left(\mathbb{S}^n \to A\right)$$

Proof. Fix an arbitrary n-type A. First, notice that for any pointed type (B,b), there is an equivalence

$$(B \to A) \simeq \sum_{a:A} ((B,b) \to_\bullet (A,a))$$

so it suffice to fix a point $a : A$, and to show that there is an equivalence

$$\left(\mathsf{K}(\mathbb{Z},\,n) \to_\bullet (A,a)\right) \simeq \left(\mathbb{S}^n \to_\bullet (A,a)\right)$$

Now, the left hand side is equivalent to the type $\mathsf{Hom}_{\mathbf{Gp}}(\mathbb{Z},\,\Omega^n(A,a))$ according to the universal property of EM-spaces. So it is equivalent to the type $\Omega^n(A,a)$ by the universal property of the group $\mathbb{Z}$. Finally, this is equivalent to the right hand side by the universal property of the n-sphere. □

By universality, the EM-space $K(\mathbb{Z}, n)$ is then the n-th truncation of the n-sphere. And we may now use two classical results (see for example the corollary 7.3.14 and the Lemma 8.3.2 in [8]) about the homotopy groups of connected and truncated types:

Lemma 4.3

Given $n : \mathbb{N}$, and a pointed type (A, a) such that A is n-connected, $\pi_k(A, a) = \mathbb{1}$ for every $k \leq n$. And, for every pointed type (A, a), and every $k \leq n$, $\pi_k(A, a) = \pi_k(\|(A, a)\|_n)$

Since $\mathsf{K}(\mathbb{Z}, n)$ is a $(n-1)$-connected n-type, and $\Omega^n(\mathsf{K}(\mathbb{Z}, n)) = \mathbb{Z}$ we know by the first point of the lemma and the definition of the n-th homotopy group that

- For every $k < n$, $\quad \pi_k(\mathsf{K}(\mathbb{Z}, n)) = \mathbb{1}$
- $\pi_n(\mathsf{K}(\mathbb{Z}, n)) = \mathbb{Z}$

Then, by the second point of the lemma, we obtain the following theorem:

Theorem 4.5 : Homotopy groups of the n-sphere

$$\text{for all } n : \mathbb{N} \text{ and } k < n, \quad \pi_k(\mathbb{S}^n) = \mathbb{1} \quad \text{and} \quad \pi_n(\mathbb{S}^n) = \mathbb{Z}$$

4.3 The fiber sequence

There is a fundamental phenomenon that arises when studying (co)homology; which is the existence of a long exact sequence through (co)homology groups induced each time there is a short exact sequence of chain complex. More suprisingly, what we only need in order to generate such a long exact sequence is only a fibration. Indeed, the sequence arising in (co)homology is the manifestation of a deeper result, that may be understood in the context of the theory of ∞-groupoid, and thus may be stated in the synthetic language of Homotopy Type Theory.

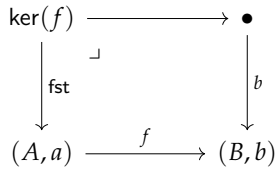
Theorem 4.6 : Fiber sequence of a fibration

If (A, a), (B, b) are two pointed types and $f : (A, a) \to_\bullet (B, b)$ is a pointed map, then there is a long exact sequence:

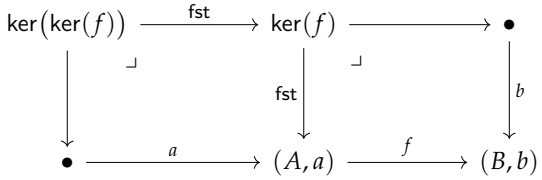
$$\ldots \to_\bullet \Omega^{n+1}(B, b) \to_\bullet \Omega^n \mathsf{ker}(f) \to_\bullet \Omega^n(A, a) \to_\bullet \Omega^n(B, b) \to_\bullet \ldots \to_\bullet \mathsf{ker}(f) \to_\bullet (A, a) \to_\bullet (B, b)$$

Where the kernel $\mathsf{ker}(f)$ of $f : (A, a) \to_\bullet (B, b)$ is defined to be the fiber $\sum_{(x:A)} f\, x =_B b$

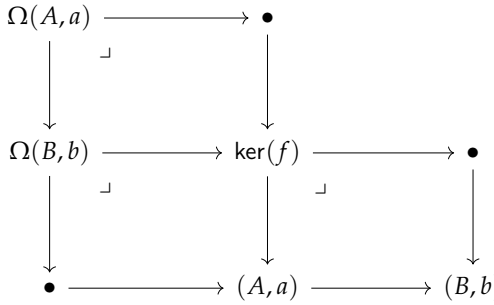
Proof. To begin with, remark that the definition of the fiber $\mathsf{ker}(f)$ yield a pullback square



But since the first projection fst is also pointed, we may form the kernel of this projection. We then obtain a second pullback:



