SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Computing Cohomology with Cubical Agda

av

Axel Ljungström

2020 - No M6

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET, 106 91 STOCKHOLM

Computing Cohomology with Cubical Agda

Axel Ljungström

Självständigt arbete i matematik 30 högskolepoäng, avancerad nivå

Handledare: Guillaume Brunerie, Anders Mortberg

2020

Abstract

Cubical Agda is a proof assistant for Cubical Type Theory (CuTT), a recent flavour of Homotopy Type Theory (HoTT), which gives a constructive interpretation of univalence. The goal of this thesis is to characterise, formally in Cubical Agda, the zeroth, first and second cohomology groups with integer coefficients of the torus and of the wedge sum of a sphere and two circles. We first introduce type theory, HoTT, CuTT and Cubical Agda. We then work our way up to cohomology, defined in terms of Eilenberg-MacLane spaces, and the Mayer-Vietoris sequence. Finally, we characterise some of the cohomology groups of the unit type, n-spheres, 0-connected types and wedge sums. Using these results, we characterise the cohomology groups in question.

Acknowledgements

I would like to thank my supervisors Guillaume Brunerie and Anders Mörtberg for all of their support and for the countless hours they have put into this project. I would also like to thank Alexander Berglund for his insightful comments on the first draft of this thesis.

Contents

1 Introduction **3**

2 Preliminaries **4**
- 2.1 Type Theory . . . 4
- 2.2 Homotopy Type Theory . . . 8
- 2.3 Cubical Type Theory . . . 15

3 Cubical Agda **16**
- 3.1 The Interval Type . . . 16
- 3.2 Transport . . . 18
- 3.3 Paths . . . 19
- 3.4 Interlude: Cubes and Fillers . . . 20
- 3.5 Partial Elements . . . 21
- 3.6 Homogeneous Composition . . . 22
- 3.7 Glue . . . 27
- 3.8 Higher inductive types . . . 27

4 Cohomology in Homotopy Type Theory **28**
- 4.1 Preliminaries . . . 28
  - 4.1.1 Foundations . . . 28
  - 4.1.2 Spheres and Pushouts . . . 33
  - 4.1.3 Homotopies and Loop Spaces . . . 35
  - 4.1.4 n-Types and Truncations . . . 36
  - 4.1.5 Connected Functions and Types . . . 41
  - 4.1.6 Freudenthal Suspension Theorem . . . 45
  - 4.1.7 The Hopf Fibration . . . 51
  - 4.1.8 Loop Spaces over $\mathbb{S}^1$ . . . 51
  - 4.1.9 Groups . . . 54
- 4.2 Cohomology with Coefficients in $\mathbb{Z}$ – Definition and Group Structure 57
- 4.3 The Mayer-Vietoris Sequence . . . 60
- 4.4 Characterisations of Cohomology Groups . . . 65
  - 4.4.1 The Unit Type . . . 65
  - 4.4.2 Spheres . . . 66
  - 4.4.3 Wedges of Spheres . . . 70
  - 4.4.4 The Torus . . . 71

5 Implementation in Cubical Agda **76**
- 5.1 Formalisation . . . 76
- 5.2 Computations . . . 78

6 Future work **79**

1 Introduction

Homotopy Type Theory (HoTT) is a constructive foundation of mathematics combining Per Martin-Löf's intensional type theory and homotopy theory. HoTT can be characterised by the following four ideas:

1. A type A corresponds to a topological space $\hat{A}$ (up to homotopy equivalence). Every term a of type A, denoted $a : A$, corresponds to a point $\hat{a} \in \hat{A}$.
2. Equalities correspond to paths; that is, given a type A and terms $a, b : A$, the identity type $a \equiv b$ is interpreted as the space of paths between a and b.
3. An equivalence (bijection) between two types is a homotopy equivalence between their corresponding spaces.
4. If two types are equivalent, then there is a witness of their equality. In other words, isomorphic structures are *the same*. This is known as *(Voevodsky's) univalence principle* [14].

Cubical Type Theory (CuTT) is slight variation of HoTT where paths (equalities) are defined as functions from a primitive unit interval rather than inductively. The main advantage of CuTT is that it gives a constructive interpretation of univalence. This gives rise to the possibility of actually using formalised proofs in CuTT to make explicit calculations which would get stuck on every application of univalence in standard HoTT.

In this thesis, we define cohomology with coefficients in $\mathbb{Z}$ in CuTT and characterise the first three cohomology groups for the torus, $\mathbb{T}^2$, and the wedge of two circles and a sphere, $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$. These spaces are interesting because they admit isomorphic cohomology groups but different cup products. The idea is that if we can define these things in a proof assistant, then we can actually make explicit computations with addition (in the cohomology groups) and cup products. In this thesis, we carry out the first step in this project by defining cohomology groups and characterising the first three cohomology groups of $\mathbb{T}^2$ and $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ formally in Cubical Agda, the cubical extension of the proof assistant Agda.

Section 2 contains a brief introduction to type theory, HoTT and CuTT. This section is aimed towards general mathematicians rather than logicians and computer scientists. Jargon from mathematical logic and category theory is avoided to an extent as great as possible.

Section 3 contains a brief introduction to Cubical Agda. This section also explains some of the fundamentals of CuTT. It also introduces some elementary lemmas which will be used later on.

Section 4 section contains the mathematics leading up to and including the definition of the group structure on cohomology groups and the characterisations of $H^n(\mathbb{T}^2)$ and $H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$ for $n = 0, 1, 2$. All proofs have been verified formally in the cubical library [3], either by myself or other contributors.

Section 5 contains some comments on the main part of the project, i.e. the formalisation in Cubical Agda.

2 Preliminaries

In this section we introduce type theory, HoTT and CuTT. The mathematics in this thesis is written in *informal* (homotopy) type theory, in the same way as classical mathematics is written in the language of informal set theory. This introduction is brief and by no means exhaustive. For a more detailed exposition, the reader is strongly encouraged to read the first two chapters of the *Homotopy Type Theory Book* [14].

2.1 Type Theory

Type theory is a formal language tracing back to the foundational crisis in the early 20th century. Originally outlined by Russell in [13] as a solution to Russell's paradox, it was first seriously developed and studied by Curry. Together with Howard, Curry found a correspondence between propositions in *intuitionistic* logic (roughly speaking, classical logic without the law of the excluded middle) and types, often referred to as the *Curry-Howard Correspondence* or the *Propositions-as-types interpretation of type theory* [10]. The essence of this correspondence is the following; for every proposition P in intuitionistic logic, there is a corresponding type P' such that constructing a term of P' is the same thing as giving a proof of P. This gives a two-fold interpretation of type theory. On the one hand, types fill the same function as sets do in classical mathematics. For instance, we have the type $\mathbb{N}$ of natural numbers, the type $\mathbb{Z}$ of integers, and so on. On the other hand, types correspond to mathematical propositions. For the most commonly occurring types, we have the following correspondences.

Type Theory	**Logic**	**Set Theory**
$a : A$	A holds (with a as proof/witness)	$a \in A$
$B : A \to \mathsf{Type}$	Predicate $B(a)$	Family of sets B_a
$A \to B$	If A, then B	$\{f \mid f : A \to B\}$
$(a : A) \to B(a)$	For all a, $B(a)$ holds	$\Pi_{a \in A} B_a$
$\sum_{a:A} B(a)$	There exists an a such that $B(a)$ holds	$\sqcup_{a \in A} B_a$
$\equiv_A$	$=$ (equality)	$\{(a, a) \mid a \in A\}$

The last three rows in the rightmost column are mainly included for completeness and can safely be ignored. We also introduce the empty type $\bot$ and the unit type $\top$. Following notation similar to that of first-order logic, we negate a statement (type) A by constructing a function from A into $\bot$ – that is, by constructing a term of the type $A \to \bot$. We give some examples of familiar mathematical statements and notation translated into type theory.

Mathematical statement	**Translation**
$f : \mathbb{N} \to \mathbb{N}$	$f : \mathbb{N} \to \mathbb{N}$
For all natural numbers n, m, if $m = -n$, then $n + m = 0$	$(n, m : \mathbb{N}) \to n \equiv -m \to n + m \equiv 0$
If n is a natural number and $n \neq 0$, then there is a natural number m s.t. $m + 1 \equiv n$	$(n : \mathbb{N}) \to (n \equiv 0 \to \bot)$ $\to \sum_{m:\mathbb{N}} m + 1 \equiv n$

Figure 1: Translations

My hope here is that the reader notes the similarity between the above and the usual renderings of mathematical statements into first-order logic. So far, the above translations just look like regular first-order logic with "$n \in \mathbb{N}$" replaced by "$n : \mathbb{N}$", "$\forall\,(a \in A)\,(\dots)$" replaced by "$(a : A) \to \dots$" and "$\exists\,(a \in A)\,(\dots)$" replaced by "$\sum_{a:A} \dots$". Although this is fairly close to the truth, there are some subtle differences. A particularly interesting difference is type theory's treatment of proofs as regular mathematical objects. For a concrete example, compare the first and second cells in the second column in Figure 1. In the first cell we write $f : \mathbb{N} \to \mathbb{N}$ to say that f is a term of the type $\mathbb{N} \to \mathbb{N}$ – that is, f is a function from $\mathbb{N}$ to $\mathbb{N}$. We turn to the second cell. The type here seems vastly different from the type $\mathbb{N} \to \mathbb{N}$. Indeed, this type represents a mathematical statement, whereas the type $\mathbb{N} \to \mathbb{N}$ represents a certain class of functions. However, a proof of this statement constitutes in constructing an element $p : (n, m : \mathbb{N}) \to n \equiv -m \to n + m \equiv 0$. This now looks much more similar to the statement in the first cell; hence, we should be able to consider p as a function, just as we did with f. There is an obvious way of doing this: p is a function that takes as input two natural numbers n and m together with a proof that $n \equiv -m$ and returns as output a proof that $n + m \equiv 0$. In other words, p is a function in the same sense as f. On the other hand, f is a proof (roughly speaking, a proof the statement "if there is a natural number, then there is a natural number") in the same way as p is. The conclusion is that type theory does not allow for a clear-cut distinction between mathematical objects and proofs *about* mathematical objects. In fact, these two notions coincide completely. In one sense, every construction in type theory is a proof. Thus, we are not always (only) interested in *what* is proved, but rather *how* something is proved. For this reason, we say that intensional type theory is *proof relevant*. We will see how this comes into play in Section 2.2.

We now know that types can essentially be treated either as sets or as propositions, and we have a rough idea of how to translate statements from standard mathematics into type theory. We also know that we can form new types (such as $\sum_{a:A} B(a)$ and $a \equiv b$) from simpler types and their terms. However, we have not said anything about (1) how to introduce new elementary types and (2) how to actually construct elements of a given type. Most types in type theory are inductively defined. What this means is that we introduce a type by inductively defining its terms. For instance, we introduce the natural

numbers inductively by

$$0 : \mathbb{N}$$
$$\mathsf{suc} : \mathbb{N} \to \mathbb{N}$$

That is, "0 is natural number and for every natural number n, there is a successor of n, i.e. $\mathsf{suc}(n)$". Hence, there are only two ways of constructing an element $a : \mathbb{N}$; either we give 0 or we give $\mathsf{suc}(n)$ for some other natural number n. The inductive types also come equipped with induction principles. For instance, if we want to prove a statement $(n : \mathbb{N}) \to B(n)$, it is enough to construct a term $b_0 : B(0)$ and, given a term $b_n : B(n)$, a term $b_{n+1} : B(n+1)$.

Other inductively defined types of particular importance are the unit type $\top$, the empty type $\bot$ and the identity type $a \equiv b$. Set theoretically, the unit type $\top$ corresponds to the class of singleton sets. Logically, it corresponds to verum. The unit type is defined by a single constructor:

$$* : \top$$

The empty type, $\bot$, corresponds set-theoretically to the empty set and logically to falsum. It has no constructors. The identity type relative to a type A and a term $a : A$, as treated in Per Martin-Löf type theory, is inductively defined as a function $A \to \mathsf{Type}$ with a sole constructor: $\mathsf{refl}_a : a \equiv a$. We often omit the subscript from refl_a when a is clear from context. Here refl_a corresponds to the trivial equality between a and itself. As the definition is inductive, it makes sense to assume the following induction principle. Suppose we have a dependent type $B : (a, b : A) \to a \equiv b \to \mathsf{Type}$ and we want to prove that $B(a, b, p)$ holds for all $a, b : A$ and $p : a \equiv b$. Then it is enough to prove $B(a, a, \mathsf{refl}_a)$. In other words, it is enough to replace b by a and assume that the identity between a and b is trivial. It is important to point out that the induction principle cannot be used to directly prove $B(a, a, q)$ for some $q : a \equiv a$. In Section 2.2, we will get some topological intuition for why this is the case. Instead, in order to apply the induction principle we have to prove the stronger statement that $B(a, b, p)$ holds for all $b : A$ and $q : a \equiv b$. We will use *path induction* to refer to this induction principle. This name will make more sense after seeing the topological interpretation of type theory in Section 2.2.

In addition to "$\equiv$", we will use the following symbols for other kinds of equality

($:\equiv$) We write $a :\equiv b$ in order to define a new term a as b.

($:=$) We write $a := b$ when a and b are definitionally/judgementally equal. That is, when a and b reduce to the exact same term (i.e., they are syntactically identical). This happens for instance when we have previously defined $a :\equiv b$, but can also occur in other cases.

($=$) The regular equality symbol will only be used informally for meta-mathematical reasoning. For instance, we may sometimes talk about "the case $n = 0$" in an inductive argument.

For examples of *path induction* in action, consider the proofs of the following propositions.

Proposition 2.1 (Transitivity of identity/composition of equalities)**.** *Let A be a type with terms $x, y, z : A$. For all terms $p : x \equiv y$ and $q : y \equiv z$, there is a term $p \cdot q : x \equiv z$.*

Proof. We are given identities $p : x \equiv y$ and $q : y \equiv z$. By path induction (twice), we may replace y and z by x and assume that p and q are just refl_x. We are now to construct a term $\mathsf{refl}_x \cdot \mathsf{refl}_x : x \equiv x$. We simply define

$$\mathsf{refl}_x \cdot \mathsf{refl}_x :\equiv \mathsf{refl}_x$$

and we are done. □

Proposition 2.2 (refl is a right-unit)**.** *Let A be a type with terms $x, y : A$ and let $p : x \equiv y$. Then $p \equiv p \cdot \mathsf{refl}_y$.*

Proof. By path induction, we may replace y by x and assume that p is just refl_x. We are now to show that

$$\mathsf{refl}_x \equiv \mathsf{refl}_x \cdot \mathsf{refl}_x$$

But this is precisely how we defined composition of identities. Thus, the statement holds (by $\mathsf{refl}_{\mathsf{refl}_x}$). □

Proposition 2.3 (Function application preserves identities)**.** *Let A and B be types with a function $f : A \to B$. Let $x, y : A$. There is a function*

$$\mathsf{cong}_f : x \equiv y \to f(x) \equiv f(y)$$

Proof. Let $x, y : A$ and $p : x \equiv y$. We need to construct $\mathsf{cong}_f(p) : f(x) \equiv f(y)$. By path induction, it suffices to construct the term when p is refl_x. We then let

$$\mathsf{cong}_f(\mathsf{refl}_x) :\equiv \mathsf{refl}_{f(x)}$$

and we are done. □

Proposition 2.4 (Transportation along equalities)**.** *Let A and B be types. Then there is a function*

$$\mathsf{transport} : A \equiv B \to A \to B$$

Proof. Let $p : A \equiv B$. By path induction, we may assume that p is refl_A. We may now define the function as

$$\mathsf{transport}(\mathsf{refl}_A, a) :\equiv a$$

□

We also have the following equivalent definition of $\mathsf{transport}$.

Proposition 2.5. *Let A be a type and let $B : A \to \mathsf{Type}$ be a dependent type. Let $x, y : A$. We then have a function*

$$\mathsf{transport}^B : x \equiv y \to B(x) \to B(y)$$

Proof. Given an identity $p : x \equiv y$, we also get an identity $p' : B(x) \equiv B(y)$. We may thus define

$$\mathsf{transport}^B(p, _) :\equiv \mathsf{transport}(p', _)$$

□

An important note is that all types are terms of higher types. For instance, the type $\mathbb{N}$ is of type Type_0, where Type_0 is the type (also know as the *universe*) of elementary types. In its turn, Type_0 is of type Type_1. In general, we have $\mathsf{Type}_n : \mathsf{Type}_{n+1}$. Consequently, many theorems we prove about terms of (often elementary) types hold for the types themselves. For instance, by letting A be Type_0 in the statement of Proposition 2.1, we see that the theorem also holds for identities between types. For the remainder of this thesis, we omit the subscript and simple write Type.

We have now answered (1). We have yet to answer (2). That is, we have yet to answer how to actually prove statements in type theory (i.e. how to construct terms of types). It is clear how to construct a term of an inductively defined type A; either we give a term $a : A$ such that a is one of the constructors from the inductive definition of A (e.g. $\mathsf{suc}\ 0$ for $\mathbb{N}$), or we give a term $f(b) : A$ where B is a type, $f : B \to A$ is a previously defined function and $b : B$. We have seen examples of both ways in the proofs of the previous propositions. Constructing a term of type $\Sigma_{a:A} B(a)$ consists in giving a term (a, b) such that $a : A$ and $b : B(a)$. Finally, constructing a term of type $(a : A) \to B(a)$ consists in giving a term of type $B(a)$, given some $a : A$. Note that $A \to B$ is just the special case of this type when B does not depend on A. We will often use λ-notation in order to construct terms of function types. This is merely a way of writing functions in a compact way. For instance, a function $f : (x : A) \to B(x)$ will often be written as $\lambda\, x \,.\, f(x)$. Here, $\lambda\, x \,.\, f(x)$ simply means "given some $(a : A)$, replace every occurrence of x in $f(x)$ by a". For instance, the function $f(x) = x^2$ can be written as $\lambda\, x \,.\, x^2$. Thus, λ-notation is simply a way of introducing functions without giving them an explicit name (e.g. "f", or similar).

2.2 Homotopy Type Theory

While the interpretation of types as sets is useful for intuition, it is perhaps not completely precise. In standard mathematics, sets are essentially structureless. This is not the case for types in intensional type theory. In particular, given a type A and terms $a, b : A$, we may look at the identity type $a \equiv b$. From the perspective of classical mathematics, this type should be either empty or contain precisely one element – either two objects are equal, or they are not. However, since intensional type theory is proof relevant, there can be several

distinct terms of this type. Proposition 2.1 tells us that there is also a binary operation $_ \cdot _ : a \equiv b \to b \equiv c \to a \equiv c$ for all $a, b, c : A$. Furthermore, we will see that this operation satisfies the groupoid laws. This allows us to interpret $_ \cdot _$ as composition of paths in a topological space. Under this interpretation, we interpret A not as a type, but as a topological space and $a \equiv b$ as the space of paths from a to b. In particular, $a \equiv a$ can be interpreted as the loop space over a. Hence we may still think of types as sets, but now also with the structure of topological spaces. This now gives a new interpretation of types.

Type Theory	**Topology**
$a : A$	A point a in the topological space A
$A \to B$	Function space
$B : A \to \mathsf{Type}$	Fibration $B(x)$ over A
$(a : A) \to B(a)$	Space of sections
$\sum_{a:A} B(a)$	Total space
$a \equiv b$	Space of paths from a to b (in some space A)

We will continue to work under this topological interpretation of type theory; in particular, we will often use "points" to refer to terms, "fibrations" to refer to dependent types, "paths" to refer to identities and "path composition" to refer to composition of identities (transitivity).

HoTT also allows for higher inductive types (HITs). The idea is that we may define a type not only by describing the type's constructors, but also by describing how to identify its terms. That is, we define a HIT A by inductively giving its constructors $c_i : A$ and then giving paths $p_{a,j}$ for some of its constructors $a : A$. For instance, $\mathbb{S}^1$ is a HIT defined by

- A base point $\mathsf{base} : \mathbb{S}^1$
- A path $\mathsf{loop} : \mathsf{base} \equiv \mathsf{base}$

Note how this corresponds precisely to the interpretation from classical algebraic topology of $\mathbb{S}^1$ as a cell complex $e^0 \cup e^1$. Here, the 0-dimensional cell e^0 corresponds to base and the 1-dimensional cell (line segment) e^1 corresponds to the path loop. In general, any finite CW complex can be described by a HIT [14]. These constructions make HoTT suitable for doing mathematics *synthetically*. That is, roughly speaking, mathematics in which algebraic structure is the defining feature of mathematical objects, rather than which points they are composed of[1]. There are two major advantages of the synthetic approach to mathematics. First, it often makes mathematical intuition precise. For instance, claims such as "a continuous function from the circle into a space A consists of a point $a \in A$ and a loop over a" hold by definition in HoTT, due to the synthetic definition the circle. Indeed, the induction principle for $\mathbb{S}^1$ says that in order to

[1]For an early example of synthetic mathematics, see e.g. Euclid's foundations of geometry [8].

construct an element $f : (x : \mathbb{S}^1) \to B(x)$, it is enough to construct an element $f(\mathsf{base}) : B(\mathsf{base})$ and an element $f(\mathsf{loop}) : f(\mathsf{base}) \equiv f(\mathsf{base})$.

The second advantage is that CW complexes defined as HITs are defined by a finite number of constructors, rather than by an infinite number of points. For instance, when we talk about points on the circle, we can really only talk about the base point or the loop. Intuitively, there are other points than base on the circle, but these are inaccessible to us. Thus HITs are finite objects, which makes them suitable for computer implementations.

The final and perhaps most important part of HoTT is Voevodsky's *univalence principle*. As is common in constructive systems, it is not possible to talk about discontinuous functions in a meaningful way. HoTT follows this tradition in that it interprets every function $f : A \to B$ as a continuous function between spaces. Thus, if a function $f : A \to B$ has an inverse $f^{-1} : B \to A$, then also f^{-1} is continuous. Hence, invertible maps correspond precisely to homotopy equivalences. Since homotopy equivalent spaces are "the same" in some structural sense, it makes sense to equate them, from our synthetic point of view. This is precisely what the univalence principle does. There are several (provably equivalent) ways to express homotopy equivalence, denoted $A \simeq B$. We define equivalences in terms of contractibility of fibres.

Definition 2.6 (Contractibility)**.** A type A is said to be *contractible* if we have a term of the following type

$$\mathsf{isContr}(A) :\equiv \sum_{a_0 : A} ((a : A) \to a \equiv a_0)$$

If we have a pair $(a_0, p) : \mathsf{isContr}(A)$, then we refer to a_0 as the *centre of contraction*.

Definition 2.7 (Equivalence)**.** We say that two types A and B are *equivalent*, denoted $A \simeq B$, if there is a map $f : A \to B$ such that the fibres of f are contractible. Formally, we define this by

$$A \simeq B :\equiv \sum_{f : A \to B} \mathsf{isEquiv}(f)$$

where

$$\mathsf{isEquiv}(f) :\equiv ((b : B) \to \mathsf{isContr}(\mathsf{fib}_f(b)))$$

and

$$\mathsf{fib}_f(b) :\equiv \sum_{a : A} (f(a) \equiv b)$$

We also say that two types A and B are *quasi-equivalent* if there are maps $f : A \to B$, and $g : B \to A$ such that these maps cancel out. Formally, a quasi-equivalence between A and B is a term of the following type

$$\sum_{f : A \to B} \mathsf{isQuasiEquiv}(f)$$

where

$$\mathsf{isQuasiEquiv}(f) :\equiv \sum_{g:B\to A} ((a:A) \to g(f(a)) \equiv a) \times ((b:B) \to f(g(b)) \equiv b)$$

Quasi-equivalences can be proved to induce equivalences [14]. We will therefore use these definitions interchangeably and without comment. We will also, with some abuse of notation, sometimes write $f : A \simeq B$ when only referring to the function $f : A \to B$ in isolation of the proof that it is an equivalence.

Proposition 2.8. *Let A and B be types with $p : A \equiv B$ and let*

$$\mathsf{idToEquiv}(p) :\equiv \mathsf{transport}(p, _\,)$$

The map $\mathsf{idToEquiv}(p) : A \to B$ is an equivalence.

Proof. The map's inverse is given by $\mathsf{transport}(p^{-1}, _\,)$. The fact that these maps cancel out follows immediately by path induction on p. □

We can now state the univalence principle.

Axiom 2.9 (Univalence)**.** *Let A and B be types. Then there is a function*

$$\mathsf{ua} : A \simeq B \to A \equiv B$$

Furthermore, ua *is an equivalence with* $\mathsf{idToEquiv}$ *as its inverse.*

We give some intuition for why $\mathsf{idToEquiv}$ is assumed to be the inverse of ua. Suppose we have an equivalence $f : A \simeq B$. Applying univalence, we get a path $\mathsf{ua}(f) : A \equiv B$. We consider the function $\mathsf{transport}(\mathsf{ua}(f), _\,) : A \to B$. This function now transports a point $a : A$ to a point b over the path $\mathsf{ua}(f)$. This corresponds to the notion of a *dependent path* from a to b. If A and B are not definitionally the same type, we cannot claim that $a \equiv b$, since this is not well-typed. We now have the next best thing – we can can claim that a and b are equal with respect to our path $\mathsf{ua}(f)$. Intuitively, this path should take elements $a : A$ to $f(a) : B$, since the path is defined in terms of f. In this sense, $\mathsf{transport}$ should also undo the application of ua and simply give f back. This is precisely what is captured by assuming that $\mathsf{idToEquiv}$ is the inverse of ua.

Univalence motivates our perhaps slightly unfamiliar choice of definition of equivalences. The notion of a quasi-equivalence appears closer to what we mean by a homotopy equivalence in classical mathematics. Indeed, this is often the definition we have in mind when actually doing mathematics in HoTT. It turns out, however, that $\mathsf{isQuasiEquiv}$ is not suitable for the definition of equivalences. The problem comes from proof-relevance. Suppose we were to use $\mathsf{isQuasiEquiv}$ in our definition of equivalences and suppose we have a map $f : A \to B$ with $p, q : \mathsf{isQuasiEquiv}(f)$. In this case, we get two elements $(f, p), (f, q) : A \simeq B$. Inverses are unique, so both p and q will point to two functions g and g' such that $g \equiv g'$. However, p and q could point to completely different proofs of, for instance, the facts that $g(f(a)) \equiv a$ and $g'(f(a)) \equiv a$ for every $a : A$. In

order for p and q to be equal, we would have to be able to transport the first proof to the second over the path $g \equiv g'$. This is not possible in general [14]. In particular then, the equivalence $(A \simeq B) \simeq (A \equiv B)$ means that (f, p) and (f, q) could correspond to two different paths $A \equiv B$. The intuition is, however, that f uniquely determines a path $A \equiv B$ and hence the role of p and q becomes unclear. Our definition of equivalences in terms of $\mathsf{isEquiv}(f)$ avoids this problem. Indeed, we can prove that for any $p, q : \mathsf{isEquiv}(f)$, we have that $p \equiv q$. In this sense, our definition of equivalences is actually closer to the definition of (homotopy) equivalences in classical mathematics – it is logically equivalent to the definition in terms of quasi-equivalences in the sense that we have

$$\mathsf{isEquiv}(f) \to \mathsf{isQuasiEquiv}(f)$$
$$\mathsf{isQuasiEquiv}(f) \to \mathsf{isEquiv}(f)$$

but it is, like the classical definition, proof-irrelevant. Indeed, we can easily define a bijection between $(A \simeq B)$ and $\sum_{f:A\to B} \|\mathsf{isQuasiEquiv}(f)\|_{-1}$, where $\|_\|_{-1}$ is an operation on types which essentially turns them into proof-irrelevant logical propositions (see Definition 4.12). This makes them correspond closer to propositions in classical mathematics.

The problem with translating the definition of equivalences in HoTT directly into classical mathematics is that, classically, contractibility of fibres does not ensure that the point of contraction in each fibre varies continuously. Consequently, we do not necessarily have that our inverse function is continuous. The key point here is that the statement "for every $b : B$, we have a point $a : A$", which is the first component of contractibility of fibres, is realised by a function $B \to A$ and, like any function in HoTT, is interpreted to be continuous.

Univalence often comes into play when applying the *encode-decode method*, which is a common proof method for characterising path spaces in HoTT. The simplest version of the encode-decode proof can be outlined as follows, following [14].

