On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory

Floris van Doorn

May 2018

Dissertation Committee:
Jeremy Avigad
Steve Awodey
Ulrik Buchholtz
Mike Shulman

Contents

Chapter 1

Introduction

The goal of this dissertation is to present synthetic homotopy theory in the setting of *homotopy type theory*. We will present various results in this framework, most notably the construction of the Atiyah-Hirzebruch and Serre spectral sequences for cohomology, which have been fully formalized in the Lean proof assistant.

Homotopy type theory, often abbreviated HoTT, is a version of type theory. Type theory is a language for formal mathematics, in which every object has a computational interpretation, so that it can also function as a programming language. It can be used as a foundation of mathematics as an alternative to set theory.

A key feature of HoTT is that the equality in a space corresponds to the path spaces; a path between two points a and b is a proof that $a = b$. Two paths that are not homotopic give different (unequal) proofs of this equality. The fact that we identify proofs of an equality with a path means that every construction in HoTT respects paths.

Many different researchers contributed to the homotopical interpretation of type theory. Steve Awodey and Michael Warren gave a model of type theory in abstract homotopy theory [AW09]. Benno van den Berg and Richard Garner published a paper addressing the coherence issue [vG12]. Independently, Vladimir Voevodsky gave a model of type theory without identity types in simplicial sets and formulated the *univalence axiom*, which he proved consistent [Voe06, Voe09]. The univalence axiom states that homotopy equivalences between two types (spaces) corresponds to equality between them [Voe14]. This means that every construction done in HoTT automatically respects homotopy equivalence, which is a very convenient property. Also, Voevodsky proved that a consequence of the univalence axiom is *function extensionality*. This states that two functions are equal when they are homotopic.

The fact that all constructions are homotopy invariant also leads to some challenges. It is not always clear whether we can define a concept of homotopy theory in homotopy type theory. For example, *singular homology* is a homotopy invariant notion, but in the construction we use the set of all simplices in a space, which is not a homotopy invariant notion. In this case, we can define homology in a different way (see Section 5.5). However, for other definitions, such as the Grassmannian manifolds, it is an open

problem whether they can be constructed in homotopy type theory.

A new concept in homotopy type theory is the concept of *higher inductive types*. These are types that generalize both cell complexes in homotopy theory, and inductively generated types (like $\mathbb{N}$) in type theory. Higher inductive types can be used to construct many spaces and operations on spaces often encountered in homotopy theory.

Type theory is a convenient language for computer proof assistants. These are programs that allow you to write formal proofs in a specified language, and then the computer checks whether the proof is correct and complete. There are many major results formalized in proof assistants, such as the four colour theorem [Gon05], Feit-Thompson theorem [G$^+$13] and the Kepler conjecture (Hales' Theorem) [H$^+$17]. HoTT is a type theory, and it has been implemented in various proof assistants, such as Coq [BGLL$^+$16], Agda [BHC$^+$], cubicaltt [CCHM], Lean [vDvRB17] and various experimental proof assistants. One disadvantage of formally verifying proofs in a proof assistant is that it takes a lot of work spelling out all details. For example, doing very basic homotopy theory (not using homotopy type theory) already takes a lot of effort [Zha17]. In HoTT this effect is mitigated, because many homotopical concepts are close to the foundations of the type theory, making formal proofs only a little more work than a paper proof.

Various results have been proven and formalized in HoTT, such as the the Seifert–van Kampen theorem [HS16], the Blakers–Massey theorem [HFLL16] and a development of cellular cohomology [BH18]. Another main result (which has not been formalized) is the computation of $\pi_4(\mathbb{S}^3)$ [Bru16], which relies on conjectured properties of the smash product, which we will discuss in Section 4.3.

HoTT gives novel proof methods and new insights to homotopy theory. A basic property of HoTT is *path induction*, which states that when proving something for a path with one free endpoint, one may assume that the path is the constant path. This corresponds to the fact that the path space with one fixed endpoint is contractible. Another technique is the encode-decode method, for calculating the path space of certain spaces [LS13]. Moreover, the proof of the Blakers–Massey theorem has been translated back to homotopy theory, resulting in a new proof with novel ideas [Rez14].

Homotopy type theory has models in most model categories [AW09, vG12], which are categorical models for homotopy theory. These models were inspired by the groupoid model [HS98]. Other models for HoTT include the simplicial set model [Voe09, KL12, Str14] and the cubical set model [BCH14, CCHM16]. More generally, all Grothendieck $(\infty, 1)$-toposes model HoTT [Cis14].[1] Moreover, it is conjectured that all elementary $(\infty, 1)$-toposes form models of HoTT [Shu17].

---

[1]General Grothendieck $(\infty, 1)$-toposes model HoTT with universes á la Tarski. This notion is weaker than universes á la Russell, which are usually considered in HoTT. We explain Russell universes in Section 2.1.3.

Type Theory

Homotopy type theory is based on Martin-Löf type theory (also called intuitionistic type theory or constructive type theory) [ML75, ML84]. In this type theory there are types, like the integers $\mathbb{Z}$, vectors $\mathbb{R}^n$; and complex functions $\mathbb{C} \to \mathbb{C}$. There are also terms, which have a unique type.[2] For example the number -2 has type $\mathbb{Z}$ (written as $-2 : \mathbb{Z}$), the vector $(1, 2, 3, \ldots, n)$ has type $\mathbb{R}^n$ and we have the exponential function $\exp : \mathbb{C} \to \mathbb{C}$. One can think of types as sets of objects (and indeed, there is a model of type theory where the types are exactly sets), but there are different interpretations, such as the types-as-spaces interpretation that homotopy type theory provides. The fact that terms have a unique type means that the $2 : \mathbb{Z}$ and the $2 : \mathbb{R}$ are different objects. It might be helpful to think of data types in a programming language, in which the `int` 2 is stored differently in memory than the `float` 2. Of course, the canonical inclusion $i : \mathbb{Z} \hookrightarrow \mathbb{R}$ does satisfy $i(2) = 2$. Type theory has a primitive notion of computation, so that for example $2 + 3$ computes to 5. Every function that is explicitly defined in type theory therefore describes an algorithm that can be executed. This means that type theory can be used as a programming language, and many programming languages make use of a type system. The congruence closure of this notion of computation is called *definitional equality* or *judgmental equality*, and if two terms are judgmentally equal, one can replace one for the other in any term.

There are several methods to construct new types out of existing ones. For example we can form the function type $A \to B$ for types A and B, the cartesian product type $A \times B$ and the coproduct or sum $A + B$. Propositions can also be interpreted as types by the *Curry-Howard isomorphism* [CF58, How80], and under this interpretation $A \times B$ is the conjunction of A and B, the sum $A + B$ is the disjunction and $A \to B$ is the implication. Furthermore, there are dependent function types $\prod_{(x:A)} P(x)$ and dependent sum types $\sum_{(x:A)} P(x)$, which correspond to the universal quantification $\forall (x : A), P(x)$ and existential quantification $\exists (x : A), P(x)$, respectively. So for example the transitivity of $\leq$ on $\mathbb{N}$ can be expressed as $\prod_{(k,m,n:\mathbb{N})} k \leq m \to m \leq n \to k \leq n$, and a term of this type is a proof that $\leq$ is transitive. The P in $\prod_{(x:A)} P(x)$ and $\sum_{(x:A)} P(x)$ is called a *dependent type*, since it is a type depending on a term $x : A$. It has type $P : A \to \mathcal{U}$, where $\mathcal{U}$ is the universe of (small) types. The dependent function type $\prod_{(x:A)} P(x)$ consists of functions f that send terms $a : A$ to a term $f(a) : P(a)$. Note that the type of $f(a)$ depends on the input a. The dependent sum type $\sum_{(x:A)} P(x)$ consists of dependent pairs (a, x) with $a : A$ and $x : P(a)$, where the type of x depends on a.

Given two terms $a, b : A$, we can form the *identity type* which we write as $a =_A b$ or $a = b$. As a proposition we view $a =_A b$ as the statement that a and b are equal. In homotopy type theory these identity types correspond to the path space of the type A.

---

[2]To be more precise: in many type theories there are terms with multiple types, for example due to universe cumulativity, but we will ignore these issues. Moreover, the type theory of Lean has unique typing [Car18].

Homotopy Type Theory

There are various versions of dependent type theory with different rules for the identity type. Some type theories have a *reflection rule*, which states that if we have a proof $p : a = b$, then a and b are judgmentally equal. Type theories with this rule are often called *extensional.* This is a convenient rule, but these type theories have meta-theoretic properties that are often seen as undesirable. For example, checking whether a term t has type A is not decidable anymore. Since this operation can be viewed as "checking the correctness of a proof," one often wants to work in a type theory with decidable type-checking.

In *intensional* type theory, without the reflection rule, multiple approaches can be taken for the identity type. In some versions, there is a rule that any two proofs of the same equality are themselves equal. This rule, often called *uniqueness of identity proofs* or *axiom K* states that if $p, q : a = b$, then there is a proof of $p = q$. In homotopy type theory, this rule is rejected. In the types-as-spaces interpretation of homotopy type theory, terms of the identity type $a =_A b$ are interpreted as paths in A from a to b. We have familiar operations on paths: given two paths $p : a =_A b$ and $q : b =_A c$, we write $p \cdot q : a = c$ for the concatenation of p and q. Furthermore, we have the inverse path $p^{-1} : b = a$ and the constant path $\mathsf{refl}_a : a = a$. We also have higher paths, the identity type $p =_{a=_A b} q$ consists of homotopies from path p to q. We can form higher path types between two homotopies, and there are also operations on these higher paths. In this way every type comes equipped with the structure of a higher groupoid.

In 2011, higher inductive types were introduced in homotopy type theory [Shu11c, Lum11, Shu11b, Shu11a]. With ordinary inductive types we specify constructors that generate the type, for example the natural numbers are generated by zero $0 : \mathbb{N}$ and the successor function $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$. Higher inductive types are generated not only by these "point constructors" but also by "path constructors," which specify the inhabitants of paths or higher paths in the type. For example, the circle $\mathbb{S}^1$ is generated by a point $\star : \mathbb{S}^1$ and a loop $\ell : \star = \star$. The rest of the structure of $\mathbb{S}^1$ is built from these constructors. Using higher inductive types we can construct many other spaces in homotopy theory, such as Eilenberg-MacLane spaces and homotopy pushouts.

As mentioned before in this introduction, we can use HoTT to do homotopy theory. We think of types as spaces and we think of maps between types as continuous maps between those spaces. Then we can define usual notions in homotopy theory, as long as they are homotopy invariant: homotopy equivalences, suspensions, spheres, etcetera. This is a *synthetic* way to do homotopy theory: many concepts, such as spaces and paths are uninterpreted constants of the type theory. This is opposed to *analytic* homotopy theory, where one studies topological spaces up to homotopy equivalence. This distinction is similar to the distinction for elementary geometry, which we can do synthetically (points and lines are undefined concepts) or analytically (we are working in $\mathbb{R}^2$). Synthetic geometry limits the things one can state or prove, but these proofs are applicable in every model of the axioms. The same is true for synthetic homotopy theory: the proofs performed synthetically are true in all models of HoTT.

In this dissertation I will not be very precise about the exact rules of the type theory we are using. We will present the constructions and proofs in such a way that they can be performed in the "HoTT book" [Uni13]. Most of the results in this dissertation have been formalized in the Lean proof assistant [dMKA+15]. The HoTT mode we used in Lean has very similar rules to the HoTT book, and the differences are not relevant for the constructions in this dissertation. A concept closely related to homotopy type theory is *univalent mathematics*, a term coined by Vladimir Voevodsky for the development of mathematics where one takes homotopy types as primitive objects, and reasons about them using type-theoretic reasoning and the univalence axiom. This is pursued in the proof assistant UniMath [VAG+]. There are also radically different type theories which are studied in homotopy type theory. These are called "cubical type theories" because they all have a primitive notion of cubes. Examples include the cubical type theory described in [CCHM16], which was implemented in the proof assistant cubicaltt [CCHM], and computational higher-dimensional type theory [AHW17], on which the proof assistant RedPRL is based [The18]. These type theories are extensions of the type theory presented in the HoTT book, which we will call book-HoTT. In book-HoTT the univalence axiom is an axiom: an uninterpreted constant of a certain type. This breaks the computational behavior of the type theory. For example not every closed term of type $\mathbb{N}$ computes to either 0 or the successor of another number. These cubical type theories add primitive concepts to the theory to make the univalence axiom provable, and therefore all terms in these system do compute.

We will often want to compare homotopy type theory with ordinary homotopy theory. We will use the adverb "classically" to refer to the concepts and theorems in homotopy theory that do not involve HoTT.[3] Conversely, we will say that something is provable in HoTT if we can prove it in book-HoTT.

Contents

In Chapter 2 we review the basic concepts in homotopy type theory. For a more detailed and thorough exposition, we refer to [Uni13]. Alternative introductions can be found in [Hou17] and [Bru16]. In Section 2.1 we introduce the basic concepts of type theory: functions, pairs, universes, and inductive types such as the identity type. In Section 2.2 we will introduce the basics of homotopy type theory. In particular we will formally state the univalence axiom and present higher inductive types. In Section 2.3 we will discuss the Lean proof assistant in more detail.

In Chapter 3 we will study higher inductive types internally in HoTT. The main problem we will focus on is the interdefinability of higher inductive types. In particular, we try to construct various higher inductive types from the homotopy pushout. We will

---

[3]This use of classically has nothing to do with the word classical in "classical logic," involving the law of excluded middle or the axiom of choice. In homotopy type theory one can consistently assume the law of excluded middle or the axiom of choice, formulated in a precise way so that it corresponds to what it usually means. However, doing so removes the computational content of all notions defined using it.

define the propositional truncation in Section 3.1, nonrecursive higher inductive types with 2-path constructors in Section 3.2 and work towards defining certain *localizations* in Section 3.3.

In Chapter 4 we present some synthetic homotopy theory in HoTT. In Section 4.1 we will describe the formalization of the long exact sequence of homotopy groups and its application to compute $\pi_3(\mathbb{S}^2)$. Although this construction has been described before in HoTT in [Uni13, Section 8.4] and [Bru16, Section 2.5.1], no formally verified proof has been given before. In Section 4.2 we will study Eilenberg-MacLane spaces, which are spaces with only one nontrivial homotopy group. Eilenberg-MacLane spaces have been defined in HoTT before [LF14]. Here we prove the (classically known) results that Eilenberg-MacLane spaces are unique, and give an equivalence of categories between the category of (abelian) groups and an appropriate class of pointed types. In Section 4.3 we will discuss the smash product. The ultimate goal is to prove that the smash product forms a 1-coherent symmetric monoidal product on pointed types, and we will give one approach towards proving this using a Yoneda-style argument.

In Chapter 5 we develop the theory of spectral sequences in HoTT. We give the construction of a spectral sequence from an exact couple (in Section 5.2) and show how to construct an exact couple from a tower of spectra (in Section 5.3). We construct the classically-known Atiyah-Hirzebruch and Serre spectral sequences for cohomology (in Section 5.4), and give some ideas towards doing the same for their counterparts in homology (in Section 5.5).

Chapter 2

Preliminaries

In this chapter we will give a brief overview of type theory and homotopy type theory. We cannot cover all the subtleties, so readers new to (homotopy) type theory should consult the homotopy type theory book [Uni13].

In Section 2.3 we will discuss the proof assistant *Lean*. All main results in this dissertation have been formalized in Lean.

2.1 Martin-Löf Type Theory

As mentioned in the introduction, homotopy type theory is based on a system called *Martin-Löf type theory* or *intuitionistic type theory*. There are types and there are terms, which have a unique type. There is a notion of computation. Two terms t and s are considered *judgmentally equal* or *definitionally equal*, denoted $t \equiv s$ if t and s compute to the same term.

We are working in dependent type theory, which means that types can depend on terms. For example, there is a type of vectors of length $n : \mathbb{N}$ in type A, denoted $\mathsf{vector}_A(n)$. In this case vector_A is a dependent type over $\mathbb{N}$. An example term in this type family is $(5, 6, 7, 8) : \mathsf{vector}_\mathbb{N}(4)$. When we say that a term has a unique type, we mean that it has a unique type up to definitional equality. In our example, we also have that $(5, 6, 7, 8) : \mathsf{vector}_\mathbb{N}(2 + 2)$, because $2 + 2 \equiv 4$. More generally, if we have two definitionally equal types $A \equiv B$ and if $t : A$, then $t : B$. Logically (under the types-as-propositions interpretation) dependent types are predicates. We will explain the topological interpretation of dependent types at the end of Section 2.1.2.

In the remainder of this section we will discuss the type formers of Martin-Löf type theory more closely.

2.1.1 Function Types

Given a type A and a family of types B depending on A, we can form the *dependent function type* (also called *product type* or *pi type*)

$$(x : A) \to B(x) \qquad \text{or} \qquad \prod_{x:A} B(x).$$

We will use the former notation in this document. A term $f : (x : A) \to B(x)$ is a function that sends each element $a : A$ to an element[4] $f(a) : B(a)$. We also use the notation fa or $f\ a$ for $f(a)$. Note that the type of $f(x)$ depends on x. We can form functions using *lambda-abstraction*. Given a term $t(x) : B(x)$, we can form the term $\lambda x.\, t(x) : (x : A) \to B(x)$, which is the function $x \mapsto t(x)$, i.e. the function that sends x to $t(x)$. We get the computation rule

$$(\lambda x.\, t(x))a \equiv t(a)$$

for $a : A$, which is called the *beta-rule* or *beta-reduction*. We also have an *eta-rule*, which states that every function is a lambda abstraction. This means that for $f : (x : A) \to B(x)$ we have

$$f \equiv \lambda x.\, f(x).$$

We will often define functions by writing $f(x) :\equiv t$ (where x may occur in t), which formally means that we define f as $\lambda x.\, t$.

An important special case occurs when B does not depend on A. In this case the dependent function type $(x : A) \to B$ is written as $A \to B$, which is the type of functions from type A to type B.

Logically, the type $A \to B$ is interpreted as the implication $A \Rightarrow B$ and the type $(x : A) \to B(x)$ is interpreted as the universal quantification $\forall(x : A), B(x)$. Topologically, a function $f : A \to B$ corresponds to a continuous map from A to B. The type $A \to B$ is the mapping space from A to B. We will explain the topological interpretation of $(x : A) \to B(x)$ at the end of Section 2.1.2.

We can define the identity function

$$\mathsf{id} \equiv \mathsf{id}_A :\equiv \lambda(x : A).\, x : A \to A$$

and the composition of functions: if $f : A \to B$ and $g : B \to C$, then $g \circ f :\equiv \lambda x.\, g(f(x)) : A \to C$. Given $b : B$, we also have a constant function $\mathbf{0}_b :\equiv \lambda x.\, b : A \to B$.

We will often write some arguments of a function implicitly. Such arguments are written with curly braces in the type. For example, given a dependent type C over $\mathbb{N}$, we write

$$g : \{n : \mathbb{N}\} \to C(n) \to C(n+1)$$

---

[4]Formally, $B(a)$ is the term $B(x)$ where we substitute a for x. In Section 2.1.3 we will see that we can treat B as a function into a universe, and that alternatively we can view $B(x)$ and $B(a)$ as function applications.

to emphasize that the first argument of g is implicit. In this case, for $c : C(n)$ we will write $g(c)$ for g applied (implicitly) to n and applied to c. The curly braces are only to indicate how we write function application for functions with this type, for all other purposes the types $\{x : A\} \to B(x)$ and $(x : A) \to B(x)$ are the same.

2.1.2 Pair Types

Given a type family B depending on a type A, we can form the *dependent pair type* (also called *dependent sum type* or *sigma type*)

$$(x : A) \times B(x) \qquad \text{or} \qquad \sum_{x:A} B(x).$$

We will use the former notation in this document. A term of type $(x : A) \times B(x)$ is a pair consisting of an element $a : A$ and an element $b : B(a)$. Given $a : A$ and $b : B(a)$, we can form the term $(a, b) : (x : A) \times B(x)$, and we have projections

$$p_1 : (x : A) \times B(x) \to A \quad \text{and} \quad p_2 : (z : (x : A) \times B(x)) \to B(p_1(z)).$$

We will sometimes write $x.i$ for $p_i(x)$. There are beta rules $p_1(a, b) \equiv a$ and $p_2(a, b) \equiv b$ and an eta rule stating that for any $z : (x : A) \times B(x)$ we have $z \equiv (p_1 z, p_2 z)$. In Lean, there is no eta rule for dependent pair types, but instead there is an induction principle, similar to those of inductive types (see Section 2.1.4).

If $B(x)$ does not depend on x, we write $(x : A) \times B$ simply as $A \times B$. In this case we retrieve the usual cartesian product of A and B.

Logically we can think of $A \times B$ as the conjunction of A and B, as described above. Furthermore, we can think of $(x : A) \times B(x)$ as a proof-relevant version of the existential quantifier $\exists (x : A).B(x)$. It is proof-relevant in the sense that from a proof of $(x : A) \times B(x)$ we can extract a witness $a : A$ such that $B(a)$ holds. In Section 2.2.4 we will define an existential quantifier from which the witness cannot be extracted.

Topologically, we think of $A \times B$ as the product space of A and B. The map $p_1 : (x : A) \times B(x) \to A$ corresponds to a *fibration*. A fibration is a map that has the homotopy lifting property with respect to any space, which is given by transport, to be defined in Section 2.2.1. Under this interpretation, $(x : A) \times B(x)$ is the total space of the fibration p_1, and $B(a)$ is the *fiber* of p_1 at point a. The type $(x : A) \to B(x)$ is the type of sections of p_1. These observations are usually summarized as "dependent types correspond to fibrations." We will often call dependent functions *sections*.

2.1.3 Universes

In our discussions below we need one or more *universes* in our type theory. There are different styles of universes in type theory [ML84], we will describe the universes á la Russell. A universe $\mathcal{U}$ is a type that has types as its terms. That is to say, if $A : \mathcal{U}$, then

A is a type. It is closed under all type-forming operations. For example, for pi-types this means that if $A : \mathcal{U}$ and for $a : A$ we have $B(a) : \mathcal{U}$, then

$$(a : A) \to B(a) : \mathcal{U}.$$

We can now interpret dependent types in $\mathcal{U}$, such as B above, as functions $B : A \to \mathcal{U}$.

In the proofs in this document we can often get away with assuming only a single universe. However, it is useful to have the property that all types have a type themselves, and we cannot do that with a single universe $\mathcal{U}$, because positing $\mathcal{U} : \mathcal{U}$ is inconsistent [Gir72]. Instead, we will assume that we have a tower of universes

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

such that for every type A there is an i such that $A : \mathcal{U}_i$. In this case every dependent type can be interpreted as a function $A \to \mathcal{U}_i$ for some i. As is customary, we usually omit writing universe levels explicitly, and we will perform constructions polymorphic over all universes. For example, if we write

$$\mathsf{id} : \{A : \mathcal{U}\} \to A \to A,$$

we really mean that for any universe level i we have

$$\mathsf{id}_i : \{A : \mathcal{U}_i\} \to A \to A.$$

One rule that is sometimes assumed is *universe cumulativity*, which states that if $A : \mathcal{U}_i$, then $A : \mathcal{U}_j$ for $j \geq i$. This can be problematic, and lead to violation of nice properties of the type theory, such as subject reduction or canonicity [Luo12]. In this document (and in Lean), we do not assume universe cumulativity. Instead, using inductive types (see Section 2.1.4) we can construct for $A : \mathcal{U}_i$ a new type $\mathsf{lift}\, A : \mathcal{U}_j$ for $j \geq i$ such that $A \simeq \mathsf{lift}\, A$.

2.1.4 Inductive Types

Inductive types are types that are inductively generated by some *constructors*. A simple example is $\mathbb{N}$, which is inductively generated by 0 and the successor function $S :\equiv \lambda x.\, x + 1$. In this section we will discuss some inductive types that we will need in this dissertation. We will talk about the empty type, the unit type, the booleans, the natural numbers and the sum type. The dependent pair type (Section 2.1.2) is also an inductive type.