But by pullback pasting, we may now identify the kernel of the kernel with the loopspace $\Omega(B, b)$ And since the first projection $\mathsf{fst} : \mathsf{ker}(\mathsf{ker}(f)) \to \mathsf{ker}(f)$ is also pointed, one may still form the pullback of this map, resulting in the following diagram:



By iterating this construction, one get the desired sequence. It is possible to carry out the exact computations of the map involved in this diagram (see for example lemma 8.4.4 in [8]), and one can identify in this way the map $\Omega(A, a) \to_\bullet \Omega(B, b)$ with the map $-\Omega f$. That is the map $\Omega f \circ (-)^{-1}$. So the sequence obtained is indeed the following:

$$\ldots \xrightarrow{(-1)^n\Omega^n\partial}_\bullet \Omega^n\mathsf{ker}(f) \xrightarrow{(-1)^n\Omega^n\mathsf{fst}}_\bullet \Omega^n(A,a) \xrightarrow{(-1)^n\Omega^n f}_\bullet \Omega^n(B,b) \to_\bullet \ldots \xrightarrow{\partial}_\bullet \mathsf{ker}(f) \xrightarrow{\mathsf{fst}}_\bullet (A,a) \xrightarrow{f}_\bullet (B,b)$$

□

For a deeper understanding of this situation, the reader may also visit the nLab, and especially the page about the fiber sequence.
There is several corollaries to the existence of this sequence. For example, since the functor π_0 preserves the exactness, one can derive from this sequence the *long exact sequence in homotopy*

$$\ldots \to_{\mathsf{gp}} \pi_{n+1}(B,b) \to_{\mathsf{gp}} \pi_n\big(\mathsf{ker}(f)\big) \to_{\mathsf{gp}} \pi_n(A,a) \to_{\mathsf{gp}} \pi_n(B,b) \to_{\mathsf{gp}} \ldots \to_\bullet \pi_0\big(\mathsf{ker}(f)\big) \to_\bullet \pi_0(A,a) \to_\bullet \pi_0(B,b)$$

Another example is the following. Since, for any pointed type X, the functor

$$(Y,y_0) \longmapsto \big((X \to Y)\,,\, y_0\big) : \mathcal{U}_\bullet \longrightarrow \mathcal{U}_\bullet$$

is exact, one get the exact sequence

$$\ldots \to_\bullet (X \to \Omega^n\mathsf{ker}(f)) \to_\bullet (X \to \Omega^n(A,a)) \to_\bullet (X \to \Omega^n(B,a)) \to_\bullet \ldots$$

And, by taking the 0-th truncation, the sequence in cohomology

$$\ldots \to_\bullet \mathrm{H}\,(X, \Omega^n\mathsf{ker}(f)) \to_\bullet \mathrm{H}\,(X, \Omega^n(A,a)) \to_\bullet \mathrm{H}\,(X, \Omega^n(B,a)) \to_\bullet \ldots$$

In the context of my internship, I needed to obtain a specific consequence of this result, relating the cohomology in groups which are linked by a short exact sequence. There is a preliminary useful result I proved for this purpose.

Definition 4.9 : Fibration sequence

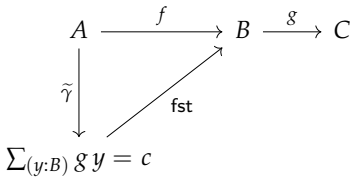
Let (A,a), (B,b), (C,c) be three pointed types with $f_\bullet = (f, f_0) : (A,a) \to_\bullet (B,b)$ and $g_\bullet = (g, g_0) : (B,b) \to_\bullet (C,c)$. then the sequence

$$(A,a) \to_\bullet (B,b) \to_\bullet (C,c)$$

is said to be a fibration when there is a path family

$$\gamma : \prod_{x:A} (g \circ f)(x) = c$$

such that the induced map to the kernel of g (*i.e* the fiber of f over b) $\widetilde{\gamma} : A \to \sum_{(y:B)} g\,y = c$ is an equivalence. In this case the following diagram commute:



Lemma 4.4

For every $n \geq 2$, and every sequence

$$(A,a) \xrightarrow{(f,f_0)}_\bullet (B,b) \xrightarrow{(g,g_0)}_\bullet (C,c) \qquad (S)$$

in the category $\mathcal{H}_n$. (S) is a fibration if and only if the looped sequence $\Omega(S)$ is a fibration.

Proof. omited for concision □

We also make use of this caracterisation of fibration in the case of group sequences:

Lemma 4.5

For every sequence

$$(A,a) \xrightarrow{(f,f_0)}_\bullet (B,b) \xrightarrow{(g,g_0)}_\bullet (C,c) \qquad (S)$$

in the category $\mathcal{H}_1$, the short group sequence

$$\mathbb{1} \to_{\mathsf{gp}} (\Omega(A,a)) \to_{\mathsf{gp}} \Omega(B,b) \to_{\mathsf{gp}} \Omega(C,c) \to_{\mathsf{gp}} \mathbb{1} \qquad (\mathsf{ext}{-}S)$$

is exact if and only if the sequence (S) of groupoid is a fibration.