Proposition 2.10. *Let (A, a) be a pointed type (i.e. suppose there is some fixed $a : A$) and consider a fibration* $\mathsf{code} : A \to \mathsf{Type}$. *Suppose that for every $x : A$, we have a function*

$$\mathsf{decode}_x : \mathsf{code}(x) \to a \equiv x$$

with some $c : \mathsf{code}(a)$ such that

$$\mathsf{decode}_a(c) \equiv \mathsf{refl}$$

and for all $y : \mathsf{code}(a)$, we have

$$\mathsf{transport}^{\mathsf{code}}(\mathsf{decode}_a(y), c) \equiv y.$$

Then $(a \equiv a) \simeq \mathsf{code}(a)$.

Proof. For each $x : A$, we define $\mathsf{encode}_x : (a \equiv x) \to \mathsf{code}(x)$ by

$$\mathsf{encode}_x(p) :\equiv \mathsf{transport}^{\mathsf{code}}(p, c)$$

We show that encode_a and decode_a cancel out. One direction is given by assumption. We now show that

$$\mathsf{decode}_x(\mathsf{encode}_x(p)) \equiv p$$

for every $x : A$ and $p : a \equiv x$. By path induction, it suffices to show that

$$\mathsf{decode}_a(\mathsf{encode}_a(\mathsf{refl})) \equiv \mathsf{refl}$$

By assumption, $\mathsf{decode}_a(c) \equiv \mathsf{refl}$, so it suffices to show that

$$\mathsf{encode}_a(\mathsf{refl}) \equiv c$$

But in this case, we have

$$\mathsf{encode}_a(\mathsf{refl}) := \mathsf{transport}^{\mathsf{code}}(\mathsf{refl}, c) := c$$

Hence, the maps cancel out and we get that $\mathsf{code}(a) \simeq (a \equiv a)$. ☐

The idea is that since loop spaces are hard to characterise as they do not allow us to use path induction, we want to prove something more general (in general, by means of constructing a fibration), which does allow us to use path induction. This method also works for other constructions relying on loop spaces (e.g. truncations of loop spaces, and similar constructions).

In general, the hardest part of designing an encode-decode proof is how to define the fibration code. If we want to construct it for a HIT A, we need to construct $\mathsf{code}(a_i)$ for the base points $a_i : A$, by sending them to appropriate types. We then need to, for every higher constructor $p : x \equiv y$, construct a path $\mathsf{code}(x) \equiv \mathsf{code}(y)$, i.e. a path between types. In general, it is not merely the existence of such a path that is interesting, but also how the path is designed. Designing such a path is precisely what the univalence principle allows us to do. In order to see how this works in practice, we revisit an old classic [14]. We sketch the proof.

Theorem 2.11. $(\mathsf{base} \equiv_{\mathbb{S}^1} \mathsf{base}) \simeq \mathbb{Z}$

Proof. We apply the encode-decode method. The first step is to define our fibration $\mathsf{code} : \mathbb{S}^1 \to \mathsf{Type}$. Since we are trying to prove that $(\mathsf{base} \equiv_{\mathbb{S}^1} \mathsf{base}) \simeq \mathbb{Z}$, we have to let $\mathsf{code}(\mathsf{base}) :\equiv \mathbb{Z}$. We now need to decide where to send loop. Formally speaking, we need to define $p : \mathbb{Z} \equiv \mathbb{Z}$ and let

$$\mathsf{cong}_{\mathsf{code}}(\mathsf{loop}) :\equiv p$$

Some readers may know the proof from traditional algebraic topology. The idea there is to consider the helix projecting down on $\mathbb{S}^1$, such that one step up in the

helix corresponds to one loop from the base point to itself. We want to mimic this idea and interpret our fibration code as the helix. We begin by defining a function $f : \mathbb{Z} \to \mathbb{Z}$ by

$$f(x) \equiv x + 1$$

It is easy to prove that f is an equivalence $\mathbb{Z} \simeq \mathbb{Z}$. Thus, by univalence, we get the path $\mathsf{ua}(f) : \mathbb{Z} \equiv \mathbb{Z}$ with the property that transportation along this path is just f. We thus let

$$\mathsf{cong}_{\mathsf{code}}(\mathsf{loop}) :\equiv \mathsf{ua}(f)$$

By this, we have captured the fundamental idea of the traditional proof – looping once is the same thing as going up one step in the helix (i.e. adding 1 to the current position). We define $\mathsf{encode}_x(p)$ for every $p : \mathsf{base} \equiv x$. By path induction, it suffices to define it when $x := \mathsf{base}$ and p is refl. We let

$$\mathsf{encode}_{\mathsf{base}}(\mathsf{refl}) \equiv 0$$

We define $\mathsf{decode}_x(y) : \mathsf{base} \equiv x$ for every $x : \mathbb{S}^1$ and $y : \mathsf{code}(x)$. We induct on x. For the base point we let

$$\begin{aligned} \mathsf{decode}_{\mathsf{base}} &: \mathbb{Z} \to \mathsf{base} \equiv \mathsf{base} \\ \mathsf{decode}_{\mathsf{base}} &:\equiv \lambda\, n\, .\, \mathsf{loop}^n \end{aligned}$$

where loop^{-} is inductively defined by

$$\begin{aligned} \mathsf{loop}^0 &:\equiv \mathsf{refl} \\ \mathsf{loop}^{\mathsf{suc}(n)} &:\equiv \mathsf{loop}^n \cdot \mathsf{loop} \\ \mathsf{loop}^{-\mathsf{suc}(n)} &:\equiv \mathsf{loop}^{-n} \cdot \mathsf{loop}^{-1} \end{aligned}$$

For $\mathsf{cong}(\mathsf{decode}_)(\mathsf{loop})$, we need to construct a dependent path between the base constructor and itself along loop. That is, we need to show that

$$\mathsf{transport}^{(x:\mathbb{S}^1)\to\mathsf{code}(x)\to\mathsf{base}\equiv x}(\mathsf{loop}, \lambda\, n\, .\, \mathsf{loop}^n) \equiv \lambda\, n\, .\, \mathsf{loop}^n$$

We have

$$\begin{aligned} &\mathsf{transport}^{(x:\mathbb{S}^1)\to\mathsf{code}(x)\to\mathsf{base}\equiv x}(\mathsf{loop}, \lambda\, n\, .\, \mathsf{loop}^n) \\ &\equiv \mathsf{transport}^{(x:\mathbb{S}^1)\to\mathsf{base}\equiv x}(\mathsf{loop}) \circ (\lambda\, n\, .\, \mathsf{loop}^n) \circ \mathsf{transport}^{(x:\mathbb{S}^1)\to\mathsf{code}(x)}(\mathsf{loop}^{-1}) \\ &\equiv (\lambda\, p\, .\, p \cdot \mathsf{loop}) \circ (\lambda\, n\, .\, \mathsf{loop}^n) \circ (\lambda\, n\, .\, (n-1)) \\ &\equiv \lambda\, n\, .\, \mathsf{loop}^{n-1} \cdot \mathsf{loop} \\ &\equiv \lambda\, n\, .\, \mathsf{loop}^n \end{aligned}$$

We do not verify these identities here. The first equality is an easy lemma about the behaviour of $\mathsf{transport}$ for fibrations with $\to$ as its main connective.

The first part of the second equality is also an easy lemma on the behaviour of $\mathsf{transport}$ for transports into path spaces. The second part of the second equality comes from our definition of code using univalence. The last equality follows by definition. We have thus defined our decode function. We now prove that these maps cancel out. Let $(x : \mathbb{S}^1)$. We show that

$$\mathsf{encode}_x(\mathsf{decode}_x(n)) \equiv n$$

for every $n : \mathbb{Z}$. We can show that $\mathbb{Z}$ is a *set*, i.e. has trivial path spaces. We will see later (see Lemma 4.39) that it therefore suffices to assume that x is base. Proving that

$$\mathsf{encode}_{\mathsf{base}}(\mathsf{decode}_{\mathsf{base}}(n)) \equiv n$$

is now easy by induction on n. We do not show the path algebra here (for the details, see the full proof in [14]). The key component is that transportation along $\mathsf{ua}(f)$ is the same as application of f.

We finally prove that

$$\mathsf{decode}_x(\mathsf{encode}_x(p)) \equiv (p)$$

for every $p : \mathsf{base} \equiv x$. This is where the essential step of the encode-decode method comes into play. By path induction, it suffices to show that

$$\mathsf{decode}_{\mathsf{base}}(\mathsf{encode}_{\mathsf{base}}(\mathsf{refl})) \equiv \mathsf{refl}$$

This now holds definitionally, since $\mathsf{encode}_{\mathsf{base}}(\mathsf{refl}) := 0$ and $\mathsf{decode}_{\mathsf{base}}(0) := \mathsf{loop}^0 := \mathsf{refl}$. □

2.3 Cubical Type Theory

CuTT is a slight alteration of HoTT which replaces the inductive definition of the identity type with a definition in terms of an interval type I, thereby mimicking the definition of paths that we are used to from classical homotopy theory. The unit interval has two constructors corresponding to the end points of the unit interval, $i_0, i_1 : I$. We say that two points $x, y : A$ are equal if there is a function $p : I \to A$ such that $p(i_0) := x$ and $p(i_1) := y$. In other words, we are to provide a term p such that $p(i_0)$ reduces to x and $p(i_1)$ reduces to y. How to work in CuTT will be explained in more detail in Section 3. In that section, we use the notation of Cubical Agda. The reader who is only interested in the basics of CuTT and not in Cubical Agda, should read the section as follows.

Notation	Read as
$\forall\{\ell\}$	Ignore
$\mathsf{Type}\,\ell$	Type
$\{a : A\}$	$(a : A)$
$\mathsf{fun}\, x_1\, x_2, \ldots, x_n$	$\mathsf{fun}(x_1, \ldots, x_n)$
$\mathsf{fun}\,\{x = y\}$	$\mathsf{fun}(y)$
$\lambda\, x \to a(x)$	$\lambda\, x \,.\, a(x)$

Furthermore, subsections 3.5 and 3.7 can safely be ignored.

The main motivation behind CuTT is that it gives a constructive interpretation of the univalence principle. A problem with standard HoTT is that univalence is something that is merely assumed as an axiom. One of the fundamental ideas of constructive mathematics is that every proof corresponds to an algorithm. In particular, we just saw an example of how to prove that $(\mathsf{base} \equiv \mathsf{base}) \simeq \mathbb{Z}$. This gives us a map $f : \mathbb{Z} \to (\mathsf{base} \equiv \mathsf{base})$ and an inverse $f^{-1} : \mathsf{base} \equiv \mathsf{base} \to \mathbb{Z}$. Since the f and f^{-1} were defined constructively, we should be able to use these functions for explicit calculations. For instance, it is provable that $f^{-1}(f(1) \cdot f(1)) \equiv 2$. Due to the functions constructive definitions, this fact should be trivial (i.e. we should not have to prove anything). If we formalise the constructions in a proof assistant, $f^{-1}(f(1) \cdot f(1))$ should simply reduce to 2. This is not the case in standard HoTT. This is because we have merely assumed univalence and not proved it. Hence it gives us no computational information. This means that if we define our function in terms of `ua`, then the function does not reduce. In CuTT, things are set up in a way that both allows us to prove univalence and such that the computation rules for `ua` are automatic. This will be discussed in more detail in the following section. For a more in depth description of CuTT, see e.g. [12].

3 Cubical Agda

The cubical mode of Agda allows us to utilise methods from CuTT. Most importantly, the Cubical mode comes with support for HITs and a primitive `Glue` type which makes possible for a constructive interpretation of univalence. The purpose of this section is to provide the general Agda user with a walkthrough of the fundamentals of Agda's cubical mode while simultaneously introducing the basics of CuTT. We do this following the official documentation of Agda 2.6.1. [1], expanding on the more technical sections.

3.1 The Interval Type

In HoTT, we interpret an equality of two terms x and y of type A as a path between two points x' and y' in a topological space A'. Classically, we would define this path as a continuous function $p : [0,1] \to A'$ such that $p(0) = x'$ and $p(1) = y'$. In order to give this a (cubical) type theoretic interpretation, we introduce a primitive interval type `I`. With it comes two terms `i0`, `i1` : `I`, corresponding respectively to the end points 0 and 1 of $[0,1]$. Hence, to to construct a path `x ≡ y`, we need to construct a function `u : I → A` such that `u i0 := x` and `u i1 := y`. For instance, we can define the constant path `refl` as follows.

```
refl : ∀ {ℓ} {A : Type ℓ} {x : A} → x ≡ x
refl {x = x} = λ i → x
```

By using curly brackets in the type of refl above, we tell Agda that these arguments are to be inferred implicitly. That is, if we have some type A of universe level ℓ and an element x : A, then we do not need to write refl ℓ A x in order to prove $x \equiv x$. We only need to write refl, and Agda will (most often) figure out what x, A and ℓ should be from the context. In the second row, when defining refl, we do not need to write out ℓ, A and x explicitly to give the definition. In this case, however, we want to mention the element x : A. Thus we write {x = x} to tell Agda that we momentarily want to refer to the implicit x explicitly as x.

Note that this differs significantly from the definition of the identity type in standard HoTT, where the identity type is inductively defined with refl's being its sole constructor. Hence, path induction is not automatic in Cubical Agda.

Already at this stage, before introducing any more machinery, we can prove function extensionality in only one line.

```
funExt : ∀ {ℓ ℓ'} {A : Set ℓ} {B : A → Set ℓ'}
         {f g : (x : A) → B x} → ((x : A) → f x ≡ g x) →
         f ≡ g
funExt p i x = p x i
```

This is an exceptionally short proof compared to the one from standard HoTT, where function extensionality is a relatively advanced theorem following from univalence.

The interval type also comes equipped with operations $\sim : I \to I$, $\wedge : I \to I \to I$ and $\vee : I \to I \to I$. Here $\sim$ can be thought of as inverting interval, thereby sending i0 to i1 and vice versa. For the sake of intuition we temporarily assume a total ordering on I with $i0 < i1$. The operation $i \wedge j$ then picks out the smallest term out of i and j whereas $i \vee j$ picks out the largest term. Importantly, when at least one of the arguments of these operations is either i0 or i1, these terms will reduce. For instance, we have

$$\begin{aligned} \sim i0 &:= i1 \\ i0 \wedge j &:= i0 \\ i1 \wedge j &:= j \\ i0 \vee j &:= j \\ i1 \vee j &:= i1 \end{aligned}$$

For a more comprehensive list of the Boolean laws, see [1].

We give two examples. The simplest application of $\sim$ is path reversal; given a path $p : x \equiv y$, we wish to construct a path $y \equiv x$. We have, by definition of p, that $p\, i0 := x$ and $p\, i1 = y$. Hence we just need to give the term $q :\equiv \lambda i \to p(\sim i)$. We have

$$q\, i0 := p\, (\sim i0) := p\, i1 := y$$

and

$$\mathtt{q\,i1} \coloneqq \mathtt{p(\sim i1)} \coloneqq \mathtt{p\,i0} \coloneqq \mathtt{x}.$$

Hence $\mathtt{q} : \mathtt{y} \equiv \mathtt{x}$. In Agda, we write this as follows.

```
sym : ∀ {ℓ} {A : Type ℓ} {x y : A} → x ≡ y → y ≡ x
sym p i = p (~ i)
```

Moreover, we can construct a path between x and any point on our path $\mathtt{p} : \mathtt{x} \equiv \mathtt{y}$.

```
pathPointsEq : ∀ {ℓ} {A : Type ℓ} {x y : A} (p : x ≡ y) (i : I) → x ≡ p i
pathPointsEq p i j = p (i ∧ j)
```

The above theorem appears to clash with function extensionality. Indeed, if we are given another path $\mathtt{q} : \mathtt{x} \equiv \mathtt{y}$, the above theorem implies (assuming path composition, which we will prove later) that for every point $\mathtt{i} : \mathtt{I}$, we have that $\mathtt{p\,i} \equiv \mathtt{q\,i}$. Since p and q both are functions from I into A, function extensionality appears to imply that $\mathtt{p} \equiv \mathtt{q}$. In other words, we seem to have a proof of axiom K (uniqueness of paths), which is well-known to be inconsistent with univalence. The key point here is that paths are not merely functions, but functions with constraints. Thus trying to give a similar proof for function extensionality for paths will not work, since the terms will not reduce. Also, note that we may not apply our current version of function extensionality in this case, because I does not belong to $\mathtt{Type}\,\ell$ for any universe level ℓ. Rather, it belongs to a separate universe $\mathtt{Set}\omega$, which thus prohibits us from letting the domain type A (in the above proof of function extensionality) be I.

3.2 Transport

Suppose we are given a path $\mathtt{p} : \mathtt{A} \equiv \mathtt{B}$ between types and a term $\mathtt{a} : \mathtt{A}$. Since A and B are equal, the construction of a should naturally give rise to a term $\mathtt{b} : \mathtt{B}$, in accordance to our path p. In standard HoTT, we prove this by constructing the `transport` function using path induction. Due to the non-inductive definition of paths in Cubical Agda however, we have to introduce `transport` differently. We introduce a primitive generalised transport.

```
transp : ∀ {ℓ} (A : (i : I) → Type ℓ) (i : I) (a : A i0) → A i1
```

This looks just like the transport function we are used to from standard HoTT, apart from the mysterious additional $\mathtt{i} : \mathtt{I}$. One can think of it as a way of guaranteeing that the usual transport laws which we are used to from standard HoTT hold. In particular, it is a way of ensuring that $\mathtt{transport\,refl} \equiv \lambda\,\mathtt{x} \to \mathtt{x}$, hence circumventing the problem that `transport` cannot be defined by path induction. Indeed, the primitive `transp` is set up so that $\mathtt{transp\,A\,i1\,a} \coloneqq \mathtt{a}$. The condition for this to be well-typed is of course that A is constant when $\mathtt{i} \coloneqq \mathtt{i1}$. Now, when $\mathtt{i} \coloneqq \mathtt{i0}$, this condition is vacuously satisfied, and hence we can define transport as follows.

```
transport : ∀ {ℓ} {A B : Type ℓ} → A ≡ B → A → B
transport p a = transp (λ i → p i) i0 a
```

As promised, we get that `transport refl` is equal to the identity function.

```
transportRefl : ∀ {ℓ} {A : Type ℓ} (x : A) → transport refl x ≡ x
transportRefl {A = A} x i = transp (λ i → A) i x
```

An important consequence of this lemma is the `J`-rule, i.e. path induction.

```
J : ∀ {ℓ ℓ'} {A : Type ℓ} {x y : A} (P : ∀ y → x ≡ y → Type ℓ')
   (d : P x refl) (p : x ≡ y) → P y p
J P d p = transport (λ i → P (p i) (λ j → p (i ∧ j))) d
```

The `J`-rule differs from its traditional version in that a term defined by `J A f` does not reduce to `f` when applied to refl. As with `transport`, this is a consequence of the fact that `J` itself is not defined by path induction. Nevertheless, we can show that they are equal up to a path.

```
JRefl : ∀{ℓ ℓ'} {A : Set ℓ} {x y : A} (P : ∀ y → x ≡ y → Type ℓ')
      (d : P x refl) → J P d refl ≡ d
JRefl P d = transportRefl d
```

3.3 Paths

As usual, we think of `transport` as constructing dependent paths. In Cubical Agda, however, dependent paths are also described explicitly by the following primitive.

```
PathP : ∀ {ℓ} (A : I → Type ℓ) → A i0 → A i1 → Type ℓ
```

A hidden constraint is that in order to construct a dependent path `p : PathP A x y`, this path must satisfy `p i0 := x` and `p i1 := y`.

Until now, we have not actually defined the identity type ≡. We define it as the special case of `PathP` when the type path is constant.

```
_≡_ : ∀ {ℓ} {A : Type ℓ} → A → A → Type ℓ
_≡_ {A = A} x y = PathP (λ i → A) x y
```

Sometimes it is useful to specify the type when proving an equality. Therefore, we also introduce an equivalent definition of ≡, with the second argument made explicit.

```
Path : ∀ {ℓ} (A : Type ℓ) → A → A → Type ℓ
Path A x y = PathP (λ i → A) x y
```

The intuitive correspondence between `transport` and the construction of dependent paths can be made explicit. In particular, we can show that if a point `a` is transported to `b` along a path `P`, then this induces a dependent path. For this, however, we shall need some more machinery, and so we postpone it for now.

3.4 Interlude: Cubes and Fillers

In order to introduce the remaining primitives in a pedagogical way, we should first make some remarks on cubes and their relation to $\texttt{I}$. Recall that we can interpret a path $\texttt{p} : \texttt{x} \equiv \texttt{y}$ as a function $\texttt{f}_1 : \texttt{I} \to \texttt{A}$ such that $\texttt{f}_1\,\texttt{i0} \coloneqq \texttt{x}$ and $\texttt{f}_1\,\texttt{i1} \coloneqq \texttt{y}$. If we are given another path $\texttt{q} : \texttt{x} \equiv \texttt{y}$, then a path $\texttt{p} \equiv \texttt{q}$ corresponds to a function $\texttt{f}_2 : \texttt{I} \to \texttt{I} \to \texttt{A}$ such that $\texttt{f}_2\,\texttt{i0} \coloneqq \texttt{p}$ and $\texttt{f}_2\,\texttt{i1} \coloneqq \texttt{q}$. By currying/schönfinkelisation, $\texttt{f}_2$ can thus be seen as a function from a square into $\texttt{A}$, with restrictions on the values which $\texttt{f}_2$ assumes on the boundaries. Generalising this, an n-dimensional path $\texttt{p}$ over $\texttt{A}$ corresponds to a function $\texttt{f}_\texttt{n}$ from an n-dimensional cube into $\texttt{A}$ with appropriate constraints on the function's behaviour on the boundary the cube. For better intuition, we represent $\texttt{p}$ by an n-dimensional cube, with its boundaries labelled by the values assumed by $\texttt{f}_\texttt{n}$.

For an example of how these notions will come into play, assume that we have $\texttt{p} : \texttt{x} \equiv \texttt{y}$, $\texttt{q} : \texttt{y} \equiv \texttt{z}$, $\texttt{r} : \texttt{x} \equiv \texttt{w}$, $\texttt{s} : \texttt{w} \equiv \texttt{z}$. We can represent these paths by an unfilled square.

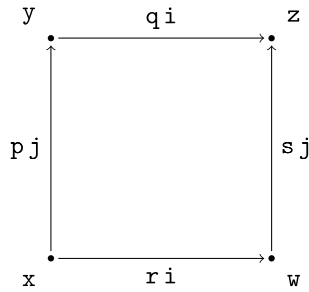


Figure 2: An unfilled square of paths

Suppose further that we wish to prove some statement relating the four paths, for instance the fact that there is a dependent path over $\lambda\,\texttt{j} \to (\texttt{p}\,\texttt{j} \equiv \texttt{s}\,\texttt{j})$ from $\texttt{r}$ to $\texttt{q}$. This is the same as defining a function from $\texttt{I} \times \texttt{I}$ into $\texttt{A}$ which reduces to the given terms on the boundary of the square. From a classical point of view, we are given a function $f_\partial : \partial([0,1] \times [0,1]) \to A$ which we need to continuously extend to function $f : [0,1] \times [0,1] \to A$, hence, in a way, filling the above square. The topological intuition is then that by the contractibility of $[0,1] \times [0,1]$, a filler allows us match up the boundaries. This intuition carries over into CuTT; we are to construct a term $\texttt{P} : \texttt{I} \to \texttt{I} \to \texttt{A}$ – often referred to as a *filler* – such that $\texttt{P}\,\texttt{i0}\,\texttt{j} \coloneqq \texttt{p}\,\texttt{j}$, $\texttt{P}\,\texttt{i}\,\texttt{i0} \coloneqq \texttt{r}\,\texttt{i}$, and so on.

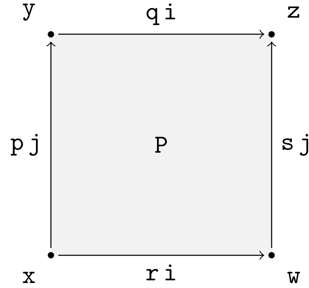


Figure 3: A filled square of paths

Sections 3.5 and 3.6 concern the construction of these cubes and their fillers.

3.5 Partial Elements

The type of representations in Section 3.4 lie at the heart of CuTT. In order to actually be able to construct them, however, we need a way of constructing them in an effective way. Suppose we are given an n-cube with some faces missing – that is, a *partial* n-cube. Intuitively, we should be able to add the missing sides to form a full n-cube. In other words, we should be able to generate n-cubes from partial cubes. Cubical Agda allows us to do precisely this. First, however, we must describe the partial n-cubes, referred to here as *partial elements*.

Cubical Agda comes with a primitive predicate `IsOne` on `I`. For `i : I`, the idea is that `IsOne i` corresponds to the constraint `i = i1`. This is necessary in order to be able to talk about constraints on `i`, since we cannot use our usual identity type to talk about identities of terms of `I`. We also need to introduce a primitive reflexivity proof `1=1 : IsOne i1`. Following the notation in [1], we use Greek letters when referring to terms of `I` that are to be though of as in the domain of `IsOne`.

We can now introduce introduce partial elements `Partial` ϕ `A` as a special case of the type `IsOne` $\phi \to$ `A`. For completeness, we also include a dependent version `PartialP`.

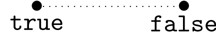
```
Partial : ∀{ℓ} (i : I) (A : Type ℓ) → Setω

PartialP : ∀ {ℓ}(φ : I) → Partial φ (Type ℓ) → Setω
```

To better understand this, we should give an example of how partial elements are introduced. We use the same example as in [1], since it is by far the easiest non-trivial one. We define a partial boolean by means of a pattern matching lambda (see [2]).

```
PartialBool : (i : I) → Partial (i ∨ ~ i) Bool
PartialBool i = λ { (i = i0) → true
                  ; (i = i1) → false }
```

We can think of this as a boolean that is able to change its values at the boundary points i0 and i1. This may look like a path true ≡ false, but naturally this would be absurd and is not the right way to interpret it. The intuition is that nothing really happens unless i := i0 or i := i1. We can see this as constructing an unfilled line (i.e. an unfilled 1-cube or, alternatively, an unfilled degenerate n-cube).



In order to do anything interesting with these constructions, we need to establish a connection between our general types and partial elements. For this we use cubical subtypes, another primitive of Cubical Agda. We introduce them as follows.

```
_[_↦_] : ∀ {ℓ} (A : Type ℓ) (ϕ : I) (u : Partial ϕ A) → Setω
A [ ϕ ↦ u ] = Sub A ϕ u
```

The idea here is to, in a way, generalise the notion of a type – given a term u : A, a term v : A [ϕ → u] by definition has the property that it reduces to u when IsOne ϕ is satisfied. The term u can itself be interpreted in such a way. Indeed, we trivially have that u := u whenever IsOne ϕ is satisfied. For this reason, we introduce the following primitive in order to turn u into a term of type A [ϕ → u].

```
inS : ∀ {ℓ} {A : Type ℓ} {ϕ : I} (u : A) → A [ ϕ ↦ (λ _ → u) ]
```

All that this does is to add the information that u reduces when IsOne ϕ holds. Hence, we should be able to transform the term back to u. We add such an operation too.

```
outS : ∀ {ℓ} {A : Type ℓ} {ϕ : I} {u : Partial ϕ A} → A [ ϕ ↦ u ] → A
```

The operations inS and outS satisfy the expected definitional equalities. For instance, we have outS (inS a) := a.

3.6 Homogeneous Composition

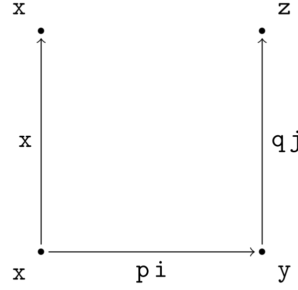
Recall, our goal is to be able to construct cubes and their fillings out of partial cubes. In the previous section, we made sense of partial cubes by introducing partial elements to the machinery. We can now finish the job by introducing homogeneous composition, a generalisation of path composition. This allows us to compose cubes. It is given by the following primitive.

```
hcomp : ∀ {ℓ} {A : Type ℓ} {ϕ : I} (u : (i : I) → Partial ϕ A) (a : A) → A
```

Partial elements specify only the sides of cubes but leave the bottom and lid open. The idea behind hcomp is that if we call it on a partial element and a path

making up the bottom of the corresponding partial cube, then it will compute the lid. More formally: given a partial element u, hcomp $\{\phi = \phi\}$ u is a function only accepting arguments u_0 such that u i0 and u_0 agree on ϕ. Practically, hcomp is far less difficult to work with than what this formal explanation makes it sound like. Consider the following example.