The empty type

The empty type $\mathbf{0} : \mathcal{U}_0$ is a type without inhabitants. There are no constructors, and we have as induction principle that if $P : \mathbf{0} \to \mathcal{U}_i$, then

$$\mathsf{ind}_{\mathbf{0}} : (x : \mathbf{0}) \to P(x).$$

This conveys that $\mathbf{0}$ indeed has no inhabitants, because if we view P as a predicate, we can prove anything about all inhabitants of $\mathbf{0}$. We can define negation $\neg A :\equiv A \to \mathbf{0}$.

The unit type

The unit type $\mathbf{1} : \mathcal{U}_0$ is a type with exactly one inhabitant $\star : \mathbf{1}$. The induction principle states that if $P : \mathbf{1} \to \mathcal{U}_i$, then

$$\mathsf{ind_1} : P(\star) \to (x : \mathbf{1}) \to P(x).$$

This states that $\star$ is the only inhabitant of $\mathbf{1}$, because if we can prove something for $\star$, then it holds for all inhabitants of $\mathbf{1}$. There is a computation rule

$$\mathsf{ind_1}(p, \star) \equiv p.$$

The booleans

The type of booleans $\mathbf{2} : \mathcal{U}_0$ has exactly two inhabitants $1_\mathbf{2}, 0_\mathbf{2} : \mathbf{2}$. Its induction principle states that if $P : \mathbf{2} \to \mathcal{U}_i$, then

$$\mathsf{ind_2} : P(1_\mathbf{2}) \to P(0_\mathbf{2}) \to (x : \mathbf{2}) \to P(x).$$

The computation rules are

$$\mathsf{ind_2}(p_{1_\mathbf{2}}, p_{0_\mathbf{2}}, 1_\mathbf{2}) \equiv p_{1_\mathbf{2}} \qquad \text{and} \qquad \mathsf{ind_2}(p_{1_\mathbf{2}}, p_{0_\mathbf{2}}, 0_\mathbf{2}) \equiv p_{0_\mathbf{2}}.$$

The natural numbers

A more interesting type is the type of natural numbers $\mathbb{N} : \mathcal{U}_0$. It has a constructor $0 : \mathbb{N}$ and a unary constructor $S : \mathbb{N} \to \mathbb{N}$, and it is freely generated by these constructors. This means that if $P : \mathbb{N} \to \mathcal{U}_i$ and if we have $p_0 : P(0)$ and $p_S : (k : \mathbb{N}) \to P(k) \to P(S\ k)$, then

$$\mathsf{ind}_\mathbb{N}(p_0, p_S) : (n : \mathbb{N}) \to P(n).$$

If we view P as a predicate, this is the usual induction principle for $\mathbb{N}$: to prove something for all numbers we need to prove it for 0 and we need to prove it for $k+1$ assuming it holds for k, for an arbitrary k. However, this induction principle also allows us to define (dependent) functions from $\mathbb{N}$. These functions satisfy the computation rules

$$\mathsf{ind}_\mathbb{N}(p_0, p_S, 0) \equiv p_0 \qquad \text{and} \qquad \mathsf{ind}_\mathbb{N}(p_0, p_S, S\ n) \equiv p_S(n, \mathsf{ind}_\mathbb{N}(p_0, p_S, n)).$$

Often, we will want to give a name f to $\mathsf{ind}_\mathbb{N}(p_0, p_S)$, and we will instead denote the recursive definition of f using pattern matching notation:

$$f(0) :\equiv p_0 \qquad \text{and} \qquad f(S\ n) :\equiv p_S(n, f(n)).$$

For example, we can define addition and multiplication $+, \cdot : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ recursively (in the second argument) as

$$n + 0 :\equiv n \qquad\qquad\qquad n \cdot 0 :\equiv 0$$
$$n + (S\ m) :\equiv S(n + m) \qquad\qquad n \cdot (S\ m) :\equiv n \cdot m + n.$$

Note that $n + 1 \equiv S\ n$, and we will often write $n + 1$ instead of $S\ n$ from now on.

The sum type

Given two types A and B, we can form the *sum type* or *coproduct* $A + B$ with constructors $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$. The induction principle states that for $P : A + B \to \mathcal{U}$ with maps $p_{\mathsf{inl}} : (a : A) \to P(\mathsf{inl}\, a)$ and $p_{\mathsf{inr}} : (b : B) \to P(\mathsf{inr}\, b)$ we get a section

$$\mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}) : (x : A + B) \to P(x)$$

with computation rules

$$\mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}, \mathsf{inl}(a)) \equiv p_{\mathsf{inl}}(a) \qquad \text{and} \qquad \mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}, \mathsf{inr}(b)) \equiv p_{\mathsf{inr}}(b)$$

Logically, the type $A + B$ is the proof-relevant disjunction of A and B. It is proof-relevant in the sense that a proof of $A + B$ is of the form $\mathsf{inl}\, a$ or $\mathsf{inr}\, b$. Therefore, a proof comes with a proof of either A or B. In Section 2.2.4 we will see a disjunction that does not have this property.

General Inductive Types

In Section 2.1.4 we saw various instances of inductive types. Also the sigma-types from Section 2.1.2 (without eta rule) are an instance of an inductive type. We will now explain inductive types and families of inductive types in general. For a more detailed description, see [Car18, Section "Inductive Types"].

When defining an inductive type, we have to list its constructors. For example, we could define the sum type as follows. Given $A\ B : \mathcal{U}$, we define

```
inductive A + B : U :=
```
• $\mathsf{inl} : A \to A + B$;
• $\mathsf{inr} : B \to A + B$.

This defines the type $A + B$ with constructors inl and inr of the specified type. Each constructor must have as target the inductive type currently being defined (in this case $A + B$).[5] Constructors can be *recursive*, meaning that the type being defined can occur in the domain of a constructor. For example, here is the type of ω-branching trees with leaves labeled by a type C.

```
inductive ω-tree_C : U :=
```
• $\mathsf{leaf} : C \to \omega\text{-tree}_C$;
• $\mathsf{node} : (\mathbb{N} \to \omega\text{-tree}_C) \to \omega\text{-tree}_C$.

A restriction on recursive constructors is that the inductive type being defined can only occur in strictly positive positions, that is as the target of one of the arguments of the constructor.

Every inductive type has an induction principle. We can algorithmically find the type of the induction principle from the constructors. The first argument of the induction

---

[5]For *higher inductive types* (Section 2.2.6) the conclusion can also be a (higher) path in the type currently being defined.

principle (often left implicit) is the *motive*, which is an arbitrary type family over the inductive type being defined, for $\omega\text{-tree}_C$ this has type $P : \omega\text{-tree}_C \to \mathcal{U}$. Then for every constructor c there is an argument that mimics the type of c and has as target $P(c(\cdots))$. For $\omega\text{-tree}_C$ these arguments have type $p_{\mathsf{leaf}} : (c : C) \to P(\mathsf{leaf}\, c)$ and

$$p_{\mathsf{node}} : (f : \mathbb{N} \to \omega\text{-tree}_C) \to ((n : \mathbb{N}) \to P(f\, n)) \to P(\mathsf{node}\, f).$$

Note that for each recursive argument f of the constructor we assume an *induction hypothesis* of type $P(f(\cdots))$. The induction principle then gives a section of P. So for example we get

$$\mathsf{ind}_{\omega\text{-tree}_C}(p_{\mathsf{leaf}}, p_{\mathsf{node}}) : (x : \omega\text{-tree}_C) \to P(x).$$

Finally, the computation rules states that if the induction principle acts on a constructor, then it will reduce to the argument corresponding to that constructor. For $\omega\text{-tree}_C$ this means (abbreviating $s :\equiv \mathsf{ind}_{\omega\text{-tree}_C}(p_{\mathsf{leaf}}, p_{\mathsf{node}})$)

$$s(\mathsf{leaf}(c)) \equiv p_{\mathsf{leaf}}(c) \qquad \text{and} \qquad s(\mathsf{node}(f)) \equiv p_{\mathsf{node}}(f, \lambda n.\, s(f\, n)),$$

where applying s to the recursive constructor leads to a recursive call of s.

One important generalization of inductive types are families of inductive types. In this case, a family of types P is being defined simultaneously indexed over some type I. In this case, constructors must have as target $P(t)$ where t is a term of type I formed by the (nonrecursive) arguments of the constructor. An example of an inductive family of types is the type of vectors in A of some length $n : \mathbb{N}$.

```
inductive vector_A : ℕ → 𝒰 :=
```
- $\mathsf{nil} : \mathsf{vector}_A(0)$;
- $\mathsf{cons} : \{n : \mathbb{N}\} \to A \to \mathsf{vector}_A(n) \to \mathsf{vector}_A(n+1)$.

Note that the parameter A remains fixed in the definition of $\mathsf{vector}_A(n)$, while the index $n : \mathbb{N}$ is not: the constructor cons constructs a vector of length $n+1$ from a vector of length n. The induction principle can again be extracted algorithmically. It is important that the motive also quantifies over all indices of the inductive family. For vectors it states that given a motive

$$P : \{n : \mathbb{N}\} \to \mathsf{vector}_A(n) \to \mathcal{U}$$

and induction steps

$$\begin{aligned} p_{\mathsf{nil}} &: P(\mathsf{nil}) \\ p_{\mathsf{cons}} &: (n : \mathbb{N}) \to (a : A) \to (x : \mathsf{vector}_A(n)) \to P(x) \to P(\mathsf{cons}(a, x)), \end{aligned}$$

we get a section

$$\mathsf{ind}_{\mathsf{vector}}(p_{\mathsf{nil}}, p_{\mathsf{cons}}) : \{n : \mathbb{N}\} \to (x : \mathsf{vector}_A(n)) \to P(x)$$

with the expected computation rules.

A very important inductive family of types is the *identity type.*[6] This is a family of types with parameters $A : \mathcal{U}$ and $a : A$ and is defined as

```
inductive Id_A(a,-) : A → U :=
• refl_a : Id_A(a,a).
```

We also denote the type $\mathsf{Id}_A(a_1, a_2)$ by $a_1 =_A a_2$ or $a_1 = a_2$ and refl_a by refl, 1_a or 1. Its induction principle states that for a family $P : (a' : A) \rightarrow a = a' \rightarrow \mathcal{U}$ and a term $p_{\mathsf{refl}} : P(a, 1_a)$ we find a section

$$\mathsf{ind}_=(p_{\mathsf{refl}}) : (a' : A) \rightarrow (p : a = a') \rightarrow P(a', p).$$

In words: we may assume that a path with free right endpoint (that is, the right hand side of the equality is a variable) is reflexivity.

Logically, the identity type corresponds to equality. Under this interpretation, a term of type $a_1 = a_2$ is a proof that a_1 and a_2 are equal. Homotopically, the identity type corresponds to the path space of A, and we will explore this interpretation more in Section 2.2.1.

2.2 Homotopy Type Theory

We will now discuss in more detail the homotopical interpretation of types, and the basic concepts of homotopy type theory.

2.2.1 Paths

Elements of an identity type form paths in the space. We can define the usual operations on paths.

Given a path $p : a =_A b$, we can define the *inverse* $p^{-1} : b =_A a$. We can do this by path induction. Define the family

$$P :\equiv \lambda(x : A). \lambda(q : a =_A x). x =_A a : (x : A) \rightarrow a =_A x \rightarrow \mathcal{U}.$$

We now have $\mathsf{refl}_a : P(a, p) \equiv a =_A a$, and therefore we get

$$p^{-1} :\equiv \mathsf{ind}_=(\mathsf{refl}_a, b, p) : b =_A a.$$

The computation rule gives that $\mathsf{refl}_a^{-1} \equiv \mathsf{refl}_a$.

We can explain the proof in words more intuitively. Path induction states that we may assume that a path with a free endpoint is reflexivity. Since p has a free endpoint (b is a variable), we may assume that $b \equiv a$ and $p \equiv \mathsf{refl}_a$. In this case, we can define

$$p^{-1} \equiv \mathsf{refl}_a^{-1} :\equiv \mathsf{refl}_a : a = a.$$

---

[6]also called *path type*, *identification type* or *equality type*.

The map path inversion we have defined this way has type

$$\{a\ b : A\} \to a = b \to b = a.$$

We can also define path *concatenation*. Given $p : a =_A b$ and $q : b =_A c$, we define $p \cdot q : a =_A c$ again by path induction. We will only give the intuitive argument and leave the formal proof to the reader. Since q has free endpoint c, we may assume that $c \equiv b$ and $q \equiv \mathsf{refl}_b$. In this case, we define $p \cdot \mathsf{refl}_b :\equiv p : a = b$.

We can also define higher paths. For example, given $p : a = b$ and $q : b = c$ and $r : c = d$, we have a path

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

which is the *associativity* of path concatenation. We can prove this by path induction on r: if r is reflexivity, then both sides reduce to $p \cdot q$.

By using path induction, we can also prove the following equalities:

$$p \cdot 1 = p \qquad\qquad p \cdot p^{-1} = 1$$
$$1 \cdot p = p \qquad\qquad p^{-1} \cdot p = 1.$$

It is trickier to prove the Eckmann-Hilton property of equality, which states that given $a : A$ and $p, q : \mathsf{refl}_a = \mathsf{refl}_a$, we have $p \cdot q = q \cdot p$. The problem is that cannot apply path induction to p or q directly. We omit the proof here and refer to [Uni13, Theorem 2.1.6].

Given a map $f : A \to B$, we can prove that f respects paths. Given a path $p : a =_A a'$, we define $\mathsf{ap}_f(p) : f(a) =_B f(b)$ by path induction: for reflexivity we define $\mathsf{ap}_f(\mathsf{refl}_a) :\equiv \mathsf{refl}_{f(a)}$. We will sometimes abuse notation and write $f(p)$ for $\mathsf{ap}_f(p)$. From a logical perspective this just states that functions respect equality, but from a homotopical perspective, this states that functions respect paths, which is in line with our intuition that all functions are continuous in HoTT.

We can compute what ap does when our map is the identity map, a constant map or a composition of maps:

$$\mathsf{ap}_{\mathsf{id}_A}(p) = p \qquad\qquad \mathsf{ap}_{0_b}(p) = \mathsf{refl}_b \qquad\qquad \mathsf{ap}_{g \circ f}(p) = \mathsf{ap}_g(\mathsf{ap}_f(p)).$$

All three of these properties are easily proven by path induction. Also, we can compute ap when we apply it to inverses or concatenations of paths:

$$\mathsf{ap}_f(p \cdot q) = \mathsf{ap}_f(p) \cdot \mathsf{ap}_f(q) \qquad\qquad \mathsf{ap}_f(p^{-1}) = (\mathsf{ap}_f(p))^{-1}.$$

Given a dependent type $P : A \to \mathcal{U}$ and a path $p : a =_A a'$, we can define the *transport* function $\mathsf{transport}^P(p) : P(a) \to P(a')$. We define it by path induction; for reflexivity we define $\mathsf{transport}^P(\mathsf{refl}_a) :\equiv \mathsf{id}_{P(a)}$. When P is known from context we will write $p_*(b)$ for $\mathsf{transport}^P(p, b)$.

By path induction we can prove basic equalities about transports. We have

$$\mathsf{transport}^P(p \cdot q, x) = \mathsf{transport}^P(q, \mathsf{transport}^P(p, x))$$
$$\mathsf{transport}^{\lambda a.\, B}(p, x) = x$$
$$\mathsf{transport}^{P \circ f}(p, x) = \mathsf{transport}^P(\mathsf{ap}_f(p), x)$$
$$f_{a'}(\mathsf{transport}^P(p, x)) = \mathsf{transport}^Q(p, f_a(x)) \qquad \text{for } f : (a : A) \to P(a) \to Q(a).$$

2.2.2 Equivalences

In this section we talk about maps between types that have an inverse in a suitable way. Before we can give the definition, we need to define homotopy.

Given two dependent maps $f, g : (a : A) \to B(a)$, a *homotopy* $h : f \sim g$ is a proof that f and g are pointwise equal:

$$(f \sim g) :\equiv (a : A) \to f(a) =_{B(a)} g(a).$$

Recall that all maps are considered continuous, so this actually gives a continuous deformation of f to g, which is exactly what a homotopy is in topology.

Definition 2.2.1. Suppose given a function $f : A \to B$.

- A *left-inverse* of f is an inhabitant of $(g : B \to A) \times g \circ f \sim \mathsf{id}_A$.

- Similarly, a *right-inverse* of f is an inhabitant of $(h : B \to A) \times f \circ h \sim \mathsf{id}_B$.

- We say that f is an *equivalence* or $\mathsf{isequiv}(f)$ if f has both a left and a right inverse. We will denote its left-inverse by f^{-1}. We can then show that f^{-1} is also a right inverse of f.

- The type of equivalences between A and B is $(A \simeq B) :\equiv (f : A \to B) \to \mathsf{isequiv}(f)$. Given an element $f : A \simeq B$, we will also use f to denote the underlying map $A \to B$.

It is easy to show that the identity map $\mathsf{id}_A : A \simeq A$ is an equivalence. Moreover, if $g : B \to C$ and $f : A \to B$ are both equivalences, then $g \circ f$ and f^{-1} are also equivalences. This shows that equivalences are reflexive, symmetric and transitive.

A very important property is that any two inhabitants of $\mathsf{isequiv}(f)$ are equal: if $p, q : \mathsf{isequiv}(f)$, then $p = q$. We will not prove this here, but it is shown in [Uni13, Theorem 4.3.2]. This property is the reason that we define the notion of equivalences this way. If we would define $\mathsf{isequiv}(f)$ by requiring a map that is both a left *and* a right inverse of f, then this property would not hold.

Given two equivalences $f, f' : A \simeq B$, it does not matter whether we compare them as functions or equivalences:

$$(f =_{A \simeq B} f') \simeq (f =_{A \to B} f') \simeq (f \sim f').$$

By path induction we also get a map $(A =_{\mathcal{U}} B) \to (A \simeq B)$, because if the path $p : A = B$ is reflexivity, we can just take $\mathsf{id}_A : A \simeq A$ as our equivalence. In plain Martin Löf type theory one cannot characterize what the type $A = B$ is. This is where the univalence axiom comes in. The *univalence axiom* states that the map

$$(A = B) \to (A \simeq B)$$

is an equivalence. In particular this means that we get a map in the other direction: given an equivalence $e : A \simeq B$, we get an equality $\mathsf{ua}(e) : A = B$.

2.2.3   More on paths

In this section we will discuss dependent paths, or pathovers; higher paths, such as squares and cubes; and paths in type formers.

Pathovers

We will often need to relate elements in two different fibers of a dependent type. Suppose we have a family $P : A \to \mathcal{U}$ with $x : P(a)$ and $x' : P(a')$. If we have a path $p : a = a'$, we can form the type $x =_p^P x'$ of *dependent paths* or *pathovers* over p. There are four equivalent ways to define this:

(i) We can define $(x =_p^P x') :\equiv (\mathsf{transport}^P(p, x) = x')$

(ii) We can define $(x =_p^P x') :\equiv (x = \mathsf{transport}^P(p^{-1}, x'))$

(iii) We can define $(x =_p^P x')$ by path induction on p. If $p :\equiv \mathsf{refl}_a$, we define $(x =_p^P x') \equiv (x =_{\mathsf{refl}_a}^P x') :\equiv (x =_{P(a)} x')$.

(iv) We can define $(x =_p^P x')$ by a family of inductive types. For fixed $A : \mathcal{U}$ and $P : A \to \mathcal{U}$ and $a : A$ and $x : P(a)$ we have the following family:

> `inductive` $x =_{(-)}^P (-) : \{a' : A\} \to a = a' \to P(a') \to \mathcal{U} :=$
> $\bullet$ $\mathsf{refl} : x =_{\mathsf{refl}_a}^P x.$

It does not matter which of these definitions we pick, because we can prove that all of them are equivalent.[7]

We have the following equivalences between pathovers:

$$(x =_p^{\lambda a.\, B} x') \simeq (x =_B x') \qquad\qquad (x =_p^{P \circ f} x') \simeq (x =_{\mathsf{ap}_f(p)}^P x').$$

---

[7]In Lean, we chose option (iv). Option (i) would probably be slightly more convenient to work with, because then this characterization becomes a definitional equality. In practice it will not matter much, though.

We can do operations on pathovers, similar to the operations on paths. We have concatenation and inversion, and we will abuse notation and denote them with the same notation.

$$(-) \cdot (-) : x_1 =_p^P x_2 \to x_2 =_q^P x_3 \to x_1 =_{p \cdot q}^P x_3$$
$$(-)^{-1} : x_1 =_p^P x_2 \to x_2 =_{p^{-1}}^P x_1.$$

We have a dependent version of ap. Given a dependent map $f : (a : A) \to P(a)$, we get

$$\mathsf{apd}_f : (p : a = a') \to f(a) =_p^P f(a').$$

A variant to apd is the following. Given $f : A \to B$, a family $P : B \to \mathcal{U}$ and a section $g : (a : A) \to P(f(a))$, we define

$$\widetilde{\mathsf{apd}}_g : (p : a = a') \to g(a) =_{\mathsf{ap}_f(p)}^P g(a'). \tag{2.2.2}$$

The difference between apd and $\widetilde{\mathsf{apd}}$ is over which path they lie.

Furthermore, if we have a map $f : A \to B$ and two families $P : A \to \mathcal{U}$ and $Q : B \to \mathcal{U}$ and a fiberwise map $g : (a : A) \to P(a) \to Q(f(a))$, then we get a fiberwise version of ap:

$$\mathsf{apo}_g : x =_p^P x' \to g_a(x) =_{\mathsf{ap}_f(p)}^Q g_{a'}(x'). \tag{2.2.3}$$

Squares

For higher paths, it is convenient to define a separate notion of a square in a type:

$$
\begin{array}{ccc}
a_{00} & \overset{p_{10}}{=\!=\!=} & a_{20} \\
\Vert {\scriptstyle p_{01}} & & \Vert {\scriptstyle p_{21}} \\
a_{02} & \underset{p_{12}}{=\!=\!=} & a_{22}
\end{array}
$$

Suppose given four paths as in the diagram above, that is

$$
\begin{aligned}
p_{10} &: a_{00} = a_{20} & p_{01} &: a_{00} = a_{02} \\
p_{12} &: a_{02} = a_{22} & p_{21} &: a_{20} = a_{22}.
\end{aligned}
$$

We have a type of squares $\mathsf{square}(p_{10}, p_{12}, p_{01}, p_{21})$, which we can define in either of the two following equivalent ways

(i) We can define $\mathsf{square}(p_{10}, p_{12}, p_{01}, p_{21}) :\equiv (p_{10} \cdot p_{21} = p_{01} \cdot p_{12})$.

(ii) $\mathsf{square}(p_{10}, p_{12}, p_{01}, p_{21})$ is defined as an inductive family of types. For a fixed $a_{00} : A$ we define the family

18

```
inductive square(−, −, −, −) : {a₂₀ a₀₂ a₂₂ : A} → a₀₀ = a₂₀ → a₀₂ = a₂₂ →
```
$\mathsf{inductive\ square}(-,-,-,-) : \{a_{20}\ a_{02}\ a_{22} : A\} \to a_{00} = a_{20} \to a_{02} = a_{22} \to$
$a_{00} = a_{02} \to a_{20} = a_{22} \to \mathcal{U} :=$
- $\mathsf{refl} : \mathsf{square}(\mathsf{refl}_{a_{00}}, \mathsf{refl}_{a_{00}}, \mathsf{refl}_{a_{00}}, \mathsf{refl}_{a_{00}}).$

We will usually write squares using diagrams as above. There are various operations on squares. For example, we can horizontally concatenate them. If we can fill each of the individual squares below, we can fill the outer rectangle (which has as top $p_{10} \cdot p_{30}$ and as bottom $p_{12} \cdot p_{32}$).

$$
\begin{array}{ccccc}
a_{00} & \overset{p_{10}}{=\!=\!=} & a_{20} & \overset{p_{30}}{=\!=\!=} & a_{40} \\
\Vert p_{01} & & \Vert p_{21} & & \Vert p_{41} \\
a_{02} & \underset{p_{12}}{=\!=\!=} & a_{22} & \underset{p_{32}}{=\!=\!=} & a_{42}
\end{array}
$$

We can also vertically concatenate squares, and horizontally or vertically invert squares.