Proof. also omited for brievety □

5 Group cohomology and HoTT

During my internship, I also learned some notions of group cohomology and basic homological algebra. I mentionned previously that we may define in HoTT a notion of cohomology in terms of maps in an EILENBERG-MACLANE space. In a famous article published in Tohoku, Grothendieck provides us with a very practical toolbox to identify different cohomologies, especially the notion of *universal δ-functor*. And the theorems related to this notion still apply in Homotopy Type Theory. Hence, if the cohomology defined with the EM-spaces turns out to be a so-called universal δ-functor. Then it will coincide with the others usual definitions of cohomology. I took part of this project to seek for such a result. Thanks to Felix CHERUBINI's work, a proof strategy has been identified, consisting in showing that we may cancel any cohomology class in a suitable extension of the coefficients module. Hence I tried to prove this result for several cohomologies revolving around the one defined in HoTT, and especially in the case of group cohomology. Even if I didn't suceed in proving it in every degree, I managed to do so in some cases, which I present in the following parts.

5.1 Some results about canceling cohomology

5.1.1 Group cohomology in coefficients

For a moment, I will work with the synthetic definition of group cohomology in terms of coefficients which I recall here.

Let G be a group and A a left G-module.

Definition 5.1: $H^k(G,\,A)$

Let $C^k(A) := G^k \to A$ for each $k \in \mathbb{N}$ and $C^{-1}(A) := 0$. We may also define a coboundary operation $\partial^k : C^{k-1}(A) \to C^k(A)$ by

$$\begin{aligned}(\partial^k a)_{g_1,\dots,g_k} \quad := \quad & g_1 \cdot a_{g_2,\dots,g_n} - a_{g_1\cdot g_2,g_3,\dots,g_n} + a_{g_1,g_2\cdot g_3,\dots,g_n} + \dots \\ & +(-1)^{n-1} a_{g_1,\dots,g_n,g_{n-1}\cdot g_n} + (-1)^n a_{g_1,\dots,g_{n-1}}\end{aligned}$$

We may check that $\partial^{k+1} \circ \partial^k = 0$ so it defines a cochain complexe. We may now define $H^k(G,\,A)$ as the quotient

$$H^k(G,\,A) := \frac{\mathsf{im}(\partial^k)}{\mathsf{ker}(\partial^{k+1})}$$

Remark
a priori this definition requires the use of higher inductives types. (see [8], section 6.10 for more details about quotients in HoTT). In the case $k = 0$, $H^0(A)$ corresponds to the G-invariants of A.

5.2 Torsors and Group cohomology in coefficients

Finally, even if the attempts toward a proof that the cohomology is a universal δ-functor were not succesfull, I managed to relate the two cohomologies (in terms of torsors and coefficients) in some specific degree or cases. This is the aim of the following subsections.

5.2.1 First cohomology group

Let G be an abstract group and $\mathbf{B}G$ the type of G torsors. We denotes $* : \mathbf{B}G$ the trivial torsor in BG. Hence we have $\Omega\,(\mathbf{B}G, *) = G$.

Let $\underline{M} : \mathbf{B}G \to \mathbf{Ab}$ be a family of abelian groups indexed over the delooping of G. It is possible to think of $\underline{M}$ as a G-module via the following assignment.

- Let $M :\equiv \underline{M}(*)$, it is an abelian (abstract group).
- To any $g : G$ corresponds a path $p_g : * =_{\mathbf{B}G} *$. Hence it yields an equality $\underline{M}_* p_g : M =_{\mathbf{Ab}} M$. Since such an equality corresponds to an isomorphism of abelian groups by univalence, we may encapsulate this data as a morphism $\rho : G^{\mathsf{op}} \to M = M$, corresponding to a right action of G on M.

The first cohomology group of the *space* $\mathbf{B}G$ with coeeficients in $\underline{M}$ according to the EILENBERG-MACLANE-spaces definition is given by $H^1(\mathbf{B}G, \underline{M}) = \pi_0\left(\prod_{(x:\mathbf{B}G)} \mathsf{EM}(\underline{M}(x), 1)\right)$. Howether, from the previous section, we also have an other definition of cohomology that we may associate to G and M, namely the cohomology of the *group* G with coefficients in the G-module M ; $H^1\,(G, M)$.

Now one may wonder if those two assignments turns out to be the same.