Example 3.1 (Binary path composition)**.** We are finally ready to prove path composition. Let p : x ≡ y, q : y ≡ z. The goal is to construct a path p · q : x ≡ z. Consider the following partial cube.



All we need to do to complete the square is to apply hcomp.

```
_·_ : ∀ {ℓ} {A : Type ℓ} {x y z : A} → x ≡ y → y ≡ z → x ≡ z
_·_ {x = x} p q i = hcomp (λ j → λ { (i = i0) → x
                                   ; (i = i1) → q j })
                        (p i)
```

Note that this is not the only way of doing it. By constructing the partial element differently, we could define path composition by

```
_□_ : ∀ {ℓ} {A : Type ℓ} {x y z : A} → x ≡ y → y ≡ z → x ≡ z
_□_ {z = z} p q i = hcomp (λ j → λ { (i = i0) → p (~ j)
                                   ; (i = i1) → z })
                        (q i)
```

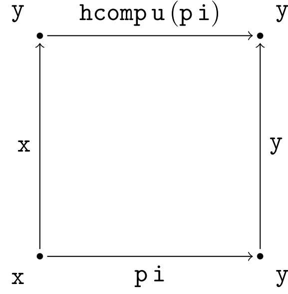
These two definitions are however provably equal up to a path.

We are left to define fillers of cubes. For this, we do not need a new primitive – a filling operation can be defined in terms of hcomp.

```
hfill : ∀ {ℓ} {A : Type ℓ} {φ : I}
        (u : ∀ i → Partial φ A)
        (u0 : A [ φ ↦ u i0 ])
        (i : I) → A
hfill {φ = φ} u u0 i = hcomp (λ j → λ { (φ = i1) → u (i ∧ j) 1=1
                                      ; (i = i0) → outS u0 })
                           (outS u0)
```

We note that the above term reduces to u_0 when i := i0 and to hcomp u u0 when i := i1, thus constructing a filler of the corresponding cube.

Example 3.2. With `hfill`, we can prove that for any path $\texttt{p} : \texttt{x} \equiv \texttt{y}$, we have that $\texttt{p} \cdot \texttt{refl} \equiv \texttt{p}$. In other words, by definition of path composition, we wish to fill the following cube.

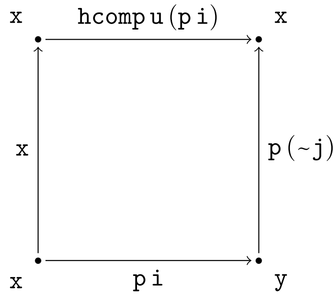


Here `u` is the partial element from the definition of path composition, i.e. s.t. `hcomp u (p i)` is $\texttt{p} \cdot \mathsf{refl}$. This is precisely the type of situation that `hfill` is defined to handle.

```
rUnit : ∀ {ℓ} {A : Type ℓ} {x y : A} (p : x ≡ y) → p · refl ≡ p
rUnit {x = x} {y = y} p j i = hfill (λ _ → λ { (i = i0) → x
                                             ; (i = i1) → y })
                                    (inS (p i))
                                    (~ j)
```

Note that the application of `inS` that will usually be needed in proofs using `hfill` merely is there for technical reasons. It can thus often be ignored – it is its argument that carries the interesting information.

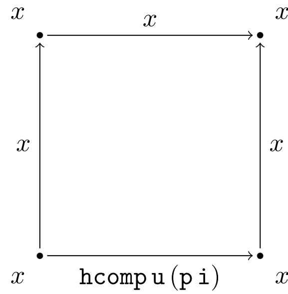
Example 3.3. So far, we have only seen examples of squares. Unfortunately, HoTT is rarely kind enough to allow us to stay in only two dimensions. Let us consider an example that forces us up one dimension. Given a path $\texttt{p} : \texttt{x} \equiv \texttt{y}$, we wish to prove that $\texttt{p} \cdot \mathsf{sym}\,\texttt{p} \equiv \texttt{refl}$. We know that `p · sym p` corresponds to the top of the following square



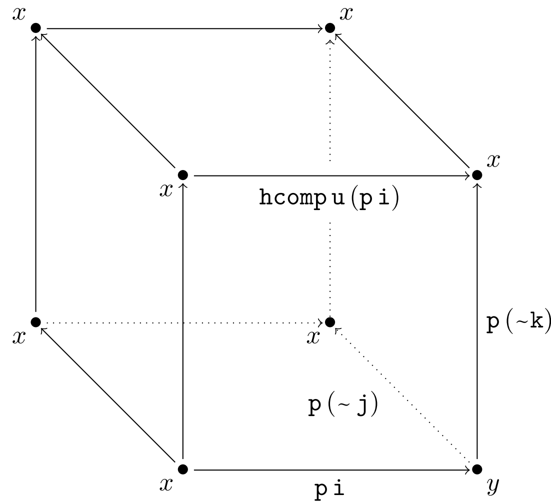
where

$$\begin{aligned}\mathtt{u\,j} :\equiv \lambda\,\{\,&(\mathtt{i} = \mathtt{i0}) \to \mathtt{x}\\ ;\,&(\mathtt{i} = \mathtt{i1}) \to \mathtt{p}\,(\mathtt{\sim j})\,\}\end{aligned}$$

However, filling the above cube will not give us the result we want – a filler would only construct a term of type $\mathtt{PathP}\,(\lambda\,\mathtt{j} \to \mathtt{x} \equiv \mathtt{p}\,(\mathtt{\sim j}))\,\mathtt{p}\,(\mathtt{p} \cdot \mathtt{sym}\,\mathtt{p})$. Rather, we want a filler of the following square.



The idea is now to build a skeleton of a cube from our two squares. The following cube does the job. Those sides without labels are to be interpreted as `refl`.



If we can find a filler of this cube, we are done – restricting this filler to the case when $\mathtt{k} := \mathtt{i1}$, we have the lid of the cube, and hence a path from $\mathtt{hcomp\,u\,(p\,i)}$ to its opposite side $\mathtt{refl\,i}$ along the sides, both being $\mathtt{refl\,j}$. In other words, we get a path $\mathtt{p} \cdot \mathtt{sym\,p} \equiv \mathtt{refl}$. The backside and the left-hand side of the cube are trivial. The front of the cube is just the filler of path composition. Thus, we only need to describe the bottom and the right-hand

side of the cube. For the bottom, we need to give a term btm : A that respects the following constraints:

$$
\begin{aligned}
\texttt{i} = \texttt{i0} &\vdash \texttt{x} \\
\texttt{i} = \texttt{i1} &\vdash \texttt{p}\,(\sim \texttt{j}) \\
\texttt{j} = \texttt{i0} &\vdash \texttt{p}\,\texttt{i} \\
\texttt{j} = \texttt{i1} &\vdash \texttt{x}
\end{aligned}
$$

We let btm :≡ p (i ∧ ~ j). Similarly, letting rSide :≡ p (~ k ∧ ~j) will do the trick for the right-hand side. Thus, we can give the filler of the cube.

```
rCancelFiller : ∀ {ℓ} {A : Type ℓ} {x y : A} (p : x ≡ y) →
                I → I → I → A
rCancelFiller {x = x} {y = y} p i j k =
                   hfill (λ k → λ { (i = i0) → x
                                   ; (i = i1) → p (~ k ∧ ~ j)
                                   ; (j = i0) → hfill (λ r → λ{ (i = i0) → x
                                                              ; (i = i1) → p (~ r) })
                                                      (inS (p i))
                                                      k
                                   ; (j = i1) → x })
                         (inS (p (i ∧ ~ j)))
                         j
```

In fact, we do not need to specify the case j = i0 in this case. Now, by restricting ourselves to its lid, we are done.

```
rCancel : ∀ {ℓ} {A : Type ℓ} {x y : A} (p : x ≡ y) → p · sym p ≡ refl
rCancel p i j = rCancelFiller p j i i1
```

The careful reader might have noted that this proof is somewhat roundabout. Indeed, it is possible to construct the lid immediately using a simple application of hcomp.

```
rCancel' : ∀ {ℓ} {A : Type ℓ} {x y : A} (p : x ≡ y) → p · sym p ≡ refl
rCancel' {x = x} p j i = hcomp (λ k → λ { (i = i0) → x ;
                                          (i = i1) → p (~ k ∧ ~ j) ;
                                          (j = i1) → x} )
                               (p (i ∧ ~ j))
```

This applies to many theorems in the cubical library. There is however value in proving it using fillers. Seeing rCancel as a special case of a more general filler, we also gain information about its relation to the other sides of the cube. Such information can be crucial for path algebraic proofs involving rCancel.

3.7 Glue

Cubical Agda includes a primitive type Glue with which we can give a constructive interpretation of the univalence axiom.

```
Glue : ∀ {ℓ ℓ'} (A : Type ℓ) {φ : I}
       → (Te : Partial φ (Σ[ T ∈ Type ℓ' ] T ≃ A))
       → Type ℓ'
```

We use a partial element in the definition of the Glue type to generalise to higher dimensional cubes. However, for our purposes here, it is enough to look at the simple case which gives univalence. Suppose we have two equivalences $\mathtt{e}_1 : \mathtt{A} \simeq \mathtt{B}$ and $\mathtt{e}_2 : \mathtt{C} \simeq \mathtt{B}$. The intuition is now that we can use e_1 to glue A to one side of B, and e_2 to glue C to the other side. Since Glue will construct an element of type Type _, and both A and C are of this type, our goal is to construct an expression on the form $\mathtt{p} : \lambda\, \mathtt{i} \to \mathtt{Glue\,X\,u}$ where $\mathtt{p\,i0} \coloneqq A$ and $\mathtt{p\,i1} \coloneqq \mathtt{C}$. Since we are to glue things onto B, we let $\mathtt{X} :\equiv \mathtt{B}$. Now, to define the partial expression u we simply need to respects the following constraints:

$$\mathtt{i} = \mathtt{i0} \vdash (\mathtt{A}\,,\, \mathtt{B} \simeq \mathtt{A})$$
$$\mathtt{i} = \mathtt{i1} \vdash (\mathtt{C}\,,\, \mathtt{B} \simeq \mathtt{C})$$

We are given these facts by assumption and so we are done. Typing it into Agda, we get

```
uaGen : ∀ {ℓ} {A B C : Type ℓ} → A ≃ B → C ≃ B → A ≡ C
uaGen {A = A} {B = B} {C = C} e₁ e₂ i =
                Glue B λ { (i = i0) → A , e₁
                         ; (i = i1) → C , e₂ }
```

As a special case when $\mathtt{C} :\equiv \mathtt{B}$ and e_2 is the identity function, we get univalence:

```
ua : ∀ {ℓ} {A B : Type ℓ} → A ≃ B → A ≡ B
ua {A = A} {B = B} e i = uaGen e  ((λ x → x) , idEquiv)
  where
  idEquiv : isEquiv (λ x → x)
  equiv-proof (idEquiv) y = (y , refl) , λ z i → z .snd (~ i)
                                       , λ j → z .snd (~ i ∨ j)
```

3.8 Higher inductive types

Finally, Cubical Agda supports HITs. For instance, we can define the circle $\mathtt{S}^1$ as follows.

```
data S¹ : Type₀ where
  base : S¹
  loop : base ≡ base
```

If we wish to define a term of type $(\mathtt{x} : \mathtt{S}^1) \to \mathtt{A\,x}$, we can pattern match on x.

Example 3.4. For a very simple example, we define the following function by induction.

$$\text{loop-sq} : S^1 \to S^1$$
$$\text{loop-sq base} = \text{base}$$
$$\text{loop-sq (loop } i) = (\text{loop} \cdot \text{loop})\ i$$

For $\mathtt{i : I}$, we have $\mathtt{loop\,i : S^1}$, so the last row is well-typed. However, when specifying which term to send `loop i` to, we have constraints on which element in $\mathtt{S^1}$ we can choose. Since `loop` is a path from `base` to itself, we have to make sure that `loop-sq` sends `loop i` to an element u_0 s.t. $\mathtt{u_0\,i0} :\equiv \mathtt{u_0\,i1} :\equiv \mathtt{loop\text{-}sq\,base}$. This is satisfied by $(\mathtt{loop} \cdot \mathtt{loop})\,\mathtt{i}$.

4 Cohomology in Homotopy Type Theory

This section contains the mathematical part of the project. The goal is to define cohomology with coefficients in $\mathbb{Z}$ and compute the first three cohomology groups of $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ and $\mathbb{T}^2$. We will do this in the style of informal type theory, following the style in [14] and, to some extent, in [5].

4.1 Preliminaries

In this subsection, we present the preliminary theory required for cohomology. Most of it comes from [14] or [6]. Exceptions are the proof of Theorem 4.10, which is a slight alteration of the proof in [5], Theorem 4.35, which is a simplification on the proof in [14] due to Evan Cavallo (see Cubical.Homotopy.Freudenthal in [4]) and Sections 4.1.8 and 4.1.9.

4.1.1 Foundations

We dedicate this subsection to the most elementary facts in CuTT.

Proposition 4.1. *Let A be a type with $x, y : A$ and $p : x \equiv y$. We have the following paths.*

(i) $\left(p^{-1}\right)^{-1} \equiv p$

(ii) $\mathsf{rCancel}(p) : p \cdot p^{-1} \equiv \mathsf{refl}$

(iii) $\mathsf{lCancel}(p) : p^{-1} \cdot p \equiv \mathsf{refl}$

(iv) $\mathsf{rUnit}(p) : p \cdot \mathsf{refl} \equiv p$

(v) $\mathsf{lUnit}(p) : \mathsf{refl} \cdot p \equiv p$

Proof.

(i) holds definitionally.

(ii) proved in Example 3.3.

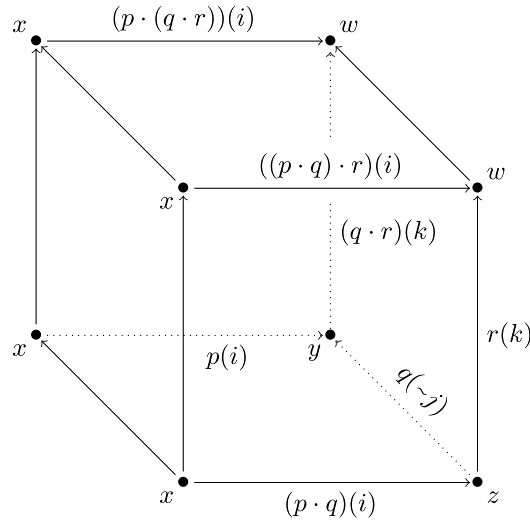
(iii) follows from (i) and (ii).

(iv) proved in Example 3.2.

(v) proved using a construction analogous to that in the proof of (iii). □

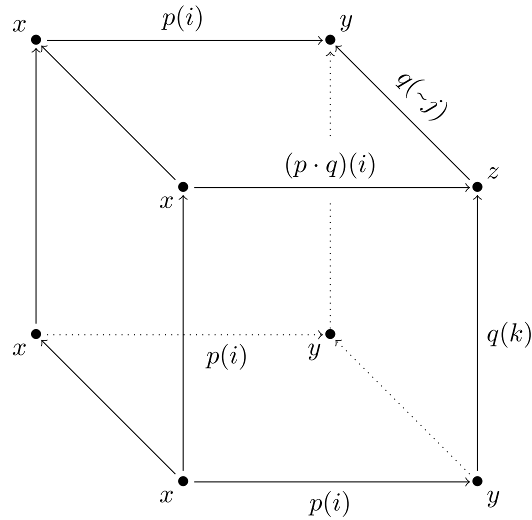
Proposition 4.2 (Associativity of path composition)**.** *Let A be a type with $x, y, z, w : A$. For any paths $p : x \equiv y$, $q : y \equiv z$, $r : z \equiv w$, we have $(p \cdot q) \cdot r \equiv p \cdot (q \cdot r)$.*

Proof. The path we wish to construct corresponds to the lid of the following cube.



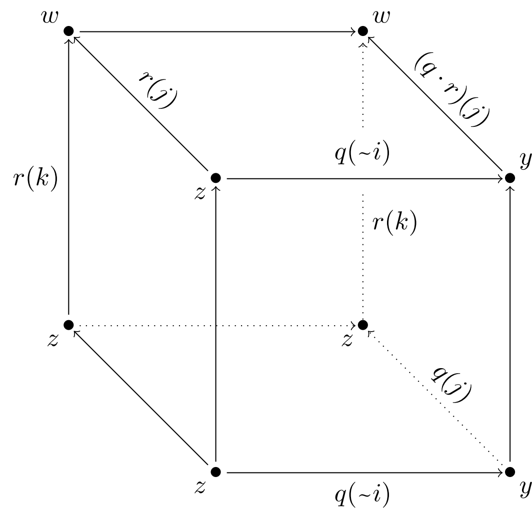
It suffices to specify fillers of the bottom and side squares. The front and back are given by the filler of path composition. The left-hand side is trivial. Thus it suffices to give fillers for the bottom and the right-hand side of the cube.

In order to construct the bottom filler, we construct the following cube.



We are interested in the lid. Again, we are done if we can give fillers for the bottom and the sides of this cube. The bottom, the back and the left-hand side are trivial. The front is just the filler of path composition. Hence it remains to give a filler for the right-hand side. The term $q(\sim j \wedge k)$ is easily verified to satisfy the boundary conditions.

In order to construct the right-hand side filler for the original cube, we note that it corresponds to the lid of the following cube.



We are done if we can construct the lid of this box. Again, it is enough to specify the bottom and the sides. The back and front are trivial and we obtain

the right-hand side by the filler of path composition. For the bottom, the term $q(\sim i \vee j)$ satisfies the constraints. For the left-hand side we give $r(j \wedge k)$.

Thus, we have constructed all sides and the bottom of the original cube and by this we have also constructed its lid. This completes the proof. □

From now on, we simply write $p \cdot q \cdot r$ instead of $p \cdot (q \cdot r)$. Note that there are different proofs of associativity of path composition, neither of which definitionally equal. One example is the proof in the cubical library. There is also a more direct proof relying more on Lemma 4.1. Using path induction, one can also easily construct several other examples.

Proposition 4.3. *Let A and B be types with $x, y, z, w : A$ and suppose there are functions $f : A \to B$ and $g : A \to A \to B$. Then there are functions*

$$\mathsf{cong}_f : x \equiv y \to f(x) \equiv f(x)$$
$$\mathsf{cong}^2_g : x \equiv y \to z \equiv w \to g(x, z) \equiv g(y, w)$$

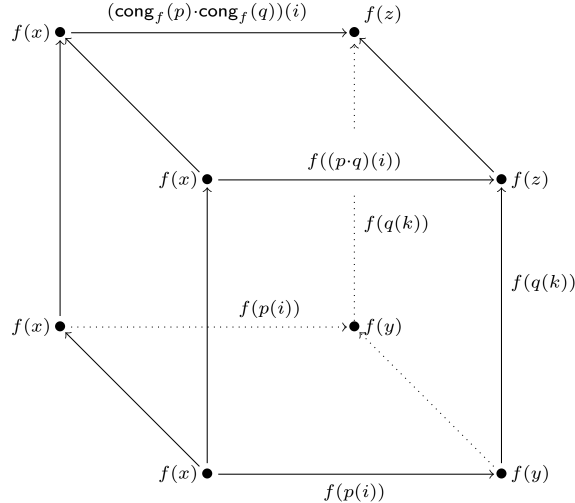
Furthermore, for any $p : x \equiv y$, $q : y \equiv z$ and $r : z \equiv w$, these functions satisfy the following functoriality principles.

$$\mathsf{cong}_f(p \cdot q) \equiv \mathsf{cong}_f(p) \cdot \mathsf{cong}_f(q) \tag{1}$$
$$\mathsf{cong}^2_g(p, r) \equiv \mathsf{cong}_{g(_,z)}(p) \cdot \mathsf{cong}_{g(y,_)}(r) \tag{2}$$

Proof. We have already seen how cong_f is constructed in Section 3. cong^2 is only a generalised version of cong. Let $p : x \equiv y$, $r : z \equiv w$ and $i : I$. We define cong^2_g by

$$(\mathsf{cong}^2_g(p, r))(i) :\equiv g(p(i), r(i))$$

We now prove (1). The statement corresponds to the lid of the following cube.



The fillers of the left-hand side, right-hand side and bottom are all trivial. The back is just the filler of path composition. The front is just f applied to the filler of path composition.

The second identity, (2), is proved in a similar manner. $\square$

We can use cong to define the inverse of function extensionality, as defined in Section 3.

Proposition 4.4. *For all $g, f : A \to B$ and paths $p : A \equiv B$ and $a : A$, there is a path*

$$\mathsf{funExt}^{-1}(p, a) : f(a) \equiv g(a)$$

Furthermore, funExt^{-1} *and* funExt *cancel out.*

Proof. The function is simply defined by

$$\mathsf{funExt}^{-1}(p, a) :\equiv \mathsf{cong}_{\lambda\, h\, .\, h(a)}(p)$$

The fact that funExt^{-1} and funExt cancel out holds definitionally. $\square$

For completeness, we include the following proposition. All it says is that a two-argument function $(f(a))(b)$ can be rewritten as a single-argument function $f(a, b)$. The careful reader will have noted that this has been used implicitly until now. We will continue to use it implicitly.

Proposition 4.5 (Currying)**.** *For all types A and fibrations $A \to \mathsf{Type}$ and $C : (a : A) \to B(a) \to \mathsf{Type}$, we have an equivalence*

$$\mathsf{curry} : \left(p : \sum_{a:A} B(a) \to C(\mathsf{fst}(p), \mathsf{snd}(p))\right) \simeq ((a : A) \to (b : B(a)) \to C(x, y))$$

As mentioned in Section 3, we also have a $\mathsf{transport}$ function in CuTT. Given a path $P : A \equiv B$ between two types A and B and an element $a : A$, we use the notation

$$\mathsf{transport}^{P}(a) : B$$

We sometimes refer to $\mathsf{transport}$ as subst in the case where P is a fibration combined with a homogeneous path. That is, if we have a fibration $B : A \to \mathsf{Type}$, two elements $a_1, a_2 : A$, a path $p : a_1 \equiv a_2$ and an element $b_1 : B(a_1)$, we write

$$\mathsf{subst}^{P}(p, b_1) : B(a_2)$$

instead of

$$\mathsf{transport}^{\lambda\, i\, .\, B(p(i))}(b_1)$$

to emphasise that we are doing a substitution.

4.1.2 Spheres and Pushouts

In Section 3.8, we gave an example of how one can construct the circle in CuTT. Similarly, we can define the 2-sphere as a HIT with the following constructors.

- $\mathsf{base} : \mathbb{S}^2$
- $\mathsf{surf} : \mathsf{refl}_{\mathsf{base}} \equiv \mathsf{refl}_{\mathsf{base}}$

We could continue like this, adding higher path constructors for each step. For the general n-sphere, we could define it by the following constructors

- $\mathsf{base} : \mathbb{S}^n$
- $\mathsf{loop} : \Omega^n(\mathbb{S}^n, \mathsf{base})$

where

$$\Omega^n \mathbb{S}^n :\equiv \left(\mathsf{refl}_{\mathsf{refl}_{\cdots_{\mathsf{refl}_{\mathsf{base}}}}} \equiv \mathsf{refl}_{\mathsf{refl}_{\cdots_{\mathsf{refl}_{\mathsf{base}}}}} \right)$$

This idea is simple, but it is not clear how to interpret it formally. The key problem is the fact that the type $\Omega^n(\mathbb{S}^n, \mathsf{base})$ does not reduce definitionally unless n is fixed. For this reason, we will introduce the general n-sphere in terms of *suspensions.*

Definition 4.6 (Suspensions)**.** Given a type A, the *suspension of* A, ΣA, is a HIT defined by

- $\mathsf{north} : \Sigma A$
- $\mathsf{south} : \Sigma A$
- $\mathsf{merid} : A \to \mathsf{north} \equiv \mathsf{south}$

Recall from classical topology that the suspension of the n-sphere is homotopy equivalent to the $(n + 1)$-sphere. Using this idea, we can define n-spheres as follows.

Definition 4.7 ($\mathbb{S}^n$)**.** For any $n \geq -1$, $\mathbb{S}^n$ is defined inductively by

- $\mathbb{S}^{-1} :\equiv \bot$
- $\mathbb{S}^{n+1} :\equiv \Sigma \mathbb{S}^n$

It is relatively straightforward to check that this definition is equivalent to, for instance, the base/loop-definitions of $\mathbb{S}^1$ and $\mathbb{S}^2$.

We can also define a sphere using *pushouts.* These correspond to homotopy pushouts in classical mathematics.

Definition 4.8 (Pushouts)**.** For any three types A, B and C together with functions $f : A \to B$ and $g : A \to C$, we define the *pushout* of f and g, denoted $B \sqcup^A_{f,g} C$, as a HIT with the following constructors.

- For every $b : B$, an element $\mathsf{inl}(b) : B \sqcup^A_{f,g} C$
- For every $c : C$, an element $\mathsf{inr}(c) : B \sqcup^A_{f,g} C$
- For every $a : A$, a path $\mathsf{push}(a) : \mathsf{inl}(f(a)) \equiv \mathsf{inr}(g(a))$

We omit f and g and simply write $B \sqcup^A C$ when the functions are clear from context.

This definition agrees with the usual category theoretic definition of pushouts with commutativity given by push.

$$\begin{array}{ccc} A & \xrightarrow{g} & C \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ B & \xrightarrow[\mathsf{inl}]{} & B \sqcup^A C \end{array}$$

One nice property of pushouts is that they give suspension as a special case. Given a type A, we can express its suspension by the pushout of the map $\lambda\, x\, .\, * : A \to \top$ with itself.

$$\begin{array}{ccc} A & \xrightarrow{\lambda\, x\, .\, *} & \top \\ {\scriptstyle \lambda\, x\, .\, *}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ \top & \xrightarrow[\mathsf{inl}]{} & \top \sqcup^A \top \end{array}$$

In particular, we get that $\mathbb{S}^{n+1} \simeq \top \sqcup^{\mathbb{S}^n} \top$ for every $n \geq 0$.

We can also use pushouts to define the wedge sum of two types. Recall from topology that the wedge sum of two pointed spaces A and B is the space obtained by glueing A and B together in their base points. We can express this by the following pushout diagram.

$$\begin{array}{ccc} \top & \xrightarrow{\lambda\, x\, .\, b} & B \\ {\scriptstyle \lambda\, x\, .\, a}\downarrow & & \downarrow{\scriptstyle \mathsf{inr}} \\ A & \xrightarrow[\mathsf{inl}]{} & A \sqcup^\top B \end{array}$$

Formally, we define the wedge sum as follows.

Definition 4.9 (Wedge sums)**.** Let (A, a) and (B, b) be two pointed types. The *wedge sum* of A and B, denoted by $A \vee B$, is defined as the pushout $A \sqcup^\top B$ of the maps $\lambda\, x\, .\, a : \top \to A$ and $\lambda\, x\, .\, b : \top \to B$.

4.1.3 Homotopies and Loop Spaces

Given a type A and two terms $x, y : A$, recall that the type $x \equiv y$ corresponds to the *path space* over x and y. We may also look at the path space over two paths in $x \equiv y$, paths between two paths in this type, and so on. In this sense, we capture the notion of *homotopy* in HoTT.

Of particular interest is often the (n-dimensional) *loop space* over a point $x : A$, denoted $\Omega^n(A, x)$. We define it inductively as follows.

$$\begin{aligned}\Omega^0(A,x) &:\equiv A\\ \Omega^1(A,x) &:\equiv (x \equiv x)\\ \Omega^{n+1}(A,x) &:\equiv (\mathsf{refl} \equiv_{\Omega^n(A,x)} \mathsf{refl})\end{aligned}$$

We simply write $\Omega(A, x)$ for $\Omega^1(A, x)$. We often omit the base-point x when it is clear from context and simply write ΩA or $\Omega^n A$. Most questions about path types, apart from those about inhabitance, can be reduced to questions about loop spaces. Indeed, as soon as we know that a path type $x \equiv y$ is inhabited, we may, by path induction, assume that $x \coloneqq y$.