Given a homotopy $h : f \sim g$ between nondependent functions $f, g : A \to B$ and a path $p : a =_A a'$, we get the following *naturality square*.

$$
\begin{array}{ccc}
f(a) & \overset{h(a)}{=\!=\!=} & g(a) \\
\mathsf{ap}_f(p) \Vert & & \Vert \mathsf{ap}_g(p) \\
f(a') & \underset{h(a')}{=\!=\!=} & g(a')
\end{array}
$$

Squareovers and cubes

Going up further, we have the type of *squareovers*. A squareover is a square in a dependent type over a square. Suppose that we have a dependent type $P : A \to \mathcal{U}$, a square s in A and a dependent path over each of the sides of the square, as in the following diagram.

$$
\begin{array}{ccc}
x_{00} & \overset{q_{10}}{=\!=\!=} & x_{20} \\
\Vert q_{01} & & \Vert q_{21} \\
x_{02} & \underset{q_{12}}{=\!=\!=} & x_{22}
\end{array}
$$

$$\downarrow$$

$$
\begin{array}{ccc}
a_{00} & \overset{p_{10}}{=\!=\!=} & a_{20} \\
p_{01} \Vert & s & \Vert p_{21} \\
a_{02} & \underset{p_{12}}{=\!=\!=} & a_{22}
\end{array}
$$

19

We have the type of *squareovers* or *dependent squares*, which fill the top square and lie over the bottom square. We can again define this using multiple methods, but the most convenient method here is to define it as an inductive family. We take as parameters the type A, the family P and the points a_{00} and x_{00} and let all the other arguments be indices. We have a "reflexivity squareover" when the square s is the reflexivity square and each of the four pathovers are reflexivity pathovers.

We can also define a type of cubes. Given six squares in a type with twelve paths as sides, fitting together in a cube, we can define the type of fillers of the cube. This is again done using a family of inductive types, where we give a cube filler when all the six sides are reflexivity squares. Of course, we could continue by defining cubeovers and 4-cubes, but we will not need them in this dissertation.

Paths in type formers

In each of the type formers of Section 2.1 we can compute what the paths in that type are, and what the operations of paths are in that type.

As a simple example, consider the cartesian product type $A \times B$. A path in the cartesian product is just a pair of paths.

$$(x =_{A \times B} y) \simeq (p_1 x =_A p_1 y) \times (p_2 x =_B p_2 y)$$

In particular, given paths $r : p_1 x = p_1 y$ and $s : p_2 x = p_2 y$, we get a path $x = y$, which we will denote (r, s). Given maps $f : A \to A'$ and $g : B \to B'$, we get the map $f \times g : A \times B \to A' \times B'$ and we can compute

$$\mathsf{ap}_{f \times g}(r, s) = (\mathsf{ap}_f(r), \mathsf{ap}_g(s))$$

. Given families $P, Q : A \to \mathcal{U}$, we can compute transport:

$$\mathsf{transport}^{\lambda a.\, P(a) \times Q(a)}(p, (x, y)) = (\mathsf{transport}^P(p, x), \mathsf{transport}^Q(p, y))$$

Pathovers in a family of cartesian products are also pairs of pathovers:

$$(x, y) =_p^{\lambda a.\, P(a) \times Q(a)} (x', y') \simeq (x =_p^P x') \times (y =_p^Q y').$$

In sigma-types the relations are a bit more difficult, since the second component depends on the first. In the type $(a : A) \times B(a)$ paths are pairs of a path and a path over that path:

$$(x =_{(a:A) \times B(a)} y) \simeq (r : p_1 x =_A p_1 y) \times (p_2 x =_r^B p_2 y)$$

We will also denote in this case the map from right to left by $(-, -)$. Given a map $f : A \to A'$ and a fiberwise map $g : (a : A) \to B(a) \to B'(f(a))$, we get a functorial action of the sigma type: $f \times g : ((a : A) \times B(a)) \to ((a' : A') \times B(a'))$. In this case, we can compute

$$\mathsf{ap}_{f \times g}(r, s) = (\mathsf{ap}_f(r), \mathsf{apo}_g(s)),$$

20

where apo is defined in (2.2.3). We leave the rule for transports as an exercise to the reader, but the rule for pathovers in a family of sigma-types is the following. For $B : A \to \mathcal{U}$ and $C : (a : A) \to B(a) \to \mathcal{U}$ we get:[8]

$$((a, b) =_p^{\lambda a.\, (b:B(a)) \times C(a,b)} (a', b')) \simeq (q : a =_p^P a') \times (y =_{(p,q)}^{\lambda(x\,:\,(a:A) \times B(a)).\, Q(p_1 x, p_2 x)} y').$$

For dependent function types the situation is a bit more complicated. Given $f, g : (a : A) \to B(a)$, by path induction we get a map

$$\mathsf{happly} : (f = g) \to f \sim g.$$

However, we cannot show in plain Martin-Löf type theory that this map gives rise to an equivalence. In homotopy type theory we can use the univalence axiom (see Section 2.2.2) to show that happly is an equivalence. We skip the proof here, but refer the reader to [Uni13, Section 4.9]. Using univalence we can also prove the other properties. The general rule for pathovers in a dependent function type is complicated, but two important special cases are the following. In the first case, the domain does not depend on the path. We have types A and B and a family $C : A \to B \to \mathcal{U}$ and then we can prove:

$$(f =_p^{\lambda a.\, (b:B) \to C(a,b)} g) \simeq (b : B) \to f(b) =_p^{C(-,b)} g(b).$$

The second case is for nondependent functions. Given a type A and two families $B, C : A \to \mathcal{U}$, we have

$$(f =_p^{\lambda a.\, B(a) \to C(a)} g) \simeq (b : B(a)) \to f(b) =_p^C g(p_*(b)).$$

We characterized paths in the universe in Section 2.2.2 using the univalence axiom. We will not need to do much path algebra in inductive types, except for the identity type, pathover type and square type. A pathover in a family of identity types is a square. Suppose given types A and B and functions $f, g : A \to B$, a path $p : a =_A a'$ and paths $q : f(a) = g(a)$ and $r : f(a') = g(a')$. Then the pathover type becomes equivalent to the square type shown below.

$$(q =_p^{\lambda a.\, f(a)=g(a)} r) \simeq \quad
\begin{array}{ccc}
f(a) & \overset{q}{=\!=\!=} & g(a) \\
\Vert \mathsf{ap}_f(p) & & \Vert \mathsf{ap}_g(p) \\
f(a') & \overset{r}{=\!=\!=} & g(a')
\end{array}$$

We also sometimes encounter a pathover in a dependent family of pathovers. In that case we get a squareover. Suppose we are given functions $f, g : A \to B$, and a homotopy $h : f \sim g$, a dependent family $C : B \to \mathcal{U}$ and sections $c : (a : A) \to C(f(a))$

---

[8]We could define a new notion "path over a pathover," but the rule given here suffices for all the cases we considered.

and $c' : (a : A) \to C(g(a))$. We want to characterize a pathover in the family $P :\equiv \lambda a.\, c(a) =^C_{h(a)} c'(a) : A \to \mathcal{U}$. If we are also given a path $p : a =_A a'$ and two pathovers $q : c(a) =^C_{h(a)} c'(a)$ and $q' : c(a') =^C_{h(a')} c'(a')$, then the pathover $q =^P_p q'$ is equivalent to the following squareover, where $\widetilde{\mathsf{apd}}$ is defined in (2.2.2), and the bottom square is a naturality square.

$$
\begin{array}{ccc}
c(a) & \overset{q}{=\!=\!=} & c'(a) \\
\widetilde{\mathsf{apd}}_b(p) \Big\| & & \Big\| \widetilde{\mathsf{apd}}_{b'}(p) \\
c(a') & \overset{q'}{=\!=\!=} & c'(a') \\
& \Big\downarrow & \\
f(a) & \overset{h(a)}{=\!=\!=} & g(a) \\
\mathsf{ap}_f(p) \Big\| & \text{nat.} & \Big\| \mathsf{ap}_g(p) \\
f(a') & \overset{}{\underset{h(a')}{=\!=\!=}} & g(a')
\end{array}
$$

Lastly, we will mention that a pathover in a family of squares is a cube, but we will not explain the details here.

2.2.4   Truncated Types

In HoTT we can define iterated path spaces in any type. In certain types, if we iterate path spaces enough times, these path spaces do not contain any information. These types are called *truncated*. The notion of an n-truncated type, was introduced in 2009 by Vladimir Voevodsky under the name "a type of h-level $n + 2$."

We define the notion that A is n-truncated, or that A is an *n-type* or $\mathsf{is\text{-}n\text{-}type}(A)$ recursively for $n \geq -2$. We say that a type A is (-2)-truncated or *contractible* if it has exactly one inhabitant, i.e. if we can prove

$$(a_0 : A) \times (a : A) \to a = a_0.$$

A type A is $(n + 1)$-truncated if for all $a\ a' : A$ the type $a =_A a'$ is n-truncated.

We can show that **1** is contractible and that every contractible type is equivalent to **1**.

The (-1)-truncated types are called *mere propositions* or *propositions* for short. A type A is a proposition precisely when any two of its inhabitants are equal, i.e. if we can prove

$$(a\ a' : A) \to a = a'.$$

We call these types propositions because these types correspond to truth values, and do not contain any further information. In particular, if a proposition is inhabited, then it

is contractible. It is easy to see that **0** and **1** are mere propositions, and in Section 2.2.2 we saw that the statement $\mathsf{isequiv}(f)$ is a mere proposition.

One level up, the 0-types are called *sets*. These are the types for which uniqueness of identity proofs holds. Examples of sets are $\mathbb{N}$ and **2**.

On the next level we have the 1-types or *groupoids*. Below we list some properties of truncated types, see [Uni13, Section 7.1] for their proofs.

Lemma 2.2.4.

- *If A is n-truncated, then A is m-truncated for all $m \geq n$.*

- *If A is n-truncated and $A \simeq B$, then B is n-truncated.*

- *If A and B are n-truncated types, then $A \times B$ and $A \simeq B$ are n-truncated. If $n \geq 0$, then $A + B$ is also n-truncated.*

- *If $B : A \to \mathcal{U}$ is a family of n-truncated types (i.e. $(a : A) \to \mathsf{is\text{-}n\text{-}type}(B(a))$), then $(a : A) \to B(a)$ is n-truncated. If moreover A is also n-truncated, then $(a : A) \times B(a)$ is also n-truncated.*

- *Given $a_0 : A$, the type $(a : A) \times (a_0 = a)$ is contractible.*

- *The type $\mathsf{is\text{-}n\text{-}type}\, A$ is a mere proposition.*

We define the *subuniverse of n-types* as $\mathcal{U}_{\leq n} :\equiv (X : \mathcal{U}) \times \mathsf{is\text{-}n\text{-}type}(X)$. For $X : \mathcal{U}_{\leq n}$ we will also write X for the underlying type of X. We write $\mathsf{Prop} :\equiv \mathcal{U}_{\leq -1}$ and $\mathsf{Set} :\equiv \mathcal{U}_{\leq 0}$.

We can do set-level mathematics in the subuniverse of sets. For example, we can define a *group* to be a set with operations satisfying the following axiomatization:[9]

$$\mathsf{Group} :\equiv (G : \mathsf{Set}) \times (m : G \to G \to G) \times (i : G \to G) \times (e : G) \times ((x\ y\ z : G) \to$$
$$m(x, m(y, z)) = m(m(x, y), z) \times m(x, e) = x \times m(x, i(x)) = e).$$

A group G is *abelian* if it moreover satisfies $m(x, y) = m(y, x)$ for all $x, y : G$. This gives the usual notion of groups, and we can perform all basic group theory in this setting.

Truncations

We can turn every type A into an n-type $\|A\|_n$ in a universal way, which is called the *n-truncation* of A. It comes with a map $|-|_n : A \to \|A\|_n$ and has the following induction principle. Suppose given $P : \|A\|_n \to \mathcal{U}$ such that $P(x)$ is n-truncated for all $x : \|A\|_n$. If we are given a dependent map $f : (a : A) \to P(|a|_n)$, we get a section

$$\mathsf{ind}_{\|-\|}(f) : (x : \|A\|_n) \to P(x)$$

such that $\mathsf{ind}_{\|-\|}(f, |a|_n) \equiv f(a)$.

We will now state some properties of the n-truncation, for the proofs we refer to [Uni13, Section 7.3].

---

[9]From these equalities the fact that e is a left-identity and i is a left-inverse can be derived.

Lemma 2.2.5.

- *The truncation is functorial. Given $f : A \to B$, we get a map $\|f\|_n : \|A\|_n \to \|B\|_n$. This map respects composition and identities: $\|g \circ f\|_n \sim \|g\|_n \circ \|f\|_n$ and $\|\operatorname{id}_A\|_n \sim \operatorname{id}_{\|A\|_n}$.*

- *A is an n-type iff $|-|_n : A \to \|A\|_n$ is an equivalence.*

- *The equality type in the truncation is truncated equality, but shifted:*

$$(|a|_{n+1} =_{\|A\|_{n+1}} |a'|_{n+1}) \simeq \|a =_A a'\|_n.$$

- *Truncating twice is the same as truncating once:*

$$\|\|A\|_n\|_k \simeq \|A\|_{\min(n,k)}.$$

In particular the *propositional truncation* $\|A\| :\equiv \|A\|_{-1}$ of A is a proposition stating that A is *merely inhabited* [AB04]. We can use it to define *proof irrelevant* versions of the disjunction or existential quantifier. We have the *mere disjunction*

$$(P \vee Q) :\equiv \|P + Q\|$$

and the *mere existential*

$$(\exists(x : A).P(x)) :\equiv \|(x : A) \times P(x)\|.$$

We say that there *merely exists* $x : A$ such that $P(x)$ holds if $\exists(x : A).P(x)$ is inhabited, to contrast with constructing an element in the untrucated dependent pair type. If we construct an element of $(x : A) \times P(x)$, we will sometimes say that there *purely exists* an x such that $P(x)$ holds, but often we will drop the adverb *purely*.

Connected types

A type is truncated if the type contains no interesting information in a high enough dimension. Dually, a type is *connected* if it contains no interesting information in a low enough dimension.

We say that a type A is *n-connected* for $n \geq -2$ if $\|A\|_n$ is contractible. From the definition we see that every type is (-2)-connected. A type is (-1)-connected precisely when it is merely inhabited. A type is called 0-connected when A has exactly one *connected component*. A 1-connected type is called *simply connected*.

Fibers

We can extend the notion of truncated types and connected types to functions. Given a function $f : A \to B$ and a point $b : B$, we define the *fiber* of f at b to be

$$\mathsf{fib}_f(b) :\equiv (a : A) \times f(a) = b.$$

The fiber of the projection $p_1 : ((a : A) \times B(a)) \to A$ at $a : A$ is equivalent to $B(a)$, which explains the terminology that $B(a)$ is the fiber of B over a.

We say that a function $f : A \to B$ is n-truncated (n-connected) when for all $b : B$ the type $\mathsf{fib}_f(b)$ is n-truncated (n-connected). The function f is (-2)-truncated precisely when it is an equivalence. The function f is (-1)-truncated, or an *embedding*, if for all $a\ a' : A$ the map $\mathsf{ap}_f : a =_A a' \to f(a) =_B f(a')$ is an equivalence. A map $f : A \to B$ between sets is an embedding iff it is *injective*, i.e. if we have a map $f(a) = f(a') \to a = a'$ for all $a\ a' : A$. On the other hand, a (-1)-connected map is called a *surjection*, which means that for every $b : B$ there merely exists an $a : A$ such that $f(a) = b$.

Every map can be factorized as an n-connected map followed by an n-truncated map in a unique way, which means that these classes form an *orthogonal factorization system* [RSS17].

Similar to the universe of n-truncated types, we have a universe of n-connected types:

$$\mathcal{U}_{>n} :\equiv (A : \mathcal{U}) \times \mathsf{is\text{-}n\text{-}connected}(A).$$

2.2.5 Pointed Types

A lot of homotopy theory is done in the $(\infty, 1)$-category of pointed types where the morphisms are maps that preserve the basepoints of the types. Below are the basic definitions for pointed types.

Definition 2.2.6.

  (i) A type A is *pointed* if A has a distinguished basepoint $a_0 : A$. For example, $\mathbf{1}$ is pointed by $\star$ and $\mathbf{2}$ is pointed with $0_{\mathbf{2}}$. We will also write $\mathbb{S}^0$ for the pointed type $\mathbf{2}$. $A \times B$ is pointed if both A and B are pointed,[10] $(a : A) \to B(a)$ is pointed if B is a family of pointed types, and $(a : A) \times B(a)$ is pointed if A is pointed and $B(a_0)$ is pointed.

  (ii) The type of *pointed types* is $\mathcal{U}^* :\equiv (A : \mathcal{U}) \times A$. Given a pointed type $A : \mathcal{U}^*$, we will also write A for its underlying type.

---

[10]More formally, we have to specify the basepoint of $A \times B$, because being pointed is structure on a type, not a property of the type, but there is only one choice of basepoint in this example and other examples where we leave the basepoint implicit.

(iii) Given two pointed types $A, B : \mathcal{U}^*$, a *pointed map* $f : A \to^* B$ is a pair consisting of a map $f : A \to B$ and a path f_0 stating that f preserves the basepoint, that is $f_0 : f(a_0) = b_0$. The type $A \to^* B$ is pointed with basepoint $\mathbf{0} \equiv \mathbf{0}_{A,B} :\equiv (\lambda a.\, b_0, \mathsf{refl}_{b_0})$.

(iv) We have an identity pointed map $\mathsf{id} \equiv \mathsf{id}_A : A \to^* A$ defined as $(\lambda x.\, x, \mathsf{refl}_{a_0})$ and if $g : B \to^* C$ and $f : A \to^* B$ we have a composite $g \circ f : A \to^* C$ defined as $(\lambda x.\, g(f(x)), \mathsf{ap}_g(f_0) \cdot g_0)$.

(v) More generally, Given a pointed type $A : \mathcal{U}^*$ and a family of types $B : A \to \mathcal{U}$ with a basepoint $b_0 : B(a_0)$, a *pointed dependent map* $f : (a : A) \to^* B(a)$ is a pair consisting of a dependent map $f : (a : A) \to B(a)$ and a path $f_0 : f(a_0) = b_0$. If we require that B is a family of pointed types, i.e. $B : A \to \mathcal{U}^*$, then $(a : A) \to^* B(a)$ is pointed with basepoint $(\lambda a.\, b_0(a), \mathsf{refl}_{b_0(a_0)})$.

(vi) Given two pointed dependent maps $f, g : (a : A) \to^* B(a)$, a *pointed homotopy* $h : f \sim^* g$ is a pointed dependent map $(a : A) \to^* f(a) = g(a)$. This is well-defined, since the type $f(a_0) = g(a_0)$ is pointed by $f_0 \cdot g_0^{-1}$. Expanding the definition, this means that h is a pair of a homotopy $h : f \sim g$ and a 2-path stating that h relates the basepoint-preserving paths of f and g. This means that we have $h_0 : h(a_0) = f_0 \cdot g_0^{-1}$, or equivalently, $h_0 : h(a_0) \cdot g_0 = f_0$. We say that a diagram of pointed types commutes if there are pointed homotopies between the corresponding composites of pointed maps.

(vii) A pointed map $e : A \to^* B$ is a *pointed equivalence* if it has a left-inverse and a right-inverse. That is, there is $\ell : B \to^* A$ such that $\ell \circ e \sim^* \mathsf{id}_A$ and $r : B \to^* A$ such that $e \circ r \sim^* \mathsf{id}_B$. The type of pointed equivalences between A and B is denoted $A \simeq^* B$. The identity map is a pointed equivalence and pointed equivalences are closed under composition.

(viii) Given $A : \mathcal{U}^*$, we define its *loop space* $\Omega A : \mathcal{U}^* :\equiv (a_0 = a_0, \mathsf{refl}_{a_0})$. We define the *iterated loop space* $\Omega^n A$ by iteration as $\Omega^0 A :\equiv A$ and $\Omega^{n+1} A :\equiv \Omega(\Omega^n A)$.

(ix) We define the n-th homotopy group of A as the set-truncation of the iterated loop space, i.e. $\pi_n(A) :\equiv \|\Omega^n A\|_0$. This is a group for $n \geq 1$ that is abelian for $n \geq 2$.

(x) Given a pointed map $f : A \to^* B$, we define the *pointed fiber of f* $\mathsf{fib}_f : \mathcal{U}^*$ as $\mathsf{fib}_f(b_0) \equiv (x : A) \times f(a) = b_0$ with basepoint (a_0, f_0). There is a pointed map $p_1 : \mathsf{fib}_f \to^* A$ defined as $(\lambda x.\, p_1(x), \mathsf{refl}_{a_0})$.

Here are some basic properties of pointed types. We omit the proofs.

Lemma 2.2.7.

(i) Suppose given a pointed map $f : A \to^* B$. The type of proofs that f is an equivalence is equivalent to the type that f is a pointed equivalence. In particular,

being a pointed equivalence is a property. Also, we can define a pointed equivalence $X \simeq^* Y$ by giving a map $e : X \to Y$ that is both an equivalence and pointed.

(ii) *Suppose given $A, B : \mathcal{U}^*$. Univalence implies* univalence for pointed types: *the canonical map $(A = B) \to (A \simeq^* B)$ is an equivalence.*

(iii) *Suppose given pointed maps $f, g : (a : A) \to^* B(a)$. Function extensionality implies* function extensionality for pointed maps: *the canonical map $(f = g) \to (f \sim^* g)$ is an equivalence.*

(iv) *We have the usual categorical laws:*

$$f \circ \mathsf{id} \sim^* f \qquad \mathsf{id} \circ f \sim^* f \qquad (h \circ g) \circ f \sim^* h \circ (g \circ f)$$
$$f \circ \mathbf{0} \sim^* \mathbf{0} \qquad \mathbf{0} \circ f \sim^* \mathbf{0}$$

The two homotopies showing $\mathbf{0} \circ \mathsf{id} \sim^* \mathbf{0}$ are equal. This is also true for the two homotopies of $\mathsf{id} \circ \mathbf{0} \sim^* \mathbf{0}$ and of $\mathsf{id} \circ \mathsf{id} \sim^* \mathsf{id}$ and of $\mathbf{0} \circ \mathbf{0} \sim^* \mathbf{0}$.