Let T be a M-torsor. In the following, we will need to understand what is the type of paths

$$\mathsf{transport}^{\mathsf{EM}(\underline{M}(\cdot),1)}(p_g, T) =_{\mathsf{EM}(\mathsf{M},1)} T$$

we unfold the definition of transport in $\mathsf{EM}(\underline{\mathsf{M}}(\cdot), 1)$ by writing $\mathsf{EM}(N, 1)$ as

$$\sum_{(A:\mathsf{Set}_{\mathcal{U}})} \sum_{(\alpha:N\times A\to A)} \left\|A\right\| \times \mathsf{isAction}(\alpha) \times \prod_{a:A} \mathsf{isEquiv}\,(\alpha(\cdot, a))$$

and T as $(A, \alpha, \_)$ Hence, we may show

$$\mathsf{transport}^{\mathsf{EM}(\underline{M}(\cdot),1)}(p_g, T) = \left(A, (m, a) \mapsto \alpha(m \cdot g^{-1}, a), \_\right)$$

where $m \cdot g := \mathsf{idtoeqv}(\rho(g))(m)$. Now, we see that the path $\mathsf{apd}_{\underline{T}}(p_g)$ is an equality

$$\mathsf{apd}_{\underline{T}}(p_g) : \left(A, (m, a) \mapsto \alpha(m \cdot g^{-1}, a), \_\right) = (A, \alpha, \_)$$

That is, a path $\sigma(g) : A = A$ such that (it is a proposition)

$$\prod_{(m:M)} \prod_{(a:A)} \alpha\left(m \cdot g^{-1}, a \cdot g^{-1}\right) \cdot g = \alpha(m, a)$$

where for $a : A$, $a \cdot g := \mathsf{idtoeqv}(\sigma(g))(a)$ Finally, this may be stated as a right G-action σ on A such that, for every $m : M$, $a : A$ and $g : G$,

$$\alpha\,(m \cdot g, a \cdot g) = \alpha(m, a) \cdot g$$

For now on, we let $g \cdot m := m \cdot g^{-1}$.

Notice that $\sum_{(x:BG)} \mathsf{EM}(\underline{M}(x), 1)$ with the pair $(*, T)$ as a distinguished element is a pointed connected groupoid. Hence, we may apply the previously stated universal property of the delooping: giving a map $\mathbf{B}G \to_\bullet \sum_{(x:BG)} \mathsf{EM}(\underline{M}(x), 1)$ is the same as giving a map $G \to_{\mathsf{gp}} \Omega\left(\sum_{(x:\mathbf{B}G)} \mathsf{EM}(\underline{M}(x), 1)\right)$. We also have a group isomorphism $\Omega\left(\sum_{(x:\mathbf{B}G)} \mathsf{EM}(\underline{M}(x), 1)\right) \to_{\mathsf{gp}} \sum_{(g:G)} \mathsf{transport}^{\mathsf{EM}(\underline{M}(\cdot),1)}(p_g, T) =_{\mathsf{EM}(\mathsf{M},1)} T$. And by expliciting the transport in $\mathsf{EM}(\underline{\mathsf{M}}(\cdot), 1)$, this group is also isomorphic to

$$\sum_{(g:G)} \sum_{(\sigma:A\simeq A)} \prod_{(m:M)} \prod_{(a:A)} \sigma(\alpha(m, a)) = \alpha(g \cdot m, \sigma(n))$$

Where the group law is given by

$$(g, \sigma) \cdot (h, \tau) = (gh, \sigma \circ \tau)$$

Given an abritrary point $a_0 : A$, we have an equivariant bijection $a \mapsto m_a$ where $\alpha(m_a, a_0) = a$ determines m_a. Under this map, we have $m_{\sigma(a)} = m_{\sigma(\alpha(m_a, a_0))} = m_{\alpha(g\cdot m_a, \sigma(a_0))} = g \cdot m_a + m_{\sigma(a_0)}$ Hence σ is fully determined by its image on a_0 and the group we care about is isomorphic to

$$G \times M$$

where the element $m : M$ corresponds to $m_{\sigma(a_0)}$. The group law is given by

$$(g, m) \cdot (h, n) = (gh, g \cdot n + m))$$

(because if $\sigma(a_0) = \alpha(m, a_0)$ and $\tau(a_0) = \alpha(n, a_0)$, then $\sigma \circ \tau(a_0) = \sigma(\alpha(n, a_0)) = \alpha(g \cdot n, \sigma(a_0)) = \alpha(g \cdot n + m, a_0))$ and we recognize here the semi-direct product $M \rtimes_\rho G$.