A particularly important fact about loop spaces is that any higher dimensional loop space is commutative with respect to path composition. The proof is inspired by the proof by Bentzen in [5]. The difference between this proof and Bentzen's is that his proof uses path induction to first prove a more general statement, whereas this proof constructs the path more directly.

Theorem 4.10 (Eckmann-Hilton)**.** *For any pointed type (A, a), $n \geq 2$ and paths $p, q : \Omega^n A$, we have that*

$$p \cdot q \equiv q \cdot p$$

Proof. We start off by showing that

$$\mathsf{cong}_{(_\cdot\mathsf{refl})}(p) \cdot \mathsf{cong}_{(\mathsf{refl}\cdot_)}(q) \equiv \mathsf{cong}_{(\mathsf{refl}\cdot_)}(q) \cdot \mathsf{cong}_{(_\cdot\mathsf{refl})}(p)$$

We construct the path explicitly. Let $i : I$. Consider the term p_i defined by

$$p_i :\equiv ((\lambda\, j\,.\, p(j \wedge {\sim}i) \cdot q(j \wedge i)) \cdot (\lambda\, j\,.\, p({\sim}i \vee j) \cdot q(i \vee j)))$$

We have

$$\begin{aligned}p_{i_0} &\coloneqq ((\lambda\, j\,.\, p(j) \cdot q(i_0)) \cdot (\lambda\, j\,.\, p(i_1) \cdot q(j)))\\ &\coloneqq ((\lambda\, j\,.\, p(j) \cdot \mathsf{refl}) \cdot (\lambda\, j\,.\, \mathsf{refl} \cdot q(j)))\\ &\coloneqq \mathsf{cong}_{(_\cdot\mathsf{refl})}(p) \cdot \mathsf{cong}_{(\mathsf{refl}\cdot_)}(q)\end{aligned}$$

and similarly for p_{i1}. Note that the composition $q(i) \cdot p(j)$ and $p(j) \cdot q(i)$ is well-typed for all $i, j : I$, since $q(i), p(j) : \mathsf{refl} \equiv \mathsf{refl}$.

It is easy to prove that $\mathsf{cong}_{(_\cdot\mathsf{refl}_a)} \equiv \mathsf{cong}_{(\mathsf{refl}_a\cdot_)}$, which follows from the fact that $\mathsf{rUnit}\,(\mathsf{refl}_a) \coloneqq \mathsf{lUnit}\,(\mathsf{refl}_a)$ (given appropriate definitions of rUnit and lUnit). Using functoriality of cong, we get

$$\mathsf{cong}_{(\mathsf{refl}_a\cdot_)}(p \cdot q) \equiv \mathsf{cong}_{(\mathsf{refl}_a\cdot_)}(q \cdot p) \tag{3}$$

The map $\mathsf{refl}_a \cdot _$ is an equivalence by lUnit, and hence we get from Theorem 2.11.1 in [14] that $\mathsf{cong}_{(\mathsf{refl}_a \cdot _)}$ is an equivalence. Thus we also get that $p \cdot q \equiv q \cdot p$. $\square$

For a shorter proof of Theorem 4.10, see Cubical.Homotopy.Loopspace in [4]. This proof does not rely on cong preserving equivalences, and instead transforms (3) into a path of the right type by an application of comp, i.e. the operation for heterogeneous composition of cubes.

4.1.4 n-Types and Truncations

Given a natural number n, we say that a type A is an *n-type* if for every $x : A$ and $m > n$ we have that $\Omega^m(A, x)$ is contractible. We say that A is a (-1)-type if it has contractible identity types and that A is a (-2)-type if it is contractible. We also give the following equivalent definition.

Definition 4.11 (n-types)**.** A type A is said to be a

1. (-2)-type if A is contractible
2. (-1)-type if for every $x, y : A$, we have a path $x \equiv y$.
3. $(n + 2)$-type if for every $x, y : A$, we have that $x \equiv y$ is an $(n + 1)$-type.

We refer to (-1)-types as *propositions* and to 0-types as *sets*.

As in classical topology, path spaces often have a very rich structure. In HoTT, however, we often cannot consider a type in isolation from its path spaces. In order to be able to disregard the higher homotopy structures of a type, we introduce *truncations*. Given a type A, its n-truncation $\|A\|_n$ is just like A, but with all paths spaces over A of dimension greater than n trivial. For instance, $\|A\|_0$ is just like A but with all path-spaces trivial.

We also define $\|A\|_{-2}$ and $\|A\|_{-1}$. The (-2)-truncation $\|A\|_{-2}$ simply turns A into a contractible (i.e. trivial) type, whereas the (-1)-truncation $\|A\|_{-1}$ turns A into a type with all elements identical. As expected, these two constructions are equal if A is pointed.

For any fixed $n \geq -2$, the definitions of truncations are simple. For instance, we can define the (-1)- and 0-truncations, referred to respectively as propositional truncations and set truncations, as follows.

Definition 4.12 (Propositional truncation)**.** For any type A, its propositional truncation $\|A\|_{-1}$ is defined as a HIT with the following constructors.

- For any $x : A$, we have a term $|x|_{-1}$
- For any two terms $x, y : \|A\|_{-1}$, we have a term $\mathsf{squash}(x, y) : x \equiv y$

Definition 4.13 (Set truncation)**.** For any type A, its set truncation $\|A\|_0$ is defined as a HIT with the following constructors.

- For any $x : A$, we have a term $|x|_0$

- For any two terms $x, y : \|A\|_0$ and two paths $p, q : x \equiv y$, we have a term $\mathsf{squash}(p, q) : p \equiv q$

These definitions are easy to mimic for any fixed n. However, the number of arguments in the second constructor increases as n increases. For this reason, we cannot give a general definition of n-truncations using a squash constructor. Instead of using squash, we introduce the constructors hub and spoke, in the style of [14].

Definition 4.14 (n-truncation)**.** For any type A and $n \geq -1$, the n-truncation $\|A\|_n$ is defined as a HIT with the following constructors.

- For each $x : A$, we have a term $|x|_n : \|A\|_n$
- For each $f : \mathbb{S}^{n+1} \to \|A\|_n$, we have a term $\mathsf{hub}_f : \|A\|_n$
- For each $f : \mathbb{S}^{n+1} \to \|A\|_n$ and each $x : \mathbb{S}^{n+1}$ we have a path $\mathsf{spoke}_f(x) : f(x) \equiv \mathsf{hub}_f$.

In order to understand what the hub and spoke constructors do, we need the following theorem. First, we define the notion of pointed maps.

Definition 4.15 (Pointed maps)**.** Given two pointed types (A, a) and (B, b), the type of pointed maps from A to B is defined by

$$((A, a) \to_* (B, b)) :\equiv \sum_{f : A \to B} f(a) \equiv b$$

Theorem 4.16. *For any pointed types (A, a) and (B, b), we have that*

$$(\Sigma A \to_* B) \simeq (A \to_* \Omega(B, b))$$

where ΣA is pointed by north *and $\Omega(B, b)$ is pointed by* refl*.*

Proof. Omitted. See Lemma 6.5.4. in [14]. □

Corollary 4.17. *For any pointed type (A, a) and $n \geq -1$ we have that*

$$\Omega^{n+1}(A, a) \simeq (\mathbb{S}^{n+1} \to_* (A, a))$$

Proof. We induct on n. For $n = -1$, we need to show that

$$A \simeq (\Sigma\bot \to_* (A, a)) \tag{4}$$

We define $f : A \to (\Sigma\bot \to_* (A, a))$ by

$$f(x) = (k_x, \mathsf{refl})$$

where $k_x : \Sigma\bot \to A$ defined by $k_x(\mathsf{north}) = a$ and $k_x(\mathsf{south}) = x$. For the inverse of f, we define $g : (\Sigma\bot \to_* (A, a)) \to A$ by $g(h, p) = h(\mathsf{south})$. For any $x : A$, we have that $g(f(x)) :\equiv x$. Thus, it remains to show that $f(g(h, p)) = (h, p)$.

Let $i : I$. We need to construct a dependent pair $(h_i, p_i) : (\Sigma\bot \to_* (A, a))$ such that $(h_{i0}, p_{i0}) \coloneqq f(g(h, p))$ and $(h_{i1}, p_{i1}) \coloneqq (h, p)$. First, we note that the first projection of $f(g(h, p))$ is the map F sending north to a and south to $h(\mathsf{south})$. Let $h_i :\equiv \lambda\, x\, .\, P(x, i)$, where $P : (x : \Sigma\bot) \to (F(x) \equiv h(x))$ is defined by

$$\begin{aligned} P(\mathsf{north}) &:\equiv p^{-1} \\ P(\mathsf{south}) &:\equiv \mathsf{refl} \end{aligned}$$

We now need to construct $p_i : h_i(\mathsf{north}) \equiv a$. Letting $p_i :\equiv \lambda j\, .\, p({\sim}i \vee j)$ does the trick.

For $n = 0$, we need to prove

$$\Omega(A, a) \simeq (\mathbb{S}^1 \to_* (A, a))$$

Using Theorem 4.16 and (4), we have that

$$\begin{aligned} (\mathbb{S}^1 \to_* (A, a)) &\simeq (\Sigma\bot \to_* \Omega(A, a)) \\ &\simeq \Omega(A, a) \end{aligned}$$

For the inductive step, we have

$$\begin{aligned} \Omega^{n+2}(A, a) &\simeq \Omega^{n+1}(\Omega(A, a)) \\ &\simeq (\mathbb{S}^{n+1} \to_* \Omega(A, a)) && \text{(inductive hypothesis)} \\ &\simeq (\Sigma\mathbb{S}^{n+1} \to_* (A, a)) && \text{(Theorem 4.16)} \\ &\coloneqq (\mathbb{S}^{n+2} \to_* (A, a)) \end{aligned}$$

□

By Corollary 4.17, in order for a pointed type (A, a) to be and n-type, it is enough to show that $(\mathbb{S}^{n+1} \to_* (A, a))$ is contractible. By introducing hub and spoke we guarantee precisely this. Naturally, this indirect definition is not as easy to work with as the direct definitions using squash. In practice, however, we almost always use truncations in contexts where their recursion and induction principles are applicable, in which case both definitions behave the same.

We proceed with some abuse of notation and assume that the definition also applies in the case when $n = -2$. In reality, we need to add two higher constructors to Definition 4.14 just in order to take care of this special case. For the sake of simplicity, we do not do this here. For a formal definition, see Cubical.HITs.Truncation.Base in [4]. For the remainder of this thesis, we also take elements in $\|A\|_{-2}$ to be on the form $|\, a\, |_{-2}$. The following theorem holds trivially for $n = -2$ (and so do most theorems for this special case).

Theorem 4.18 (Truncation Elimination)**.** *Let A be a type, $n \geq -2$ and $B : \|A\|_n \to n{-}\mathsf{type}$. Suppose $B(|\, a\, |_n)$ holds for every $a : A$. Then $B(x)$ holds for every $x : \|A\|_n$.*

Proof. We induct on x. We are given the case $x \coloneqq |\, a\, |_n$ for some $a : A$. The other cases are immediate from Corollary 4.17. □

In essence, the principle says that in order to prove a statement of type $B(x)$ for every $x : \|A\|_n$ for some type A (or, equivalently, define a dependent function $(x : \|A\|_n) \to B(x)$), it is enough to do so for elements $|\, a\,|_n : \|A\|_n$. Using this, we can easily prove the following simple but important facts.

Proposition 4.19. *For all $n \geq -2$ and n-types A, we have that $\|A\|_n \simeq A$*

Proof. We define $f : \|A\|_n \to A$ by truncation elimination, using that A is an n-type:

$$f(|\, a\,|_n) = a$$

The inverse map is just $|\, -\,|_n : A \to \|A\|_n$. We have $|\, f(|\, a\,|_n)\,|_n :\equiv |\, a\,|_n$ and $f(|\, a\,|_n) :\equiv a$. Hence, by another application of truncation elimination, we have established the equivalence. □

Proposition 4.20. *Let $n \geq -2$ and let $m \geq n$. Given a type A, we have that*

$$\|A\|_n \simeq \|\|A\|_m\|_n \,.$$

Proof. The equivalence is given by $f : \|A\|_n \to \|\|A\|_m\|_n$ defined by

$$f(|\, a\,|_n) = \|\, a\,|_m|_n$$

We define the quasi inverse g by

$$g(|\, x\,|_n) = g'(x)$$

where $g' : \|A\|_m \to \|A\|_n$

$$g'(|\, a\,|_m) = |\, a\,|_n$$

using that $\|A\|_n$ also is an m-type. These maps immediately cancel out by further truncation elimination. □

Suppose we want to use truncation elimination to prove a statement on the form $(x : \|A\|_n) \to \cdots \to B(x)$ where $B : \|A\|_n \to n\text{-type}$. The following proposition tells us that this is allowed, no matter what intermediary types this statement contains.

Proposition 4.21. *Let A be a type and let $B : A \to n\text{-type}$. Then $((a : A) \to B(a))$ is an n-type.*

Proof. We induct on n. For $n = -2$, we have that $B(a)$ is contractible for every $a : A$. Denote the centre of contraction by c_a. We define $f : ((a : A) \to B(a))$ by

$$f(a) = c_a$$

Hence $((a : A) \to B(a))$ is pointed. Now let $g : ((a : A) \to B(a))$. We want to show that $f \equiv g$. By function extensionality, it is enough to show that

$f(a) \equiv g(a)$ given an arbitrary $a : A$. This follows, since $B(a)$ is contractible. The case when $n = -1$ is follows by the same argument.

Suppose the statement holds for $n \geq -1$. We wish to show that it holds for $n+1$. Let $f, g : ((a : A) \to B(a))$. We wish to show that $f \equiv g$ is an n-type. By function extensionality, we have that

$$f \equiv g \simeq ((a : A) \to f(a) \equiv g(a))$$

By the inductive hypothesis, we thus only need that $f(a) \equiv g(a)$ is an n-type for every $a : A$. But this follows from the fact that $B(a)$ is an $(n+1)$-type. $\square$

For now, there is no easy way of dealing with paths in truncations of types. Suppose for instance that we have a path $|\,x\,|_n \equiv |\,y\,|_n$ for two points $x, y : A$ and we want to say something about the path space $x \equiv y$. In general, there is no map of type $|\,x\,|_n \equiv |\,y\,|_n \to x \equiv y$. The following theorem, however, gives us the next best thing.

Theorem 4.22. *Let $x\,y : A$ and $n \geq -2$. Then*

$$\|x \equiv y\|_n \simeq (|\,x\,|_{n+1} \equiv |\,y\,|_{n+1})$$

Proof. The proof uses the encode-decode method and uses the fact that n-type is an $(n+1)$-type (see Theorem 7.3.11 in [14]). We begin by defining a fibration $P : \|A\|_{n+1} \to \|A\|_{n+1} \to n\text{-}\mathsf{type}$. Using that n-type is an $(n+1)$-type, we may define it by truncation elimination.

$$P(|\,x\,|_{n+1}, |\,y\,|_{n+1}) :\equiv \|x \equiv y\|_n$$

We now define $\mathsf{decode} : (z, w : \|A\|_{n+1}) \to P(z, w) \to z \equiv w$ by truncation elimination.

$$\mathsf{decode}(|\,x\,|_{n+1}, |\,y\,|_{n+1}, |\,p\,|_n) :\equiv \mathsf{cong}_{|-|_{n+1}}(p)$$

In order to define the encode function, we first define a function

$$f : (z : \|A\|_{n+1}) \to P(z, z)$$

Again, truncation elimination is permissible since $P(z, w)$ is an n-type, and hence an $(n+1)$-type for every $z,\ w : \|A\|_{n+1}$. We define f by

$$f(|\,x\,|_{n+1}) :\equiv |\,\mathsf{refl}_x\,|_n$$

We can now define $\mathsf{encode} : (a, b : \|A\|_{n+1}) \to a \equiv b \to P(a, b)$ by

$$\mathsf{encode}(a, b, p) :\equiv \mathsf{transport}^{(\lambda\, i\, .\, P(a, p(i)))}(f(a))$$

Let $a, b : \|A\|_{n+1}$. The fact that $\mathsf{decode}(a, b, -) \circ \mathsf{encode}(a, b, -) \equiv \lambda\, x\, .\, x$ is immediate by function extensionality and truncation elimination.

We finally show

$$\mathsf{encode}(|a|_{n+1}, |b|_{n+1}, \mathsf{decode}(|a|_{n+1}, |b|_{n+1}, |p|_n)) \equiv |p|_n$$

for all $a, b : A$ and $p : a \equiv b$. The truncation elimination is justified here, since paths over n-types are $(n-1)$-types, i.e. also $(n+1)$-types. By path induction, we may assume that $b :\equiv a$ and $p :\equiv \mathsf{refl}$. Thus, we get

$$\begin{aligned}\mathsf{encode}(|a|_{n+1}, |a|_{n+1}, \mathsf{decode}(|a|_{n+1}, |a|_{n+1}, |\mathsf{refl}|_n)) &\equiv \mathsf{transport}^{\mathsf{refl}}(|\mathsf{refl}|_n) \\ &\equiv |\mathsf{refl}|_n\end{aligned}$$

and we are done. We have shown that

$$P(a, b) \simeq (a \equiv b)$$

for every a, $b : \|A\|_{n+1}$. In particular, for every x, $y : A$, we have

$$\|x \equiv y\|_n \simeq (|x|_{n+1} \equiv |y|_{n+1})$$

which is what we wanted to prove. □

A particularly important consequence of this is the fact that path spaces over truncations of types admit an elimination rule.

Corollary 4.23. *Let $x, y : A$ and let B be an n-type for some $n \geq -2$. Suppose that we have a function $f : x \equiv y \to B$. Then we also have a function*

$$g : |x|_{n+1} \equiv |y|_{n+1} \to B$$

Proof. Let $p : |x|_{n+1} \equiv |y|_{n+1}$. We are to construct an element of B. By Proposition 4.30, we have an element $p' : \|x \equiv y\|_n$. Since B is an n-type, we may assume that $p :\equiv |q|_n$ for some $q : x \equiv y$. Hence we get $f(q) : B$. □

4.1.5 Connected Functions and Types

Using the machinery of truncations, we are now able to give the type theoretical analogue of *connectedness*. Following [6], we define it as follows.

Definition 4.24. Let $n \geq -2$. A map $f : A \to B$ is said to be *n-connected* if $\|\mathsf{fib}_f(b)\|_n$ is contractible for every $b : B$. Furthermore, a type A is said to be n-connected if $\|A\|_n$ is contractible.

These definitions of connectedness agree with the traditional definitions of n-connected spaces and $n+1$-connected functions in traditional algebraic topology (see e.g. [7]). We first state an obvious but important fact about connected types.

Proposition 4.25. *If A is an n-connected type, then A is m-connected for every $m \leq n$.*

Proof. Immediate by Theorem 4.20. □

Connected maps yield a useful induction principle. In essence, this allows us to specify maps from pointed types by only constructing them for the base points (modulo some connectedness conditions). We give the induction principle in its full form as follows.

Theorem 4.26. *Let A and B be types with $f : A \to B$. Given any fibration $P : B \to \mathsf{Type}$ over B, we write*

$$\begin{aligned} g_P &: ((b : B) \to P(b)) \to ((a : A) \to P(f(a))) \\ g_P &:\equiv \lambda\, h \,.\, h \circ f \end{aligned} \tag{5}$$

Let $n \geq -2$. TFAE:

(i) f is n-connected

(ii) g_P is an equivalence for every $P : B \to n{-}\mathsf{type}$

(iii) g_P has a section for every $P : B \to n{-}\mathsf{type}$

Proof. We begin with the implication (i) $\Longrightarrow$ (ii). Suppose that $f : A \to B$ is n-connected and let $P : B \to n{-}\mathsf{type}$. We are to show that g_P is an equivalence. We begin by establishing an equivalence

$$((b : B) \to P(b)) \simeq ((a : A) \to P(f(a))) \tag{6}$$

Simultaneously, we trace a given $h : (b : B) \to P(b)$ being mapped through this equivalence, in order to show that the equivalence in (6) agrees with g_P. First, we note that we have the following equivalences.

$$\begin{aligned} ((b : B) \to P(b)) &\simeq ((b : B) \to \|\mathsf{fib}_f(b)\|_n \to P(b)) \\ &\simeq ((b : B) \to \mathsf{fib}_f(b) \to P(b)) \end{aligned}$$

The first equivalence is trivial since $\|\mathsf{fib}_f(b)\|_n$ is contractible. The second equivalence comes from truncation elimination, using that $P(b)$ is an n-type for every $b : B$. These compositions send $h : ((b : B) \to P(b))$ to $\lambda\, x\, y \,.\, h(x)$. We continue by currying $\mathsf{fib}_f(b)$ and get an equivalence

$$(b : B)(a : A)(p : f(a) \equiv b) \to P(b)$$

This trivially sends $\lambda\, x\, y \,.\, h(x)$ to $\lambda\, x\, y\, p \,.\, h(x)$. Due to the argument of type $(f(a) \equiv b)$, this type is trivially equivalent to the type

$$(a : A) \to P(f(a))$$

This sends $\lambda\, x\, y \,.\, h(x)$ to $\lambda\, a \,.\, h(f(a)) :\equiv g_P(h)$, and we are done.

The implication (ii) $\Longrightarrow$ (iii) is trivial. Hence it remains to show that (iii) $\Longrightarrow$ (i). Our goal is to show that for every $b : B$, we have that $P(b) :\equiv \|\mathsf{fib}_f(b)\|_n$ is contractible. Since $P(b)$ is an n-truncation, it is an n-type. Thus

we get from (iii) that g_P has a section h, and so we immediately get an element $c : (b : B) \to \|\mathsf{fib}_f(b)\|_n$ by setting $c :\equiv h(\lambda x\,.\ |\, x, \mathsf{refl}_{f(x)}\,|_n)$. Since h is a section, we have

$$|\, a, \mathsf{refl}_{f(a)}\,|_n = c(f(a))$$

We choose this as our centre of contraction. We now want to show that for every $b : B$ and $w : \|\mathsf{fib}_f(b)\|_n$, we have $w \equiv c(b)$. By truncation elimination, we may assume that w is on the form $|\, a, p\,|_n$. Hence it suffices to show that for every $a : A$ and $p : f(a) \equiv b$, we have $|\, a, p\,|_n \equiv c(b)$. By path induction we may assume that $b :\equiv f(a)$ and $p :\equiv \mathsf{refl}_{f(a)}$. Hence it suffices to show that $|\, a, \mathsf{refl}_{f(a)}\,|_n \equiv c(f(a))$, which we already know is true. This concludes the proof. ☐

This theorem becomes especially powerful in combination with the following lemma.

Lemma 4.27. *For every pointed type (A, a_0), we have that A is $(n+1)$-connected iff the map $\lambda x\,.\,a_0 : \top \to A$ is n-connected.*

In order to prove Lemma 4.27, we first need some preliminary lemmas. The following lemma is an immediate consequence of Theorem 4.26.

Lemma 4.28. *A type A is n-connected iff the map*

$$\lambda\, b\, a\,.\, b : B \to (A \to B) \tag{7}$$

is an equivalence for every n-type B.

We also need the following simple lemma.

Lemma 4.29. *Let A be an $(n+1)$-connected type. Then for any $x, y : A$, we have that $x \equiv y$ is n-connected.*

Proof. We want to show that $\|x \equiv y\|_n$ is contractible. From Theorem 4.22 we have that $\|x \equiv y\|_n \simeq (|\ x\ |_{n+1} \equiv |\ y\ |_{n+1})$. By assumption, $\|A\|_{n+1}$ is contractible. Hence there is a centre of contraction $p : |\ x\ |_{n+1} \equiv |\ y\ |_{n+1}$. Furthermore, any contractible type is also a set, and hence for each $q : |\, x\,|_{n+1} \equiv |\, y\,|_{n+1}$, we have that $p \equiv q$. Thus $|\, x\,|_{n+1} \equiv |\, y\,|_{n+1}$ is contractible. Hence $x \equiv y$ is n-connected. ☐

We are now ready to prove Lemma 4.27.

Proof of Lemma 4.27. We begin with the left-to-right direction. Suppose that A is $(n+1)$-connected. We want to show that $\mathsf{fib}_{\lambda x\,.\,a_0}(a)$ is n-connected for every $a : A$, i.e. that $\sum_{x:\top}(a_0 \equiv a)$ is n-connected. Now, of course, we have $\sum_{x:\top}(a_0 \equiv a) \simeq (a_0 \equiv a)$. But this is n-connected by Lemma 4.29. Hence $\lambda x\,.\,a_0$ is n-connected.

For the other direction, we assume that $\lambda x\,.a_0$ is n connected. Let B be an $(n+1)$-type. By Lemma 4.28, it suffices to show that the map f defined as in (7) is an equivalence. Consider the map $g : (A \to B) \to B$ given by

$\lambda h \,.\, h(a_0)$. We have that $g(f(b)) \coloneqq b$ for every $b : B$. So it remains to show that $f(g(h)) \equiv h$ for every $h : A \to B$. We prove something slightly stronger; namely that for any $h : (A \to B)$ and $a : A$, we have that $h(a) \equiv h(a_0)$. Note that $h(a) \equiv h(a_0)$ is an n-type for every $a : A$, since B is an $(n+1)$-type. Using this fact, and the fact that $\lambda x \,.a_0$ is n connected, we get, by setting $P :\equiv ((a : A) \to\ a \,.\, h(a) \equiv h(a_0))$ in the statement of Lemma 4.26, that there is an equivalence $((a : A) \to h(a) \equiv h(a_0)) \simeq ((b : \top) \to h(a_0) \equiv h(a_0))$. Since $((b : \top) \to h(a_0) \equiv h(a_0))$ is trivially pointed, also $((a : A) \to h(a) \equiv h(a_0))$ is pointed. Thus, for every $h : A \to B$ and $a : A$ we have that $h(a) \equiv h(a_0)$. This finishes the proof. □

Proposition 4.30. *If $f : A \to B$ is n-connected, then the induced map $\|f\|_n : \|A\|_n \to \|B\|_n$ is an equivalence.*

Proof. We begin by constructing an inverse map $g : \|B\|_n \to \|A\|_n$. Let con be the proof that f is n-connected, i.e.

$$\mathsf{con} : (b : B) \to \sum_{x : \|\mathsf{fib}_f(b)\|_n} \big((y : \|\mathsf{fib}_f(b)\|_n) \to y \equiv x\big)$$

We define g by truncation elimination:

$$g(|\, b \,|_n) :\equiv \|\mathsf{fst}\|_n \,(\mathsf{fst}(\mathsf{con}\,(b)))$$

We first show that $\|f\|_n \,(g\,(|\, b \,|_n)) \equiv|\ b\ |_n$ for every $b : B$. We have $g(|\ b\ |_n) \equiv \|\mathsf{fst}\|_n \,(\mathsf{fst}(\mathsf{con}(b)))$. By truncation elimination, we may assume that $\mathsf{fst}(\mathsf{con}(b)) \equiv|\ a, p\ |_n$ where $a : A$ and $p : f(a) \equiv b$. Hence we have $\|f\|_n \,(\|\mathsf{fst}\|_n \,(|\, a, p \,|_n)) \equiv|\, f(a) \,|_n$. Now, we get

$$\mathsf{cong}_{|-|_n}(p) : |\, f(a) \,|_n \equiv |\, b \,|_n$$

and we are done.

We finally show that $g\,(\|f\|_n \,(|\, a \,|_n)) \equiv|\ a\ |_n$ for every $a : A$. We have $\|f\|_n \,(|\, a \,|_n) \equiv|\, f(a) \,|_n$. Hence, we are to show that

$$\|\mathsf{fst}\|_n \,(\mathsf{fst}\,(\mathsf{con}(|\, f(a) \,|_n))) \equiv|\, a \,|_n$$

But $\|\mathsf{fib}_f(f(a))\|_n$ is n-connected, and so $\mathsf{fst}\,(\mathsf{con}(|\ a\ |_n)) \equiv|\ a, \mathsf{refl}\ |_n$. Finally, we have $\|\mathsf{fst}\|_n \,(|\ a, \mathsf{refl}\ |_n) \equiv|\ a\ |_n$. Thus, we are done. Hence, $\|f\|_n$ is an equivalence. □

Theorem 4.31. *For every $n \geq -1$, the $\mathbb{S}^{n+1}$ is n-connected.*

The theorem is an easy corollary of the following lemma.