(v) *We can form iterated pointed maps $(A \to^* B \to^* C) :\equiv (A \to^* (B \to^* C))$. To show that such a map preserves the basepoint, we need to give an equality between pointed maps, or equivalently, we can give a pointed homotopy between pointed maps. For example, the above homotopies involving $\mathbf{0}$ imply that we have precomposition and postcomposition maps. For $f : A \to^* B$ we have a pointed map $(-) \circ f : (B \to^* C) \to^* A \to^* C$ and for $g : B \to^* C$ we have a pointed map $g \circ (-) : (A \to^* B) \to^* A \to^* C$. We will also write $f \to C$ resp. $A \to g$ for these maps. Precomposition and postcomposition commute, which means that the following square commutes.*

$$
\begin{array}{ccc}
(A \to^* B) & \xrightarrow{\;g \circ (-)\;} & (A \to^* B) \\
{\scriptstyle (-) \circ f}\downarrow & & \downarrow{\scriptstyle (-) \circ f} \\
(A' \to^* B) & \xrightarrow{\;g \circ (-)\;} & (A' \to^* B')
\end{array}
$$

Moreover, if f or g are constant, then these maps are pointed homotopic to constant maps, which gives a pointed map

$$(-) \circ (-) : (B \to^* C) \to^* (A \to^* B) \to^* A \to^* C.$$

(vi) *There are also dependent versions of these composition maps. In particular, if $g : (a : A) \to B(a) \to^* C(a)$, then we have a map*

$$g \circ (-) : ((a : A) \to^* B(a)) \to^* (a : A) \to^* C(a).$$

We have an equivalence

$$\mathsf{fib}_{g \circ (-)} \simeq^* ((a : A) \to^* \mathsf{fib}_{ga}).$$

(vii) Ω *and* Ω^n *are pointed functors. For* Ω *this means that given a pointed map* $f : A \to^* B$*, we can define* $\Omega f : \Omega A \to^* \Omega B$*, with pointed homotopies* $\Omega(g \circ f) \sim^*$ $\Omega g \circ \Omega f$ *and* $\Omega\,\mathsf{id} \sim^* \mathsf{id}$ *and* $\Omega\mathbf{1} \simeq^* \mathbf{1}$*. This also implies that* $\Omega\mathbf{0} \sim^* \mathbf{0}$ *and that if* $e : A \simeq^* B$ *then* $\Omega e : \Omega A \simeq^* \Omega B$*.*

(viii) *There is a pointed version of function extensionality for pointed types. If* B *is a family of pointed types, we have a pointed equivalence*

$$e_B : \Omega((a : A) \to^* B(a)) \simeq^* ((a : A) \to^* \Omega B(a)).$$

This equivalence is natural in B*. This means that given a fiberwise pointed map* $f : (a : A) \to B(a) \to^* C(a)$*, the following square commutes.*

$$
\begin{array}{ccc}
\Omega((a : A) \to^* B(a)) & \xrightarrow{\ e_B\ } & ((a : A) \to^* \Omega B(a)) \\
\Big\downarrow{\scriptstyle \Omega(f \circ (-))} & & \Big\downarrow{\scriptstyle \Omega f \circ (-)} \\
\Omega((a : A) \to^* C(a)) & \xrightarrow{\ e_C\ } & ((a : A) \to^* \Omega C(a))
\end{array}
$$

(ix) *The fiber of a pointed map is functorial. This means that given a commuting square, we get a pointed map from the fiber of the top map to the fiber of the bottom map.*

$$
\begin{array}{ccccc}
\mathsf{fib}_f & \xrightarrow{\ p_1\ } & A & \xrightarrow{\ f\ } & B \\
\vdots & & \Big\downarrow{\scriptstyle g} & & \Big\downarrow{\scriptstyle h} \\
\mathsf{fib}_{f'} & \xrightarrow{\ p_1\ } & A' & \xrightarrow{\ f'\ } & B'
\end{array}
$$

Moreover, if the left and the right sides of the squares are equivalences, then the functorial action is an equivalence. Lastly, p_1 *is natural, which means that the left square commutes.*

(x) *Given a pointed map* $f : A \to^* B$*, we have a equivalence* $\Omega\,\mathsf{fib}_f \simeq^* \mathsf{fib}_{\Omega f}$ *that is natural in* f*. This means that if we have a commuting square with top* f *and bottom* f'*, then the following square commutes (the left and the right side come from the functorial action of* fib*).*

$$
\begin{array}{ccc}
\Omega\,\mathsf{fib}_f & \xrightarrow{\ \sim\ } & \mathsf{fib}_{\Omega f} \\
\Big\downarrow & & \Big\downarrow \\
\Omega\,\mathsf{fib}_{f'} & \xrightarrow{\ \sim\ } & \mathsf{fib}_{\Omega f'}
\end{array}
$$

(xi) We have a pointed equivalence $(\mathbb{S}^0 \to^* X) \simeq^* X$ natural in X.

(xii) A pointed type A is n-connected iff $\pi_k(A)$ is trivial (contractible) for all $k \leq n$. If a type A is n-truncated, then $\pi_k(A)$ is trivial for $k > n$ (however, the converse is not true in general).

2.2.6 Higher Inductive Types

Higher inductive types are a generalization of inductive types where we specify not only the generating points in the type by constructors, but also the generating paths and higher paths. The idea is that the type together with its (higher) path spaces are freely generated by these constructors.

A simple example is the *interval*. The interval I is generated by two points $0, 1 : I$ and a path $\mathsf{seg} : 0 = 1$. Using a syntax similar to that of inductive types, we could write

$\mathsf{HIT}\ I : \mathcal{U} :=$
- $0, 1 : I$;
- $\mathsf{seg} : 0 =_I 1$.

Note that this is not an inductive type, since the last constructor does not specify an element in I, but an element in the path space of I. We get an induction principle for higher inductive types, similar to the induction principle for inductive types. We first give a special case, the nondependent induction principle, also called the *recursion principle*. For the interval this states the following. Given a type X, if we have points $x_0\ x_1 : X$ and a path $p : x_0 =_X x_1$, then we get a map $\mathsf{rec}_I(x_0, x_1, p) : I \to X$. On the points this has the expected computation rules:

$$\mathsf{rec}_I(x_0, x_1, p, 0) \equiv x_0 \qquad \text{and} \qquad \mathsf{rec}_I(x_0, x_1, p, 1) \equiv x_1.$$

We want a similar computation rule on paths. We can apply the induction principle to seg using ap. The resulting computation rule is

$$\mathsf{ap}_{\mathsf{rec}_I(x_0, x_1, p)}(\mathsf{seg}) = p.$$

Note that for this case we postulate a member of the identity type instead of making this a definitional equality. There are various reasons for this. Firstly, in this type theory, there is no justification for this equality to be definitional. There are various ways to define ap, and there is no good reason for the computation rules to favor this definition. Secondly, in the early proof assistants for HoTT there was no support for definitional computation rules on path constructors, but there was a trick to get it for the point constructors [Lic11]. In fact, calling this rule a "computation rule" is not quite accurate, since there is no computation going on. We will still keep using this terminology, so that we have the same terminology as for inductive types. In the cubical type theories mentioned in the introduction we can make these terms reduce judgmentally, making them convenient for working with higher inductive types.

The induction principle for the interval is the following. Suppose given a family $P : I \to \mathcal{U}$ with elements $x_0 : P(0)$ and $x_1 : P(1)$. We need to relate x_0 and x_1 in some way, but we cannot ask that they are equal, since they live in different types. Instead, we require a pathover $p : x_0 =^P_{\mathsf{seg}} x_1$. In this case we get a dependent map $\mathsf{ind}_I(x_0, x_1, p) : (i : I) \to P(i)$ with computation rules on points

$$\mathsf{ind}_I(x_0, x_1, p, 0) \equiv x_0 \qquad \text{and} \qquad \mathsf{ind}_I(x_0, x_1, p, 1) \equiv x_1.$$

For the computation rule on paths, we need to use apd to apply the induction principle to seg, and we get

$$\mathsf{apd}_{\mathsf{ind}_I(x_0, x_1, p)}(\mathsf{seg}) = p.$$

A more interesting example of a higher inductive type is the *(graph) quotient* which we will call a *quotient* in this dissertation. Given $A : \mathcal{U}$ and $R : A \to A \to \mathcal{U}$, the quotient is the following higher inductive type.

$\mathtt{HIT}\ \mathsf{quotient}_A(R) :=$
- $i : A \to \mathsf{quotient}_A(R)$;
- $\mathsf{glue} : (a\ a' : A) \to R(a, a') \to i(a) = i(a')$.

We will sometimes use the notation $[-]_0$ for i and $[-]_1$ for glue.

A very similar higher inductive type is the *homotopy pushout*, or *pushout* for short. Given two maps $f : A \to B$ and $g : A \to C$, their pushout is the following HIT.

$\mathtt{HIT}\ \mathsf{pushout}(f, g) :=$
- $\mathsf{inl} : B \to \mathsf{pushout}(f, g)$
- $\mathsf{inr} : C \to \mathsf{pushout}(f, g)$
- $\mathsf{glue} : (a : A) \to \mathsf{inl}(f(a)) = \mathsf{inr}(g(a))$

We denote $\mathsf{pushout}(f, g)$ by $B +_A C$ if f and g are clear from the context. In this section, we will define other higher inductive types in terms of the pushout. However, we could also start with the quotient, by the following lemma.

Lemma 2.2.8. *The pushout and quotient are interdefinable in MLTT.*

Proof. We will only give the definitions of the pushout and the quotient in terms of the other. Showing that these definitions are correct is easy, and we omit it here.

If we have quotients, we can define the pushout of $f : A \to B$ and $g : B \to C$ as the quotient of $B + C$ under the relation $R : B + C \to B + C \to \mathcal{U}$, which is inductively generated by $\mathsf{mk} : (a : A) \to R(f(a), g(a))$.

On the other hand, if we have pushouts, we can define the quotient of A under R as follows. Let $T :\equiv (a\ a' : A) \times R(a, a')$ be the total space of R. Then the quotient of A under R is the pushout of $f :\equiv \langle \pi_1, \pi_2 \rangle : T + T \to A$ and $g :\equiv \langle \mathsf{id}, \mathsf{id} \rangle : T + T \to T$. □

Many higher inductive types can be defined in terms of the homotopy pushout (or equivalently, the quotient):

- The *cofiber* of a map $f : A \to B$ is defined as $C_f :\equiv B +_A \mathbf{1}$. The maps are f and $!$.

- The *suspension* ΣA of type A is defined as $\Sigma A :\equiv \mathbf{1} +_A \mathbf{1}$, i.e. as the cofiber of the map $A \to \mathbf{1}$. The points are called N and S and glue is called merid.

- The *wedge sum* of a family of pointed types $A : I \to \mathcal{U}^*$ is defined as the cofiber of the map $I \to (i : I) \times A(i)$, which sends i to the pair $(i, \mathsf{pt}_{A(i)})$. The binary wedge $A \vee B$ of two pointed types $A\ B : \mathcal{U}^*$ can equivalently be described as the pushout of $A +_\mathbf{1} B$ where the maps come from the basepoints of A and B.

- The *smash product* $A \wedge B$ of A and B can be defined as the cofiber of the map $A \vee B \to A \times B$, which sends $\mathsf{inl}(a)$ to (a, b_0) and $\mathsf{inr}(b)$ to (a_0, b) and $\mathsf{glue}(\star)$ to $\mathsf{refl}_{(a_0,b_0)}$. We will discuss the smash product in Section 4.3.

- The *n-sphere* $\mathbb{S}^n$ is defined inductively for $n \geq 0$: $\mathbb{S}^0 :\equiv \mathbf{2}$ and $\mathbb{S}^{n+1} :\equiv \Sigma \mathbb{S}^n$. The n-sphere is pointed with point N for $n \geq 1$ and with $0_\mathbf{2}$ for $n = 0$. We could also start counting at $n = -1$, defining $\mathbb{S}^{-1} = \mathbf{0}$, but we often only want to consider the pointed spheres.

Another higher inductive type that we will study is the *sequential colimit* or *colimit* for short. This is the following HIT for $A : \mathbb{N} \to \mathcal{U}$ and $f : (n : \mathbb{N}) \to A(n) \to A(n+1)$:

$\mathsf{HIT}\ \mathsf{colim}(A, f) :=$
- $\iota : (n : \mathbb{N}) \to A(n) \to \mathsf{colim}(A, f)$;
- $\kappa : (n : \mathbb{N}) \to (a : A(n)) \to i_{n+1}(f_n(a)) = i_n(a)$.

We can define $\mathsf{colim}(A, f)$ using quotients, namely as $\mathsf{quotient}(B, R)$ where $B = (n : \mathbb{N}) \times A(n)$ is the total space of A and $R : B \to B \to \mathcal{U}$ is inductively generated by $\mathsf{mk} : (n : \mathbb{N}) \to (a : A(n)) \to R(f_n(a), a)$. We will discuss the colimit more in Section 3.3

We will use the following properties of these higher inductive types. For the proof we refer to [Uni13, Chapter 8]

Lemma 2.2.9.

- *If A is n-connected, then ΣA is $(n+1)$-connected.*

- *The suspension is left-adjoint to the loop space: $\Sigma \dashv \Omega$. That means that for any two pointed types A and B there is a pointed equivalence*

$$(\Sigma A \to^* B) \simeq^* (A \to^* \Omega B)$$

that is natural in A and B.

- *We have the following equivalence: $\Omega \mathbb{S}^1 \simeq \mathbb{Z}$. Therefore $\mathbb{S}^1$ is 1-truncated and $\pi_1(\mathbb{S}^1) \simeq \mathbb{Z}$.*

In particular, by the above lemma we know that $\mathbb{S}^n$ is $(n-1)$-connected, and hence that $\pi_k(\mathbb{S}^n)$ is trivial for $k < n$.

Another higher inductive type is the *torus*, which is the following higher inductive type

```
HIT T² :=
• ⋆ : T²;
• ℓ₁ ℓ₂ : ⋆ = ⋆;
• ℓ₁ · ℓ₂ = ℓ₂ · ℓ₁.
```

The last constructor of the torus is a 2-path constructor. In general, HITs can have as constructor any higher path. We say that a HIT is an *n-HIT* if its highest path constructor has dimension n. So the torus is a 2-HIT and all the other HITs we have seen are 1-HITs.

Higher inductive types can also have *recursive* constructors. If a higher inductive type has at least one recursive constructor, we will call it a recursive HIT. For example, we can encode the propositional truncation as a HIT with a recursive path constructor:

```
HIT ‖A‖ :=
• |−| : A → ‖A‖;
• (x₁ x₂ : ‖A‖) → x₁ = x₂.
```

Higher truncations can also be encoded using HITs [Uni13, Section 7.3].

2.3   Lean

Lean [dMKA⁺15] is an interactive theorem prover that is mainly developed at Microsoft Research and Carnegie Mellon University.[11] The project was started in 2013 by Leonardo de Moura to bridge the gap between interactive theorem proving and automated theorem proving. Lean is an open-source program released under the Apache License 2.0.

In its short history, Lean has undergone several major changes. The second version (Lean 2) supports two kernel modes. The standard mode is for proof irrelevant reasoning, in which `Prop`, the bottom universe, contains types whose objects are considered to be judgmentally equal. This is incompatible with homotopy type theory, so there is a second HoTT mode without `Prop`. In 2016, the third major version of Lean (Lean 3) was released [EUR⁺17]. In this version, many components of Lean have been rewritten. Of note, the unification procedure has been restricted, since the full higher-order unification that is available in Lean 2 can lead to timeouts and error messages that are unrelated to the actual mistakes. Due to certain design decisions, such as proof erasure in the virtual machine and a function definition package that requires axiom K [GMM06], the homotopy type theory mode is currently not natively supported in Lean 3. However, a trick found by Gabriel Ebner allows us to build a homotopy type theory library in Lean 3. In this library, we do not use singleton elimination, which is the feature of `Prop` that is inconsistent with univalence. Singleton elimination is the property that some `Prop`-valued inductive types can eliminate to all universe levels. Gabriel Ebner also wrote a piece of code that no definition in this library uses singleton elimination in its definition. Porting the HoTT library from Lean 2 to Lean 3 is a lot of work, because

---

[11]The contents of this section are based on [vDvRB17], which was written with Jakob von Raumer and Ulrik Buchholtz.

of the changes in the elaborator and in the syntax. All major results in this dissertation are only formalized in Lean 2 and not yet in Lean 3. The HoTT 3 library can be found at https://github.com/gebner/hott3.

The HoTT kernel of Lean 2 provides the following primitive notions:

- *Type universes* `Type.{u} : Type.{u + 1}` for each universe level $u \in \mathbb{N}$. In Lean, this chain of universes is non-cumulative, and all universes are predicative.

- *Function types* `A → B : Type.{max u v}` for types `A : Type.{u}` and `B : Type.{v}` as well as *dependent function types* `Πa, B a : Type.{max u v}` for each type `A : Type.{u}` and type family `B : A → Type.{v}`. These come with the usual β and η rules.

- *inductive types* and *inductive type families*, as proposed by Peter Dybjer [Dyb94]. Every inductive definition adds its constructors and dependent recursors to the environment. Pattern matching is *not* part of the kernel

- two kinds of *higher inductive types*: n-truncation and (typal) quotients.

Outside the kernel, Lean's elaborator uses backtracking search to infer implicit information. It does the following simultaneously.

- The elaborator fills in *implicit arguments* that can be inferred from the context, such as the type of the term to be constructed and the given explicit arguments. Users mark implicit arguments with curly braces. For example, the type of equality is `eq : Π{A : Type}, A → A → Type`, which allows the user to write `eq a₁ a₂` or `a₁ = a₂` instead of `@eq A a₁ a₂`. The symbol `@` allows the user to fill in implicit arguments explicitly. The elaborator supports both first-order unification and higher-order unification.

- We can mark functions as *coercions*, which are then "silently" applied when needed. For example, we have the type of equivalences `A ≃ B`, which is a structure consisting of a function `A → B` with a proof that the function is an equivalence. The map `(A ≃ B) → (A → B)` is marked as a coercion. This means that we can write `f a` for `f : A ≃ B` and `a : A`, and the coercion is inserted automatically.

- Lean was designed with *type classes* in mind, which can provide canonical inhabitants of certain types. This is especially useful for algebraic structures and for type properties like truncatedness and connectedness. Type class instances can refer to other type classes, so that we can chain them together. This makes it possible for Lean to automatically infer why types are n-truncated if our reasoning requires this, for example when we are eliminating out of a truncated type. For example we show that the type of functors between categories `C` and `D` is equivalent to an iterated sigma type.

```
(Σ (F₀ : C → D) (F₁ : Π {a b}, hom a b → hom (F₀ a) (F₀ b)),
 (Π (a), F₁ (ID a) = ID (F₀ a)) ×
 (Π {a b c} (g : hom b c) (f : hom a b),
   F₁ (g ∘ f) = F₁ g ∘ F₁ f)) ≃ functor C D
```

Note the use of coercions here: `F₀ : C → D` really means a function from the objects of `C` to the objects of `D`. From this equivalence, Lean's type class inference can automatically infer that `functor C D` is a set if the objects of `D` form a set. Type class inference will repeatedly apply the rules when sigma-types and pi-types are sets, and use the facts that hom-sets are sets and that equalities in sets are sets (in total 20 rules are applied for this example).

- Instead of giving constructions by explicit terms, we can also make use of Lean's *tactics*, which give us an alternative way to construct terms step by step. This is especially useful if the proof term is large, or if the elaboration relies heavily on higher-order unification.

- We can define custom syntax, including syntax with binding. In the following example we declare two custom notations.

```
infix · := concat
notation `Σ` binders `, ` r:(scoped P, sigma P) := r
```

The first line allows us to write `p · q` for path concatenation `concat p q`. The second line allows us to write `Σ x, P x` instead of `sigma P`. This notation can also be chained: `Σ (A : Type) (a : A), a = a` means `sigma (λ(A : Type), sigma (λ(a : A), a = a))`.

All main results in this dissertation have been formalized in Lean. Some corollaries or examples have not been formalized, in which case we will explicitly mention this. The formalizations are separated in two Github repositories: the Lean-HoTT library[12] and the "spectral" repository, which was originally a repository to formalize spectral sequences, but now also contain many other results in synthetic homotopy theory.[13]

---

[12]https://github.com/leanprover/lean2/blob/master/hott/hott.md
[13]https://github.com/cmu-phil/Spectral/

Below is a table with the locations of the formal results in the libraries.

Theorem	File	Name
Theorem 3.1.8	`hott/hit/prop_trunc.hlean`	`ptrunc_equiv_trunc`
Theorem 3.2.2	`hott/hit/two_quotient.hlean`	`simple_two_quotient.rec`
Theorem 3.3.26	`Spectral/colimit/seq_colim.hlean`	`sigma_seq_colim_over_equiv`
Corollary 3.3.28	`Spectral/colimit/seq_colim.hlean`	`seq_colim_eq_equiv`
Theorem 4.1.1	`hott/homotopy/LES_of_homotopy_groups.hlean`	`is_exact_LES_of_homotopy_groups`
Corollary 4.1.11	`hott/homotopy/sphere2.hlean`	$\pi 2S2$ and $\pi nS3\_eq\_\pi nS2$
Corollary 4.1.13	`hott/homotopy/sphere2.hlean`	πnSn and $\pi 3S2$
Theorem 4.2.7	`Spectral/homotopy/EM.hlean`	`AbGrp_equivalence_cptruncconntype`$'$
Theorem 4.3.28	`Spectral/homotopy/smash_adjoint.hlean`	`smash_adjoint_pmap`
Theorem 5.2.6	`Spectral/algebra/spectral_sequence.hlean`	`is_built_from_infpage`
Theorem 5.3.7	`Spectral/algebra/spectral_sequence.hlean`	`converges_to_sequence`
Theorem 5.4.10	`Spectral/cohomology/serre.hlean`	`atiyah_hirzebruch_convergence`
Theorem 5.4.12	`Spectral/cohomology/serre.hlean`	`serre_convergence`

Chapter 3

Higher Inductive Types

In this chapter we will study properties of Higher Inductive Types (HITs), which we introduced in Section 2.2.6. There is no uniformly accepted scheme of which HITs are allowed, and the semantics of HITs is a topic of current research. There are semantic interpretations of a large class of Higher Inductive Types [LS17], but there are still open questions. Firstly, a general scheme for higher inductive types is unknown, although [ACD⁺16] is a step in the right direction. Secondly, it is unknown whether universes can be closed under higher inductive types. This is unknown even in the case for homotopy pushouts. In this chapter, we do not study the semantics of higher inductive types. Instead, we will work internally in a type theory that has some specific HITs, and construct other HITs from the ones we started with.

In particular, we are interested in the case where we start with the quotient, or equivalently, the homotopy pushout.

One HIT from Section 2.2.6 that we have not yet defined using quotients is the n-truncation. In Section 3.1 we will define the propositional truncation using quotients. A construction of the n-truncations is given by the join construction [Rij17]. This shows that we can define certain recursive HITs using quotients. We will make a start on defining a bigger class of recursive HITs using quotients in Section 3.3.

Another class of HITs we want to construct is HITs with higher path constructors. We construct nonrecursive 2-HITs in Section 3.2, using a method very similar to the hubs and spokes method [Uni13, Section 6.7].

One might wonder after these examples whether all HITs can be reduced to quotients. This turns out to be false. In [LS17, Section 9] the authors describe a specific recursive 1-HITs that cannot be reduced to quotients. Still, it is worthwhile to see which higher inductive types can be constructed from quotients, for example if one is interested in a model of HoTT with homotopy pushouts, but without the extra structure to model all HITs.

3.1 Propositional Truncation

In this section we will construct the propositional truncation from quotients.[14]

Given a type A, define $\{A\}$ as the quotient of A by the indiscrete relation $R :\equiv \lambda(a\ a' : A), \mathbf{1}$. We will call the type $\{A\}$ the *one-step truncation*, since repeating it will give the propositional truncation. We will denote its point constructor by $f : A \to \{A\}$ and its path constructor by $e : (x\ y : A) \to f(x) = f(y)$. We call a function $g : A \to B$ *weakly constant* if $(x\ y : A) \to g(x) = g(y)$ is inhabited. Note that maps $\{A\} \to B$ correspond exactly to weakly constant maps $A \to B$.

Given a type A, we define a sequence $\{A\}_- : \mathbb{N} \to \mathcal{U}$ by

$$
\begin{aligned}
\{A\}_0 &:\equiv A \\
\{A\}_{n+1} &:\equiv \{\{A\}_n\}
\end{aligned}
\tag{3.1.1}
$$

We have map $f_n :\equiv f : \{A\}_n \to \{A\}_{n+1}$, which is the constructor of the one-step truncation. This gives the sequence

$$
A \xrightarrow{f} \{A\} \xrightarrow{f} \{\{A\}\} \xrightarrow{f} \cdots
\tag{3.1.2}
$$

We define $\{A\}_\infty = \mathsf{colim}(\{A\}_-, f_-)$. We will prove that $\{A\}_\infty$ is the propositional truncation of A, in the sense that the construction $A \mapsto \{A\}_\infty$ has the same formation, introduction, elimination and computation rules for the propositional truncation.

We have already shown the formation rule of the propositional truncation (note that $\{A\}_\infty$ lives in the same universe as A).

We also easily get the point constructor of the propositional truncation, because that is just the map $i_0 : A \to \{A\}_\infty$. The path constructor $(x, y : \{A\}_\infty) \to x = y$, i.e. the statement that $\{A\}_\infty$ is a mere proposition, is harder to define. We will postpone this until after we have defined the elimination and computation rules.