Now, observe that giving a family of torsors $\underline{T} : \prod_{(x:\mathbf{B}G)} \mathsf{EM}(\underline{M}(x), 1)$ with $\underline{T}(*) = T$ is actually the same as giving a non dependant map $\widetilde{\underline{T}} : \mathbf{B}G \to \sum_{(x:BG)} \mathsf{EM}(\underline{M}(x), 1)$ which is constant on the first component. With the previously noticed group isomorphism, those maps corresponds to the maps $G \to M \rtimes_\rho G$ which are a section of the projection on G. That is to families $(m_g)_{g:G}$ such that $m_{g \cdot h} = g \cdot m_h + m_g$. Hence, when giving $a_0 : A$, giving $\underline{T}$ with $\underline{T}(*) = T$ is the same as giving a cocyle in the G-module M. But now we may check that this cocycle depend on the choice of a_0 we made only up to a coboundary, because when giving another $a_1 = \alpha(u, a_0)$ on A, we have for every $(g, \sigma) : \sum_{(g:G)} \sum_{(\sigma:A \simeq A)} \prod_{(m:M)} \prod_{(a:A)} \sigma(\alpha(m, a)) = \alpha(g \cdot m, \sigma(n))$, if $\sigma(a_0) = \alpha(m, a_0)$ then

$$\sigma(a_1) = \sigma(\alpha(u, a_0)) = \alpha(g \cdot u, \sigma(a_0)) = \alpha(g \cdot u, \alpha(m, a_0)) = \alpha(g \cdot u + m, a_0) = \alpha(g \cdot u + m - u, a_0)$$

Hence by choosing a_1 instead of a_0, the cocycle $(m_g)_{g:G}$ become $(n_g)_{g:G}$ with, for each $g : G$:

$$n_g = g \cdot u + m_g - u$$

Thus $\left[(n_g)_{g:G}\right] = \left[(m_g)_{g:G}\right]$ does not depend on the choice of a distinguished point on A. Now, given a group morphism

$$(id_G, \sigma) : G \to \sum_{(g:G)} \sum_{(\sigma:A \simeq A)} \prod_{(m:M)} \prod_{(a:A)} \sigma(\alpha(m, a)) = \alpha(g \cdot m, \sigma(n))$$

we may caracterise the constructed cocycle class κ by

$$Q(\kappa) :\equiv \prod_{(a_0:A)} \sum_{(m:C^1(G,M))} ([m] = \alpha) \times \prod_{g:G} \alpha(m_g, a_0) = \sigma_g(a_0)$$

By choosing a distinguished point on A, we may derive that the property $Q(\kappa)$ determines fully κ. Hence it shows

$$\mathsf{isProp}\left(\sum_{\kappa:H^1(G,M)} Q(\kappa)\right)$$

and we may use $\left\|A\right\|$ to obtain an element of this sigma type by picking a point on A. This defines a map

$$\prod_{x:\mathbf{B}G} \mathsf{EM}(\underline{M}(x), 1) \longrightarrow H^1(G, M)$$

and because $H^1(G, M)$ is a set, it induces a map

$$H^1(\mathbf{B}G, \underline{M}) \xrightarrow{\Phi} H^1(G, M)$$

Now, we see the reciproque: Given a cocyle $(m_g)_{g:G}$ in M, defines A to be the M-torsor M with the action given by translation $\alpha(m, a) :\equiv m + a$. The map

$$\begin{array}{ccc} G & \longrightarrow & M \times G \\ g & \longmapsto & (m_g, g) \end{array}$$

defines a group morphism $G \to M \rtimes_\rho G$ which is the identity on the second component. Hence, by choosing $a_0 := 0 : M$ as a distinguished element on m, it defines a family of torsors $\underline{T} : \prod_{(x:\mathbf{B}G)} \mathsf{EM}(\underline{M}(x), 1)$, with $\underline{T}(*) = M$ the trivial M-torsor defined above. If instead of $(m_g)_{g:G}$ we choose the cocycle $(n_g)_{g:G}$ in the same class than $(m_g)_{g:G}$ the torsor family obtained correspond to the morphism

$$\begin{array}{ccc} G & \longrightarrow & M \times G \\ g & \longmapsto & (n_g, g) \end{array}$$

If $u : M$ is such that for every $g : G$, $n_g = g \cdot u + m_g - u$, then we notice that the map

$$(m, g) \mapsto (g \cdot u + m - u, g)$$

defines a group isomorphism $M \rtimes_\rho G \simeq M \rtimes_\rho G$ which identifies the morphisms associated to $(m_g)_{g:G}$ and $(n_g)_{g:G}$. Hence the families of torsors induced by both cocycles are equals, and they defines the same class in $H^1(\mathbf{B}G, \underline{M})$. This construction yield a map

$$H^1(M, G) \xrightarrow{\Psi} H^1(\mathbf{B}G, \underline{M})$$

And now we may check (this is indeed by construction) that (Φ, Ψ) defines an equivalence.

5.2.2 Second cohomology

Now, we may ask the same question for the second group. In what follows, we unfold the definitions of $EM(\cdot, 2)$ do describe what it means to give a dependant family $\mathcal{G} : \prod_{(x:\mathbf{B}G)} \mathsf{EM}\left(\underline{M}(x), 2\right)$.