Lemma 4.32. *Given two maps $f : C \to A$ and $g : C \to B$ where f is n-connected, the map $\mathsf{inr} : B \to A \sqcup^C B$ is n-connected.*

Proof. Let $D :\equiv A \sqcup^C B$ and $P : D \to n-\mathsf{type}$. By Theorem 4.26 it suffices to show that the associated map g_P has a section. In this case, this means that given a map $h : (b : B) \to P(\mathsf{inr}(b))$, we have map $k : (d : D) \to P(d)$ such that $k \circ \mathsf{inr} \equiv h$. We begin by constructing k. We first define the fibration $Q : A \to n\text{-}\mathsf{type}$ by

$$Q(a) :\equiv P(\mathsf{inl}(a))$$

We now get a function $F : (c : C) \to Q(f(c))$ defined by

$$F(c) :\equiv \mathsf{transport}^{\lambda\, i\, .\, P((\mathsf{push}(c))(\sim i))}(h(g(c)))$$

In order to define k for a point on the form $\mathsf{inl}(a)$ for some $a : A$, we need to construct an element of type $P(\mathsf{inl}(a))$. That is, we need to construct some $u : (a : A) \to Q(a)$. First, note that since f is n-connected, we have by Theorem 4.26 that the associated map

$$g_Q : (a : A \to Q(a)) \to ((a : C) \to Q(f(c))) \tag{8}$$

is an equivalence. Hence, since we have $F : (c : C) \to Q(f(c))$, this also gives us an element $u : (a : A) \to Q(a)$. To be specific, we pick $u :\equiv \mathsf{fst}(\mathsf{fib}_{g_Q}(F))$.

We are now ready to define k. First, we let

$$k(\mathsf{inl}(a)) :\equiv u(a)$$
$$k(\mathsf{inr}(b)) :\equiv h(b)$$

We now need to define $k((\mathsf{push}(c))(i))$ where $c : C$. That is, we need to construct a dependent path from $u(f(c))$ to $h(g(c))$ over $\lambda\, i\, .\, P((\mathsf{push}(c))\,(i))$. Since u is in the fibre of g_Q, we have that $u(f(c)) \equiv F(c)$. Thus, it suffices to construct the dependent path from $F(c)$ to $h(g(c))$. We get this path by transporting $F(c)$ along $\lambda\, i\, .P((\mathsf{push}(c))\,(i))$, since

$$\begin{aligned} & \mathsf{transport}^{\lambda\, i\, .P((\mathsf{push}(c))\,(i))}(F(c)) \\ := {} & \mathsf{transport}^{\lambda\, i\, .P((\mathsf{push}(c))\,(i))}\left(\mathsf{transport}^{\lambda\, i\, .P((\mathsf{push}(c))\,(\sim i))}(h(g(c))\right) \\ \equiv {} & h(g(c)) \end{aligned}$$

Now, with k defined, all we need to do is to verify that we have $k \circ \mathsf{inr} \equiv h$ for every $s : (b : B) \to P(\mathsf{inr}(b))$. But this holds definitionally, and thus we are done. □

4.1.6 Freudenthal Suspension Theorem

In order to define a group structure on cohomology types, we shall need an equivalence

$$\|\mathbb{S}^n\|_n \simeq \left\|\Omega\mathbb{S}^{n+1}\right\|_n \tag{9}$$

This can either be proved directly as in [11] or by means of the Freudenthal Suspension Theorem. Here, we choose the latter approach. The following proofs are based on simplified versions of the proofs in [14] due to Evan Cavallo. His proofs can be found in Cubical.Homotopy.WedgeConnectivity and Cubical.Homotopy.Freudenthal in [4].

Lemma 4.33. *Let $n, m \geq -2$ and let $N = (n+2)+(m+2)$. Let $P : B \to (N-2)-\mathsf{type}$ and suppose $f : A \to B$ is m-connected. Then the for every $h : (a : A) \to P(f(a))$ we have that the fibre of g_P from (5) over h is an n-type.*

Proof. We induct on n. For the base case, suppose $n = -2$. We are to show that $\mathsf{fib}_{g_P}(h)$ is a (-2)-type, i.e. contractible, for every $h : (a : A) \to P(f(a))$. In other words, we are to show that g_P is an equivalence. Since $P(b)$ is an m-type, Theorem 4.26 tells us that it suffices to show that f is m-connected, which is precisely what we have assumed.

For the inductive step, we assume that the theorem holds for n and show it for $n+1$. Let $h : (a : A) \to P(f(a))$ and $(\ell_1, p_1), (\ell_2, p_2) : \mathsf{fib}_{g_P}(h)$. We are done if we can show that $(\ell_1, p_1) \equiv (\ell_2, p_2)$ is an n-type. It follows by an easy lemma that

$$((\ell_1, p_1) \equiv (\ell_2, p_2)) \simeq \mathsf{fib}_{\mathsf{cong}_f}(p_1 \cdot p_2^{-1})$$

Let $P' :\equiv \lambda\, b \,.\, \ell_1(b) \equiv \ell_2(b)$. We also have a trivial equivalence

$$\mathsf{fib}_{\mathsf{cong}_f}(p_1 \cdot p_2^{-1}) \simeq \mathsf{fib}_{g_P}(\mathsf{funExt}^{-1}(p_1 \cdot p_2^{-1}))$$

so it suffices to show that $\mathsf{fib}_{g_P}(\mathsf{funExt}^{-1}(p_1 \cdot p_2^{-1}))$ is an n-type. Note that for every $b : B$, we have that $\ell_1(b) \equiv \ell_2(b)$ is an $(N-3)$-type, since $P(b)$ is an $(N-2)$-type. Hence we can apply the inductive hypothesis, thus getting that $\mathsf{fib}_{g_P}(\mathsf{funExt}^{-1}(p_1 \cdot p_2^{-1}))$ is an n-type. This concludes the proof. □

Lemma 4.34 (Wedge Connectivity Lemma)**.** *Let (A, a_0) and (B, b_0) be two pointed types, suppose that they are n- and m-connected for $n, m \geq -1$ respectively and let $P : A \to B \to (n+m)-\mathsf{type}$. Suppose there are two maps*

$$f : (a : A) \to P(a, b_0)$$
$$g : (b : B) \to P(a_0, b)$$

with a path $p : f(a_0) \equiv g(b_0)$. Then there exists a map

$$F : (a : A)(b : B) \to P(a, b)$$

satisfying

$$F(a, b_0) \equiv f(a)$$
$$F(a_0, b) \equiv g(b)$$

for every $a : A$ and $b : B$.

Proof. First, we define a fibration $Q : A \to \mathsf{Type}$ by

$$Q(a) :\equiv \sum_{h:(b:B)\to P(a,b)} (h(b_0) \equiv f(a))$$

The first step is showing that Q is an $(n-1)$-type. We trivially have that

$$Q(a) \simeq Q'(a) :\equiv \sum_{h:(b:B)\to P(a,b)} (h \circ (\lambda\,(x:\top)\,.\,b_0) \equiv \lambda\,x\,.\,f(a))$$

Since B is m-connected, we may apply Lemma 4.27 to conclude that the trivial map $\overrightarrow{b_0} :\equiv \lambda\,x\,.\,b_0 : \top \to B$ is $(m-1)$-connected. The type $Q'(a)$ is just the fibre of $g_{P(a,_)}$ over $\lambda x\,.\,f(a)$. Applying Lemma 4.33 we thus get that it is an $(n-1)$-type.

The second step is showing that the fibre of

$$g_Q :\equiv \lambda\,h\,.\,\lambda\,(x:\top)\,.\,h(a_0) : ((a:A) \to Q(a)) \to \top \to Q(a_0)$$

over $\lambda\,(x:\top)\,.\,(g\,,\,p^{-1})$ is contractible. Since Q is an $(n-1)$-type and the map $\lambda\,(x:\top)\,.\,a_0$ is $(n-1)$ connected, we have that g_Q is an equivalence and hence has contractible fibres.

Let P be the proof that $\mathsf{fib}_{g_Q}(\lambda\,(x:\top)\,.\,(g\,,\,p^{-1}))$ is contractible. Note that $\mathsf{fst}(\mathsf{fst}(P)) : (a:A) \to Q(a)$. Note further that given $a : A$, we have $(\mathsf{fst}((\mathsf{fst}(\mathsf{fst}(P)))(a))) : (b:B) \to P(a,b)$. We can now use this in order to construct our function $F : (a:A)(b:B) \to P(a,b)$. We define it by

$$F(a,b) :\equiv (\mathsf{fst}((\mathsf{fst}(\mathsf{fst}(P)))(a)))(b)$$

It is far from beautiful, but it gets the job done.

We now show that for every $a : A$, we have $F(a,b_0) \equiv f(a)$. We have that

$$(\mathsf{fst}(\mathsf{fst}(P)))(a) : Q(a)$$

Hence

$$\mathsf{snd}((\mathsf{fst}(\mathsf{fst}(P)))(a)) : F(a,b_0) \equiv f(a)$$

Finally, we show that for every $b : B$, we have a path $F(a_0,b) \equiv g(b)$. Note that

$$\mathsf{snd}(\mathsf{fst}(P)) : \mathsf{fst}(\mathsf{fst}(P)) \circ (\lambda\,x\,.\,a_0) \equiv \lambda\,(x:\top)\,.\,(g\,,\,p^{-1})$$

Applying both sides to $* : \top$ gives a path

$$\mathsf{fst}(\mathsf{fst}(P))(a_0) \equiv (g\,,\,p^{-1})$$

Applying fst on both sides gives

$$\mathsf{fst}(\mathsf{fst}(\mathsf{fst}(P))(a_0)) \equiv g$$

i.e.

$$F(a_0,_) \equiv g$$

Hence, we get $F(a_0,b) \equiv g(b)$. □

Theorem 4.35 (Freudenthal Suspension Theorem)**.** *Let (A, a_0) be a pointed type which is n-connected for some $n \geq 0$. Then the map $\sigma : A \to \Omega(\Sigma A)$ defined by*

$$\sigma(x) :\equiv \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(a_0))^{-1}$$

is $2n$-connected.

Proof. We utilise the encode-decode method and prove something more general. We first construct a family of types $\mathsf{Code} : \Sigma A \to \mathsf{north} \equiv y \to \mathsf{Type}$. For the base points, we let

$$\mathsf{Code}(\mathsf{north}, p) :\equiv \|\mathsf{fib}_\sigma(p)\|_{2n}$$
$$\mathsf{Code}(\mathsf{south}, p) :\equiv \|\mathsf{fib}_{\mathsf{merid}}(p)\|_{2n}$$

For $\mathsf{Code}((\mathsf{merid}\,(x_0))(i), p_i)$, with $p_i : \mathsf{north} \equiv (\mathsf{merid}\,(x_0))(i)$ we need to construct a dependent path from $\|\mathsf{fib}_\sigma(p_{i_0})\|_{2n}$ to $\|\mathsf{fib}_{\mathsf{merid}}\,(p_{i_1})\|_{2n}$ Unfortunately, not just any path will do. In particular, we want this equivalence to depend on x_0. First, we note that there is an obvious dependent path from $\mathsf{fib}_{\mathsf{merid}}\,(p_{i_1})$ to $\mathsf{fib}_{\lambda\, x\, .\, \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(x_0))^{-1}}(p_{i_0})$, since get have a dependent path Q from $\lambda\, x\, .\, \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(x_0))^{-1}$ to merid over the filler for path composition. For the remaining part, we fix $p : \mathsf{north} \equiv \mathsf{north}$ and construct an equivalence

$$\|\mathsf{fib}_\sigma(p)\|_{2n} \simeq \left\|\mathsf{fib}_{\lambda\, x\, .\, \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(x_0))^{-1}}(p)\right\|_{2n} \tag{10}$$

We begin by constructing the map. By truncation elimination, it is enough to give a map of type

$$\mathsf{fib}_\sigma(p) \to \left\|\mathsf{fib}_{\lambda\, x\, .\, \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(x_0))^{-1}}(p)\right\|_{2n}$$

or, by currying, a map

$$F : (a : A) \to (\sigma(a) \equiv p) \to \left\|\mathsf{fib}_{\lambda\, x\, .\, \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(x_0))^{-1}}(p)\right\|_{2n}$$

We proceed by an application of Lemma 4.34. Let

$$P :\equiv \lambda\, a\, .\, \lambda\, b\, .\, \left((\sigma(b) \equiv p) \to \left\|\mathsf{fib}_{\lambda\, x\, .\, \mathsf{merid}\,(x) \cdot (\mathsf{merid}\,(a))^{-1}}(p)\right\|_{2n}\right) : A \to A \to \mathsf{Type}$$

This is a $2n$-type. We construct two functions $f : (a : A) \to P(a, a_0)$ and $g : (a : A) \to P(a_0, a)$. We define f by

$$(f(a))(r) :\equiv |\, a, c_r\, |_{2n} \tag{11}$$

where

$$c_r : \mathsf{merid}\,(a) \cdot (\mathsf{merid}\,(a))^{-1} \equiv p$$

is given by the composite path

$$\begin{aligned} \mathsf{merid}\,(a) \cdot (\mathsf{merid}\,(a))^{-1} &\equiv \mathsf{refl} && \text{(by } \mathsf{rCancel}) && (12)\\ &\equiv \mathsf{merid}\,(a_0) \cdot (\mathsf{merid}\,(a_0))^{-1} && \text{(by } \mathsf{rCancel}) && (13)\\ &\equiv p && \text{(by } r) && (14) \end{aligned}$$

We construct g by

$$(g(a))(r) :\equiv |\, a, r \,|_{2n} \tag{15}$$

Finally, it is easy to see that $f(a_0) \equiv g(a_0)$. Putting all of these facts together, we apply Lemma 4.34 to get a map $F : (a : A)(b : A) \to P(a, b)$ with homotopies

$$\begin{aligned} \mathsf{left} : (a : A) &\to F(a, a_0) \equiv f(a) \\ \mathsf{right} : (a : A) &\to F(a_0, a) \equiv g(a) \end{aligned}$$

Going back to (10), we can finally define our equivalence $G_{x_0,p}$ by undoing the currying. We define it by

$$G_{x_0,p}(|\, z \,|_{2n}) :\equiv (\mathsf{uncurry}(F(x_0)))(z)$$

We have to verify that $G_{x_0,p}$ is an equivalence. We do this by showing that its fibres are contractible. The property of being contractible is a proposition, i.e. a (-1)-type. Hence it is also an $(n-1)$-type, since $n \geq 0$. The map $\lambda\, x \,.\, a_0 : \top \to A$ is $(n-1)$-connected by Lemma 4.27. Define $R : A \to (n-1)\text{-}\mathsf{type}$ by

$$R(a) :\equiv (t : \left\|\mathsf{fib}_{\lambda\, y \,.\, \mathsf{merid}\,(y) \cdot (\mathsf{merid}\,(a))^{-1}}(p)\right\|_{2n}) \to \mathsf{isContr}(\mathsf{fib}_{G_{a,p}}(t))$$

Theorem 4.26 now gives us an equivalence

$$((a : A) \to R(a)) \simeq ((x : \top) \to R(a_0))$$

In particular we get a map of type $((a : \top) \to R(a_0)) \to ((a : A) \to R(a))$. Hence, we get a map of type $(R(a_0) \to (a : A) \to R(a))$. So we are done if we can construct an element of $R(a_0)$, i.e. a function

$$\phi : (t : \left\|\mathsf{fib}_{\lambda\, y \,.\, \mathsf{merid}\,(y) \cdot (\mathsf{merid}\,(a_0))^{-1}}(p)\right\|_{2n}) \to \mathsf{isContr}(\mathsf{fib}_{G_{a_0,p}}(t))$$

Again, $\mathsf{isContr}$ is a proposition and hence also a $2n$-type. Thus, we only need to define $\phi(|\, t \,|_{2n})$ for some $t : \mathsf{fib}_{\lambda\, y \,.\, \mathsf{merid}\,(y) \cdot (\mathsf{merid}\,(a_0))^{-1}}(p)$. We deduce from Theorem 4.26 that it is enough to show that $G_{a_0,p}$ is an equivalence. Applying right, it is easy to see that $G_{a_0,p}$ is just the identity function and hence an equivalence. So $G_{a_0,p}$ and hence also $G_{x_0,p}$ is an equivalence.

We now construct a function $\mathsf{encode} : (y : \Sigma A) \to (p : \mathsf{north} \equiv y) \to \mathsf{Code}(y, p)$. By path induction, we only need to define $\mathsf{encode}(\mathsf{north}, \mathsf{refl})$. We let

$$\mathsf{encode}(\mathsf{north}, \mathsf{refl}) :\equiv |\, a_0, \mathsf{rCancel}(\mathsf{merid}\,(a_0)) \,|_{2n}$$

We are done if we can show that $\mathsf{Code}(\mathsf{north}, p)$ is contractible for every $p : \mathsf{north} \equiv \mathsf{north}$. Since $\mathsf{Code}(\mathsf{north}, _) \simeq \mathsf{Code}(\mathsf{south}, _)$, it is enough to show that $\mathsf{Code}(\mathsf{south}, p)$ is contractible for $p : \mathsf{north} \equiv \mathsf{south}$. We choose $\mathsf{encode}(\mathsf{south}, p)$ as the centre of contraction.

We first show that $\mathsf{encode}(\mathsf{south}, \mathsf{merid}\,(a)) \equiv |\, a, \mathsf{refl}\,|_{2n}$ for every $a : A$. Since we defined encode by path induction, this is interpreted as a transport which can be verified to be equal to

$$\mathsf{transport}^{\lambda\, i\, .\, \|\mathsf{fib}_{Q(i)}(\lambda\, j\, .\mathsf{merid}\,(a)(i \wedge j))\|_{2n}}(G_{a,\mathsf{refl}}(|\, a_0, \mathsf{rCancel}(\mathsf{merid}\,(a_0))\,|_{2n}))$$

By left, we get a path

$$G_{a,\mathsf{refl}}(|\, a_0, \mathsf{rCancel}\ (\mathsf{merid}\,(a_0))\,|_{2n}) \equiv |\, a, c_{\mathsf{rCancel}(\mathsf{merid}\,(a_0))}\,|_{2n}$$

where $c_{_}$ is the path from (12). Furthermore, it follows by simple path algebra that

$$c_{\mathsf{rCancel}(\mathsf{merid}\,(a_0))} \equiv \mathsf{rCancel}(\mathsf{merid}\,(a))$$

Hence, it remains to show that

$$\begin{aligned} &\mathsf{transport}^{\lambda\, i\, .\, \|\mathsf{fib}_{Q(i)}(\lambda\, j\, .\mathsf{merid}\,(a)(i \wedge j))\|_{2n}}(|\, a, \mathsf{rCancel}(\mathsf{merid}\,(a))\,|_{2n}) \\ \equiv\ & |\, a, \mathsf{refl}\,|_{2n} \end{aligned}$$

This dependent path is easily constructed using the filler for $\mathsf{rCancel}$ (see Section 3.3).

Finally, we prove the contractibility criterion. That is, given $|\, z\,|_{2n} : \mathsf{Code}(\mathsf{south}, p)$, we show that $\mathsf{encode}(\mathsf{south}, p) \equiv |\, z\,|_{2n}$. Currying z, this is the same as proving that for all $a : A$ and $q : \mathsf{merid}\,(a) \equiv p$, we have

$$\mathsf{encode}(\mathsf{south}, p) \equiv |\, a, q\,|_{2n}$$

By path induction on q, we may assume that $p := \mathsf{merid}\,(a)$. Hence it suffices to show that

$$\mathsf{encode}(\mathsf{south}, \mathsf{merid}\,(a)) \equiv |\, a, \mathsf{refl}\,|_{2n}$$

which is is precisely what we showed above. So $\|\mathsf{fib}_{\mathsf{merid}}(p)\|_{2n}$ is contractible for every p : $\mathsf{north} \equiv \mathsf{south}$ and hence $\|\mathsf{fib}_{\sigma}(p)\|_{2n}$ is contractible for every $p : \mathsf{north} \equiv \mathsf{north}$. Thus σ is $2n$-connected. □

For our purposes, the most important consequence of this is the following corollary.

Corollary 4.36 (Freudenthal Equivalence)**.** *Let $n \geq 0$ and let (A, a_0) be a pointed and n-connected type. We then have that*

$$\|A\|_{2n} \simeq \|\Omega(\Sigma A)\|_{2n}$$

Proof. Follows immediately by Theorem 4.35 and Proposition 4.30. □

4.1.7 The Hopf Fibration

For the constructions of the equivalence (9), the Freudenthal Suspension Theorem will only work when $n \geq 2$. For $n = 1$, we will rely on the *Hopf Fibration.* For the proof of this theorem/construction, see either [14] or the formalised proof in the cubical library (see Cubical.HITs.Hopf in [4]).

Theorem 4.37. *There is a fibration* $\mathsf{Hopf} : \mathbb{S}^2 \to \mathsf{Type}$ *such that*

$$\mathsf{Hopf}(\mathsf{north}) \simeq \mathbb{S}^1$$
$$\sum_{x:\mathbb{S}^2} \mathsf{Hopf}(x) \simeq \mathbb{S}^3$$

Note that, since $\mathsf{merid}\,(\mathsf{north}_{\mathbb{S}^1}) : \mathsf{north} \equiv_{\mathbb{S}^2} \mathsf{south}$, we also have

$$\mathsf{Hopf}(\mathsf{south}) \simeq \mathbb{S}^1$$

We also need the following lemma. Both identities follow immediately by construction.

Lemma 4.38. *For every* $x : \mathbb{S}^1$*, we have*

$$\mathsf{subst}^{\mathsf{Hopf}}((\mathsf{merid}(x)), \mathsf{north}) \equiv x \tag{16}$$
$$\mathsf{subst}^{\mathsf{Hopf}}((\mathsf{merid}(\mathsf{north}))^{-1}, x) \equiv x \tag{17}$$

4.1.8 Loop Spaces over $\mathbb{S}^1$

The final component we need for the cohomology group structure is the fact that $\Omega\mathbb{S}^1 \simeq \mathbb{Z}$, which we saw how to prove in Theorem 2.11. The proof of this gives us the following two maps

$$\mathsf{winding} \quad : \mathbb{Z} \to \Omega(\mathbb{S}^1)$$
$$\mathsf{winding}^{-1} : \Omega\mathbb{S}^1 \to \mathbb{Z}$$

where winding is defined by

$$\mathsf{winding}(k) :\equiv \mathsf{loop}^k$$

and where $\mathsf{winding}^{-1}$ has the property that

$$\mathsf{winding}^{-1}(\mathsf{loop}^k) \equiv k$$

Theorem 2.11 is incredibly useful for characterising paths over $\mathbb{S}^1$ and has several useful corollaries. However, we have one problem. Theorem 2.11 only concerns paths of type $\mathsf{base} \equiv \mathsf{base}$; often we want to say things about paths of type $x \equiv x$ for *any* $x : \mathbb{S}^1$. Fortunately, there is an easy fix. We first need the following lemma which strengthens the induction principle for $\mathbb{S}^1$ when mapping into propositions.

Lemma 4.39. *Let $P : \mathbb{S}^1 \to \mathsf{Type}$ and suppose that $P(\mathsf{base})$ is an proposition. If we have a term of type $P(\mathsf{base})$, then we also have terms of type $P(x)$ for each $x : \mathbb{S}^1$.*

Proof. Let $p : P(\mathsf{base})$. We define $f : (x : \mathbb{S}^1) \to P(x)$ by induction on x. For $x :\equiv \mathsf{base}$, we just give p. For $x :\equiv \mathsf{loop}(i)$ we need to construct a term $q_i : P(\mathsf{loop}(i))$ such that $q_{i0} :\equiv p$ and $q_{i1} :\equiv p$. We begin by constructing a term $r_i : P(\mathsf{loop}(i))$. We define it by

$$r_i :\equiv \mathsf{transport}^{\lambda\, j\, .\, P(\mathsf{loop}(i \wedge j))}(p)$$

Now, since $P(\mathsf{base})$ is a proposition, we have two paths $r_{i0} \equiv p$ and $r_{i1} \equiv p$. Composing these paths with $\lambda\, i\, .\, r_i$, we get a path $q : \lambda\, i\, .\, P(\mathsf{loop}(i))$ such that $q(i1) :\equiv q(i0) :\equiv p$. Thus, we are done. □

Using this, we get all the corollaries from Theorem 2.11 in their general form, which finally gives us a general version of the theorem.

Corollary 4.40. *$\mathbb{S}^1$ is a 1-type.*

Proof. By path induction, it suffices to show that $\Omega(\mathbb{S}^1, x)$ is a set for every $x : \mathbb{S}^1$. We know from Theorem 7.1.10 in [14] that the type $\mathsf{isSet}(A)$ is a proposition for any type A. Hence by Lemma 4.39, it suffices to show the claim when $x :\equiv \mathsf{base}$. Now we note that, by Theorem 2.11, $\Omega(\mathbb{S}^1, \mathsf{base}) \simeq \mathbb{Z}$. Thus, it is enough to show that $\mathbb{Z}$ is a set. But this is a well-known result following from the fact that $\mathbb{Z}$ has decidable equality (see e.g. Theorems 7.2.5 and 7.2.6 in [14]). □

Corollary 4.41. *For any $x : \mathbb{S}^1$ and $p, q : \Omega(\mathbb{S}^1, x)$, we have that*

$$p \cdot q \equiv q \cdot p$$

Proof. By Corollary 4.40, we have that $p \cdot q \equiv q \cdot p$ is a proposition. Hence, by Lemma 4.39 it suffices to show the statement when $x :\equiv \mathsf{base}$ and $p, q : \Omega(\mathbb{S}^1, \mathsf{base})$. The following equality follows easily from the construction of $\mathsf{winding}^{-1}$.

$$\mathsf{winding}^{-1}(p \cdot q) \equiv \mathsf{winding}^{-1}(p) + \mathsf{winding}^{-1}(p)$$

Since $\mathbb{Z}$ is commutative we get

$$\mathsf{winding}^{-1}(p) + \mathsf{winding}^{-1}(p) \equiv \mathsf{winding}^{-1}(q) + \mathsf{winding}^{-1}(p)$$

For any $n, m : \mathbb{Z}$, we have

$$\begin{aligned} \mathsf{winding}(n + m) &\equiv \mathsf{loop}^{n+m} \\ &\equiv \mathsf{loop}^{n} \cdot \mathsf{loop}^{m} \\ &\equiv \mathsf{winding}(n) \cdot \mathsf{winding}(m) \end{aligned}$$

Hence

$$\begin{aligned} &\mathsf{winding}(\mathsf{winding}^{-1}(q) + \mathsf{winding}^{-1}(p)) \\ &\equiv \mathsf{winding}(\mathsf{winding}^{-1}(q)) \cdot \mathsf{winding}(\mathsf{winding}^{-1}(p)) \\ &= q \cdot p \end{aligned}$$

And so we have

$$p \cdot q \equiv \mathsf{winding}(\mathsf{winding}^{-1}(p \cdot q)) \equiv q \cdot p$$

□

Using this, we can easily generalise Theorem 2.11.

Theorem 4.42. *For any* $x : \mathbb{S}^1$*, we have* $\Omega(\mathbb{S}^1, x) \simeq \mathbb{Z}$*.*

Proof. By Theorem 2.11, it is enough to show that

$$\Omega(\mathbb{S}^1, x) \simeq \Omega(\mathbb{S}^1, \mathsf{base})$$

for every $x : \mathbb{S}^1$. We begin by constructing $f_x : \Omega(\mathbb{S}^1, x) \to \Omega(\mathbb{S}^1, \mathsf{base})$. We define it by induction on x. We let $f_{\mathsf{base}}(p) :\equiv p$. For $f_{\mathsf{loop}(i)}(p)$, we need to construct an element $r_i : \mathsf{loop}(i) \equiv \mathsf{loop}(i)$ satisfying

$$\begin{aligned} r_{i0} &\coloneqq p \\ r_{i1} &\coloneqq p \end{aligned}$$

We define $q_i : \mathsf{loop}(i) \equiv \mathsf{loop}(i)$ by

$$q_i :\equiv [\lambda\, j \,.\, \mathsf{loop}(j \wedge i)] \cdot p \cdot \left[\lambda\, j \,.\, \mathsf{loop}((\sim j) \wedge i)\right]$$

When $i \coloneqq i0$, we just get

$$q_{i0} \coloneqq \mathsf{refl} \cdot p \cdot \mathsf{refl} \equiv p$$

When $i \coloneqq i1$ we get

$$\begin{aligned} q_{i1} &\coloneqq \mathsf{loop} \cdot p \cdot \mathsf{loop}^{-1} \\ &\equiv \mathsf{loop} \cdot \mathsf{loop}^{-1} \cdot p && \text{(by Corollary 4.41)} \\ &\equiv p \end{aligned}$$

And so, by composing the above, we get our term r_i. Hence, we may define $f(\mathsf{loop}(i), p) \equiv r_i$.