The elimination principle — or induction principle — for the propositional truncation is the following statement. Suppose we are given a family of propositions $P : \{A\}_\infty \to \mathsf{Prop}$ with a section $h : (a : A) \to P(i_0(a))$. We then have to construct a map $k : (x : \{A\}_\infty) \to P(x)$. To construct k, take an $x : \{A\}_\infty$. Since x is in a colimit, we can apply induction on x. Notice that we construct an element in $P(x)$, which is a mere proposition, so we only have to define k on the point constructors. This means that we can assume that $x \equiv i_n(a)$ for some $n : \mathbb{N}$ and $a : \{A\}_n$. Now we apply induction on n.

If $n \equiv 0$, then we can choose $k(i_0(a)) :\equiv h(a) : P(i_0(a))$.

If $n \equiv \ell + 1$ for some $\ell : \mathbb{N}$, we know that $a : \{\{A\}_\ell\}$, so we can induct on a. The path constructor of this induction is again automatic. For the point constructor, we can assume that $a \equiv f(b)$. In this case we need to define $k(i_{\ell+1}(f(b))) : P(i_{\ell+1}(f(b)))$. By induction hypothesis, we have an element $y : P(i_\ell(b))$. Now we can transport x along the equality $(g_\ell(b))^{-1} : i_\ell(b) = i_{\ell+1}(f(b))$. This gives the desired element in $P(i_{\ell+1}(f(b)))$.

We can write the proof in pattern matching notation:

---

[14]The contents of this section have been published in [vD16]. However, Corollary 3.1.10 is new.

- $k(i_0(a)) :\equiv h(a)$

- $k(i_{n+1}(f_n(a))) :\equiv (g_n(b))_*^{-1}(k(i_n(b)))$

The definition $k \ (i_0 \ a) :\equiv h \ a$ is also the judgmental computation rule for the point constructors of the propositional truncation.

For the remainder of this section we will prove that $\{A\}_\infty$ is a mere proposition. We will need the following two lemmas.

Lemma 3.1.3. *Let X be a type with $x : X$. Then the type $(y : X) \to x = y$ is a mere proposition.*

Proof. To prove that $(y : X) \to x = y$ is a mere proposition, we assume that it is inhabited and show that it is contractible. Let $f : (y : X) \to x = y$. From this, we conclude that X is contractible with center x. Now given any $g : (y : X) \to x = y$, we know that f and g are pointwise equal, because their codomain is contractible. By function extensionality we conclude that $f = g$, finishing the proof. □

Lemma 3.1.4. *If $g : X \to Y$ is weakly constant, then for every $x, x' : X$, the function $ap_g : x = x' \to g(x) = g(x')$ is weakly constant. That is, $g(p) = g(q)$ for all $p, q : x = x'$.*

Proof. Let $q : (x, y : X) \to g(x) = g(y)$ be the proof that g is weakly constant, and fix $x : X$. We first prove that for all $y : X$ and $p : x = y$ we have

$$g(p) = q(x, x)^{-1} \cdot q(x, y). \tag{3.1.5}$$

This follows from path induction, because if p is reflexivity, then $g(\mathsf{refl}_x) \equiv \mathsf{refl}_{g(x)} = q(x, x)^{-1} \cdot q(x, x)$. The right hand side of (3.1.5) does not depend on p, hence ap_g is weakly constant. □

To prove that $\{A\}_\infty$ is a mere proposition, we need to show $(x, y : \{A\}_\infty) \to x = y$. Since $(y : \{A\}_\infty) \to x = y$ is a mere proposition, we can use the induction principle for the propositional truncation on x, which we have just proven for $\{A\}_\infty$. This means we only have to show that for all $a : A$ we have $(y : \{A\}_\infty) \to i_0(a) = y$. We do not know that $i_0(a) = y$ is a mere proposition,[15] so we will just use the regular induction principle for colimits on y. We then have to construct two inhabitants of the following two types:

(i) For the point constructor we need $p(a, b) : i_0(a) = i_n(b)$ for all $a : A$ and $b : \{A\}_n$.

(ii) We have to show that p respects path constructors:

$$p(a, f(b)) \cdot g(b) = p(a, b). \tag{3.1.6}$$

We have a map $f^n : A \to \{A\}_n$ defined by induction on n, which repeatedly applies f. We also have a path $g^n(a) : i_n(f^n(a)) = i_0(a)$, which is a concatenation of instances of g.

---

[15] Of course, we do know that it is a mere proposition after we have finished the proof that $\{A\}_\infty$ is a mere proposition.
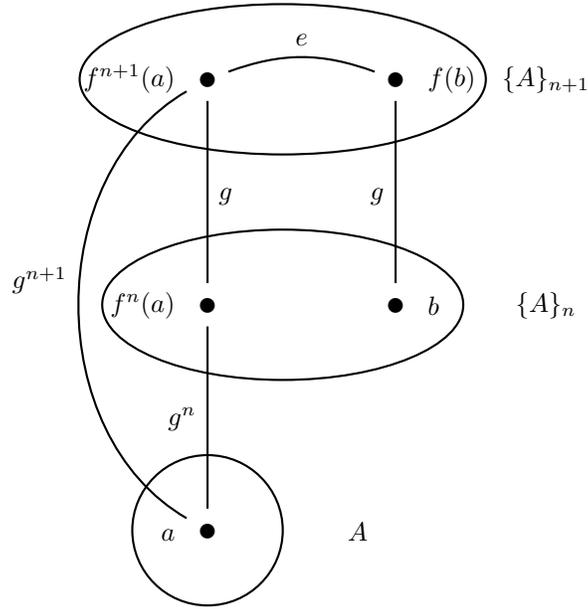
Figure 3.1: The definition of p. The applications of i and the arguments of the paths are implicit.

We can now define $p(a, b)$ as displayed in Figure 3.1, which is the concatenation

$$
\begin{aligned}
i_0(a) &= i_{n+1}(f^{n+1}(a)) && \text{(using } g^{n+1}) \\
&\equiv i_{n+1}(f(f^n(a))) \\
&= i_{n+1}(f(b)) && \text{(using } e) \\
&= i_n(b) && \text{(using } g)
\end{aligned}
$$

Note that by definition $g^{n+1}(a) \equiv g(f^n(a)) \cdot g^n(a)$, so the triangle on the left of Figure 3.1 is a definitional equality.

Now we have to show that this definition of p respects the path constructor of the colimit, which means that we need to show (3.1.6). This is displayed in Figure 3.2. We only need to fill the square in Figure 3.2. To do this, we first need to generalize the statement, because we want to apply path induction. Note that if we give the applications of i explicitly, the bottom and the top of this square are

$$
i\big(e(f^{n+1}(a), f(b))\big)
$$

and

$$
i\big(e(f^{n+2}(a), f(f(b)))\big),
$$

respectively. This means we can apply the following lemma to prove this equality.

Lemma 3.1.7. *Suppose we are given $x\ y : \{A\}_n$, $p : x = y$ and $p' : f(x) = f(y)$. Then we can fill the outer square in Figure 3.3, i.e.*

$$
g(x) \cdot i(p) = i(p') \cdot g(y).
$$

39
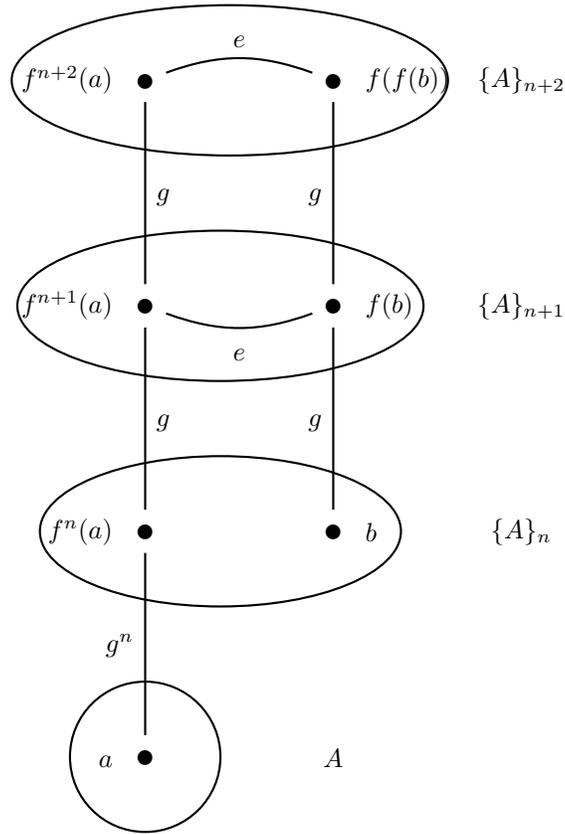
Figure 3.2: The coherence condition for p. The applications of i and the arguments of the paths are implicit.
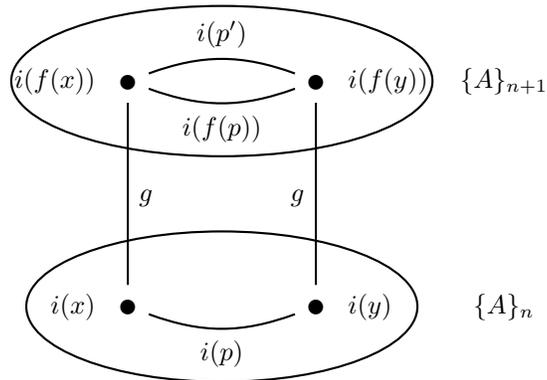
Figure 3.3: The situation in Lemma 3.1.7.

Proof. We can fill the inner square of the diagram by induction on p, because if p is reflexivity, then the inner square reduces to

$$g(x) \cdot \mathsf{refl}_{i(x)} = \mathsf{refl}_{i(f(x))} \cdot g(x).$$

To show that the two paths in the top are equal, first note that $i_k : \{A\}_k \to \{A\}_\infty$ is weakly constant. To see this, look at Figure 3.1. The path from $f^n(a)$ to b in that figure gives a proof of $i_n(f^n(a)) = i_n(b)$ that does not use the form of $f^n(a)$, so we also have $i_k(u) = i_k(v)$ for $u, v : \{A\}_k$. Since i_{n+1} is weakly constant, by Lemma 3.1.4 the function

$$\mathrm{ap}_{i_{n+1}} : f(x) = f(y) \to i_{n+1}(f(x)) = i_{n+1}(f(y))$$

is also weakly constant. This means that the two paths in the top are equal, proving the Lemma. $\qquad\square$

We have now given the proof of the following theorem:

Theorem 3.1.8. *The map $A \mapsto \{A\}_\infty$ satisfies all the properties of the propositional truncation $\|-\|$, including the universe level and judgmental computation rule on point constructors.*

We will mention two corollaries of this result. An alternate proof of the first one is given in [KECA14].

Corollary 3.1.9. *Given a weakly constant function $h : A \to A$, there is a function $\|A\| \to A$.*

Proof. The weakly constant function h gives a function $\tilde{h} : \{A\} \to A$. The HIT $\{-\}$ is functorial (just like all other HITs), so by its functorial action we get a map $\{\tilde{h}\} : \{\{A\}\} \to \{A\}$, which we can compose with $\tilde{h}$ to get a map $\{\{A\}\} \to A$. By induction on n we get a map $k_n : \{A\}_n \to A$. Formally, we define

$$k_0(a) :\equiv a$$
$$k_{n+1}(x) :\equiv \tilde{h}(\{k_n\}(x))$$

However, this sequence of maps does not form a cocone, because the triangles do not commute. (For example for the first triangle we have to show $h(a) = a$ for all a.) But we can easily modify the definition by postcomposing with h. Define $h_n :\equiv h \circ k_n : \{A\}_n \to A$. Now we get a cocone; all triangles commute because h is weakly constant. By the universal property of the sequential colimit we get a map $\|A\| \to A$. $\qquad\square$

We can also construct maps out of the propositional truncation into a set by giving a weakly constant function. An alternate proof was given in [Kra15].

Corollary 3.1.10. *Suppose given a weakly constant function $g : A \to B$ where B is a set. Then there is a map $\tilde{g} : \|A\| \to B$ such that $\tilde{g}(|a|) = g(a)$.*

Proof. First note that given any map $h : \{X\} \to B$, we get a map $h' : \{\{X\}\} \to B$ such that $h' \circ f \sim h$. Namely, on point constructors we define $h'(f(x)) :\equiv h(x)$ for $x : \{X\}$. Now given $x', y' : \{X\}$, we want to define h' on $e(x', y')$. We perform induction on both x and y. In the case that both x and y are point constructors, $x' \equiv f(x)$ and $y' \equiv f(y)$ we can define

$$\mathsf{ap}_{h'}(e(x', y')) := \mathsf{ap}_h(e(x, y)) : h'(f(x')) \equiv h(f(x)) = h(f(y)) \equiv h'(f(y')).$$

In the other three cases, we are constructing a 2-path (or 3-path) in B, which is automatically filled because B is a set. This finishes the construction of h', which satisfies $h' \circ f \sim h$ by definition.

Now we can define a cocone $g_n : \{A\}_n \to B$ as follows. g_0 and g_1 are given by g. We now define $g_{n+2} :\equiv g'_{n+1}$. These g's form a cocone because $g'_{n+2} \circ f \sim g_{n+1}$. This gives a map $\tilde{g} : \|A\| \to B$ such that $\tilde{g}(|a|) = g(a)$. □

An alternative construction of the propositional truncation using non-recursive HITs has been given in [Kra16]. All results in this section have been fully formalized.

3.2 Non-recursive 2-HITs

We can also define nonrecursive 2-HITs using quotients.[16] There are various 2-HITs we would like to construct, such as the torus (as formulated in Section 2.2.6), groupoid quotients, and Eilenberg-MacLane spaces $K(G, 1)$. The construction of 2-HITS uses a method similar to the hubs-and-spokes method described in [Uni13, Sect. 6.7].

The idea behind the hubs-and-spokes method is that for any path $p : x =_A x$ we can define a map $f : \mathbb{S}^1 \to A$ with $\mathsf{ap}_f(\mathsf{loop}) = p$ by circle induction. Then we can prove the equivalence

$$(p = 1) \simeq (x_0 : A) \times (z : \mathbb{S}^1) \to f(z) = x_0.$$

This equivalence informally states that filling in a loop is the same as adding a new point x_0, the *hub*, and *spokes* $f(z) = x_0$ for every $z : \mathbb{S}^1$, similar to the spokes in a wheel. This means that in a higher inductive type, we can replace a 2-path constructor $p = 1_x$ by a new point constructor $x_0 : A$ and a family of 1-path constructors $(z : \mathbb{S}^1) \to f(z) = x_0$. 2-path constructors of the form $p = q$ can be replaced by the equivalent path constructor $p \cdot q^{-1} = 1$.

This construction reduces certain 2-HITs to 1-HITs. However, this reduction is not a quotient, since this family of path constructors refers to other path constructors (in the definition of f), which is not allowed in quotients. If we use quotients, we need to take the quotient twice. We first define a quotient with only the 1-paths (and the hubs), and then use another quotient to add the spokes. In this section we will describe this construction of 2-HITs from quotients.

To be more formal, let us first prove a slightly more general version of the above equivalence.

---

[16] A summary of this section also appeared in [vDvRB17].

42

Lemma 3.2.1. *Given a path $p : a =_A a$ and $f : A \to B$, we have an equivalence*

$$e : ((b_0 : B) \times (z : \mathbb{S}^1) \to f(\mathbb{S}^1.\text{rec}(a, p, z)) = b_0) \simeq (\text{ap}_f(p) = 1),$$

where $\mathbb{S}^1.\text{rec} : (y : P) \to y = y \to \mathbb{S}^1 \to P$ is the nondependent eliminator of the circle $\mathbb{S}^1$.

Proof. This follows from the following chain of equivalences.

$$(b_0 : B) \times (z : \mathbb{S}^1) \to f(\mathbb{S}^1.\text{rec}(a, p, z)) = b_0$$
$$\simeq (b_0 : B) \times (q : f(a) = b_0) \times q =_{\text{loop}}^{f(\mathbb{S}^1.\text{rec}(a,p,-))=b_0} q$$
$$\simeq 1_{f(a)} =_{\text{loop}}^{f(\mathbb{S}^1.\text{rec}(a,p,-))=f(a)} 1_{f(a)}$$
$$\simeq \text{ap}_{f \circ \mathbb{S}^1.\text{rec}(a,p)}(\text{loop}) = \text{ap}_{\mathbf{0}_{f(a)}}(\text{loop})$$
$$\simeq \text{ap}_f(p) = 1. \qquad \square$$

More formally, for $A : \mathcal{U}$ and $R : A \to A \to \mathcal{U}$ we will define words in R to be the following inductive family of types:

inductive $\text{words}_R : A \to A \to \mathcal{U} :=$
- $[-] : \{a\ a' : A\} \to R(a, a') \to \text{words}_R(a, a')$;
- $\langle - \rangle : \{a\ a' : A\} \to a = a' \to \text{words}_R(a, a')$;
- $-^{-1} : \{a\ a' : A\} \to \text{words}_R(a, a') \to \text{words}_R(a', a)$;;
- $- \cdot - : \{a_1\ a_2\ a_3 : A\} \to \text{words}_R(a_1, a_2) \to \text{words}_R(a_2, a_3) \to \text{words}_R(a_1, a_3)$.

A *specification for a (nonrecursive) 2-HIT* consists of a type A and two families $R : A \to A \to \mathcal{U}$ and $S : \{a\ a' : A\} \to \text{words}_R(a, a') \to \text{words}_R(a, a') \to \mathcal{U}$. Using this, we define the 2-HIT $\text{two-quotient}(A, R, S)$ with constructors

HIT $\text{two-quotient}(A, R, S) :=$
- $[-]_0 : A \to \text{two-quotient}(A, R, S)$;
- $[-]_1 : \{a\ a' : A\} \to R(a, a') \to [a]_0 = [a']_0$;
- $[-]_2 : \{a\ a' : A\} \to \{t\ t' : \text{words}_R(a, a')\} \to S(t, t') \to \overline{[t]_1} = \overline{[t']_1}$.

where $\overline{[t]_1}$ is the action of $[-]_1$ on words in R. So if $t : \text{words}_R(a, a')$, then $\overline{[t]_1} : [a]_0 = [a']_0$ is defined by recursion over t. For example, the recursive steps for concatenation of words is

$$\overline{[t_1 \cdot t_2]_1} :\equiv \overline{[t_1]_1} \cdot \overline{[t_2]_1}.$$

Before we define $\text{two-quotient}(A, R, S)$, we first define a special case with only reflexivities on the right hand side of 2-path constructors. This is the following HIT, where Q has type $\{a : A\} \to \text{words}_R(a, a) \to \mathcal{U}$.

HIT $\text{simple-two-quotient}(A, R, Q) :=$
- $[-]_0 : A \to \text{simple-two-quotient}(A, R, Q)$;
- $[-]_1 : \{a\ a' : A\} \to R(a, a') \to [a]_0 = [a']_0$;
- $[-]_2 : \{a : A\} \to \{t : \text{words}_R(a, a)\} \to Q(t) \to \overline{[t]_1} = 1$.

To define this, we first define a new type where we add a *hub* to A for every path specified by Q.

$$B :\equiv A + (a : A) \times (t : \mathsf{words}_R(a, a)) \times Qt.$$

Then we quotient this as specified by R, to obtain the 1-paths.

$$C :\equiv \mathsf{quotient}_B(R_B),$$

where the inductive family of types R_B is defined as follows.

```
inductive R_B : B → B → U :=
```
• $\{a\ a' : A\} \to R(a, a') \to R_B(\mathsf{inl}\,a, \mathsf{inl}\,a')$.

We now define $D :\equiv \mathsf{simple\text{-}two\text{-}quotient}(A, R, Q') :\equiv \mathsf{quotient}_C(R_C)$ where R_C is defined as the following inductive family of types (we write $u_t :\equiv \mathbb{S}^1.\mathsf{rec}([\mathsf{inl}\,a]_0, \overline{[t]_1}) : \mathbb{S}^1 \to C$)

```
inductive R_C : C → C → U :=
```
• $\{a : A\} \to \{t : \mathsf{words}_R(a, a)\} \to (q : Q(t)) \to (x : \mathbb{S}^1) \to R_C(u_t(x), [\mathsf{inr}(a, t, q)]_0)$.

We will now define the expected constructors, eliminators, and computation rules for this two-quotient.

Theorem 3.2.2. *The type $D :\equiv \mathsf{simple\text{-}two\text{-}quotient}(A, R, Q)$ is the HIT as specified above. This means that*

- *There is a 0-path constructor $\langle - \rangle_0 : A \to D$;*

- *There is a 1-path constructor $\langle - \rangle_1 : \{a\ a' : A\} \to R(a, a') \to \langle a \rangle_0 = \langle a' \rangle_0$;*

- *There is a 2-path constructor $\langle - \rangle_2 : \{a : A\} \to \{t : \mathsf{words}_R(a, a)\} \to Q(t) \to \overline{\langle t \rangle_1} = 1$;*

- *There is an induction principle that states the following: given a family $P : D \to \mathcal{U}$ with $s_0 : (a : A) \to P\langle a \rangle_0$ and*

$$s_1 : \{a\ a' : A\} \to (r : R(a, a')) \to s_0(a) =^P_{\langle r \rangle_1} s_0(a')$$

$$s_2 : \{a : A\} \to \{t : \mathsf{words}_R(a, a)\} \to (q : Q(t)) \to \overline{s_1}(t) =_{\langle q \rangle_2} 1,$$

  then P has a section $f : (d : D) \to P(d)$ that computes on the point and 1-path constructors: $f\langle a \rangle_0 \equiv s_0(a)$ and $\mathsf{apd}_f \langle r \rangle_1 = s_1(r)$.

- *There is a recursion principle that states the following: given $P : \mathcal{U}$ with $p_0 : A \to P$ and*

$$p_1 : \{a\ a' : A\} \to R(a, a') \to p_0(a) = p_0(a')$$

$$p_2 : \{a : A\} \to \{t : \mathsf{words}_R(a, a)\} \to Q(t) \to \overline{p_1}(t) = 1,$$

  then there is a map $g : D \to P$ that computes on the point 1-path and 2-path constructors. This means that $g\langle a \rangle_0 \equiv p_0(a)$ for $a : A$, and that there is a path $\iota_1 : \mathsf{ap}_g \langle r \rangle_1 = p_1(r)$ for $r : R(a, a')$ and a filler of the following square for $q : Q(t)$.

44

$$\mathsf{ap}_g\overline{\langle t\rangle_1} \overset{\mathsf{ap}_{\mathsf{ap}_g}\langle q\rangle_2}{=\joinrel=\joinrel=\joinrel=\joinrel=} 1$$

$$\overline{\iota_1}\parallel \qquad\qquad \parallel 1$$

$$\overline{p_1}(t) \overset{p_2(q)}{=\joinrel=\joinrel=\joinrel=} 1$$

Remark 3.2.3.

- We do not prove a computation rule for the induction principle on 2-paths. Although we strongly expect this to be true, it will involve an elaborate computation. This computation rule is not necessary to define simple-two-quotient(A, R, Q) up to equivalence. If we had another type with these exact constructors, eliminators and computation rules, we can prove that it is equivalent to this one. Furthermore, in many examples of two-quotients we will 1-truncate the result, such as for Eilenberg-MacLane spaces and groupoid quotients (see Section 4.2). After the 1-truncation, the computation rules on the 2-paths are automatic, since these 3-paths can be constructed just from the assumption that the type family is truncated.

- We do not define the recursion principle as a special case of the induction principle. We can define it is a much simpler way, so that we can compute its action on 2-paths more easily.

- We use overlines to denote elimination out of the inductive type words_R. The exact type and definition of the overline depends on the type of the object we overline. For example

$$\overline{\langle-\rangle_1} : \mathsf{words}_R(a, a') \to \langle a\rangle_0 = \langle a'\rangle_0$$

is defined recursively by path concatenation and path inversion. In contrast

$$\overline{s_1} : (t : \mathsf{words}_R(a, a')) \to s_0(a) =^P_{\langle t\rangle_1} s_0(a')$$

is defined recursively by pathover concatenation and pathover inversion and

$$\overline{\iota_1} : (t : \mathsf{words}_R(a, a')) \to \mathsf{ap}_g\overline{\langle t\rangle_1} = \overline{p_1}(t)$$

is defined recursively by horizontal concatenation and horizontal inversion and by using the rules $\mathsf{ap}_g(p \cdot q) = \mathsf{ap}_g(p) \cdot \mathsf{ap}_g(q)$ and $\mathsf{ap}_g(p^{-1}) = (\mathsf{ap}_g(p))^{-1}$.