Giving such an element $\mathcal{G}$ is the same as giving an element of

$$\sum_{(\underline{A}:\mathbf{B}G\to\mathcal{H}_1)} \prod_{(x:\mathbf{B}G)} \prod_{(a:\underline{A}(x))} \left(\underline{A}(x),\, a\right) \simeq_\bullet \left(\mathsf{EM}\left(\underline{M}(x),\, 1\right),\, *\right)$$

Since $\mathcal{H}_1$ is a groupoid, giving such a map A corresponds to chosing the image $A :\equiv \underline{A}(*)$ and a group homomorphism $G \to_{\mathsf{gp}} \Omega\left(\mathcal{H}_1,\, A\right)$. *i.e.* to choose an homogeneous groupoid A with a left G-action. Then, defining an element of $\prod_{(x:\mathbf{B}G)} \prod_{(a:\underline{A}(x))} \left(\underline{A}(x),\, a\right) \simeq_\bullet \left(\mathsf{EM}\left(\underline{M}(x),\, 1\right),\, *\right)$ is the same as defining an element $\Gamma : \prod_{(a:A)} \left(A,\, a\right) \simeq_\bullet \left(\mathsf{EM}\left(M,\, 1\right),\, *\right)$ together with a group homomorphism

$$G \to_{\mathsf{gp}} \sum_{g:G} \mathsf{transport}^{(x,a)\mapsto \prod_{(a:\underline{A}(x))}\left(\underline{A}(x),a\right)\simeq_\bullet \mathsf{EM}\left(\underline{M}(x),1\right)}\left(p_g,\, \Gamma\right) = \Gamma$$

which is the identity on the first component. By writing Γ as $(T,\, \alpha,\, \zeta)$ with $T : \prod_{(x:\mathbf{B}G)}(Ax) \to \mathsf{Set}_{\mathcal{U}}$ the carrier of the torsor, $\alpha(a,\, b)$ the action on $T(a,\, b)$ and ζ_a the distinguished point on $T(a,\, a)$ which corresponds to a proof that $b \mapsto \left(T(a,\, b),\, \alpha(a,\, b),\, \_\right)$ is pointed, The inner part of this sigma type may be rewritten as

$$\begin{aligned}\prod_{(a,\,b:A)} \sum_{(\sigma_{g,\,a,\,b}:T(a,\,g\cdot b)\simeq T(g^{-1}\cdot a,\,b)} \prod_{(m:M)} \prod_{(t:T(a,\,g\cdot b))} \sigma_{g,\,a,\,b}\left(\alpha(a,\, g\cdot b)(g\cdot m,\, t)\right) = \alpha\left(g^{-1}\cdot a,\, b\right)\left(m,\, \sigma_{g,\,a,\,b}(t)\right)\\ \times \prod_{(a:A)} \sigma_{g,\,g\cdot a,\,a}\left(\zeta_{g\cdot a}\right) = \zeta_a\end{aligned}$$

Which may be rephrased again as

$$\begin{aligned}\prod_{(a,\,b:A)} \sum_{(\sigma_{g,\,a,\,b}:T(a,\,b)\simeq T(g\cdot a,\,g\cdot b)} \prod_{(m:M)} \prod_{(t:T(a,\,b))} \alpha\left(g\cdot a,\, g\cdot b\right)\left(g\cdot m,\, \sigma_{g,\,a,\,b}(t)\right) = \sigma_{g,\,a,\,b}\left(\alpha(a,\, b)(m,\, t)\right)\\ \times \prod_{(a:A)} \sigma_{g,\,a,\,a}\left(\zeta_a\right) = \zeta_{g\cdot a}\end{aligned}$$

Denote this type by $\mathfrak{S}_g$, then the group law on $\sum_{(g:G)} \mathfrak{S}_g$ will be given by

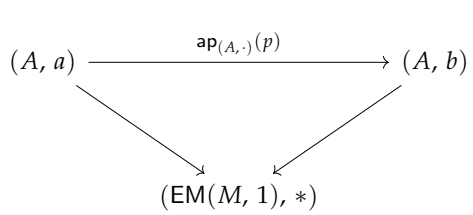
$$\left(g,\, \sigma_g\right)\cdot\left(h,\, \sigma_h\right) = \left(gh,\, \left(a,\, b \mapsto \sigma_{g,\,h\cdot a,\,h\cdot b}\circ\sigma_{h,\,a,\,b}\right)\right)$$