The inverse function $g_x : \Omega(\mathbb{S}^1, \mathsf{base}) \to \Omega(\mathbb{S}^1, x)$ is defined in a similar manner. In particular, we let $g_{\mathsf{base}}(p) :\equiv p$.

Fix $x : \mathbb{S}^1$. We now show that the maps cancel out. The proof is identical in both directions, so we only do one here. Let $p : \Omega(\mathbb{S}^1, x)$. We need to show that $g_x(f_x(p)) \equiv p$. By Corollary 4.40, we have that $g_x(f_x(p)) \equiv p$ is a proposition. Hence we may assume that $x \coloneqq \mathsf{base}$. But now $g_x(f_x(p)) \coloneqq p$, since f_{base} and g_{base} are both just the identity function on $\Omega(\mathbb{S}^1, \mathsf{base})$. □

4.1.9 Groups

We need to introduce some elementary group theory. In order to define a group structure on a type G, we require that G is a set. This means that most elemntary definitions and lemmas concerning groups from standard mathematics carry over into HoTT. We define a group as a tuple

$$(G, \mathsf{isSet}(G), 0_G, -_G, +_G, \mathsf{isGroup}(G, 0_G, -_G, +_G))$$

where

- G is the underlying type
- $\mathsf{isSet}(G)$ is a proof that G is a set
- $0_G : G$ is the unit element
- $-_G : G \to G$ denotes the inversion operation
- $+_G : G \to G \to G$ denotes addition
- $\mathsf{isGroup}(G, 0_G, -_G, +_G)$ is a proof that $+_G$, $-_G$ and 0_G satisfy the usual group laws (i.e. associativity, unit laws, etc.)

We use e.g. $+_G$ rather than $\cdot_G$ since all groups considered in this thesis are abelian. This is however not assumed in any of the proofs in this subsection. We often abuse notation and use G to refer to the underlying type of a group G. We also often omit the subscripts from the unit element as well as the inversion and addition operations.

Furthermore, a morphism from a group G into a group H is a pair

$$(\phi, \mathsf{isMorph}(\phi))$$

where

- $\phi : G \to H$
- $\mathsf{isMorph}(\phi) : (g_1, g_2 : G) \to \phi(g_1 + g_2) \equiv \phi(g_1) + \phi(g_2)$

We say that two groups G and H are isomorphic, denoted $G \cong H$, if there is an equivalence $f : G \simeq H$ such that f is a morphism. So far, groups theory does not differ much from how it is done in standard mathematics. For instance, the following propositions are proved in the same way as we are used to from standard mathematics.

Proposition 4.43. *Let $\phi : G \to H$ be a morphism of groups. We have:*

(i) $-0 \equiv 0$

(ii) $\phi(0) \equiv 0$

(iii) $\phi(-g) \equiv -\phi(g)$*, for all* $g : G$

(iv) *If ϕ has an inverse function $\psi : H \to G$, then also ψ is a morphism.*

We define predicates $\mathsf{isInKer}$ and isInIm, respectively expressing that an element of a group is in the kernel or in the image of some morphism. Let $\phi : G \to H$ be a morphism of groups and let $g : G$ and $h : H$. We define the predicates as follows

$$\mathsf{isInKer}_\phi(g) :\equiv (\phi(g) \equiv 0)$$
$$\mathsf{isInIm}_\phi(h) :\equiv \left\| \sum_{g:G} (\phi(g) \equiv h) \right\|_{-1}$$

Note that we need to truncate in order to distinguish the proposition "h is in the image of ϕ" from the fibre over h. We do not need to truncate when defining $\mathsf{isInKer}$, since this type already is a proposition (since groups are sets). Using this, we can express what it means for ϕ to be injective or surjective. We define the corresponding predicates by

$$\mathsf{isInjective}(\phi) :\equiv ((g : G) \to \mathsf{isInKer}_\phi(g) \to g \equiv 0)$$
$$\mathsf{isSurjective}(\phi) :\equiv ((h : H) \to \mathsf{isInIm}_\phi(h))$$

We get the following lemma.

Lemma 4.44. *Let G and H be groups and suppose there is a surjective and injective morphism $\phi : G \to H$. Then $G \cong H$.*

Proof. Since ϕ is assumed to be a morphism, it is enough to show that it induces an equivalence $G \simeq H$. We construct its inverse. We do this by simultaneously proving that the inverse function is right inverse. Let $F : (h : H) \to \sum_{g:G}(\phi(g) \equiv h)$. Let $h : H$. In order to define $F(h)$, we will want to apply our term $\mathsf{inim}_h : \mathsf{isInIm}_\phi(h)$ to extract an element $g : G$ and a proof $p : \phi(g) \equiv h$. However, in order to use this, we need to apply truncation elimination. Thus we need to prove that $\sum_{g:G}(\phi(g) \equiv h)$ is a proposition. Let $(g_1, p_1), (g_2, p_2) : \sum_{g:G}(\phi(g) \equiv h)$. In order to prove that $(g_1, p_1) \equiv (g_2, p_2)$, it suffices to show that $g_1 \equiv g_2$, since $\phi(g) \equiv h$ is a proposition for any $g : G$. We have

$$\begin{aligned} \phi(g_1 - g_2) &\equiv \phi(g_1) - \phi(g_2) \\ &\equiv h - h \\ &\equiv 0 \end{aligned}$$

Thus $(g_1 - g_2)$ is in the kernel of ϕ. Since ϕ is assumed to be injective, we have $g_1 \equiv g_2$, and we are done. We may now define $F(h)$. By truncation elimination, we may assume that $\mathsf{inim}_h :=\mid g, p \mid_{-1}$ for some $g : G$ and $p : \phi(g) \equiv h$. Hence we may just let

$$F(h) :\equiv (g, p)$$

We may thus define our inverse map $\phi^{-1} : H \to G$ by

$$\phi^{-1}(h) :\equiv \mathsf{fst}(F(h))$$

The fact that $\phi(\phi^{-1}(h)) \equiv h$ is now given by $\mathsf{snd}(F(h))$. We finally prove that $\phi^{-1}(\phi(g)) \equiv g$ for every $g : G$. That is, we want to show that

$$\mathsf{fst}(F(\phi(g))) \equiv g$$

However, F is defined by truncation truncation, and hence does not reduce since $\mathsf{inim}_{\phi(g)}$ is not on the form $|_|_{-1}$. However, $\mathsf{isInim}_{\phi}(\phi(g))$ is a proposition, and thus we may swap $\mathsf{inim}_{\phi(g)}$ for $|g, \mathsf{refl}_{\phi(g)}|_{-1}$, in which case $\mathsf{fst}(F(\phi(g)))$ reduces to g. □

The above lemma will later be used in order to construct group isomorphisms. We will also use arguments from exact sequences. In general, an exact sequence is a collection of groups $G_1, G_2, \ldots$ and morphisms $\phi_i : G_i \to G_{i+1}$ such that $\mathsf{Im}(\phi_i) = \mathsf{Ker}(\phi_{i+1})$ for all $i = 1, 2, \ldots$. In this setting, this equality means that $\mathsf{Im}(\phi_i) \subseteq \mathsf{Ker}(\phi_{i+1})$ and $\mathsf{Ker}(\phi_{i+1}) \subseteq \mathsf{Im}(\phi_i)$. These types are defined by

$$\mathsf{Im}(\phi_i) \subseteq \mathsf{Ker}(\phi_{i+1}) :\equiv ((g : G_{i+1}) \to \mathsf{isInIm}_{\phi_i}(g) \to \mathsf{isInKer}_{\phi_{i+1}}(g))$$
$$\mathsf{Ker}(\phi_{i+1}) \subseteq \mathsf{Im}(\phi_i) :\equiv ((g : G_{i+1}) \to \mathsf{isInKer}_{\phi_{i+1}}(g) \to \mathsf{isInIm}_{\phi_i}(g))$$

We get the following useful way of construct isomorphisms from exact sequence. Let $\mathbf{0}$ denote the trivial group, i.e. $\top$ with the trivial group structure.

Lemma 4.45. *Suppose the following sequence of groups is exact.*

$$\mathbf{0} \xrightarrow{\quad\psi\quad} G \xrightarrow{\quad\phi\quad} H \xrightarrow{\quad\xi\quad} \mathbf{0}$$

Then ϕ induces an isomorphism $G \cong H$.

Proof. Since ϕ is assumed to be a morphism, it suffices, by Lemma 4.44, to show that ϕ is injective and surjective. We prove that it is injective. Let $g : G$ and suppose that g is in the kernel of ϕ. Since the above sequence is exact, we have an element $u : \mathsf{isInIm}_{\psi}(g)$. We are trying to prove a proposition, so we may assume that u is on the form $|u_0|_{-1}$ where $u_0 : \sum_{t:\mathbf{0}}(\psi(t) \equiv g)$. Since $\mathbf{0}$ contains only 0 and we have $\psi(0) \equiv 0$ (since ψ is a morphism), we have that $\psi(\mathsf{fst}(u_0)) \equiv 0$. Thus, we get

$$g \equiv \psi(\mathsf{fst}(u_0)) \equiv 0$$

which is what we wanted to show. The fact that ϕ is surjective follows from a similar argument. □

4.2 Cohomology with Coefficients in $\mathbb{Z}$ – Definition and Group Structure

We are now ready to define cohomology. We recall from traditional algebraic topology (see e.g. [9]) that an *Eilenberg-MacLane space* A is a pointed topological space such that $\pi_n(A)$ is non-trivial for a unique $n \geq 1$. In this case, we may simply write $K(G, n)$ for A, where G is some abelian group such that $\pi_n(X) \cong G$. For every CW complex X, we have that its nth cohomology group with coefficients in G is isomorphic to the set of homotopy classes of functions $X \to K(G, n)$, i.e.

$$H^n(X) \cong \langle X \to K(G, n)\rangle \tag{18}$$

For every CW complex X, let $\hat{H}^n(X) = \langle X \to K(G, n)\rangle$. Given some $f : X \to Y$, precomposition with f gives us a canonical map $f^* : \hat{H}^n(Y) \to \hat{H}^n(X)$. By this, we may see $\hat{H}^n$ as a contravariant functor from the category of CW complexes to the category of abelian groups, as we usually do with H^n. Importantly, we have that (18) is a natural isomorphism between H^n and $\hat{H}^n$.

In HoTT, we have no good way of defining cohomology in the usual way by means of chain complexes. However, it turns out that the definition of cohomology in terms of the right hand side of the above isomorphism is well-behaved. In order to get cohomology with coefficients in $\mathbb{Z}$, we thus need to define a family of types K_n, $n = 1, 2, 3, \ldots$, such that K_n is $(n-1)$-connected and $\Omega^n K_n \simeq \mathbb{Z}$. We know that $\Omega\mathbb{S}^1 \simeq \mathbb{Z}$ and hence, since $\mathbb{S}^1$ is a 1-type, that $\Omega^n \left\|\mathbb{S}^1\right\|_1 \simeq \mathbb{Z}$. Furthermore, we will soon see that for every $n \geq 1$, we have that $\|\mathbb{S}^n\|_n \simeq \Omega \left\|\mathbb{S}^{n+1}\right\|_{n+1}$. Thus, we have by induction that $\Omega^n \|\mathbb{S}^n\|_n \simeq \mathbb{Z}$. Furthermore, we have that $\mathbb{S}^{n+1}$ and hence $\left\|\mathbb{S}^{n+1}\right\|_{n+1}$ is n-connected. Hence, these types are Eilenberg-MacLane spaces. We may therefore define our family of types by

$$K_n :\equiv \left\|\mathbb{S}^{n+1}\right\|_n$$

for $n \geq 1$. We also define

$$K_0 :\equiv \mathbb{Z}$$

Note that we will often consider K_n as a pointed type, pointed by 0 when $n = 0$ and $|\,\mathsf{north}\,|_n$ when $n > 0$. We now copy the definition of cohomology from above. Note that we also have to apply set truncation in order to turn the space into a group.

Definition 4.46 (Cohomology with coefficients in $\mathbb{Z}$)**.** Given a type A, its nth cohomology group $H^n(A)$ is given by the type $\|A \to K_n\|_0$.

In order to give $H^n(A)$ a group structure, we need a way to define the sum of two elements $a, b : H^n(A)$. First, we need to define addition in K_n. That is, we need an appropriate operation $+_k : K_n \to K_n \to K_n$. For this, we first need the following lemma. For every $n \geq 1$, let $\varphi_{\mathbb{S}^n} : \mathbb{S}^n \to \Omega\mathbb{S}^{n+1}$ be defined by

$$\varphi_{\mathbb{S}^n}(x) :\equiv \mathsf{merid}(x) \cdot (\mathsf{merid}(\mathsf{north}))^{-1}$$

Lemma 4.47. *For all $n \geq 0$, we have an equivalence $\sigma_n : K_n \simeq \Omega K_{n+1}$ where*

$$\sigma_0(k) = \mathsf{cong}_{|-|_1}(\mathsf{loop}^k)$$
$$\sigma_{n+1}(|x|_{n+1}) = \mathsf{cong}_{|-|_{n+2}}(\varphi_{\mathbb{S}^{n+1}}(x))$$

Proof. We proceed by induction on n. When $n = 0$, the statement follows by Theorem 2.11.

For $n \geq 1$, we have that

$$\sigma_n \equiv f \circ \|\varphi_{\mathbb{S}^n}\|_n$$

where $f : \|\Omega\mathbb{S}^{n+1}\|_n \simeq \Omega\,\|\mathbb{S}^{n+1}\|_{n+1}$ is the equivalence from Theorem 4.22. Hence we only need to show that $\|\varphi_{\mathbb{S}^n}\|_n$ is an equivalence.

For $n = 1$, we define $d : \Omega\mathbb{S}^2 \to \mathbb{S}^1$ using Theorem 4.37 by

$$d(p) :\equiv \mathsf{subst}^{\mathsf{Hopf}}(p, \mathsf{north}_{\mathbb{S}^1})$$

We consider the composition $d \circ \varphi_{\mathbb{S}^1}$.

$$\begin{aligned}
\mathsf{subst}^{\mathsf{Hopf}}(\varphi_{\mathbb{S}^1}(x), \mathsf{north}_{\mathbb{S}^1}) &\equiv \mathsf{subst}^{\mathsf{Hopf}}(\mathsf{merid}\,(x) \cdot \mathsf{merid}\,(\mathsf{north})^{-1}, \mathsf{north}_{\mathbb{S}^1}) \\
&\equiv \mathsf{subst}^{\mathsf{Hopf}}(\mathsf{merid}\,(\mathsf{north})^{-1} \\
&\qquad (\mathsf{subst}^{\mathsf{Hopf}}(\mathsf{merid}\,(x), \mathsf{north}))) \\
&\equiv \mathsf{subst}^{\mathsf{Hopf}}(\mathsf{merid}\,(x), \mathsf{north}) && \text{by (17)} \\
&\equiv x && \text{by (16)}
\end{aligned}$$

Thus, $d \circ \varphi_{\mathbb{S}^1}$ is just the identity. Consequently, so is the composition of the induced functions $\|d\|_1 \circ \|\varphi_{\mathbb{S}^1}\|_1$. Thus, in order to show that $\|\varphi_{\mathbb{S}^1}\|_1$ is an equivalence, it suffices to show that $\|d\|_1$ is an equivalence. By Proposition 4.30, it is enough to show that d is 1-connected, i.e. that $\|\mathsf{fib}_d(x)\|_1$ is contractible for every $x : \mathbb{S}^1$. We have that $\mathsf{isContr}(\|\mathsf{fib}_d(x)\|_1)$ is a proposition, and so by Lemma 4.39 it is enough to show the statement for $x := \mathsf{north}$. By construction of the Hopf fibration, we have that $\mathsf{fib}_d(\mathsf{north}) \equiv \Omega\mathbb{S}^3$. By Lemma 4.31 and Lemma 4.29, this is 1-connected, and hence $\|\Omega\mathbb{S}^3\|_1$ is contractible, which is what we wanted to show.

When $n \geq 2$, the map $\varphi_{\mathbb{S}^n}$ is just the map from the Freudenthal Suspension Theorem. This map is $2n$-connected, and hence, by Proposition 4.25, n-connected. Thus, $\|\varphi_{\mathbb{S}^n}\|_n$ is n-connected by Proposition 4.30. □

Using Lemma 4.47, we now see that K_n inherits path composition in ΩK_{n+1}. We use this fact to define addition in K_n. We define it for any $a, b : K_n$ by

$$a +_k b :\equiv \sigma_n^{-1}(\sigma_n(a) \cdot \sigma_n(b))$$

When $n = 0$, it is easy to see that $a +_k b \equiv a + b$, where $a + b$ denotes regular addition in the integers.

We define the unit and subtraction in K_n in the obvious ways.

$$0_k :\equiv \mathsf{pt}(K_n)$$
$$-_k\, x :\equiv \sigma_k^{-1}((\sigma_k(x))^{-1})$$

We can now give K_n its algebraic structure.

Proposition 4.48. *Let $x, y, z : K_n$. Then we have paths*

$$\mathsf{lUnit}_k(x) : 0_k +_k x \equiv x \tag{19}$$
$$\mathsf{rUnit}_k(x) : x +_k 0_k \equiv x \tag{20}$$
$$\mathsf{rCancel}_k(x) : x +_k (-_k\, x) \equiv 0_k \tag{21}$$
$$\mathsf{lCancel}_k(x) : (-_k\, x) +_k x \equiv 0_k \tag{22}$$
$$\mathsf{assoc}_k(x, y, z) : (x +_k y) +_k z \equiv x +_k (y +_k z) \tag{23}$$
$$\mathsf{\text{-}distr}(x, y) : -_k\, (x +_k y) \equiv (-_k\, x) +_k (-_k\, y) \tag{24}$$
$$\mathsf{comm}_k(x, y) : x +_k y \equiv y +_k x \tag{25}$$

Proof. The proofs of (19) – (24) all follow the same idea. In essence, the statements can all be translated into the corresponding statements about path composition. For instance, (21) can be proved as follows:

$$\begin{aligned} x +_k (-_k\, x) &:= \sigma_k^{-1}(\sigma_k(x) \cdot \sigma_k(\sigma_k^{-1}((\sigma_k(x))^{-1}))) \\ &\equiv \sigma_k^{-1}(\sigma_k(x) \cdot (\sigma_k(x))^{-1}) \\ &\equiv \sigma_k^{-1}(\mathsf{refl}) \\ &\equiv \sigma_k^{-1}(\sigma_k(\mathsf{pt}(K_n))) \\ &\equiv \mathsf{pt}(K_n) \\ &:= 0_k \end{aligned}$$

The fact that $\sigma_k(\mathsf{pt}(K_n)) \equiv \mathsf{refl}$ is immediate by construction.

We prove (25). Since $K_n \simeq \Omega K_{n+1}$, we are done if we can show that ΩK_{n+1} is commutative w.r.t. path composition. We have that

$$\Omega K_{n+1} \simeq \Omega(\Omega K_{n+2}) := \Omega^2 K_{n+2}$$

and hence we are done by Theorem 4.10. □

We are now ready to give the group structure of cohomology groups. We first define addition, subtraction and the unit. By truncation elimination, we only need to define them for $|\, f\,|_0, |\, g\,|_0 \colon H^n(A)$ for some type A.

$$|\, f\,|_0 +_h |\, g\,|_0 :\equiv |\, \lambda\, x\, .\, f(x) +_k g(x)\,|_0$$
$$-_k\, |\, f\,|_0 :\equiv |\, \lambda\, x\, .\, -_k\, f(x)\,|_0$$
$$0_h :\equiv |\, \lambda\, x\, .\, 0_k\,|_0$$

The group structure follows from Proposition 4.48.

Proposition 4.49. *Let A be a type with $x, y, z : H^n(A)$ for some $n \geq 0$. Then we have paths*

$$\begin{aligned}
&\mathsf{lUnit}_h(x) : 0_k +_h x \equiv x \\
&\mathsf{rUnit}_h(x) : x +_h 0_h \equiv x \\
&\mathsf{rCancel}_h(x) : x +_h (-_h\, x) \equiv 0_h \\
&\mathsf{lCancel}_h(x) : (-_h\, x) +_h x \equiv 0_h \\
&\mathsf{assoc}_h(x, y, z) : (x +_h y) +_h z \equiv x +_h (y +_h z) \\
&-\mathsf{distr}_h(x, y) : -_h\, (x +_h y) \equiv (-_h\, x) +_h (-_h\, y) \\
&\mathsf{comm}_h(x, y) : x +_h y \equiv y +_h x
\end{aligned}$$

We often omit the subscripts k and h and write $x + y$, and $-x$ rather than e.g. $x +_h y$ and $-_h x$.

Proposition 4.50 (Commutativity of ΩK_n)**.** *For every $n \geq 0$, $x : K_n$ and $p, q : \Omega(K_n, x)$, we have that $p \cdot q \equiv q \cdot p$.*

Proof. For $n = 0$, the statement follows immediately from the fact that $\mathbb{Z}$ is a set. We sketch the argument for $n \geq 1$. We use the fact $\Omega K_n \simeq K_{n+1}$. When $x := \mathsf{north}$, the statement follows immediately from Theorem 4.10. We reduce to this case for an arbitrary x. We define a map $f : x \equiv x \to 0_k \equiv 0_k$ by

$$f(p) :\equiv \mathsf{rCancel}_k^{-1}(x) \cdot \mathsf{cong}_{_-x}(p) \cdot \mathsf{rCancel}_k(x)$$

This map is easily verified to be an equivalence, using that the map $\lambda\, y\, .\, y - x$ is an equivalence. Thus it has an inverse f^{-1}. Furthermore, we have $f(p \cdot q) \equiv f(p) \cdot f(q)$, which follows from the fact that

$$\mathsf{cong}_{_-x}(p \cdot q) \equiv \mathsf{cong}_{_-x}(p) \cdot \mathsf{cong}_{_-x}(q)$$

We can now prove the statement. We have

$$\begin{aligned}
p \cdot q &\equiv f^{-1}(f(p \cdot q)) \\
&\equiv f^{-1}(f(p) \cdot f(q)) \\
&\equiv f^{-1}(f(q) \cdot f(p)) && \text{(commutativity of } \Omega(K_n, 0_k)) \\
&\equiv f^{-1}(f(q \cdot p)) \\
&\equiv q \cdot p
\end{aligned}$$

□

4.3 The Mayer-Vietoris Sequence

The Mayer-Vietoris sequence is a long exact sequence of cohomology groups which is incredibly useful for characterising cohomology groups (see e.g. [9]). In particular, for $n \geq 1$, it will give us isomorphisms

$$\begin{aligned}
H^n(A \vee B) &\cong H^n(A) \times H^n(B) \\
H^n(\mathbb{S}^n) &\cong \mathbb{Z}
\end{aligned}$$

We follow the construction in [6] closely. The goal is to construct the following long exact sequence given types A, B and C, functions $f : C \to A$ and $g : C \to B$ and $D :\equiv A \sqcup^{C}_{f,g} B$.

$$\begin{array}{ccccc}
H^0(D) & \xrightarrow{i_0} & H^0(A) \times H^0(B) & \xrightarrow{\Delta_0} & H^0(C) \\
& & \xleftarrow{d_0} & & \\
H^1(D) & \xrightarrow{i_1} & H^1(A) \times H^1(B) & \xrightarrow{\Delta_1} & H^1(C) \\
& & \xleftarrow{d_1} & & \\
H^2(D) & \dashrightarrow & \cdots & &
\end{array}$$

Figure 4: The Mayer-Vietrois Sequence

where i_n, Δ_n and d_n are families of morphisms will soon be defined.

We now give the definitions of i_n, Δ_n and d_n. First, we note that any function $h : X \to Y$, we can an induced function $h^* : H^n(Y) \to H^n(X)$ by

$$h^*(|\,\alpha\,|_0) \equiv |\,\alpha \circ h\,|_0$$

Using this, we can easily define $i_n : H^n(D) \to H^n(A) \times H^n(B)$ and $\Delta_n : H^n(A) \times H^n(B) \to H^n(C)$. We define the functions by

$$i_n(\alpha) :\equiv (\mathsf{inl}^*(\alpha), \mathsf{inr}^*(\alpha))$$
$$\Delta_n(\alpha, \beta) :\equiv f^*(\alpha) - g^*(\beta)$$

We now define $d_n : H^n(C) \to H^{n+1}(D)$ by

$$d_n(|\,\alpha\,|_0) \equiv |\,d'(\alpha)\,|_0$$

where $d'_n : (C \to K_n) \to D \to K_{n+1}$ is defined by

$$d'_n(\alpha, \mathsf{inl}(a)) :\equiv 0$$
$$d'_n(\alpha, \mathsf{inr}(b)) :\equiv 0$$
$$d'_n(\alpha, (\mathsf{push}(c))(i)) :\equiv (\sigma_n(\alpha(c)))(i)$$

Proposition 4.51. *The maps i_n, Δ_n and d_n are morphisms for all $n \geq 0$.*

Proof. For i_n, the proposition is trivial (modulo induction on n and truncation elimination). For Δ_n, the proposition is proved easily by some elementary algebra in K_n. For d_n, we need to do some more work. By truncation elimination, we are done if we can prove that

$$d_n(|\,f\,|_0 + |\,g\,|_0) \equiv d_n(|\,f\,|_0) + d_n(|\,g\,|_0)$$

i.e.

$$|\, d'(\lambda\, x\, .\, (f(x) + g(x)))\, |_0 \equiv |\, \lambda\, x\, .\, (d'(f))(x) + (d'(g))(x)\, |_0$$

In particular, we are done if we can show that

$$d'((\lambda\, x\, .\, (f(x) + g(x))), \delta) \equiv d'(f, \delta) + d'(g, \delta)$$

for every $\delta : D$. We induct on δ. For the base cases, we only need to prove that $0 \equiv 0 + 0$. In both cases, we give the path $\mathsf{lUnit}_k^{-1}(0) : 0 \equiv 0 + 0$. When $\delta = (\mathsf{push}(c))(i)$, we need to fill the following square.

$$\begin{array}{ccc}
0{+}0 & \xrightarrow{(\sigma_n(f(c)))(i)+(\sigma_n(g(c)))(i)} & 0{+}0 \\
\uparrow {\scriptstyle (\mathsf{lUnit}_k(0))(\sim j)} & & \uparrow {\scriptstyle (\mathsf{lUnit}_k(0))(\sim j)} \\
0 & \xrightarrow[(\sigma_n(f(c)+g(c)))(i)]{} & 0
\end{array}$$

The top path could be rephrased as

$$\mathsf{cong}^2_{+}(\sigma_n(f(c)), \sigma_n(g(c)))$$

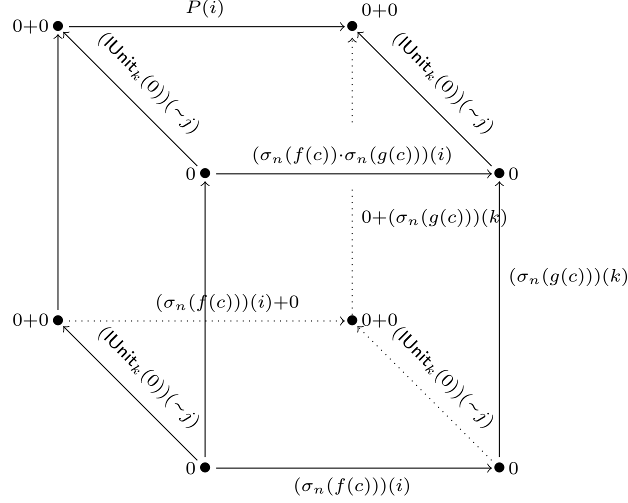
By functoriality of cong^2, this is the same as

$$P :\equiv \mathsf{cong}_{(_+0)}(\sigma_n(f(c))) \cdot \mathsf{cong}_{(0+_)}(\sigma_n(g(c)))$$

For the bottom path, we have that $\sigma_n(f(c) + g(c)) \equiv \sigma_n(f(c)) \cdot \sigma_n(g(c))$. Hence it is enough to fill the following square.

$$\begin{array}{ccc}
0{+}0 & \xrightarrow{P(i)} & 0{+}0 \\
\uparrow {\scriptstyle (\mathsf{lUnit}_k(0))(\sim j)} & & \uparrow {\scriptstyle (\mathsf{lUnit}_k(0))(\sim j)} \\
0 & \xrightarrow[(\sigma_n(f(c))\cdot\sigma_n(g(c)))(i)]{} & 0
\end{array}$$

We see it as the lid of the following cube.