Proof. **Constructors.**
We define for $a : A$ the point constructor

$$\langle a\rangle_0 :\equiv [[\mathsf{inl}\, a]_0]_0 : D$$

and for $r : R(a, a')$ the 1-path constructor

$$\langle r \rangle_1 :\equiv \mathsf{ap}_{[-]_0}[r]_1 : \langle a \rangle_0 = \langle a' \rangle_0$$

from the path constructors of C.

The 2-path constructor $\langle q \rangle_2 : \overline{\langle t \rangle_1} = 1$ for $q : Q(t)$ is defined as the concatenation $\overline{\langle t \rangle_1} = \mathsf{ap}_{[-]_0}\overline{[t]_1} = 1$. Here the first equality is by a general lemma about words_R that states that $\overline{\mathsf{ap}_f(h(r))} = \mathsf{ap}_f(\overline{h(r)})$. The second path uses Lemma 3.2.1 and is defined as $e([[\mathsf{inr}(a, t, q)]_0]_0, [q, -])$, where

$$[q, -] : [\mathbb{S}^1.\mathsf{rec}([\mathsf{inl}\, a]_0, \overline{[t]_1}, x)]_0 \equiv [u_t(x)]_0 = [[\mathsf{inr}(a, t, q)]_0]_0$$

is the path constructor $[q, x]_1$ of D.

Induction Principle.
For the induction principle, suppose given P, s_0, s_1 and s_2 as in the theorem statement. We first define $f_0 : (c : C) \to P[c]_0$ by induction on $c : C$. We define

$$f_0[\mathsf{inl}\, a] :\equiv s_0(a)$$

and (denoting $b :\equiv \mathsf{base} : \mathbb{S}^1$)

$$f_0[\mathsf{inr}(a, t, q)] :\equiv \mathsf{transport}^P([q, b]_1, s_0(a)).$$

For the path constructor, we need to construct for $r : R(a, a')$ the pathover

$$\mathsf{apd}_{f_0}[r]_1 : s_0(a) =_{[r]_1}^{P[-]_0} s_0(a').$$

Here we can use s_1, and then apply the equivalence

$$y =_{\mathsf{ap}_f(p)}^{P} y' \simeq y =_p^{P \circ f} y'.$$

Note that this equivalence holds by reflexivity in a cubical type theory. In the remainder of this proof we will denote any occurrence of this and similar by a tilde for readability. So we define

$$\mathsf{apd}_{f_0}[r]_1 :\equiv \widetilde{s_1}(r).$$

This defines f_0, which is f applied to the point constructor of D, that is, $f[c]_0 :\equiv f_0(c)$. Now we need to define for $x : \mathbb{S}^1$ the pathover

$$\mathsf{ap}_f[q, x]_1 : f_0(u_t(x)) =_{[q,x]_1}^{P} ([q, b]_1)_*(s_0 a).$$

We will fill this pathover by induction to x. For $x \equiv b$ we can constructor the resulting pathover by

$$1_{[q,b]_1} : \mathsf{ap}_f[q, b]_1 : s_0 a =_{[q,b]_1}^{P} ([q, b]_1)_*(s_0 a),$$

46

where in general $1_p : y =_p^P p_*(y)$ can be easily defined by induction on p. When x varies along loop, we need to construct a pathover between two pathovers and this corresponds to the following squareover. The bottom square is a square in D, namely the naturality square of

$$[(q,-)]_1 : (x : \mathbb{S}^1) \to [u_t(x)]_0 = [[\mathsf{inr}(a,t,q)]_0]_0$$

applied to the path loop, and the top square is the squareover we need to fill.

$$
\begin{array}{ccc}
s_0(a) & \overset{1_{[q,b]_1}}{=\!=\!=} & [q,b]_{1*}(s_0(a)) \\
\Big\| \widetilde{\mathsf{apd}}_{f_0 \circ u_t}(\mathsf{loop}) & & \Big\| \widetilde{\mathsf{apd}}_{0_{[q,b]_{1*}(s_0(a))}}(\mathsf{loop}) \\
s_0(a) & \overset{1_{[q,b]_1}}{=\!=\!=} & [q,b]_{1*}(s_0(a))
\end{array}
$$

$$\downarrow$$

$$
\begin{array}{ccc}
\langle a \rangle_0 & \overset{[q,b]_1}{=\!=\!=} & [[\mathsf{inr}(a,t,q)]_0]_0 \\
\Big\| \mathsf{ap}_{[u_t(-)]_0}(\mathsf{loop}) & & \Big\| \mathsf{ap}_{0_{[[\mathsf{inr}(q)]_0]_0}}(\mathsf{loop}) \\
\langle a \rangle_0 & \overset{[q,b]_1}{=\!=\!=} & [[\mathsf{inr}(a,t,q)]_0]_0
\end{array}
$$

We will first focus on the left side of the squareover. We compute

$$
\begin{aligned}
\widetilde{\mathsf{apd}}_{f_0 \circ u_t}(\mathsf{loop}) &= \widetilde{\mathsf{apd}}_{f_0}(\mathsf{ap}_{u_t}(\mathsf{loop})) \\
&= \widetilde{\mathsf{apd}}_{f_0}(\overline{[t]}_1) \\
&= \widetilde{\overline{s_1(t)}} \\
&= \widetilde{\overline{s_1(t)}} \\
&= (\widetilde{\langle q \rangle_2^{-1}})_* 1 & \text{(using } s_2\text{)} \\
&\equiv: \widetilde{1}.
\end{aligned}
$$

Here with $\widetilde{1}$ we mean the pathover $1 : s_0(a) =_{1_{\langle a \rangle_0}}^P s_0(a)$ but transported along the path

$$1_{\langle a \rangle_0} \overset{\langle q \rangle_2}{=\!=} \overline{\langle t \rangle_1} = \mathsf{ap}_{[-]_0}\overline{[t]}_1 = \mathsf{ap}_{[-]_0}(\mathsf{ap}_{u_t}(\mathsf{loop})) = \mathsf{ap}_{[u_t(-)]_0}(\mathsf{loop}).$$

By unfolding the definition of $\langle q \rangle_2$ this can be simplified to the following concatenation:

$$1_{\langle a \rangle_0} \overset{e}{=} \mathsf{ap}_{[-]_0}\overline{[t]}_1 = \mathsf{ap}_{[-]_0}(\mathsf{ap}_{u_t}(\mathsf{loop})) = \mathsf{ap}_{[u_t(-)]_0}(\mathsf{loop}).$$

The right side of the squareover is easier to manipulate:

$$\widetilde{\mathsf{apd}}_{0_{[q,b]_{1*}(s_0(a))}}(\mathsf{loop}) = \widetilde{1},$$

where in this case we mean the pathover $1 : s_0(a) =^P_{1_{\langle a \rangle_0}} s_0(a)$ transported along the path

$$1_{\langle a \rangle_0} = \mathsf{ap0}_{[[\mathsf{inr}(q)]_0]_0}(\mathsf{loop}).$$

Now in both the left and the right side these transports only act on the path they lie over. This means that we can "push them down" to the base square.

After we do that, we have a vertically degenerate squareover, and we only have to show that the square over which it lies is also vertically degenerate, which is a straightforward calculation.

This finishes the definition of f. The computation rule $f\langle a \rangle_0 \equiv s_0(a)$ follows directly from the computation rule for the quotient. Furthermore, we have

$$\mathsf{apd}_f\langle r \rangle_1 \equiv \mathsf{apd}_f\mathsf{ap}_{[-]_0}[r]_1 = \widetilde{\mathsf{apd}}_{f \circ [-]_0}[r]_1 \equiv \widetilde{\mathsf{apd}}_{f_0}[r]_1 = s_1(r).$$

Recursion Principle.

For the recursion principle, suppose given P, p_0, p_1, p_2 as in the theorem statement. We first define $g_0 : C \to P$ by

$$g_0[\mathsf{inl}\ a]_0 :\equiv p_0(a)$$
$$g_0[\mathsf{inr}(a, t, q)]_0 :\equiv p_0(a)$$
$$\mathsf{ap}_{g_0}[r]_1 :\equiv p_1(r).$$

We define $g : D \to P$ by $g[c]_0 :\equiv g_0(c)$ and then we need to define $\mathsf{ap}_g[q, x]_1 : g_0(u_t(x)) = p_0(a)$, which we do by induction to x. For $x \equiv b$, this can be done by reflexivity, so $\mathsf{ap}_g[q, b]_1 :\equiv 1_{p_0(a)}$. When x varies over loop, we need to fill the following square.

$$
\begin{array}{ccc}
p_0(a) & \overset{1}{=\!=\!=\!=} & p_0(a) \\[2pt]
\mathsf{ap}_{g_0 \circ u_t}(\mathsf{loop}) \Big\| & & \Big\| \mathsf{ap0}_{p_0(a)}(\mathsf{loop}) \\[2pt]
p_0(a) & \overset{1}{=\!=\!=\!=} & p_0(a)
\end{array}
$$

This can be done by the following calculation.

$$\mathsf{ap}_{g_0 \circ u_t}(\mathsf{loop}) = \mathsf{ap}_{g_0}\mathsf{ap}_{u_t}(\mathsf{loop}) = \mathsf{ap}_{g_0}\overline{[t]}_1 = \overline{p_1}(t) \overset{p_2}{=} 1 = \mathsf{ap0}_{p_0(a)}(\mathsf{loop}).$$

This completes the definition of g. The computation rule $g\langle a \rangle_0 \equiv p_0(a)$ follows from the computation rule for quotients on points. We can define the computation rule on paths as the composite

$$\iota_1 : \mathsf{ap}_g\langle r \rangle_1 \equiv \mathsf{ap}_g\mathsf{ap}_{[-]_0}[r]_1 = \mathsf{ap}_{g_0}[r]_1 = p_1(r).$$

The fact that g has the correct computation rule for 2-paths requires some complicated path algebra, which we will omit here. $\qquad\square$

We can now define the general version of the 2-quotient, $\mathsf{two\text{-}quotient}(A, R, S)$, to be equal to $\mathsf{simple\text{-}two\text{-}quotient}(A, R, Q)$ where Q is the inductive family

```
inductive Q : {a : A} → words_R(a, a) → U :=
```
  • $(a\ a' : A) \to (t\ t' : \mathsf{words}_R(a, a')) \to (s : S(t, t')) \to Q(t \cdot t'^{-1})$.

We then show that $\mathsf{two\text{-}quotient}(A, R, S)$ and $\| \mathsf{two\text{-}quotient}(A, R, S)\|_n$ have the right elimination principles and computation rules (it requires some work to show that the eliminator of the truncated 2-quotient has the right computation rules on 2-paths).

This allows us to define all nonrecursive HITs with point, 1-path and 2-path constructors. For example, we define the torus $T^2 := \mathsf{two\text{-}quotient}(\mathbf{1}, R, S)$ where $R(\star, \star) = \mathbf{2}$ (giving two path constructors p and q from the basepoint to itself) and Q is generated by the constructor $s_0 : S(0_\mathbf{2} \cdot 1_\mathbf{2})(1_\mathbf{2} \cdot 0_\mathbf{2})$, which determines a path $p \cdot q = q \cdot p$. We also define the *groupoid quotient*: For a groupoid G we define its quotient as $\| \mathsf{two\text{-}quotient}(G, \mathsf{hom}_G, S)\|_1$ where:

```
inductive S :=
```
  • $(a\ b\ c : G) \to (g : \mathsf{hom}(b, c)) \to (f : \mathsf{hom}(a, b)) \to S(g \circ f)(f \cdot g)$

If G is just a group (considered as a groupoid with a single object), then the groupoid quotient of G is exactly the Eilenberg-MacLane space $K(G, 1)$. For more information, see Section 4.2.

3.3   Colimits

We can ask whether we can use the construction of Section 3.1 can be generalized to construct other higher inductive types.[17] The general idea is that we can construct a recursive higher inductive type as a sequential colimit of repeatedly applying a nonrecursive version of the HIT. This does not work in general: if a constructor is infinitary, there is no reason why the type after ω many steps is the desired type. However, this does work for a general class of higher inductive types, the *ω-compact localizations*. In this section we will show various properties of colimits that are used in the proof of this fact. The full proof will appear in an upcoming preprint.

Definition 3.3.1. Suppose given a type A, families $P, Q : A \to \mathcal{U}$ and $F : \{a : A\} \to P(a) \to Q(a)$.
   A type X is *F-local* if for all $a : A$ the map

$$\psi_X(a) :\equiv \lambda f.\ f \circ F(a) : (Q(a) \to X) \to (P(a) \to X)$$

is an equivalence.

   The *F-localization* $L_F X$ or LX of X turns X into a F-local type in a universal way. This means there is a map $\ell_X : X \to LX$ such that for any F-local type Y there is an equivalence of maps $(LX \to Y) \to (X \to Y)$ given by precomposition with ℓ_X. $L_F X$ can be given as a higher inductive type with the following constructors:

---

[17]The work in this section is joint work with Egbert Rijke and Kristina Sojakova.

```
HIT L F X : Type :=
| incl : X → L X
| rinv : Π{a} (f : P a → L X), Q a → L X
| isri : Π{a} (f : P a → L X) (x : P a), rinv f (F x) = f x
| linv : Π{a} (f : P a → L X), Q a → L X
| isli : Π{a} (f : Q a → L X) (x : Q a), linv (f ∘ F) x = f x.
```

For a sequence $(A_n, f_n)_n$ we denote the colimit by $\mathsf{colim}(A)$ or A_∞. Also, for any type X, we can define a new sequence $(X \to A_n, f_n \circ (-))_n$. Note that there is a canonical map

$$\xi_X : \mathsf{colim}(X \to A_n) \to (X \to A_\infty).$$

It is defined by $\xi_X(i_n(f)) :\equiv i_n \circ f$ and $\xi_X(\kappa(f)) :\equiv \kappa_n \circ f$, where κ is the path constructor of the colimit.

Definition 3.3.2. A type X is said to be ω-*compact* if the map ξ_X is an equivalence for all sequences $(A_n, f_n)_n$.

Examples of ω-compact types are the finite types. Moreover, the ω-compact types are closed under dependent pair types and pushouts. A non-example of an ω-compact type is $\mathbb{N}$. We will omit the details here.

Theorem 3.3.3. *Assume that for all $a : A$ the types $P(a)$ and $Q(a)$ are ω-compact. Then we can construct the F-localization in MLTT+quotients.*

We will not prove this theorem here, but defer it to an upcoming preprint. However, we will develop machinery here that is crucial to prove this theorem. In particular we prove that sigma-types commute with sequential colimits.

Type Sequences

Definition 3.3.4. A *type sequence* (A, f) consists of a diagram of the form

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$

Thus, the type of all sequences of types is

$$\mathrm{Seq} :\equiv (A : \mathbb{N} \to \mathcal{U}) \times (n : \mathbb{N}) \to A_n \to A_{n+1}$$

Recall that the relation $\leq$ on the natural numbers is defined as an inductive family of types $\leq : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$ with

$$r : (n : \mathbb{N}) \to n \leq n$$
$$s : (n, m : \mathbb{N}) \to n \leq m \to n \leq m + 1.$$

It follows that $n \leq m$ is a a mere proposition for each $n, m : \mathbb{N}$.

Definition 3.3.5. Let (A, f) be a type sequence. For any $n, m : \mathbb{N}$, we define

$$f^{n \leq m} : A_n \to A_m.$$

where we leave the proof that $n \leq m$ implicit.

Construction. We define $f^{n \leq m}$ by induction on the proof that $n \leq m$ by taking

$$f^{n \leq n} :\equiv \mathsf{id}_{A_n}$$
$$f^{n \leq m+1} :\equiv f_m \circ f^{n \leq m} \hspace{3cm} \square$$

Definition 3.3.6. Let (A, f) be a type sequence. For any $n, k : \mathbb{N}$, we define $f_n^k : A_n \to A_{n+k}$ to be $f^{n \leq n+k}(p)$, where p is the canonical proof that $n \leq n + k$.

Definition 3.3.7. A *sequence (B, g) of types over (A, f)* consists of a diagram of the form

$$B_0 \xrightarrow{g_0} B_1 \xrightarrow{g_1} B_2 \xrightarrow{g_2} \cdots$$
$$\downarrow \qquad \downarrow \qquad \downarrow$$
$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$

where each g_n has type $(a : A_n) \to B_n(x) \to B_{n+1}(f_n(a))$, implicitly rendering the squares commutative.

We say that a sequence (B, g) over (A, f) is *equifibered* if each g_n is a family of equivalences.

Definition 3.3.8. Let (A, f) and (A', f') be type sequences. A *natural transformation* $(A, f) \to (A', f')$ is a pair (τ, H) consisting of a family of maps

$$\tau : (n : \mathbb{N}) \to A_n \to A_n'$$

and a family H_n of homotopies witnessing that the diagram

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots$$
$$\downarrow^{\tau_0} \qquad \downarrow^{\tau_1} \qquad \downarrow^{\tau_2}$$
$$A_0' \xrightarrow{f_0'} A_1' \xrightarrow{f_1'} A_2' \xrightarrow{f_2'} \cdots$$

commutes.

Definition 3.3.9. A *natural equivalence* is a natural transformation (τ, H) such that each τ_n is an equivalence. The type of natural equivalences from (A, f) to (A', f') is called $\mathsf{NatEq}((A, f), (A', f'))$.

Lemma 3.3.10. *The canonical dependent function* idtonateq

$$((A, f) = (A', f')) \to \mathsf{NatEq}((A, f), (A', f'))$$

that sends $\mathsf{refl}_{(A,f)}$ *to the identity natural transformation, is an equivalence.*

51

Proof. Straightforward application of univalence. □

Every type sequence (B, g) over (A, f) gives rise to a natural transformation, by the following definition.

Definition 3.3.11. Let (B, g) be a sequence over (A, f). Then we define the sequence $\Sigma((A, f), (B, g))$ to consist of the diagram

$$(a : A_0) \times B_0(a) \xrightarrow{(f_0, g_0)} (a : A_1) \times B_1(a) \xrightarrow{(f_1, g_1)} (a : A_2) \times B_2(a) \xrightarrow{(f_2, g_2)} \cdots$$

where we take the usual definition

$$(f_n, g_n) :\equiv \lambda(a, b). (f_n(a), g_n(a, b)).$$

Furthermore, we define a natural transformation

$$(\pi, \theta) : \Sigma((A, f), (B, g)) \to (A, f)$$

by taking

$$\pi_n :\equiv \mathsf{pr}_1 \qquad\qquad : ((a : A_n) \times B_n(a)) \to A_n$$
$$\theta_n(a, b) :\equiv \mathsf{refl}_{f_n(a)} \qquad\qquad : f_n(\mathsf{pr}_1(a, b)) = \mathsf{pr}_1(f_n(a), g_n(b)).$$

We will now look at the shift operation on type sequences, in particular to bring up subtleties that come up in the formalization of mathematics in homotopy type theory. The issue we face is that equality in the natural numbers is not always strict. For instance, when addition is defined by induction on the second argument, then $n + 0$ is judgmentally equal to n, while $0 + n$ is not. This implies that sometimes we might have to *transport* along the equalities in the natural numbers (such as $n = 0 + n$), and this complicates the formalization process.

We define the shift operation.

Definition 3.3.12. For any type sequence (A, f) we define a new type sequence $(S(A), S(f))$ by taking

$$S(A)_n :\equiv A_{n+1}$$
$$S(f)_n :\equiv f_{n+1}.$$

Of course we can iterated the shift operation, defining a type sequence $(S^k(A), S^k(f))$ for every $k : \mathbb{N}$. However, while the type $S^k(A)_n$ is A_{n+k}, the function $S^k(f)_n$ is some function $A_{n+k} \to A_{(n+1)+k}$ that is not judgmentally equal to a function of the form f_m for some $m : \mathbb{N}$. Therefore, we make an alternative definition of the k-shift that is different from S^k, the type sequence obtained from iterating the shift S.

Definition 3.3.13. Given a type sequence (A, f), we define $S_k(A, f) \equiv (S_k(A), S_k(f))$ to be the type sequence given by

$$S_k(A)_n :\equiv A_{k+n}$$
$$S_k(f)_n :\equiv f_{k+n}.$$

Given a dependent sequence (B, g) over (A, f), we also define $S_k(B, g) \equiv (S_k(B), S_k(g))$ by

$$S_k(B)_n :\equiv B_{k+n}$$
$$S_k(g)_n :\equiv g_{k+n}.$$

Note that the sequence $(S_{k+1}(A), S_{k+1}(f))$ is not judgmentally equal to the sequence $S(S_k(A), S_k(f))$, since in general we do not have $(k+1) + n \equiv (k+n) + 1$. Therefore we have the following lemma.

Lemma 3.3.14. *For any $k, n : \mathbb{N}$ and $a : A_k$, one has $q_{k,n}(a) : f_k^{n+1}(a) =_{p(k,n)}^A f_{k+1}^n(f_k(a))$ where $p(k,n) : (k+n) + 1 = (k+1) + n$ is the canonical path in $\mathbb{N}$.*

Proof. By induction on $n : \mathbb{N}$. □

Corollary 3.3.15. *For any type sequence (A, f), the type sequence $(S_{k+1}(A), S_{k+1}(f))$ is naturally equivalent to the type sequence $(S(S_k(A)), S(S_k(f)))$.*

Sequential Colimits

Remark 3.3.16. The induction principle for sequential colimits tells us how to construct a dependent function $f : (a : A_\infty) \to P(a)$ for a type family $P : A_\infty \to \mathcal{U}$.

Given $s : (a : A_\infty) \to P(a)$, we get

$$\lambda n. \lambda a. s(\iota_n(a)) : (n : \mathbb{N})(a : A_n) \to P(\iota_n(a))$$
$$\lambda n. \lambda a. \mathsf{apds}\kappa_n(a) : (n : \mathbb{N})(a : A_n) \to s(\iota_n(a)) =_{\kappa_n(a)}^P s(\iota_{n+1}(f_n(a)))$$

In other words, we have a canonical map

$$\left((a : A_\infty) \to P(a)\right) \to$$
$$\left((h : (n : \mathbb{N})(a : A_n) \to P(\iota_n(a))) \times (n : \mathbb{N})(a : A_n) \to h_n(a) =_{\kappa_n(a)}^P h_{n+1}(f_n(a))\right)$$

Now we can state the induction principle and computation rule concisely: the canonical map described above comes equipped with a section. We assume that that the computation rule is strict on the point constructors.

The universal property of sequential colimits is a straightforward consequence of the induction principle.

53

Theorem 3.3.17. *Let (A, f) be a type sequence, and let X be a type. Then the canonical map*

$$(A_\infty \to X) \to (h : (n : \mathbb{N}) \to A_n \to X) \times (n : \mathbb{N}) \to h_n \sim h_{n+1} \circ f_n$$

is an equivalence.

The following theorem is a descent theorem for sequential colimits.

Theorem 3.3.18. *Consider a sequence (A, f). The type $A_\infty \to \mathcal{U}$ is equivalent to the type of equifibered type sequences over (A, f).*

Proof. By the universal property of A_∞ and by univalence we have

$$(A_\infty \to \mathcal{U}) \simeq (B : (n : \mathbb{N}) \to A_n \to \mathcal{U}) \times (n : \mathbb{N}) \to B_n \sim B_{n+1} \circ f_n$$
$$\simeq (B : (n : \mathbb{N}) \to A_n \to \mathcal{U}) \times (n : \mathbb{N})(x : A_n) \to B_n(x) \simeq B_{n+1}(f_n(x)) \square$$

Lemma 3.3.19. *Suppose given a natural transformation $(\tau, H) : (A, f) \to (A', f')$.*

(i) *We get a function $\mathsf{colim}(\tau, H)$ or $\tau_\infty : A_\infty \to A'_\infty$.*

(ii) *The sequential colimit is 1-functorial. This means the following three things. If $(\sigma, K) : (A', f') \to (A'', f'')$, then $(\tau \circ \sigma)_\infty \sim \tau_\infty \circ \sigma_\infty$. Moreover, $1_\infty \sim \mathsf{id}$, where 1 is the identity natural transformation. Lastly, if $(\tau', H') : (A, f) \to (A', f')$ and $q : (n : \mathbb{N}) \to \tau_n \sim \tau'_n$ and we can fill the following square for all $a : A_n$*

$$
\begin{array}{ccc}
\tau_{n+1}(f_n a) & \xlongequal{q_{n+1}(f_n a)} & \tau'_{n+1}(f_n a) \\
\Big\| H_n(a) & & \Big\| H'_n(a) \\
f'_n(\tau_n(a)) & \xlongequal{\mathsf{ap}_{f'_n}(q_n(a))} & f'_n(\tau'_n(a))
\end{array}
$$

then $\tau_\infty \sim \tau'_\infty$.