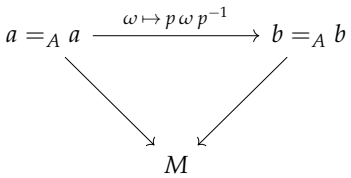
So a map $G \to_{\mathsf{gp}} \sum_{(g:G)} \mathfrak{S}_g$ may be seen as a M-equivariant G-action on the family of torsors T. Actually, there is an intrinsic description of this structure, which may be expressed on the path types of A itself: given $a,\, b : A$, the equivalence $\left(\underline{A}(*),\, a\right) \simeq_\bullet \left(\mathsf{EM}\left(\underline{M}(*),\, 1\right),\, *\right)$ induces an equivalence on paths

$$\left(a =_A b\right) \simeq \left(* =_{\mathsf{EM}(\underline{M},1)} T(a,\, b)\right) \simeq T(a,\, b)$$

where the left action of $a =_A a$ corresponds to the M-torsor structure on $T(a,\, b)$ under the choice of the distinguished point $\zeta_a : a = a$. Since we have a family of such equivalences $\left(\underline{A}(x),\, a\right) \simeq_\bullet \left(\mathsf{EM}\left(\underline{M}(x),\, 1\right),\, *\right)$ for $x : \mathbf{B}G$, the underlying map of $\left(A,\, a\right) \simeq_\bullet \left(\mathsf{EM}\left(M,\, 1\right),\, *\right)$ is G-equivariant, hence so is the family of induced maps $a =_A b \simeq T(a,\, b)$ for $b : A$, that is we have some sort of G-action on the paths of A (induced by the G-action on A) which sends an element $g : G$ and a path $p : a =_A b$ to a path $g\cdot p : g\cdot a =_A g\cdot b$, which corresponds to the isomorphisms $\sigma_{g,\,a,\,b}$ previously described.

Given two points $a,\, b : A$ and any path $p : a =_A b$, we also have an equivalence $\left(A,\, a\right) \simeq_\bullet \left(A,\, b\right)$, which induces on paths $\left(a =_A a\right) \simeq \left(b =_A b\right)$ by conugaison with p (this may be seen by path induction on p). Since there is a family of maps $\prod_{(a:A)}\left(A,\, a\right) \simeq_\bullet \left(\mathsf{EM}(M,\, 1),\, *\right)$, the following diagrams should commutes





Hence, for $m : M$ and $g : G$, if m corresponds to $\omega : a =_A a$, the element $g\cdot m$ of M corresponds to the element $p\left(g\cdot\omega\right)p^{-1}$ in $a =_A a$, for any $p : a = g\cdot a$. Furthermore we may lift the G-action on M to a G-action on each $a =_A a$ by the formula $g * \omega := p\left(g\cdot\omega\right)p^{-1}$ for any choice of a path $p : a =_A g\cdot a$ (This does not depend on the choice of p because $a =_A a$ is abelian since merely identical to M as a group).

Assuming that G is finite, I describe now the construction of a 2-cocycle associated to a given gerbe $\mathcal{G} : \prod_{(x:\mathbf{B}G)} \mathsf{EM}(\underline{M}(x), 2)$. We describe such an element $\mathcal{G}$ by the previously described structure and with the notations already introduced. First, choose an element $a : A$ on A. By using the finiteness of G, we may choose a family of paths $p_g : a =_A g \cdot a$. Then we define a path $\omega_{g,h}$ as

$$\omega_{g,h} := p_g \, (g \cdot p_h) \, p_{gh}^{-1}$$

which corresponds under the identification $(a =_A a) \simeq T(a, a) \simeq M$ to an element $c_{g,h} : M$. Howether, we may work inside the loop space by using the intrinsic presentation previously described. First, we may see that the cocycle ω is closed, that is, for every g, h, $k : G$,

$$(g * \omega_{h,k}) \, \omega_{gh,k}^{-1} \, \omega_{g,hk} \, \omega_{g,h}^{-1} = 1_a$$

which, by abelianity is the same as the relation

$$(g * \omega_{h,k}) \, \omega_{g,hk} = \omega_{g,h} \, \omega_{gh,k}$$

i.e

$$p_g \, (g \cdot (p_h \, (h \cdot p_k) \, p_{hk}^{-1})) \, p_g^{-1} \, p_g \, (g \cdot p_{hk}) \, p_{ghk}^{-1} = p_g \, (g \cdot p_h) \, p_{gh}^{-1} \, p_{gh} \, (gh \cdot p_k) \, p_{ghk}^{-1}$$

i.e

$$p_g \, (g \cdot (p_h \, (h \cdot p_k))) \, p_{ghk}^{-1} = p_g \, (g \cdot p_h) \, (gh \cdot p_k) \, p_{ghk}^{-1}$$

which is now clearly satisfied. Then we should check that modifying our choices of p_g only modifies ω up to a 2-coboundary. If instead of p_g we choose the paths $q_g = \theta_g \, p_g$, the resulting cocycle, which we denote ξ may be related to ω as follows: For each g, $h : G$,

$$\begin{aligned}
\xi_{g,h} &= q_g \, (g \cdot q_h) \, q_{gh}^{-1} \\
&= (\theta_g \, p_g) \, (g \cdot \theta_h) \, (g \cdot p_h) \, q_{gh}^{-1} \, \theta_{gh}^{-1} \\
&= (\theta_g \, p_g) \, (g \cdot \theta_h) \, p_g^{-1} \, \theta_{gh}^{-1} \, p_g \, (g \cdot p_h) \, q_{gh}^{-1} \\
&= \theta_g \, (g * \theta_h) \, \theta_{gh}^{-1} \, \omega_{g,h} \\
&= \left(\partial^2 \theta\right)_{g,h} \, \omega_{g,h}
\end{aligned}$$