The left-hand side is trivial. The front and the back are filled by the filler of path composition. The right-hand side is is just

$$(\mathsf{lUnit}_k((\sigma^n(g(c))))(k))(\sim j)$$

We now need to fill the bottom. This should be filled just like the right-hand side, but the problem is that we would need $\mathsf{lUnit}_k(0)$ to be swapped for $\mathsf{rUnit}_k(0)$. However, we can prove that

$$\mathsf{lUnit}_k(0) \equiv \mathsf{rUnit}_k(0)$$

This follows easily from the fact that addition in K_n is commutative. Thus we may swap $\mathsf{lUnit}_k(0)$ for $\mathsf{rUnit}_k(0)$, and hence the bottom may be filled in the same way as the right-hand side.

□

We finally prove that the sequence in fact is exact.

Theorem 4.52. *The Mayer-Vietoris sequence (Figure 4) is exact. That is, for every $n \geq 0$, the maps i_n, Δ_n and d_n satisfy the following*

$$\mathsf{Im}(i_n) \subseteq \mathsf{Ker}(\Delta_n) \tag{26}$$

$$\mathsf{Ker}(\Delta) \subseteq \mathsf{Im}(i_n) \tag{27}$$

$$\mathsf{Im}(\Delta_n) \subseteq \mathsf{Ker}(d_n) \tag{28}$$

$$\mathsf{Ker}(d_n) \subseteq \mathsf{Im}(\Delta_n) \tag{29}$$

$$\mathsf{Im}(d_n) \subseteq \mathsf{Ker}(i_{n+1}) \tag{30}$$

$$\mathsf{Ker}(i_{n+1}) \subseteq \mathsf{Im}(d_n) \tag{31}$$

Proof. For the proofs of (26) to (30), we refer to the proofs in [6] since these do not leave much room for a cubical approach. We prove (31). Suppose $\mid F \mid_0$ is in the kernel of i_{n+1} for $F : D \to K_{n+1}$. We are done if we can construct an element $\mid X, \mathsf{id} \mid_{-1}$: $\left\| \sum_{\delta : H^n(D)} (d_n(\delta) = \mid F \mid_0) \right\|_{-1}$. We have that $i_{n+1}(\mid F \mid_0) \coloneqq (\mid \mathsf{inl}^*(F)) \mid_0, \mid \mathsf{inr}^*(F) \mid_0) \equiv (0, 0)$ by assumption. Hence we have

$$\mid \mathsf{inl}(F) \mid_0 \equiv \mid \mathsf{inr}(F) \mid_0 \equiv 0 \coloneqq \mid \lambda\, x\, .\, 0_k \mid_0$$

Since we are proving a proposition, Corollary 4.23 gives us paths

$$p_l : \mathsf{inl}^*(F) \equiv \lambda\, x\, .\, 0_k$$
$$p_r : \mathsf{inr}^*(F) \equiv \lambda\, x\, .\, 0_k$$

We define $X :\equiv \mid \lambda\, c\, .\, \sigma_n^{-1}(P(c)) \mid_0$, where P is defined by the following square:

$$\begin{array}{ccc} 0 & \xrightarrow{(P(c))(i)} & 0 \\ {\scriptstyle (p_l(j))(f(c))} \uparrow & & \uparrow {\scriptstyle (p_r(j))(g(c))} \\ F(\mathsf{inl}(f(c))) & \xrightarrow[F((\mathsf{push}(c))(i))]{} & F(\mathsf{inr}(g(c))) \end{array}$$

Let $G :\equiv \lambda\, c\, .\, \sigma_n^{-1}(P(c))$ We need to show that $d_n(\mid G \mid_0) \equiv \mid F \mid_0$. We have that $d_n(\mid G \mid_0) \coloneqq \mid d_n'(G) \mid_0$. Thus it is enough to show that

$$(d_n'(G))(\delta) \equiv F(\delta)$$

for every $\delta : D$. We induct on δ. For $\delta = \mathsf{inl}(x)$ and $\delta = \mathsf{inr}(x)$, we give the paths

$$\mathsf{cong}_{\lambda\, h\, .\, h(x)}(p_l^{-1}) : 0 \equiv F(\mathsf{inl}(x))$$
$$\mathsf{cong}_{\lambda\, h\, .\, h(x)}(p_r^{-1}) : 0 \equiv F(\mathsf{inr}(x))$$

For $\delta = (\mathsf{push}(c))(i)$, we need to fill the following square:

$$\begin{array}{ccc} F(\mathsf{inl}(f(c))) & \xrightarrow{F((\mathsf{push}(c))(i))} & F(\mathsf{inr}(g(c))) \\ {\scriptstyle (p_l(\sim j))(f(c))} \uparrow & & \uparrow {\scriptstyle (p_r(\sim j))(g(c))} \\ 0 & \xrightarrow[(\sigma_n(\sigma_n^{-1}(P(c))))(i)]{} & 0 \end{array}$$

We swap the bottom for $(P(c))(i)$, since $\sigma^n(\sigma_n^{-1})(P(c)) \equiv P(c)$. This gives us back the square used in the definition of P, flipped vertically. Hence we simply use the filler of the square defining P, and we are done. □

4.4 Characterisations of Cohomology Groups

We are now ready to characterise some concrete cohomology groups. Our goal is to be able to characterise $H^n(\mathbb{T}^2)$ and $H^n(\mathbb{S}^1 \vee \mathbb{S}^1 \vee \mathbb{S}^2)$ for $n = 0, 1, 2$. In order to do this, we first need to know some things about the cohomology groups of $\top$ and $\mathbb{S}^n$. Before we start, we prove the following lemma.

Lemma 4.53. *For every* 0*-connected and pointed type* (A, a_0)*, we have that* $H^0(A) \cong \mathbb{Z}$*.*

Proof. We first need to show that $H^0(A) \simeq \mathbb{Z}$ as types. We define $f : H^0(A) \to \mathbb{Z}$ and $g : \mathbb{Z} \to H^0(A)$ by

$$\begin{aligned} f(|\, h \,|_0) &:\equiv h(a_0) \\ g(b) &:\equiv |\, \lambda\, x \,.\, b \,|_0 \end{aligned}$$

Note that truncation elimination is permissible in the definition of f, since $\mathbb{Z}$ is a set. We have that $f(g(b)) \coloneqq b$ for every $b : \mathbb{Z}$. So we are left to show that

$$\underbrace{g(f(|\, h \,|_0))}_{|\lambda\, x \,.\, h(a_0)|_0} \equiv |\, h \,|_0$$

for every $h : A \to \mathbb{Z}$. We are done if we can show the following slightly stronger statement.

$$(\lambda\, x \,.\, h(a_0)) \equiv h$$

Let $a : A$. We wish to construct a path $h(a_0) \equiv h(a)$. Naturally, this would follow if we could construct a path $a \equiv a_0$. By Corollary 4.23, it is enough to show that $|\, a \,|_0 \equiv |\, a_0 \,|_0$. This is immediate, since A is 0-connected, i.e. $\|A\|_0$ is contractible. Thus, we have shown that $H^0(A) \simeq \mathbb{Z}$.

We now need to show that either f or g is a morphism. The fact that g is a morphism follows immediately from the fact that addition in $+_k$ agrees with regular integer addition. Hence, we have shown that $H^0(A) \cong \mathbb{Z}$. □

4.4.1 The Unit Type

Unsurprisingly, characterising the cohomology groups of $\top$ is straightforward. We first note that $\top$ is pointed and n-connected for any $n \geq -2$ since it is contractible. In particular, it is 0-connected. Using Lemma 4.53, we thus get that $H^0(\top) \cong \mathbb{Z}$. In general, we get the following behaviour for $H^n(\top)$.

Proposition 4.54.

$$H^n(\top) \cong \begin{cases} \mathbb{Z} & n = 0 \\ \mathbf{0} & n \geq 1 \end{cases}$$

Proof. It remains to show that $H^n(\top) \cong \mathbf{0}$ for $n \geq 1$. It suffices to show that $H^n(\top)$ is contractible. We have

$$\begin{aligned} H^n(\top) &:= \|\top \to \|\mathbb{S}^n\|_n\|_0 \\ &\simeq \|\|\mathbb{S}^n\|_n\|_0 \\ &\simeq \|\mathbb{S}^n\|_0 && \text{(Theorem 4.20)} \end{aligned}$$

Since $\mathbb{S}^n$ is $(n-1)$-connected and $n \geq 1$, we have that $\mathbb{S}^n$ is 0-connected. Hence we $\|\mathbb{S}^n\|_0$ is contractible, and we are done. □

4.4.2 Spheres

In this section, we will often use the definition of $\mathbb{S}^{n+1}$ as the pushout $\top \sqcup^{\mathbb{S}^n} \top$. We start off with the 0th cohomology groups of $\mathbb{S}^n$.

Proposition 4.55.

$$H^0(\mathbb{S}^n) \cong \begin{cases} \mathbb{Z} \times \mathbb{Z} & n = 0 \\ \mathbb{Z} & n \geq 1 \end{cases}$$

Proof. For $n \geq 1$, this follows immediately from Lemma 4.53. We are left to show that that $H^0(\mathbb{S}^0) \cong \mathbb{Z} \times \mathbb{Z}$. We represent the maps $\mathbb{S}^0 \to \mathbb{Z}$ by $f_{a,b}$, where $a, b : \mathbb{Z}$ and $f_{a,b}$ is the map sending north to a and south to b. Clearly, this gives us an equivalence $(\mathbb{S}^0 \to \mathbb{Z}) \simeq \mathbb{Z} \times \mathbb{Z}$. By truncating, we get

$$H^0(\mathbb{S}^0) \simeq \|\mathbb{Z} \times \mathbb{Z}\|_0 \simeq \mathbb{Z} \times \mathbb{Z}$$

The fact that this equivalence is a morphism is immediate from the fact that $+_k$ agrees with integer addition. □

We now turn to $H^1(\mathbb{S}^1)$. This becomes somewhat trickier to compute than usual, since we do not have the reduced version of the Mayer-Vietoris sequence at our disposal. Nevertheless, the sequence can still be used by carefully tracing the maps. We can also characterise $H^1(\mathbb{S}^1)$ directly without using Mayer-Vietoris. We give both proofs below.

Proposition 4.56. $H^1(\mathbb{S}^1) \cong \mathbb{Z}$

Proof by Mayer-Vietoris. From Mayer-Vietoris, we get the following exact sequence

$$\mathbb{Z} \times \mathbb{Z} \xrightarrow{\Delta_0'} H^0(\mathbb{S}^0) \xrightarrow{d_0} H^1(\mathbb{S}^1) \xrightarrow{i_1'} \mathbf{0} \times \mathbf{0}$$

Where Δ_0' and i_1' are just Δ_0 and i_1 factored through the isomorphisms between the corresponding cohomology groups and the groups above. For $a : \mathbb{Z}$, define $F_a, G_a : \mathbb{S}^0 \to \mathbb{Z}$ as follows

$$\begin{aligned} F_a(x) &:\equiv a \\ G_a(\mathsf{north}) &:\equiv a \\ G_a(\mathsf{south}) &:\equiv 0 \end{aligned}$$

Define $d : H^0(S^0) \to H^1(\mathbb{S}^1)$ by

$$d(|\, f\,|_0) :\equiv d_0(|\, G_{f(\mathsf{north})}\,|_0)$$

We first study the image of Δ_0'. Let $a, b : \mathbb{Z}$. It is easy to see that $\Delta_0'(a,b) \equiv |\, F_{a+b}\,|_0$. Since $\mathsf{Ker}(d_0)$ is precisely $\mathsf{Im}(\Delta_0')$, we have that the $\mathsf{Ker}(d_0)$ consists of the elements $|\, F_a\,|_0$ for every $a : \mathbb{Z}$. Hence, $d(|\, f\,|_0) \equiv 0$ iff $f(\mathsf{north}) \equiv 0$. We now look at the composite map $\phi : \mathbb{Z} \to H^1(\mathbb{S}^1)$

$$\begin{aligned} \phi(a) &:\equiv d(|\, G_a\,|_0) \\ &\coloneqq d_0(|\, G_a\,|_0) \end{aligned}$$

By the above argument, the kernel of ϕ is trivial. Thus, we only need to show that ϕ is surjective. Let $x : H^0(\mathbb{S}^1)$. By the surjectivity of d_0, we may assume there is some $f : \mathbb{S}^0 \to \mathbb{Z}$ such that $d_0(|\, f\,|_0) \equiv x$. We have

$$\begin{aligned} x &\equiv d_0(|\, f\,|_0) + \underbrace{d_0(|\, F_{f(\mathsf{south})}\,|_0)}_{0} \\ &\equiv d_0(|\, f\,|_0 + |\, F_{f(\mathsf{south})}\,|_0) \\ &\equiv d_0(|\, G_{f(\mathsf{north})}\,|) \\ &\coloneqq \phi(f(\mathsf{north})) \end{aligned}$$

So ϕ is surjective. Furthermore it is clearly a morphism, since d_0 is a morphism. Thus, we have shown that $H^1(\mathbb{S}^1) \cong \mathbb{Z}$. □

As promised, we also give a direct proof. This uses the winding numbers. In order to do this, we interpret $\mathbb{S}^1$ under the loop/base-definition. We first need the following technical lemma:

Lemma 4.57. *For any $s : \mathbb{S}^1$, let $B_s : \Omega(\mathbb{S}^1, s) \to \Omega(\mathbb{S}^1, \mathsf{base})$. Let A be a type pointed by $a : A$ and suppose we have maps*

$$\begin{aligned} f, g &: \mathbb{S}^1 \to_* \Omega A \\ F &: \Omega A \to \mathbb{S}^1 \end{aligned}$$

Let $x, y : \mathbb{S}^1$. Then

$$\begin{aligned} &B_{F(f(\mathsf{base})\cdot g(\mathsf{base}))}(\mathsf{cong}^2_{\lambda x\, \lambda y\, .\, F(f(x)\cdot g(x))}(\mathsf{loop}, \mathsf{loop}) \\ &\equiv B_{F(f(\mathsf{base}))}(\mathsf{cong}_{F\circ f}(\mathsf{loop})) \cdot B_{F(g(\mathsf{base}))}(\mathsf{cong}_{F\circ g}(\mathsf{loop})) \end{aligned}$$

Proof. We first outline the proof and explain the steps afterwards.

$$\begin{aligned} &B_{F(f(\mathsf{base})\cdot g(\mathsf{base}))}(\mathsf{cong}^2_{\lambda x\, \lambda y\, .\, F(f(x)\cdot g(x))}(\mathsf{loop}, \mathsf{loop}) \\ &\equiv B_{F(f(\mathsf{base})\cdot g(\mathsf{base}))}(\mathsf{cong}_{\lambda x\, .\, F(f(x)\cdot g(\mathsf{base}))}(\mathsf{loop}) \cdot \mathsf{cong}_{\lambda x\, .\, F(f(\mathsf{base})\cdot g(x))}(\mathsf{loop})) \\ &\equiv (B_{F(f(\mathsf{base})\cdot g(\mathsf{base}))}(\mathsf{cong}_{\lambda x\, .\, F(f(x)\cdot g(\mathsf{base}))}(\mathsf{loop}))) \\ &\qquad \cdot\ (B_{F(f(\mathsf{base})\cdot g(\mathsf{base}))}(\mathsf{cong}_{\lambda x\, .\, F(f(\mathsf{base})\cdot g(x))}(\mathsf{loop}))) \\ &\equiv B_{F(f(\mathsf{base}))}(\mathsf{cong}_{F\circ f}(\mathsf{loop})) \cdot B_{F(g(\mathsf{base}))}(\mathsf{cong}_{F\circ g}(\mathsf{loop})) \end{aligned}$$

The first equality comes from functionality of cong^2. The last equality is just deletion of $f(\mathsf{base})$ and $g(\mathsf{base})$, using the fact that f and g are pointed maps. The second equality comes from the fact that B_s preserves path composition. We prove this. Let $x : \mathbb{S}^1$ and $p, q : x \equiv x$. We want to show that $B_x(p \cdot q) \equiv B_x(p) \cdot B_x(q)$. We know that $\mathbb{S}^1$ is a 1-type, and hence we are trying to prove a proposition. Thus, we may assume $x := \mathsf{base}$. But then the statement follows trivially, since B_{base} is just the identity function. □

We get the following technical lemma as a corollary.

Lemma 4.58. *Let $f, g : \mathbb{S}^1 \to \mathbb{S}^1$. Then*

$$B_{f(\mathsf{base})+g(\mathsf{base})}(\mathsf{cong}_{\lambda x\,.\,f(x)+g(x)})(\mathsf{loop}) \tag{32}$$
$$\equiv B_{f(\mathsf{base})}(\mathsf{cong}_f(\mathsf{loop})) \cdot B_{g(\mathsf{base})}(\mathsf{cong}_g(\mathsf{loop})) \tag{33}$$

where $+$ denotes $+_k$, modulo the fact that $K_1 \simeq \mathbb{S}^1$.

Proof. First, let $x, y : \mathbb{S}^1$ such that $f(\mathsf{base}) \equiv x$ and $g(\mathsf{base}) \equiv y$ – we can prove the existence of such elements by letting $x := f(\mathsf{base})$ and $y := g(\mathsf{base})$. We want to prove (33), which is a 2-dimensional path over $\mathbb{S}^1$. Since $\mathbb{S}^1$ is a 1-type, (33) is a proposition. Thus, we may assume that $x := y := \mathsf{base}$. Thus we also have $f(\mathsf{base}) \equiv g(\mathsf{base}) \equiv \mathsf{base}$. With some abuse of notation, we note that

$$\begin{aligned}
\sigma_1 \circ f &: \mathbb{S}^1 \to \Omega K_2\\
\sigma_1 \circ g &: \mathbb{S}^1 \to \Omega K_2\\
\sigma_1^{-1} &: \Omega K_2 \to \mathbb{S}^1
\end{aligned}$$

and that

$$\begin{aligned}
(\sigma_1 \circ f)(\mathsf{base}) &\equiv \sigma_1(\mathsf{base}) \equiv \mathsf{refl}\\
(\sigma_1 \circ g)(\mathsf{base}) &\equiv \sigma_1(\mathsf{base}) \equiv \mathsf{refl}
\end{aligned}$$

Hence, we may apply Lemma 4.57. We get

$$\begin{aligned}
&B_{f(\mathsf{base})+g(\mathsf{base})}(\mathsf{cong}_{\lambda x\,.\,f(x)+g(x)})(\mathsf{loop})\\
:= &B_{f(\mathsf{base})+g(\mathsf{base})}\left(\mathsf{cong}^2_{\lambda x\,\lambda y\,.\,f(x)+g(y)}(\mathsf{loop}, \mathsf{loop})\right)\\
:= &B_{f(\mathsf{base})+g(\mathsf{base})}\left(\mathsf{cong}^2_{\lambda x\,\lambda y\,.\,\sigma_1^{-1}(\sigma_1(f(x))+\sigma_1(g(y)))}(\mathsf{loop}, \mathsf{loop})\right)\\
\equiv &B_{\sigma_1^{-1}(\sigma_1(f(\mathsf{base})))}\left(\mathsf{cong}_{\sigma_1^{-1}\circ\sigma_1\circ f}(\mathsf{loop})\right)\\
&\quad\cdot B_{\sigma_1^{-1}(\sigma_1(g(\mathsf{base})))}\left(\mathsf{cong}_{\sigma_1^{-1}\circ\sigma_1\circ g}(\mathsf{loop})\right) \qquad \text{(Lemma 4.57)}\\
\equiv &B_{f(\mathsf{base})}(\mathsf{cong}_f(\mathsf{loop})) \cdot B_{g(\mathsf{base})}(\mathsf{cong}_g(\mathsf{loop}))
\end{aligned}$$

□

Direct proof of Proposition 4.56. For this proof we use the $\mathsf{loop}/\mathsf{base}$ definition of $\mathbb{S}^1$. We have that $H^1(\mathbb{S}^1) \simeq \left\|(\mathbb{S}^1 \to \mathbb{S}^1)\right\|_0$, since $\mathbb{Z}$ is a set. We begin by proving that

$$(\mathbb{S}^1 \to \mathbb{S}^1) \simeq \mathbb{S}^1 \times \mathbb{Z} \tag{34}$$

We define $F : (\mathbb{S}^1 \to \mathbb{S}^1) \to \mathbb{S}^1 \times \mathbb{Z}$ by

$$F(f) :\equiv (f(\mathsf{base}), \mathsf{winding}(B_{f(\mathsf{base})}(\mathsf{cong}_f(\mathsf{loop}))))$$

We define its inverse $G : \mathbb{S}^1 \times \mathbb{Z} \to (\mathbb{S}^1 \to \mathbb{S}^1)$ by

$$\begin{aligned}(G(x,a))(\mathsf{base}) &:\equiv x \\ (G(x,a))(\mathsf{loop}(i)) &:\equiv (B_x^{-1}(\mathsf{winding}^{-1}(a)))(i)\end{aligned}$$

The fact that these maps cancel out follows from the fact that B_x and $\mathsf{winding}$ are equivalences. This proves (34). We now get

$$\begin{aligned}H^1(\mathbb{S}^1) := \left\|\mathbb{S}^1 \to \mathbb{S}^1\right\|_0 &\simeq \left\|\mathbb{S}^1 \times \mathbb{Z}\right\|_0 \\ &\simeq \left\|\mathbb{S}^1\right\|_0 \times \left\|\mathbb{Z}\right\|_0 \\ &\simeq \mathbb{Z}\end{aligned}$$

The last equivalence follows from the facts that $\mathbb{S}^1$ is 0-connected and that $\mathbb{Z}$ is a set. The induced equivalence $F' : H^1(\mathbb{S}^1) \to \mathbb{Z}$ is the map

$$F'(|\,f\,|_0) :\equiv \mathsf{winding}(B_{f(\mathsf{base})}(\mathsf{cong}_f(\mathsf{loop}))$$

We show that this map is a morphism. Let $f, g : \mathbb{S}^1 \to \mathbb{S}^1$. We need to show that, with some abuse of notation

$$F'(|\,f\,|_0 + |\,g\,|_0) \equiv F'(|\,f\,|_0) + F'(|\,g\,|_0)$$

We have that

$$\begin{aligned}F'(|\,f\,|_0 + |\,g\,|_0) &:= F'(|\,\lambda\, x\,(f(x) + g(x))\,|_0) \\ &:= \mathsf{winding}(B_{f(\mathsf{base})+g(\mathsf{base})}(\mathsf{cong}_{\lambda\, x\,.\,(f(x)+g(x))}(\mathsf{loop}))) \\ &:= \mathsf{winding}(B_{f(\mathsf{base})+g(\mathsf{base})}(\mathsf{cong}_{\sigma_1^{-1}\circ(\lambda\, x\,.\,(\sigma_1(f(x))\cdot\sigma_1(g(x))))}(\mathsf{loop})))\end{aligned}$$

Since $\mathsf{winding}$ is a morphism, we are done if we can show that

$$\begin{aligned}&B_{f(\mathsf{base})+g(\mathsf{base})}(\mathsf{cong}_{\sigma_1^{-1}\circ(\lambda\, x\,.\,(\sigma_1(f(x))\cdot\sigma_1(g(x))))}(\mathsf{loop})) \\ &\equiv B_{f(\mathsf{base})}(\mathsf{cong}_f(\mathsf{loop})) \cdot B_{g(\mathsf{base})}(\mathsf{cong}_g(\mathsf{loop}))\end{aligned}$$

But this is precisely what we proved in Lemma 4.58. Thus, F' is morphism and we have that $H^1(\mathbb{S}^1) \cong \mathbb{Z}$. □

The higher cohomology groups now follow easily by induction, using the Mayer-Vietoris sequence.

Proposition 4.59. *$H^n(\mathbb{S}^n) \cong \mathbb{Z}$ for every $n \geq 1$.*

Proof. We induct on n. We know that $H^1(\mathbb{S}^1) \cong \mathbb{Z}$. Suppose $H^n(\mathbb{S}^n) \cong \mathbb{Z}$. We prove that $H^{n+1}(\mathbb{S}^{n+1}) \cong \mathbb{Z}$. Using that $H^m(\top) \cong \mathbf{0}$ for every $m \geq 1$, the Mayer-Vietoris sequence gives us an exact sequence

$$\mathbf{0} \longrightarrow H^n(\mathbb{S}^n) \longrightarrow H^{n+1}(\mathbb{S}^{n+1}) \longrightarrow \mathbf{0}$$

Hence, $H^{n+1}(\mathbb{S}^{n+1}) \cong H^n \cong \mathbb{Z}$ and we are done. □

We also have that $H^n(\mathbb{S}^m) \cong \mathbf{0}$ whenever $m \neq n$. and $n, m \geq 1$ This follows easily from the Mayer-Vietoris sequence. Only some special cases are formalised in Agda, since inequalities in theorem statements are somewhat clumsy to work with. We do not prove this fact here. The case when $n = 1$ follows easily by tracing the maps in the Mayer-Vietoris sequence, and the case when $n \geq 1$ follows easily by induction.

4.4.3 Wedges of Spheres

Our goal in this section is to compute the first three cohomology groups of the wedge of one sphere and two circles. Fortunately, this is easy given the previous section. For the 0th cohomology group, we need to following lemma.

Lemma 4.60. *If (A, a_0) and (B, b_0) are two pointed and 0-connected types, then $A \vee B$ is pointed and 0-connected.*

Proof. The wedge of A and B is trivially pointed by $\mathsf{inl}(a_0)$. We choose $|\mathsf{inl}(a_0)|_0$ as the centre of contraction. Let $|x|_0$: $\|A \vee B\|_0$. We want to show that $|\mathsf{inl}(a_0)|_0 \equiv |x|_0$. This is a proposition, and so we only need to show it when $x := \mathsf{inl}(a)$ and when $x := \mathsf{inr}(b)$ for some $a : A$ and $b : B$. First, assume that $x := \mathsf{inl}(a)$. Since A is connected, we have that $|a|_0 \equiv |a_0|_0$. By Corollary 4.23, we may assume that $a \equiv a_0$. Thus, we also get $|\mathsf{inl}(a)|_0 \equiv |\mathsf{inl}(a_0)|_0$. Now, assume that $x := \mathsf{inr}(b)$. We have that $\mathsf{push}(*) : \mathsf{inl}(a_0) \equiv \mathsf{inr}(b_0)$, and hence we only need to show that $\mathsf{inr}(b_0) \equiv \mathsf{inr}(b)$. Again, this follows from the fact that B is 0-connected. □

For the higher cohomology groups, we only need the following fact.