(iii) *If τ is a natural equivalence, then τ_∞ is an equivalence.*

Proof.

(i) We define $\tau_\infty(\iota_n(a)) :\equiv \iota_n(\tau_n(a))$ and

$$\mathsf{ap}_{\tau_\infty}(\kappa_n(a)) := \mathsf{ap}_{\iota_{n+1}}(H(a)) \cdot \kappa_n(\tau_n(a)) : \iota_{n+1}(\tau_{n+1}(f_n a)) = \iota_n(\tau_n(a)).$$

(ii) All three parts are by induction on the element of A_∞, and all parts are straight-forward.

54

(iii) We define $(\tau_\infty)^{-1} :\equiv (\tau^{-1})_\infty$ where τ^{-1} is the natural transformation by inverting τ_n for each n. Now we can check that this is really the inverse by using all three parts of the 1-functoriality.

$$\tau_\infty^{-1} \circ \tau_\infty \sim (\tau^{-1} \circ \tau)_\infty \sim 1_\infty \sim \mathsf{id}_{A_\infty}.$$

For the second homotopy we need to show that we can fill a certain square, which is straightforward. The other composite is homotopic to the identity by a similar argument. $\qquad\square$

The following lemma states that ι_0 is an equivalence if all maps in the sequence are an equivalence. We will have a more general result in Corollary 3.3.30(v), but in that proof we will use some special cases of this lemma.

Lemma 3.3.20. *Suppose given a sequence (A, f) where f_n is an equivalence for all n. Then $\iota_0 : A_0 \to A_\infty$ is an equivalence.*

Proof. First note that the map $f^{0 \leq n} : A_0 \to A_n$ is an equivalence, which is an easy induction on the proof that $0 \leq n$, because $f^{0 \leq 0} \equiv \mathsf{id}$ is an equivalence and $f^{0 \leq n+1} \equiv f_n \circ f^{0 \leq n}$ is a composition of two equivalences.

Also note that we have paths $\kappa^{n \leq m}(a) : \iota_m(f^{n \leq m}(a)) = \iota_n(a)$ for $a : A_n$.

Now we define $\iota_0^{-1} : A_\infty \to A_0$ as

$$\iota_0^{-1}(\iota_n(a)) :\equiv (f^{0 \leq n})^{-1}(a)$$

and we define

$$\mathsf{ap}_{\iota_0^{-1}}(\kappa_n(a)) : (f^{0 \leq n})^{-1}(f_n^{-1}(f_n(a))) = (f^{0 \leq n})^{-1}(a)$$

as $\mathsf{ap}_{(f^{0 \leq n})^{-1}}(\ell_n(a))$, where $\ell_n(a) : f_n^{-1}(f_n(a))$ is the canonical path.

Now $\iota_0^{-1} \circ \iota_0 \sim \mathsf{id}$ is true by definition. To show that for $x : A_\infty$ we have $p(x) : \iota_0(\iota_0^{-1}(x)) = x$, we use induction on x. If $x \equiv \iota_n(a)$, we have

$$\begin{aligned}
\iota_0(\iota_0^{-1}(\iota_n(a))) &\equiv \iota_0((f^{0 \leq n})^{-1}(a)) \\
&= \iota_n(f^{0 \leq n}((f^{0 \leq n})^{-1}(a))) \\
&= \iota_n(a).
\end{aligned}$$

If we write $r^{0 \leq n} : f^{0 \leq n} \circ (f^{0 \leq n})^{-1} \sim \mathsf{id}$ for the canonical homotopy, then we explicitly define $p(\iota_n(a))$ as

$$p(\iota_n(a)) :\equiv (\kappa^{0 \leq n}((f^{0 \leq n})^{-1}(a)))^{-1} \cdot \mathsf{ap}_{\iota_n}(r^{0 \leq n}(a)).$$

If x varies over $\kappa_n(a)$, then we need to fill the following square.

$$\begin{array}{ccc}
\iota_0(\iota_0^{-1}(\iota_{n+1}(f_n a))) & \xlongequal{\;p(\iota_{n+1}(f_n a))\;} & \iota_{n+1}(f_n a) \\
\Big\| {\scriptstyle \mathsf{ap}_{\iota_0 \circ \iota_0^{-1}}(\kappa_n(a))} & & \Big\| {\scriptstyle \kappa_n(a)} \\
\iota_0(\iota_0^{-1}(\iota_n(a))) & \xlongequal{\;p(\iota_n(a))\;} & \iota_n(a)
\end{array}$$

55

If we unfold the definitions of ι_0^{-1} and p, we can fill this as the horizontal concatenation of the following two squares (where we have left out some arguments to the paths)

$$
\begin{array}{ccc}
\iota_0\big((f^{0\le n})^{-1}(f_n^{-1}(f_na))\big) & \overset{(\kappa^{0\le n})^{-1}}{=\!=\!=} & \iota_n\big(f^{0\le n}((f^{0\le n})^{-1}(f_n^{-1}(f_na)))\big) \\[4pt]
\Big\| \mathsf{ap}_{\iota_0\circ(f^{0\le n})^{-1}}(\ell) & & \Big\| \mathsf{ap}_{\iota_n\circ f^{0\le n}\circ(f^{0\le n})^{-1}}(\ell) \\[4pt]
\iota_0\big((f^{0\le n})^{-1}(a)\big) & \overset{(\kappa^{0\le n})^{-1}}{=\!=\!=\!=} & \iota_n\big(f^{0\le n}((f^{0\le n})^{-1}(a))\big)
\end{array}
$$

$$
\begin{array}{ccc}
\iota_n\big(f^{0\le n}((f^{0\le n})^{-1}(f_n^{-1}(f_na)))\big) & \overset{\kappa^{-1}\cdot\mathsf{ap}_{\iota_{n+1}}(\mathsf{ap}_f(r^{0\le n})\cdot r)}{=\!=\!=\!=\!=\!=\!=\!=} & \iota_{n+1}(f_na) \\
\end{array}
$$

$\mathsf{ap}_{\iota_n}(r^{0\le n})$

$\mathsf{ap}_{\iota_n\circ f^{0\le n}\circ(f^{0\le n})^{-1}}(\ell)\qquad\iota_0(f_n^{-1}(f_na))\qquad\kappa$

$\mathsf{ap}_{\iota_n}(\ell)$

$\iota_n\big(f^{0\le n}((f^{0\le n})^{-1}(a))\big)\quad\overset{\mathsf{ap}_{\iota_n}(r^{0\le n})}{=\!=\!=\!=\!=}\quad\iota_n(a)$

The first square is a naturality square, as is the bottom-left part of the second square. We can use the triangle equalities of f to rewrite the r in the top part to $\mathsf{ap}_f(\ell)$. After doing that, the top-right square becomes the following naturality square.

$$
\begin{array}{ccc}
\iota_n\big(f_n(f^{0\le n}((f^{0\le n})^{-1}(f_n^{-1}(f_na))))\big) & \overset{\mathsf{ap}_{\iota_{n+1}\circ f}(r^{0\le n}\cdot\ell)}{=\!=\!=\!=} & \iota_{n+1}(f_na) \\[4pt]
\Big\|\kappa & & \Big\|\kappa \\[4pt]
\iota_0(f_n^{-1}(f_na)) & \overset{\mathsf{ap}_{\iota_n}(r^{0\le n}\cdot\ell)}{=\!=\!=\!=} & \iota_n(a)
\end{array}
$$

$\square$

Lemma 3.3.21. *For any type sequence (A,f), the colimits of (A,f) and $S(A,f)$ are equivalent.*

Proof. We construct a map $\varphi : A_\infty \to S(A)_\infty$ by induction on A_∞, by taking

$$
(x : A_n) \mapsto \iota_n^{S(A),S(f)}(f_n(x))
$$
$$
(x : A_n) \mapsto \kappa_n^{S(A),S(f)}(f_n(x)).
$$

Next, we construct a map $\psi : S(A)_\infty \to A_\infty$ by induction on $S(A)_\infty$, by taking

$$
(x : S(A)_n) \mapsto \iota_{n+1}^{A,f}(x)
$$
$$
(x : S(A)_n) \mapsto \kappa_{n+1}^{A,f}(x).
$$

Then we prove that $\psi \circ \varphi \sim \mathsf{id}$ by induction on A_∞, by taking

$$
(x : A_n) \mapsto \kappa_n^{A,f}(x)
$$

56

Now we compute

$$\mathsf{ap}_{\psi \circ \varphi}(\kappa_n^{A,f}(x))) = \mathsf{ap}_\psi(\mathsf{ap}_\varphi(\kappa_n^{A,f}(x)))$$
$$= \mathsf{ap}_\psi(\kappa_n^{S(A),S(f)}(f_n(x)))$$
$$= \kappa_{n+1}^{A,f}(f_n(x))$$

from the computation rules of A_∞ and $S(A)_\infty$.

We construct the homotopy $\varphi \circ \psi \sim \mathsf{id}$ by induction on A_∞, by taking

$$(x : S(A)_n) \mapsto \kappa_{n+1}(S(f)_n(x))$$

Now we compute

$$\mathsf{ap}_{\varphi \circ \psi}(\kappa_n^{S(A),S(f)}(x)) = \mathsf{ap}_\varphi(\mathsf{ap}_\psi(\kappa_n^{S(A),S(f)}(x)))$$
$$= \mathsf{ap}_\varphi(\kappa_{n+1}^{A,f}(x))$$
$$= \kappa_{n+1}^{S(A),S(f)}(f_{n+1}(x)). \qquad \square$$

Lemma 3.3.22. *For any type sequence (A, f), we have an equivalence*

$$\mathsf{kshift\_equiv}_k : \mathsf{colim}(A, f) \simeq \mathsf{colim}(S_k(A, f)).$$

The shift operations and the corresponding equivalences on the sequential colimits can be used to turn an arbitrary sequence (B, g) over (A, f) into an equifibered sequence over (A, f).

Definition 3.3.23. Given a dependent sequence (B, g) over (A, f) and $x : A_0$, we define a type sequence $(B[x], g[x])$ by

$$B[x]_n :\equiv B_n(f^n(x))$$
$$g[x]_n :\equiv g_n(f^n(x), -).$$

Definition 3.3.24. Given any sequence (B, g) over (A, f), we define an equifibered sequence $(\square B, \square g)$ over the sequence (A, f).

Construction. For $x : A_n$ we define

$$(\square B)_n(x) :\equiv S_n(B)[x]_\infty \equiv \mathsf{colim}_m(B_{n+m}(f^m(x))).$$

Now note that

$$(\square B)_{n+1}(f(x)) \equiv \mathsf{colim}_m(B_{(n+1)+m}(f^m(f(x))))$$
$$\simeq \mathsf{colim}_m(B_{n+(m+1)}(f^{m+1}(x)))$$
$$\simeq \mathsf{colim}_m(B_{n+m}(f^m(x)))$$
$$\equiv (\square B)_n(x)$$

The first equivalence $u_{n,m}$ is given by transporting along the dependent path in Lemma 3.3.14 in the family B. This forms a natural equivalence, because transport is natural. The second equivalence is given by applying Lemma 3.3.21. We call the composite equivalence F, which shows that $\square B$ is an equifibered sequence. $\qquad \square$

Definition 3.3.25. Let (B, g) be a sequence over (A, f). Then we define

$$B_\infty : A_\infty \to \mathcal{U}$$

to be the family over A_∞ associated to the equifibered sequence $(\square B, \square g)$ via the equivalence of Theorem 3.3.18.

By construction of B_∞ we get the equality

$$r(y) : \mathsf{transport}^{B_\infty}(\kappa_n(x), y) = F(y)$$

for $y : B_\infty(\iota_{n+1}(f_n(x)))$ witnessing that B_∞ is defined by the equivalence F on the path constructor.

We now state our main result, which could be seen as a flattening lemma for sequential colimits, with the added generality that the sequence (B, g) over (A, f) is not required to be equifibered.

Theorem 3.3.26. *Let* $P :\equiv (P, f)$ *be a sequence over* $A :\equiv (A, a)$. *Then we have a commuting triangle*

in which α is an equivalence.

The strategy of the proof is to first show that $(x : A_\infty) \times P_\infty(x)$ has the induction principle of $\mathsf{colim}((x : A_n) \times P_n(x), (a_n, f_n))_n$. This simplifies giving the equivalence, because $(x : A_\infty) \times P_\infty(x)$ is a 2-HIT, being a sigma-type of two 1-HITs, while $\mathsf{colim}((x : A_n) \times P_n(x), (a_n, f_n))_n$ is a 1-HIT. Before we continue, we first define α.

The map α is defined by induction on $\mathsf{colim}(\Sigma(A, P))$. On the point constructors we define

$$\alpha(\iota_n(x, y)) :\equiv (\iota_n(x), \iota_0(y)).$$

For the path constructor we need to define

$$\kappa'_n(x, y) : (\iota_{n+1}(a_n x), \iota_0(f_n(x, y))) = (\iota_n(x), \iota_0(y))$$

The first components are equal by $\kappa_n(x)$. By the definition of P_∞, transporting along $\kappa_n(x)$ takes $\iota_0(f_n(x, y))$ to $\iota_1(f_n(x, y))$, which is equal to $\iota_0(y)$ by $\kappa_0(y)$. Explicitly, we define

$$\mathsf{ap}_\alpha(\kappa_n(x, y)) := \kappa'_n(x, y) :\equiv (\kappa_n(x), r(\iota_0(f_n(x, y))) \cdot \kappa_0(y)).$$

Theorem 3.3.27. *Let* $E : (x : A_\infty) \to P_\infty(x) \to \mathcal{U}$ *such that*

(i) *For each* $n : \mathbb{N}$, $x : A_n$, $y : P_n(x)$, *a term* $e_n(x, y) : E(\iota_n(x), \iota_0(y))$.

58

(ii) For each $n : \mathbb{N}$, $x : A_n$, $y : P_n(x)$, a path

$$w_n(x, y) : e_{n+1}(a_n x, f_n(x, y)) =^E_{\kappa'_n(x,y)} e_n(x, y).$$

Then there exists a function $s : (x : A_\infty)(y : P_\infty(x)) \to E(y)$.

Proof. We define the function s by induction on both x and y. We need to consider four cases, since both x and y can be a point constructor or vary over a path constructor.

(point-point) Fix $x : A_n$, we first define $g(n, x) : (p : P_\infty(\iota_n(x))) \to E(\iota_n(x), p)$. To obtain $g(n, x)$, we do induction on $p : P_\infty(\iota_n(x))$. Fix $y : P_{n+k}(a_n^k(x))$, we need to construct a term of type $g_*(k, n, x, y) : E(\iota_n(x), \iota_k(y))$. Proceed by induction on k. We can define

$$g_*(0, n, x, y) :\equiv e_n(x, y) : E(\iota_n(x), \iota_0(y)).$$

Assume that $g_*(k)$ is defined. We need to define $g_*(k + 1, n, x, y) : E(\iota_n(x), \iota_{k+1}(y))$, where $y : P(n + (k + 1), a_n^{k+1}(x))$. However, the type of y is equivalent to the type $P((n + 1) + k, a_{n+1}^k(a_n(x)))$ via the equivalence $u_{n,k}$. Therefore, it suffices to define for $z : P_{(n+1)+k}(a_{n+1}^k(a_n(x)))$

$$g_*(k + 1, n, x, u_{n,k}(z)) : E(\iota_n(x), \iota_{k+1}(u_{n,k}(z))).$$

By induction hypothesis we have $g_*(k, n + 1, a_n(x), z) : E(\iota_{n+1}(a_n(x)), \iota_k(z))$, so it suffices to show that

$$\kappa^*_{n,k}(x, z) : (\iota_{n+1}(a_n(x)), \iota_k(z)) = (\iota_n(x), \iota_{k+1}(u_{n,k}(z))).$$

This construction is similar to that of $\kappa'_n(x, y)$. The first components are equal by $\kappa_n(x)$, and for the second components we need to show that $\mathsf{transport}^{P_\infty}(\kappa_n(x), \iota_k(z)) = \iota_{k+1}(u_{n,k}(z))$. This follows from the computation rule of P_∞ on paths, since the equivalence used to define P_∞ sends $\iota_k(z)$ to $\iota_{k+1}(u_{n,k}(z))$. Specifically,

$$\kappa^*_{n,k}(x, z) :\equiv (\kappa_n(x), r(\iota_k(y))).$$

This finishes the construction of g_*, hence also of g on points. By construction, we get the following equation:

$$\mu_{n,k}(x, z) : g_*(k, n + 1, a_n(x), z) =^E_{\kappa^*_{n,k}(x,z)} g_*(k + 1, n, x, u_{n,k}(z))$$

(point-path) To define g on paths $\kappa_k(y) : \iota_{k+1}(f(y)) = \iota_k(y)$, we need to give a dependent path

$$\nu(k, n, x, y) : g_*(k + 1, n, x, f(y)) =^{E(\iota_n(x))}_{\kappa_k(y)} g_*(k, n, x, y).$$

We do this by induction on k. For $k = 0$ note that $u_{n,0}$ is the identity function, and the goal definitionally reduces to

$$\nu(k, n, x, y) : \mathsf{transport}^E(\kappa^*_{n,0}(x, f_n(x, y)), e_{n+1}(a_n(x), f_n(x, y))) =^{E(\iota_n(x))}_{\kappa_0(y)} e_n(x, y).$$

Note that $\kappa'_n(x, y) = \kappa^*_{n,0}(f_n(x, y) \cdot (1, \kappa_0(y)))$, which means we get this from $w_n(x, y)$. Now suppose that $\nu(k)$ is defined. We need to define for $y : P(n + (k + 1), a_n^{k+1}(x))$

$$\nu(k + 1, n, x, y) : g_*(k + 2, n, x, f(y)) =^{E(\iota_n(x))}_{\kappa_k(y)} g_*(k + 1, n, x, y).$$

Now we again write $y = u_{n,k}(z)$ for $z : P((n + 1) + k, a_{n+1}^k(a_n(x)))$ and we equivalently need to give

$$\nu(k + 1, n, x, u_{n,k}(z)) : g_*(k + 2, n, x, f(u_{n,k}(z))) =^E_{(1, \kappa_k(y))} g_*(k + 1, n, x, u_{n,k}z).$$

We will define this as the composition of a square that we will give later in the proof.

(path-point) We have defined s on points constructors of A_∞. To define it on the path $\kappa_n(x) : \iota_{n+1}(a_n(x)) = \iota_n(x)$ we need a path $g(n + 1, a_n(x)) = g(n, x)$ over $\kappa_n(x)$. By function extensionality, we can characterize dependent paths in a function type, which means we need to show:

$$(p : P_\infty(\iota_{n+1}(a_n(x)))) \to g(n + 1, a_n(x), p) =^E_{(\kappa_n(x), 1)} g(n, x, \mathsf{transport}^{P_\infty}(\kappa_n(x), p)).$$

Now for $p : P_\infty(\iota_{n+1}(a_n(x)))$, we can apply the path $r(p)$, which means we need to construct the following path (note that $r(p)$ is added to the path, since g is a dependent function):

$$g(n + 1, a_n(x), p) =^E_{(\kappa_n(x), r(p))} g(n, x, F(p)).$$

We proceed by induction on p. If $p \equiv \iota_k(y)$ for $k : \mathbb{N}$, $y : P_{(n+1)+k}(a_{n+1}^k(a_n(x)))$, then $F(p) \equiv \iota_{k+1}(u_{n,k}(y))$ and we need a path

$$g_*(k, n + 1, a_n(x), y) =^E_{(\kappa_n(x), r(\iota_k(y)))} g_*(k + 1, n, x, u_{n,k}(y)).$$

Now the path $(\kappa_n(x), r(\iota_k(y))) \equiv \kappa^*_{n,k}(x, y)$, hence this dependent path is given by $\mu_{n,k}(x, y)$.

(path-path) If p varies over $\kappa_k(y)$, we need to give a dependent path in a family of dependent paths. This is equivalent to filling the following dependent square in the family E, which lies over the naturality square form by applying $\lambda p. (\kappa_n(x), r(p))$ to

60

the path $\kappa_k(y)$.[18]

$$g_*(k+1, n+1, a_n(x), f_{(n+1)+k})(y)) \xupdownarrow[\kappa^*_{n,k+1}(x, f_{n+k}(y))]{\mu_{n,k+1}(x, f_{(n+1)+k}(y))} g_*(k+2, n, x, u_{n,(k+1)}(f_{(n+1)+k}(y)))$$

$$\left\| \begin{array}{c} (1, \kappa_k(y)) \; \Big\| \; \mathsf{apd}_{g(n+1, a_n(x))}(\kappa_k(y)))) \end{array} \right. \qquad\qquad\qquad \left. \begin{array}{c} (1, \kappa_k(y)) \; \Big\| \; \mathsf{apd}_{g(n,x) \circ F}(\kappa_k(y)))) \end{array} \right.$$

$$g_*(k, n+1, a_n(x), y) \xupdownarrow[\kappa^*_{n,k}(x,y)]{\mu_{n,k}(x,y)} g_*(k+1, n, x, u_{n,k}(y))$$

Below and to the left of each equal sign we give the path in $(x : A_\infty) \times P_\infty(x)$ over which the pathover lie. Above and to the right of each equal sign we give the value of the dependent path.

Now $\mathsf{apd}_{g(n+1, a_n(x))}(\kappa_k(y))$ (occurring in the left pathover) is equal to $\nu(k, n + 1, a_n(x), y)$ by definition of g. On the right, we have (δ is the naturality of $u_{n,k}$)

$$\begin{aligned}
\mathsf{apd}_{g(n,x) \circ F}(\kappa_k(y)) &= \mathsf{apd}_{g(n,x)}(F(\kappa_k(y))) \\
&= \mathsf{apd}_{g(n,x)}(\iota_{k+2}(\delta(y)) \cdot \kappa_{k+1}(u_{n,k}(y))) \\
&= \mathsf{apd}_{g_*(k+2,n,x)}(\delta(y)) \cdot \mathsf{apd}_{g(n,x)}(\kappa_{k+1}(u_{n,k}(y))) \\
&= \mathsf{apd}_{g_*(k+2,n,x)}(\delta(y)) \cdot \nu(k+1, n, x, u_{n,k}(y))
\end{aligned}$$

Now we can move the first part of the expression to the top of the square, which means we need to fill the following squareover (where we made some arguments implicit).

$$g_*(f(y)) \xupdownarrow[\kappa^*(f(y))]{\mu(f(y))} g_*(u(f(y))) \xupdownarrow[(1, \delta(y))]{\mathsf{apd}_{g_*(k+2,n,x)}(\delta(y))} g_*(f(u(y)))$$

$$\left\| \begin{array}{c} (1, \kappa(y)) \Big\| \nu(k, n+1, a_n(x), y) \end{array} \right. \qquad\qquad \left. \begin{array}{c} (1, \kappa(u_{n,k}(y))) \Big\| \nu(k+1, n, x, u_{n,k}(y)) \end{array} \right.$$

$$g_*(y) \xupdownarrow[\kappa^*(y)]{\mu(y)} g_*(u_{n,k}(y))$$

Note that in this squareover g does not occur, and ν only occurs on the left side (applied to k) and on the right side (applied to $k + 1$). Therefore, the top, bottom and left side form a valid open box, and we define $\nu(k+1, n, x, u_{n,k}(y))$ to be the composition of *this* open box. This inductively defines ν, and makes the filler for this square automatic. This finishes the proof. □

[18]The left and right sides of the square are not quite correct, the dependent function applied to $\kappa_k(y)$ are pathovers lying over $\kappa_k(y)$, and not $(1, \kappa_k(y))$. However, pathovers lying over $\kappa_k(y)$ in the family $E(\iota_n(x))$ are equivalent to pathovers lying over $(1, \kappa_k(y))$ in the family E, and this equivalence commutes with all operations we perform, therefore we omit them in this proof. The following calculations are only type correct when these equivalences are inserted back. Furthermore, we omit some other details. For example, if $p = q$, then $\mathsf{apd}_g(p)$ and $\mathsf{apd}_g(q)$ have different types: the former is a dependent path over p and the latter one over q. However, if you modify the path over which they lie, they become equal. These "modifications" can be pushed down to the square in $(x : A_\infty) \times P_\infty(x)$, and the proof still goes through. For the full details, consult the formal proof.