Hence the class $[\omega]$ does not depend on our choice of $(p_g)_{g:G}$. We also need to check that it does not depend on the choice $a : A$ that we have made. So suppose that we choose instead $b : A$, the conjugaison by a path $r : a = b$ translate the construction of ω to a cocycle on $b =_A b$, and since the conjugaison commutes with the identifications $(a =_A a) \simeq M$ and $(b =_A b) \simeq M$ it defines the same cocycle in the abstract group M. Because of the unique choice principle, this yield a map

$$\left(\prod_{x:\mathbf{B}G} \mathsf{EM}(\underline{M}(x), 2) \right) \longrightarrow H^2(G, M)$$

And because the target type is a set, this map factors through a map

$$H^2(\mathbf{B}G, \underline{M}) \longrightarrow H^2(G, M)$$

6 Conclusion

To close this report from the mathematical point of view, I would say that Type Theory has proven to me in this work its great capacity for expression. It allows in the context of cohomology, to describe and unify different definitions, topological or algebraic, while bringing a new look at the question and in a more synthetic language than usual. the definition of cohomology groups as a truncation of the type of sections

$$H^n(X, M) = \pi_0 \left(\prod_{x:X} \mathsf{EM}(M(x), n) \right)$$

allows us to recover the simplicial cohomology in the case where X is seen as a space and M as a constant module. But also to find the notion of cohomology of a group with coefficients in a module when X is seen as (the classifying space of) a group, and M as a module equipped with an action of this group.

These two points of view actually unite in the field of site cohomology, where the base space happens to be both a generalization of topological spaces (or let us rather say locales) and groups. Unfortunately, I did not have time to explore these objects below the surface during my internship.

From the point of view of the mathematician, this internship gave me the opportunity to discover a mathematical landscape very different from the one I was accustomed to during my previous years of study. I would say that the point that struck me the most was the understanding of Vladimir VOEVODSKY's hierarchy of types. The latter makes it possible to unify logic and mathematical structure, while giving both of them a geometric flavor. I also think that the language of Type Theory allows a new and much more accessible description of complex mathematical objects, such as ∞-groupoids, while the usual definitions of these will only be presented to me in class starting next year.

I will also remember from these four months at Chalmers the closeness between computer scientists and mathematicians, which was very prolific. Especially in the field of Type Theory, where constructivism and calculus are central and essential concepts of the theory. Ideas coming from computer science are new in mathematics, and sometimes bring a more pragmatic vision. For example by structuring the formalization language so that its storage and processing can be computerized and checked more easily. Beyond mathematical knowledge, I also learned about researching in mathematics, or talk about it. The long exchange sessions with Thierry COQUAND allowed me to become aware of the extent of the subjects with which I was confronted. I had to learn in breadth more than in depth, and see my subject of study evolve as the internship progressed. Certainly this stay in Sweden was an extremely pleasant and enriching experience, from all points of view.

References

[1] Benno van den Berg and Richard Garner. "Types are weak omega-groupoids". In: *Proceedings of the London Mathematical Society* 102.2 (Oct. 2010), pp. 370–394. DOI: 10.1112/plms/pdq026. URL: https://doi.org/10.1112%2Fplms%2Fpdq026.

[2] Marc Bezem et al. *Symmetry*. https://github.com/UniMath/SymmetryBook. Commit: 506a252. Mar. 5, 2022.

[3] Pierre Deligne. "Le symbole modéré". fr. In: *Publications Mathématiques de l'IHÉS* 74 (1991), pp. 147–181. URL: http://dml.mathdoc.fr/item/PMIHES_1991__73__147_0.

[4] Jean Giraud. *Cohomologie non abelienne*. Springer Berlin, Heidelberg, 1971.

[5] Martin Hofmann and Thomas Streicher. "The Groupoid Interpretation of Type Theory". In: (Apr. 2002).

[6] Clara Löh. *Group Cohomology, Sommersemester 2019 Universit ät Regensburg*. URL: https://loeh.app.uni-regensburg.de/teaching/grouphom_ss19/lecture_notes.pdf.

[7] Per Martin-Löf. "An Intuitionistic Theory of Types: Predicative Part". In: *Logic Colloquium '73*. Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Elsevier, 1975, pp. 73–118. DOI: https://doi.org/10.1016/S0049-237X(08)71945-1.

[8] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book/, 2013. URL: http://homotopytypetheory.org/book/.