Proposition 4.61. *Let A and B be two pointed types. Then $H^n(A \vee B) \cong H^n(A) \times H^n(B)$ for all $n \geq 1$.*

Proof. We proceed by induction on n. For the inductive step, the statement follows immediately from the Mayer-Vietoris sequence. Thus it is enough to prove it for $n = 1$. Mayer-Vietoris gives us an exact sequence

$$H^0(\top) \xrightarrow{d_0} H^1(A \vee B) \xrightarrow{i_1} H^1(A) \times H^1(B) \xrightarrow{\Delta_1} H^1(\top)$$

Since $H^1(\top)$ is trivial, i_1 is surjective. Thus, we only need to show that it is injective. We show this by showing that $d_0 : H^0(\top) \to H^1(A \vee B)$ is the constant 0-map (hence proving that i_1 has a trivial kernel). Let $x : H^0(\top)$. We are trying to prove a proposition, and consequently we may assume that $x :=| f |_0$ for some $f : \top \to \mathbb{Z}$. The function f is uniquely determined by the value $f(*)$. Clearly, it is in the image of Δ_0, since

$$\Delta_0(|\lambda\, x\, .\, f(*)\,|_0, |\,\lambda\, x\, .\, 0\,|_0) \equiv | f |_0$$

Since $\mathsf{Im}(\Delta_0) \subseteq \mathsf{Ker}(d_0)$, we get that $d_0(f) \equiv 0$. This concludes the proof. $\square$

Using this and our characterisation of the spheres, we get the full characterisation of the cohomology groups of $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$.

Proposition 4.62.

$$H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \cong \begin{cases} \mathbb{Z} & n = 0, 2 \\ \mathbb{Z} \times \mathbb{Z} & n = 1 \\ \mathbf{0} & n \geq 3 \end{cases}$$

Proof. When $n = 0$, we have that both $\mathbb{S}^2$ and $\mathbb{S}^1$ are connected and pointed types. Thus, by Lemma 4.60, $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ is connected and pointed. Hence the statement follows from Lemma 4.53.

When $n \geq 1$ we only need Proposition 4.61. For $n = 1$, we get

$$\begin{aligned} H^1(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) &\cong H^1(\mathbb{S}^2) \times H^1(\mathbb{S}^1) \times H^1(\mathbb{S}^1) \\ &\cong \mathbf{0} \times \mathbb{Z} \times \mathbb{Z} \\ &\cong \mathbb{Z} \times \mathbb{Z} \end{aligned}$$

For $n = 2$, we get

$$\begin{aligned} H^2(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) &\cong H^2(\mathbb{S}^2) \times H^2(\mathbb{S}^1) \times H^2(\mathbb{S}^1) \\ &\cong \mathbb{Z} \times \mathbf{0} \times \mathbf{0} \\ &\cong \mathbb{Z} \end{aligned}$$

Finally, for $n \geq 3$, we get

$$\begin{aligned} H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) &\cong H^n(\mathbb{S}^2) \times H^n(\mathbb{S}^1) \times H^n(\mathbb{S}^1) \\ &\cong \mathbf{0} \times \mathbf{0} \times \mathbf{0} \\ &\cong \mathbf{0} \end{aligned}$$

$\square$

4.4.4 The Torus

Finally, we look at the first three cohomology groups of the tours, and check that they agree with those of $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$. We define the torus as $\mathbb{T}^2 :\equiv \mathbb{S}^1 \times \mathbb{S}^1$. The 0th cohomology group is easy to find.

Proposition 4.63. $H^0(\mathbb{T}^2) \cong \mathbb{Z}$

Proof. By Lemma 4.60, we are done if we can show that $\mathbb{T}^2$ is connected. But this is immediate, since $\|\mathbb{S}^1\|_0$ is contractible and

$$\begin{aligned}\|\mathbb{T}^2\|_0 &:= \|\mathbb{S}^1 \times \mathbb{S}^1\|_0 \\ &\simeq \|\mathbb{S}^1\|_0 \times \|\mathbb{S}^1\|_0\end{aligned}$$

□

The remaining two groups are somewhat harder to work with. In particular, Mayer-Vietoris is not of much help. For $H^1(\mathbb{T}^2)$, however, we have already done most of the work in the direct proof of Proposition 4.56.

Proposition 4.64. $H^1(\mathbb{T}^2) \cong \mathbb{Z} \times \mathbb{Z}$

Proof. We use K_1 and $\mathbb{S}^1$ interchangeably, since they are identical. We first note the following:

$$\begin{aligned}H^1(\mathbb{T}^2) &:= \|\mathbb{S}^1 \times \mathbb{S}^1 \to K_1\|_0 \\ &\simeq \|\mathbb{S}^1 \to (\mathbb{S}^1 \to \mathbb{S}^1)\|_0 \\ &\simeq \|\mathbb{S}^1 \to (\mathbb{S}^1 \times \mathbb{Z})\|_0 && \text{by (34)} \\ &\simeq \|\mathbb{S}^1 \to \mathbb{S}^1\|_0 \times \|\mathbb{S}^1 \to \mathbb{Z}\|_0 \\ &\simeq H^1(\mathbb{S}^1) \times H^0(\mathbb{S}^1) \\ &\simeq \mathbb{Z} \times \mathbb{Z}\end{aligned}$$

The fact that this equivalence is a morphism follows by the same argument as in the second proof of Proposition 4.56. □

Now, there is only $H^2(\mathbb{T}^2)$ left. We use a similar proof here and first characterise the function space $(\mathbb{S}^1 \to K_2)$. The idea is that a map from $\mathbb{S}^1 \to K_2$ consists of a point $x : \mathbb{K}_2$ and a loop $\ell : x \equiv x$. A loop in K_2 may be interpreted under the equivalence $\Omega K_2 \simeq K_1$. Hence, the function space should be equivalent to the product $K_1 \times K_2$. We prove this.

Lemma 4.65. $(\mathbb{S}^1 \to K_2) \simeq K_1 \times K_2$.

Proof. We begin by defining a map $F : (\mathbb{S}^1 \to K_2) \to K_1 \times K_2$. Let $f : (\mathbb{S}^1 \to K_2)$. For the point in K_2, we simply choose $f(\mathsf{north})$. For the point in K_1, we use σ_1^{-1} and construct a loop in ΩK_2. Recall that a loop in $\mathbb{S}^n$ under the definition using meridians corresponds to the path $\mathsf{loop}' :\equiv \mathsf{merid}\,(\mathsf{south}) \cdot (\mathsf{merid}\,(\mathsf{north}))^{-1}$. The immediate candidate for our element of ΩK_2 would then be $\mathsf{cong}_f(\mathsf{loop}')$. However, this is not well-typed, since we do not have $f(\mathsf{north}) := |\,\mathsf{north}\,|_2$. In order to make sure that our loop is well-typed, we need to use our algebraic structure on K_2. We define F by

$$\begin{aligned}\mathsf{snd}(F(f)) &:\equiv f(\mathsf{north}) \\ \mathsf{fst}(F(f)) &:\equiv \sigma_1^{-1}(P_f)\end{aligned}$$

where P_f is the composite path

$$\begin{aligned} |\,\mathsf{north}\,|_2 &\equiv f(\mathsf{north}) - f(\mathsf{north}) \\ &\equiv f(\mathsf{south}) - f(\mathsf{south}) \\ &\equiv |\,\mathsf{north}\,|_2 \end{aligned}$$

given by

$$\begin{aligned} &\mathsf{rCancel}_k(f(\mathsf{north}))^{-1} \\ &\cdot \mathsf{cong}^2_{\lambda\, x\, \lambda\, y\, .\, (f(x) - f(y))}(\mathsf{merid}(\mathsf{south}), \mathsf{merid}(\mathsf{north})) \\ &\cdot \mathsf{rCancel}_k(f(\mathsf{south})) \end{aligned}$$

We now define a map $G : K_1 \times K_2 \to (\mathbb{S}^1 \to K_2)$ by

$$\begin{aligned} (G(a,b))(\mathsf{north}) &:\equiv b + 0 \\ (G(a,b))(\mathsf{south}) &:\equiv b + 0 \\ (G(a,b))((\mathsf{merid}(\mathsf{north}))(i)) &:\equiv b + 0 \\ (G(a,b))((\mathsf{merid}(\mathsf{south}))(i)) &:\equiv b + (\sigma_1(a))(i) \end{aligned}$$

Note that we have to map elements to $b + 0$ rather than just b, if the want to the last step of the definition above to be well-typed; in general, we do not have $a + 0 :\equiv a$. This unfortunate fact makes the proof that F and G cancel out somewhat more complicated. We begin by showing that $F(G(a,b)) \equiv (a,b)$. We have

$$\begin{aligned} \mathsf{fst}(F(G(a,b))) :\equiv \sigma_1^{-1}[&\mathsf{rCancel}_k(b+0)^{-1} \\ &\cdot \mathsf{cong}^2_{\lambda\, x\, \lambda\, y\, .\, ((G(a,b))(x) - (G(a,b))(y))}(\mathsf{merid}(\mathsf{south}), \mathsf{merid}(\mathsf{north})) \\ &\cdot \mathsf{rCancel}_k(b+0)] \end{aligned}$$

By definition of $G(a,b)$ for $\mathsf{merid}(\mathsf{north})$ and $\mathsf{merid}(\mathsf{south})$, this is just

$$\begin{aligned} \sigma_1^{-1}[&\mathsf{rCancel}_k(b+0)^{-1} \\ &\cdot \mathsf{cong}_{\lambda\, x\, .\, ((b+x) - (b+0))}(\sigma_1(a)) \\ &\cdot \mathsf{rCancel}_k(b+0)] \end{aligned}$$

By functoriality of cong^2, the composite path inside of the parenthesis is equal to

$$\begin{aligned} &\mathsf{rCancel}_k^{-1}(b+0) \\ &\cdot \mathsf{cong}_{\lambda\, x\, .\, ((G(a,b))(x) - (b+0))}(\mathsf{merid}(\mathsf{south})) \\ &\cdot \mathsf{cong}_{\lambda\, x\, .\, ((b+0) - (G(a,b))(x))}(\mathsf{merid}(\mathsf{north})) \\ &\cdot \mathsf{rCancel}_k(b+0) \end{aligned}$$

We are done if we can show that this is equal to $\sigma_1(a)$, since σ_1 and σ_1^{-1} are mutually inverse. Given a path $p : 0_k \equiv 0_k$, a function $f : K_2 \to K_2$ such that

$f \equiv \lambda x . x$ and a path $q : f(0_k) \equiv 0_k$, the following fact follows by some simple path algebra using the commutativity of ΩK_n.

$$q^{-1} \cdot \mathsf{cong}_f(p) \cdot q \equiv p$$

We let $p :\equiv \sigma_1(b)$, $f :\equiv \lambda x . ((a + x) - (a + 0))$ and $q := \mathsf{rCancel}_k(b + 0)$ in the above equality. Clearly, f is equal to the identity function. Thus, we get

$$\mathsf{rCancel}_k(b + 0)^{-1} \cdot \mathsf{cong}_{\lambda x . ((b+x)-(b+0))}(\sigma_1(b)) \cdot \mathsf{rCancel}_k(b + 0) \equiv \sigma_1(b)$$

which is precisely what we needed. Thus $\mathsf{fst}(F(G(a, b))) \equiv a$. We also have $\mathsf{snd}(F(G(a, b))) := b + 0 \equiv b$. Hence $F(G(a, b)) \equiv (a, b)$.

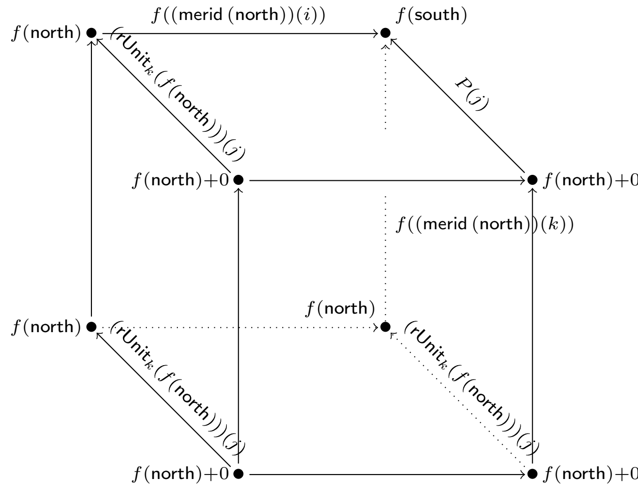
The other direction requires some more work. Let $f : \mathbb{S}^1 \to K_2$. We want to show that $G(F(f)) \equiv f$. By function extensionality, we only need to show that $(G(F(f)))(x) \equiv f(x)$ for every $x : \mathbb{S}^1$. For the base cases, we have

$$(G(F(f)))(\mathsf{north}) \equiv (G(F(f)))(\mathsf{south}) \equiv f(\mathsf{north}) + 0$$

For these cases we give the following paths.

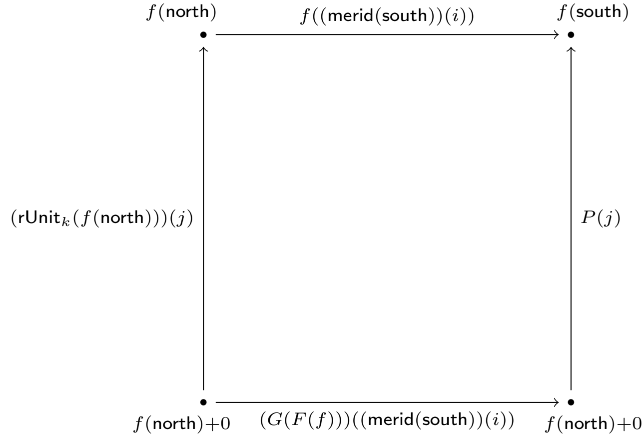
$$\mathsf{rUnit}_k(f(\mathsf{north})) : (G(F(f)))(\mathsf{north}) \equiv f(\mathsf{north})$$
$$P :\equiv \mathsf{rUnit}_k(f(\mathsf{north})) \cdot \mathsf{cong}_f(\mathsf{merid}(\mathsf{north})) : (G(F(f)))(\mathsf{south}) \equiv f(\mathsf{south})$$

For the $\mathsf{merid}\,(\mathsf{north})$ case, we need the lid of the following cube



The front, bottom and left-hand side are trivial. The right-hand side is given by the filler for path composition. The backside is given by $f((\mathsf{merid}(\mathsf{north}))(i \wedge k))$. This establishes the $\mathsf{merid}(\mathsf{north})$ case.

For the $\mathsf{merid}(\mathsf{south})$ case, we need to fill the following square.



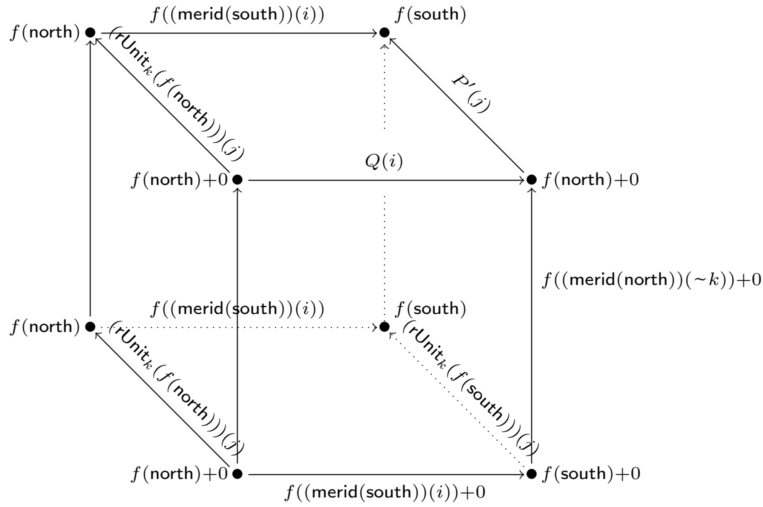
It follows from the commutativity of ΩK_n and functoriality of cong that

$$\begin{aligned}\mathsf{cong}_{G(F(f))}(\mathsf{merid}(\mathsf{south})) \equiv\, &\mathsf{cong}_{\lambda\, x\,.\,(f(x)+0)}(\mathsf{merid}\,(\mathsf{south}))\\ &\cdot\ \mathsf{cong}_{\lambda\, x\,.\,(f(x)+0)}(\mathsf{merid}\,(\mathsf{north}))^{-1}\end{aligned} \tag{35}$$

Let Q denote the path on the right-hand side in (35). We also have $P \equiv P'$, where

$$P'(j) :\equiv \mathsf{cong}_{\lambda\, x\,.\,(f(x)+0)}(\mathsf{merid}\,(\mathsf{south})) \cdot \mathsf{rUnit}_k(f(\mathsf{south}))$$

We may now replace the bottom of the square by Q and the right-hand side of the square by P'. We consider the new square as the lid of the following cube.



This is easy to fill. The backside and the left-hand side are trivial. The front is just the filler of path composition. The right hand side is essentially the dual of path composition, and is filled similarly. The bottom is filled by

$$(\mathsf{rUnit}_k(f((\mathsf{merid}\,(\mathsf{south})(i)))))(j)$$

This finishes the proof. ☐

Using this, we now easily find $H^2(\mathbb{T}^2)$.

Proposition 4.66. $H^2(\mathbb{T}^2) \cong \mathbb{Z}$

Proof. We have the following equivalences

$$\begin{aligned}
H^2(\mathbb{T}^2) &:= \left\|\mathbb{S}^1 \times \mathbb{S}^1 \to K_2\right\|_0 \\
&\simeq \left\|\mathbb{S}^1 \to (\mathbb{S}^1 \to K_2)\right\|_0 \\
&\simeq \left\|\mathbb{S}^1 \to K_1 \times K_2\right\|_0 && \text{(Lemma 4.65)} \\
&\simeq \left\|\mathbb{S}^1 \to K_1\right\|_0 \times \left\|\mathbb{S}^1 \to K_2\right\|_0 \\
&:= H^1(\mathbb{S}^1) \times H^2(\mathbb{S}^1) \\
&:= \mathbb{Z} \times \top \\
&:= \mathbb{Z}
\end{aligned}$$

The fact that this equivalence is a morphism follows from the fact that the map $G : K_1 \times K_2 \to (\mathbb{S}^1 \to K_2)$ from the proof of Lemma 4.65 preserves addition in the natural sense, which is an easy lemma. ☐

5 Implementation in Cubical Agda

5.1 Formalisation

The main part of this thesis is the formalisation of the mathematics in Section 4. The formalisation consists of three pull requests (chronologically: 1, 2, 3) making up ~4400 lines of code in total. This is excluding a 2600 line pull request containing the Freudenthal Suspension Theorem and some preliminary lemmas from Section 4.1.5 (available here). The formalisation took approximately 4½ months to complete. I started off with no experience in Cubical Agda and some very elementary familiarity with standard Agda. Consequently, the most time consuming part of the project was learning Cubical Agda and getting used to cubical reasoning.

The following paragraphs outline the main parts of the formalisation.

Preliminary lemmas The cubical library is relatively young. Consequently, much of the theory from [14] and [6] needed for the cohomology group structure had to be formalised. This especially concerned theory about truncations and connected functions/types. Most proofs here follow [14] closely and appear to

leave little room for cubical improvement. The formalisations concerning theory about truncations include e.g. Theorem 4.22 and Proposition 4.20. Connected functions and types were not yet defined in the library and consequently all of Section 4.1.5 had to be formalised. Evan Cavallo worked independently on some of these theorems and consequently the current version of the module on connected functions and types contain proofs from the both of us.

Another notable preliminary lemma was the Eckmann-Hilton argument. My original proof followed the HoTT book closely and consisted of 35 lines, excluding theorem statements. After Bentzen referred me to his cubical proof in [5], I completely reworked the proof accordingly. The resulting proof consists of only 4 lines, excluding theorem statements, and uses only one application of comp. This is a clear illustration of how the composition operations in CuTT can be used to significantly simplify lengthy path algebraic arguments in standard HoTT. In this case, they also removed the need for a path induction, thus shortening Bentzen's already very short proof further.

Freudenthal Suspension Theorem Formalising the Freudenthal Suspension Theorem was the first big step of this project. My proof follows [14] closely and, disregarding preliminary lemmas, consists of approximately 1600 lines of code (although it could most likely be abbreviated significantly by renaming complicated terms). Shortly after I finished my proof, Cavallo published an independent proof consisting of only ~100 lines of code, excluding preliminary lemmas. The difference in length comes from the fact that my proof heavily relies on path induction whereas Cavallo's proof utilises both the Glue and hcomp machineries of Cubical Agda. The difference in length is astonishing and prove a strong case for the cubical approach to HoTT. In addition, his proof is a slight modification of the proof in [14], which simplifies much of the path algebra involved in the proof. Computationally, Cavallo's proof also performs better. His proof terminates in ~15 seconds when compiled ignoring abstracts and ~3 seconds with some abstracts turned on. My proof appears not to terminate without abstracts. Interestingly, this seems to be unrelated to the length of the proof. Termination fails already when trying to transport an element over the Code-fibration. This is avoided in Cavallo's proof by using the Glue-type to define Code, thus avoiding unnecessary applications of univalence.

Group structure on cohomology The first step was to formalise the equivalence $K_n \simeq \Omega K_{n+1}$. Mathematically, this was straightforward. However, because the proofs of $\Omega\mathbb{S}^1 \simeq \mathbb{Z}$ and of the Hopf fibration used the base/loop-definition of $\mathbb{S}^1$, whereas the definition of $\mathbb{S}^1$ using suspensions is used for cohomology groups, a large portion of the work went into translating between the these two definitions.

Unfortunately, $+_k$ and $+_h$ would perform very poorly (and in some cases not terminate at all) unless most of the proof of $K_n \simeq \Omega K_{n+1}$ was put as abstract. The inverse map from ΩK_{n+1} to K_n needed to be hidden for $n \geq 1$. For $n = 1$, it would perform poorly but terminate. For $n > 1$, it would not even terminate. This most likely comes from the fact that it is induced by the

Freudenthal Suspension Theorem and several other relatively complex theorems on connected maps.

Mayer-Vietoris Sequence The formalisation of the Mayer-Vioteris sequence follows Brunerie's proof in [6] closely. Naturally, since Brunerie's proof is written in the style of informal type theory there were some technicalities that needed to be verified. One example is the fact that the right-unit law for addition in K_n agrees with the left-unit law for 0_k, which was needed in order to prove that the map d_n is a morphism.

Cohomology groups For the characterisation of the various cohomology groups, the first thing I had to do was to disable η-equality for the various records involved in the definition of groups. With η enabled, many proofs would fail to terminate even in very simple cases. Also, defining cohomology groups and direct products of groups using copatterns sped up type checking significantly.

There were some issues concerning the function i_n from the Mayer-Vietoris sequence. Often when proving something about $i_n(x)$ for some x in the domain of i_n, I would have to first apply truncation elimination on x in order to make it take the form $|\,y\,|_0$, even in cases when this should not be needed. It is likely that this is caused by a performance issue in Cubical Agda. This trick also sped up type checking in some cases where it was not actually needed for termination.

5.2 Computations

Unfortunately, the formalised characterisations of $H^n(\mathbb{T}^2)$ and $H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$ suffer from some performance issues. The main culprits are most likely the Mayer-Vietoris Sequence, the Hopf Fibration and the Freudenthal Suspension Theorem. For every $n = 0, 1, 2$, the following functions have been defined in Agda by means of the group characterisations of $H^n(\mathbb{T}^2)$ and $H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$.

$$\begin{aligned} \mathsf{from}^T_n &: H^n(\mathbb{T}^2) \to G_n \\ \mathsf{from}^\vee_n &: H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \to G_n \\ \mathsf{to}^T_n &: G_n \to H^n(\mathbb{T}^2) \\ \mathsf{to}^\vee_n &: G_n \to H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \end{aligned}$$

where

$$G_n :\equiv \begin{cases} \mathbb{Z} & n = 0, 2 \\ \mathbb{Z} \times \mathbb{Z} & n = 1 \end{cases}$$

The idea is that $\mathsf{from}_n(\mathsf{to}_n(a))$ should reduce to a when normalising in Agda. Unsurprisingly, this works well for the 0th cohomology groups. For instance, both $\mathsf{from}^T_0(\mathsf{to}^T_0(100000))$ and $\mathsf{from}^\vee_0(\mathsf{to}^\vee_0(100000))$ reduce to 1000000 in less than a second. On the other hand, $\mathsf{from}_2(\mathsf{to}_2(a))$ fails in both cases and does so even

for $a = 0$. This is most likely since the maps rely on $+_k$ in both cases. In the case $n = 1$, the isomorphisms $H^1(\mathbb{T}^2) \cong \mathbb{Z} \times \mathbb{Z}$ and $H^1(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \cong \mathbb{Z} \times \mathbb{Z}$ are both proved relatively directly. We get some reductions in reasonable time, most likely due to this fact. We have that $\mathsf{from}_1^{\vee}(\mathsf{to}_1^{\vee}(x, y))$ reduces to (x, y) in reasonable time for small values of x and y. For $H^1(\mathbb{T}^2)$, we get that $\mathsf{from}_1^{\vee}(\mathsf{to}_1^{\vee}(x, y))$ reduces to (x, y) only when $x = 0$. The reduction is again inefficient for larger values of y. When $x \neq 0$, reduction appears to fail. Again, this most likely boils down to the fact that the map is defined in terms of $+_h$.

For addition $+_h$, we get termination issues in all cases but for H^0. That is, $\mathsf{from}_n(\mathsf{to}_n(a) +_h \mathsf{to}_n(b))$ fails to reduce when $n \geq 1$ both for $H^n(\mathbb{T}^2)$ and $H^n(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$.

6 Future work

The next step is to define the cup product in Cubical Agda. This project was started but put on hold. Some of the proofs involved are, despite being intuitively easy, hard to do formally. When this is done, we need to try to improve the definition of $+_h$ in order to be able to compare it to the cup product. There are two main steps in doing this.

1. Many proofs in the cubical library concerning $\mathbb{S}^1$ or $\mathbb{S}^2$ use the base/loop and base/surf definitions instead of the definition by suspensions. Currently, many lines of code are dedicated to translating between these definitions. It would be a good idea to settle for one of them. This should not have a very big impact on the overall computation times, but could hopefully help somewhat.

2. As a second step, one could attempt to define the equivalence $K_n \simeq \Omega K_{n+1}$ in terms of the equivalence $\left\|\Omega \mathbb{S}^{n+1}\right\|_n \simeq \|\mathbb{S}^n\|_n$ as proved in [11]. This could potentially improve the performance of $+_k$, and consequently of $+_h$, since this proof does not rely on any theory about connected maps or on the Freudenthal Suspension Theorem. Nevertheless, similar theorems are required for the cup product, at least if it is to be defined along the lines of [6].

References

[1] *Agda 2.6.1 Documentation : Cubical.* `https://agda.readthedocs.io/en/v2.6.1/language/cubical.html`. Accessed: 2020-04-03.

[2] *Agda 2.6.1 Documentation : Lambda Abstraction.* `https://agda.readthedocs.io/en/v2.6.1/language/lambda-abstraction.html`. Accessed: 2020-04-03.

[3] Several Authors. *The Cubical Library.* `https://github.com/agda/cubical`. 2020.

[4] Several Authors. *The Cubical Library (frozen branch as of 05-08-2020)*. `https://github.com/aljungstrom/cubical/tree/MA-Thesis`. 2020.

[5] Bruno Bentzen. "Naive cubical type theory". In: *ArXiv* abs/1911.05844 (2019).

[6] Guillaume Brunerie. "On the homotopy groups of spheres in homotopy type theory". In: *ArXiv* abs/1606.05916 (2016).

[7] Tammo tom Dieck. *Algebraic topology*. Zürich: European Mathematical Society, 2008. ISBN: 978-3-03719-048-7.

[8] Euklides and Isaac Barrow. *Euclide's elements the whole fifteen books compendiously demonstrated*. London: Printed by R. Daniel for William Nealand, 1660.

[9] Allen Hatcher. *Algebraic topology*. Cambridge: Cambridge University Press. ISBN: 9780521795401.

[10] W. A. Howard. "The Formulæ-as-Types Notion of Construction". In: *The Curry-Howard Isomorphism*. Ed. by Philippe De Groote. Academia, 1995.

[11] Daniel R. Licata and Guillaume Brunerie. "$\pi_n(S^n)$ in Homotopy Type Theory". In: *Certified Programs and Proofs*. Ed. by Georges Gonthier and Michael Norrish. Cham: Springer International Publishing, 2013, pp. 1–16. ISBN: 978-3-319-03545-1.

[12] Anders Mörtberg and Loïc Pujet. "Cubical Synthetic Homotopy Theory". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 158–171. ISBN: 9781450370974. DOI: `10.1145/3372885.3373825`. URL: `https://doi.org/10.1145/3372885.3373825`.

[13] Bertrand Russell. *The principles of mathematics*. 2. ed. London: Allen & Unwin.

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