Proof (of Theorem 3.3.26). We first define a map

$$\beta : \big((x : A_\infty) \times P_\infty(x)\big) \to \mathsf{colim}(\Sigma(A, P)).$$

We do this by induction on $x : A_\infty$ and $p : P_\infty(x)$ individually, so we get four cases again (we do not use our newly defined induction principle, because we have not proven a computation rule for it).

(point-point) Suppose $x : A_n$ and $y : P_{n+k}(a_n^k(x))$. We define

$$\beta(\iota_n(x), \iota_k(y)) :\equiv \iota_{n+k}(a_n^k(x), y).$$

(point-path) To show that the second argument respects $\kappa_k(y)$, we define

$$\mathsf{ap}_{\beta(\iota_n(x))}(\kappa_k(y)) := \kappa_{n+k}(a_n^k(x), y) : \iota_{n+(k+1)}(a_n^{k+1}(x), f(y)) = \iota_{n+k}(a_n^k(x), y).$$

(path-point) To show that the first argument respects $\kappa_n(x)$, we need to give a dependent path

$$\beta(\iota_{n+1}(a_n(x))) =_{\kappa_n(x)}^{P_\infty(-)\to\mathsf{colim}(\Sigma(A,P))} \beta(\iota_n(x)).$$

By function extensionality, this is equivalent to showing for $p : P_\infty(\iota_{n+1}(a_n(x)))$ that

$$\beta(\iota_{n+1}(a_n(x)), p) = \beta(\iota_n(x), \mathsf{transport}^{P_\infty}(\kappa_n(x), p)).$$

We apply $\mathsf{ap}_{\beta(\iota_n(x))}^{-1}(r(p))$ on the right, so that we have to show

$$\beta(\iota_{n+1}(a_n(x)), p) = \beta(\iota_n(x), F(p)).$$

Now we apply induction on p. If $p \equiv \iota_k(y)$, then $F(p) \equiv \iota_{k+1}(u_{n,k}(y))$ and we need to show

$$\mu_{n,k}(x, y) : \iota_{(n+1)+k}(a_{n+1}^k(a_n(x)), y) = \iota_{n+(k+1)}(a_n^{k+1}, u_{n,k}(y)).$$

But the triples $((n + 1) + k, a_{n+1}^k(a_n(x)), y)$ and $(n + (k + 1), a_n^{k+1}, u_{n,k}(y))$ are equal: the first two components by Lemma 3.3.14 and the last component because $u_{n,k}$ was defined by transporting along the equality of the first components. Let us call this equality s. So we define $\mu_{n,k}(x, y)$ by applying ι to s.

(path-path) Suppose p varies along $\kappa_k(y)$, we need to construct a proof of a pathover in an equality type. This is equivalent to filling the following square.

$$
\begin{array}{ccc}
\beta(\iota_{n+1}(a_n(x)), \iota_{k+1}(f(y))) & \overset{\mu_{n,k+1}(x,f(y))}{=\!=\!=\!=\!=} & \beta(\iota_n(x), \iota_{k+2}(u_{n,k+1}(f(y)))) \\[2pt]
\Big\| \mathsf{ap}_{\beta(\iota_{n+1}(a_nx))}(\kappa_k(y)) & & \Big\| \mathsf{ap}_{\beta(\iota_n(x))\circ F}(\kappa_k(y)) \\[2pt]
\beta(\iota_{n+1}(a_n(x)), \iota_k(y)) & \overset{\mu_{n,k}(x,y)}{=\!=\!=\!=\!=} & \beta(\iota_n(x), \iota_{k+1}(u_{n,k}(y)))
\end{array}
$$

By simplifying the left and right path, this reduces to

$$
\begin{array}{ccc}
\beta(\iota(a(x)), \iota(f(y))) & \overset{\mu(f(y))}{=\!=\!=} \beta(\iota(x), \iota(u(f(y)))) \overset{\mathsf{ap}_{\beta(\iota(a(x)),\iota(-))}(\kappa(y))}{=\!=\!=\!=\!=\!=\!=} & \beta(\iota(x), \iota(f(u(y)))) \\[2pt]
\Big\| \kappa(a^k(a(x)),y) & & \Big\| \kappa(a^k(x),u(y)) \\[2pt]
\beta(\iota(a(x)), \iota_k(y)) & \overset{\mu(y)}{=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=\!=} & \beta(\iota(x), \iota(u(y)))
\end{array}
$$

Now the concatenation of the two paths on the top reduces to the function $i :\equiv \lambda n. \lambda x. \lambda y. \iota_{n+1}(a(x), f(y))$ applied to s. Then the square is exactly the naturality square of the homotopy $\kappa : i \sim \iota$ applied to the path s. This finishes the definition of β.

Now we need to show that $\beta \circ \alpha \sim \mathsf{id}$. Take $p : \mathsf{colim}(\Sigma(A, P))$, we apply induction to p. If $p \equiv \iota_n(x, y)$, then the equality holds by reflexivity:

$$\beta(\alpha(p)) \equiv \beta(\iota_n(x), \iota_0(y)) \equiv \iota_n(x, y) \equiv p.$$

If p varies over $\kappa_n(x, y)$, we need to fill a square with two degenerate sides, so we need to prove that $\mathsf{ap}_{\beta \circ \alpha}(\kappa_n(x, y)) = \kappa_n(x, y)$. We can show this as follows.

$$
\begin{aligned}
&\mathsf{ap}_{\beta \circ \alpha}(\kappa_n(x, y)) \\
={}& \mathsf{ap}_\beta(\kappa_n(x), r(\iota_0(f(y))) \cdot \kappa_0(y)) \\
={}& \mathsf{ap}_\beta(\kappa'_n(x, y)) \\
={}& \mu_{n,1}(x, f(y)) \cdot \mathsf{ap}^{-1}_{\beta(\iota_n(x))}(r(\iota_0(f(y)))) \cdot \mathsf{ap}_{\beta(\iota_n(x))}(r(\iota_0(f(y))) \cdot \kappa_0(y)) \\
={}& \mu_{n,1}(x, f(y)) \cdot \mathsf{ap}_{\beta(\iota_n(x))}(\kappa_0(y)) \\
={}& \mathsf{ap}_{\beta(\iota_n(x))}(\kappa_0(y)) \\
={}& \kappa_n(x, y)
\end{aligned}
$$

In the third step we use that $\mathsf{ap}_\beta(p, q) = \mathsf{apd}_\beta(p)(f(y)) \cdot \mathsf{ap}_{\beta\iota_n(x)(q)}$ and in the fifth step that $\mu_{n,k}(x, y) = 1$ for any *numeral* k.

Finally we need to show that $\alpha \circ \beta \sim \mathsf{id}$. Take $p : (x : A_\infty) \times P_\infty(x)$. We apply the induction principle proven in Theorem 3.3.27 to p. Suppose that $p \equiv (\iota_n(x), \iota_0(y))$. Then the equality holds by reflexivity:

$$\alpha(\beta(p)) \equiv \alpha(\iota_n(x, y)) \equiv (\iota_n(x), \iota_0(y)) \equiv p.$$

If p varies over $\kappa'_n(x, y)$, then we have to show (similar to the proof $\beta \circ \alpha \sim \mathsf{id}$) that

$$\mathsf{ap}_{\alpha \circ \beta}(\kappa'_n(x, y))\kappa'_n(x, y).$$

But by the previous computation, $\mathsf{ap}_\beta(\kappa'_n(x, y)) = \kappa_n(x, y)$, so we have

$$\mathsf{ap}_{\alpha \circ \beta}(\kappa'_n(x, y)) = \mathsf{ap}_\alpha(\kappa_n(x, y)) = \kappa'_n(x, y).$$

This finishes the proof. $\qquad\square$

Corollary 3.3.28. *Consider a sequence $(A_n, f_n)_n$. Then for any $a, a' : A_n$ there is an equivalence*

$$(\iota_n(a) =_{A_\infty} \iota_n(a')) \simeq \mathsf{colim}(f^k(a) =_{A_{n+k}} f^k(a')).$$

Proof. We first prove this for $n \equiv 0$. Note that for any $a : A_0$, we have the diagram

$$
\begin{array}{ccccccc}
\lambda(a' : A_0).\, a = a' & \longrightarrow & \lambda(a' : A_1).\, f(a) = a' & \longrightarrow & \lambda(a' : A_2).\, f^2(a) = a' & \longrightarrow & \cdots \\
\downarrow & & \downarrow & & \downarrow & & \\
A_0 & \xrightarrow{\ f_0\ } & A_1 & \xrightarrow{\ f_1\ } & A_2 & \xrightarrow{\ f_2\ } & \cdots
\end{array}
$$

This defines a type family $P : A_\infty \to \mathcal{U}$ with

$$P(\iota_n(a')) :\equiv \mathsf{colim}_k(f^{0 \leq n+k}(a) =_{A_{n+k}} f^k(a')).$$

Now we use Theorem 3.3.26 to see that the total space of P is contractible.

$$
\begin{aligned}
(a' : A_\infty) \times P(a') &\simeq \mathsf{colim}_n((a' : A_n) \times f^n(a) = a') \\
&\simeq \mathsf{colim}_n(\mathbf{1}) \\
&\simeq \mathbf{1}.
\end{aligned}
$$

Since $\iota_0(\mathsf{refl}\, a) : P(\iota_0(a))$ and noting that $f^{0 \leq 0+k}(a) \equiv f^k(a)$ we can now conclude by the total space method to characterize the identity type that

$$(\iota_0(a) =_{A_\infty} \iota_0(a')) \simeq P(\iota_0(a')) \equiv \mathsf{colim}(f^k(a) =_{A_{0+k}} f^k(a')).$$

For general n, we use Lemma 3.3.22, which gives us an equivalence $\mathsf{kshift\_equiv}_n : A_\infty \simeq \mathsf{colim}(S_n(A, f))$. For $a, a' : A_n$ we can now compute:

$$
\begin{aligned}
(\iota_n(a) =_{A_\infty} \iota_n(a')) &\simeq (\mathsf{kshift\_equiv}_n(\iota_n(a)) =_{\mathsf{colim}(S_n(A,f))} \mathsf{kshift\_equiv}_n(\iota_n(a'))) \\
&\simeq (\iota_0(a) =_{\mathsf{colim}(S_n(A,f))} \iota_0(a')) \\
&\simeq \mathsf{colim}(S_n(f)^k(a) =_{S_n(A)_{0+k}} S_n(f)^k(a')) \\
&\simeq \mathsf{colim}(f^k(a) =_{A_{n+k}} f^k(a'))..
\end{aligned}
$$

The last equivalence comes from a natural equivalences of the sequences, because there is a dependent path between $S_n(f)^k(a)$ and $f^k(a)$ over the canonical path that $n + (0 + k) = n + k$. $\qquad \square$

Corollary 3.3.29. *Suppose given a natural transformation* $\tau : (A', f') \to (A, f)$ *and a point* $a : A_n$. *Then*

$$\mathsf{fib}_{\tau_\infty}(\iota_n(a)) \simeq \mathsf{colim}(\square \mathsf{fib}_\tau[a]) \equiv \mathsf{colim}_k(\mathsf{fib}_{\tau_{n+k}}(f^k(a))).$$

Proof. Consider the following diagram, where the equivalences on the top are given by Theorem 3.3.26 and the fact that the total space of the fiber of a function is the domain of that function.

This diagram commutes: the left triangle commutes by Theorem 3.3.26 and the right triangle commutes by the 1-functoriality of the colimit, Lemma 3.3.19. Therefore,

$$\mathsf{fib}_{\tau_\infty}(\iota_n(a)) \simeq \mathsf{fib}_{\pi_1}(\iota_n(a)) \simeq (\mathsf{fib}_\tau)_\infty(\iota_n(a)) \equiv \mathsf{colim}(\square \mathsf{fib}_\tau[a]).$$

$\qquad \square$

Corollary 3.3.30. *Consider a sequence (A, f) and some $k \geq -2$.*

   (i) *If A_n is k-truncated for all $n : \mathbb{N}$, then A_∞ is k-truncated.*

   (ii) *We have an equivalence*

$$\|A_\infty\|_k \simeq \mathsf{colim}(\|A_n\|_k, \|f_n\|_k)_n.$$

   (iii) *If A_n is k-connected for all $n : \mathbb{N}$, then A_∞ is k-connected.*

   (iv) *Given a natural transformation $(\tau, H) : (A, f) \to (A', f')$ such that τ_n is k-truncated (k-connected) for all n, then τ_∞ is k-truncated (k-connected).*

   (v) *If f_n is k-truncated (k-connected) for all n, then ι_0 is k-truncated (k-connected).*

Remark 3.3.31. By Lemma 3.3.22 we can generalize the quantification "for all $n : \mathbb{N}$" in this Corollary to the weaker "there exists an $m : \mathbb{N}$ such that for all $n \geq m$". In part (v) the conclusion then becomes that ι_m is k-truncated (k-connected).

Proof.

   (i) We prove this by induction on k. Suppose $k = -2$, then f_n is an equivalence for all n. Therefore $A_\infty \simeq A_0$ by Lemma 3.3.20, hence A_∞ is contractible.

       Now suppose $k \equiv k' + 1$. Take $x, x' : A_\infty$, we need to show that $x = x'$ is k'-truncated. Since being truncated is a mere proposition, by induction on x and x' we may assume that $x \equiv \iota_n(a)$ and $x' \equiv \iota_m(a')$. Now $\iota_n(a) = \iota_{\mathsf{max}(n,m)}(f^{n \leq \mathsf{max}(n,m)}(a))$ and $\iota_m(a') = \iota_{\mathsf{max}(n,m)}(f^{m \leq \mathsf{max}(n,m)}(a'))$, therefore the type $\iota_n(a) = \iota_m(a')$ is equivalent to

$$\iota_{\mathsf{max}(n,m)}(f^{n \leq \mathsf{max}(n,m)}(a)) = \iota_{\mathsf{max}(n,m)}(f^{m \leq \mathsf{max}(n,m)}(a')).$$

       Therefore it suffices to show that the latter equality type is k'-truncated. By Corollary 3.3.28 we need to show that

$$\mathsf{colim}(f^\ell(f^{n \leq \mathsf{max}(n,m)}(a)) = f^\ell(f^{m \leq \mathsf{max}(n,m)}(a')))_\ell$$

       is k'-truncated, which follows from the induction principle and the fact that $A_{\mathsf{max}(n,m)+\ell}$ is $(k'+1)$-truncated.

   (ii) From the functoriality of the sequential colimit, we get a function

$$A_\infty \to \mathsf{colim}_n(\|A_n\|_k, \|f_n\|_k).$$

       Because the right hand side is k-truncated, this induces a map

$$g : \|A_\infty\|_k \to \mathsf{colim}(\|A_n\|_k, \|f_n\|_k)_n.$$

       For the other direction, we define the function

$$h : \mathsf{colim}(\|A_n\|_k, \|f_n\|_k)_n \to \|A_\infty\|_k$$

by
$$h(\iota_n(|a|_k)) :\equiv |\iota_n(a)|_k$$
and
$$\mathsf{ap}_h(\kappa_n(|a|_k)) := \mathsf{ap}_{|-|_k}(\kappa_n(a)).$$

It is straightforward to show that both $h \circ g$ and $g \circ h$ are homotopic to the identity.

(iii) Since A_n is k-connected, $\|A_n\|_k$ is contractible, and therefore $\mathsf{colim}_n(\|A_n\|_k) \simeq \|A_\infty\|_k$ is contractible.

(iv) A function is k-truncated (k-connected) whenever its fibers are k-truncated (k-connected). Let $x : A_\infty$. We need to show a proposition, so we may assume that $x \equiv \iota_n(a)$ for some $a : A_n$. Now $\mathsf{fib}_{\tau_\infty}(\iota_n(a)) \simeq \mathsf{colim}(\square\mathsf{fib}_\tau[a])$ by Corollary 3.3.29. Since $\mathsf{fib}_{\tau_n}(x)$ is k-truncated (k-connected) for all n, we know that $\mathsf{colim}(\square\mathsf{fib}_\tau[a])$ is k-truncated (k-connected) for all n, by part (i) or (iii).

(v) Consider the natural transformation

$$
\begin{array}{ccccccccc}
A_0 & = & A_0 & = & A_0 & = & A_0 & = \cdots = & \mathsf{colim}(A_0)_n \\
\| & & \|{\scriptstyle f} & & \|{\scriptstyle f^{0\leq 2}} & & \|{\scriptstyle f^{0\leq 3}} & & \downarrow \\
A_0 & \longrightarrow & A_1 & \longrightarrow & A_2 & \longrightarrow & A_3 & \longrightarrow \cdots \longrightarrow & A_\infty
\end{array}
$$

The maps $f^{0\leq n} : A_0 \to A_n$ are k-truncated (k-connected) and form a natural transformation. Therefore, by part (iv) the map $f^{0\leq\infty} : \mathsf{colim}_n(A_0) \to A_\infty$ is k-truncated (k-connected). The fiber of ι_0 over $x : A_\infty$ is the same as the fiber of $f^{0\leq\infty}$ over x, and therefore ι_0 is k-truncated (k-connected).

$\square$

We can use this machinery, in particular Theorem 3.3.26, to define the localization for maps between ω-compact types. We will omit the construction here, but this will be published in an upcoming preprint.

Chapter 4

Homotopy Theory

As discussed in the introduction, one very useful application of HoTT is synthetic homotopy theory. Many results in homotopy theory have been stated and proven in HoTT in a synthetic way. Most of these results have also been formalized in a proof assistant. This is important, because one of the advantages of HoTT is to make verification of proofs by a proof assistant practically possible. Formalizing results that have been proved internally in HoTT provides more evidence for this.

In this chapter we will look at various topics in homotopy theory and give proofs for them in HoTT that are fully checked by the Lean proof assistant. In Section 4.1 we will describe a formalization of the proof that $\pi_3(\mathbb{S}^2) = \mathbb{Z}$. This was already known to be provable in HoTT, but no fully formalized proof has been given before. We will discuss some new properties proven about Eilenberg-MacLane spaces in HoTT in Section 4.2, namely that the Eilenberg-MacLane space functor induces an equivalence of categories. In Section 4.3 we prove the adjunction of the smash product and pointed maps, from which we can conclude that the smash product is associative.

None of these results have been formalized before, even including formalization in foundations other than HoTT. In fact, not much homotopy theory has been formalized in other foundations. The most notable examples of formalizations are the formalization of basic properties of the fundamental group [Zha17] and the formalization of singular homology theory [H⁺, `Multivariate/homology.ml`].

4.1   Computing $\pi_3(\mathbb{S}^2)$

Computing that $\pi_3(\mathbb{S}^2) = \mathbb{Z}$ has been done before in Homotopy Type Theory, but it has not been formalized in a proof assistant before. In this section we will discuss some considerations of formalizing the proof that $\pi_3(\mathbb{S}^2) = \mathbb{Z}$. The Hopf fibration was formalized in Lean by Ulrik Buchholtz and was formalized before in Agda by Guillaume Brunerie. The remaining results are formalized by the author.

4.1.1 The long exact sequence of homotopy groups

We start with an important result in homotopy theory, the long exact sequence of homotopy groups.

This has been proven before in HoTT. Two different proofs are given in [Uni13, Section 8.4] and [Bru16, Section 2.5.1], although these proofs have not been formalized. There have been previous formalizations of parts of this result [AKL15, Voe15, VAG$^+$]; however none of these formalizations are complete in the sense that they can be used to deduce the results in this section.

The statement is as follows.

Theorem 4.1.1 (Long exact sequence of homotopy groups)**.** *Suppose $f : X \to Y$ is a pointed map. Then the following is an exact sequence*

$$\vdots$$

$$
\begin{array}{ccccc}
\pi_2(F) & \xrightarrow{\;\;\pi_2(p_1)\;\;} & \pi_2(X) & \xrightarrow{\;\pi_2(f)\;} & \pi_2(Y) \\
& & & & \\
& \pi_1(\delta) & & & \\
\pi_1(F) & \xrightarrow{\;\;\pi_1(p_1)\;\;} & \pi_1(X) & \xrightarrow{\;\pi_1(f)\;} & \pi_1(Y) \\
& & & & \\
& \pi_0(\delta) & & & \\
\pi_0(F) & \xrightarrow{\;\;\pi_0(p_1)\;\;} & \pi_0(X) & \xrightarrow{\;\pi_0(f)\;} & \pi_0(Y)
\end{array}
$$

Here $F :\equiv \mathsf{fib}_f$ is the fiber of f, $p_1 : F \to X$ is the first projection, and $\delta : \Omega Y \to F$ is defined in the proof.

First of all, we have to carefully formulate the statement of this theorem in type theory. The naive thing to do is to say that there is a sequence $A : \mathbb{N} \to \mathsf{Set}^*$ and maps $f : (n : \mathbb{N}) \to A_{n+1} \to A_n$ such that

$$A_0 :\equiv \pi_0(Y), \quad A_1 :\equiv \pi_0(X), \quad A_2 :\equiv \pi_0(F),$$

and so forth. Continuing, this means that

$$A_{3n} = \pi_n(Y), \quad A_{3n+1} = \pi_n(X), \quad A_{3n+2} = \pi_n(F).$$

However, there is no way to make these equalities definitional, the elimination principle for the natural numbers does not allow for computation rules like that. This means that the map $f_{3n} : A_{3n+1} \to A_{3n}$ cannot be compared directly to $\pi_n(f)$ since the domain and codomain are note definitionally equal. Setting things up this way is possible, but makes reasoning about it unnecessarily complicated. Instead, we change the indexing set, using $\mathbb{N} \times \mathsf{fin}_3$ instead of $\mathbb{N}$. We will work with a general notion of sequences with a flexible choice of indexing set.

Definition 4.1.2. A *successor structure* is a type I with endomap $S : I \to I$ called the *successor*. We will write $i + n$ for $i : I$ and $n : \mathbb{N}$ to mean iterated application of the successor function, $i + n :\equiv S^n(i)$.

A *chain complex* indexed by a successor structure I is a family of pointed sets $A : I \to \mathsf{Set}^*$ and maps $f : (i : I) \to A_{i+1} \to A_i$ with the property that $(i : I) \to (a : A_{i+2}) \to f_i(f_{i+1}(a)) = a_0^i$ where a_0^i is the basepoint of A_i. We call a chain complex *exact* or a *long exact sequence* if

$$(i : I) \to (a : A_{i+1}) \to f_i(a) = a_0^i \to \|(a' : A_{i+2}) \times f_{i+1}(a') = a\|.$$

A *type-valued chain complex* is the same, except that A_i is only required to be a pointed type (not a pointed set). A type-valued chain complex is *exact* or a *type-valued exact sequence* if the above property holds without any propositional truncation, i.e. if

$$(i : I) \to (a : A_{i+1}) \to f_i(a) = a_0^i \to (a' : A_{i+2}) \times f_{i+1}(a') = a.$$

Remark 4.1.3. Note that a type-valued exact sequence gives part of the structure of a *fiber sequence*. A *fiber sequence* is a sequence where A_{i+2} "is" the fiber of f_i. This means that $(A_{i+2}, f_{i+1}) = (\mathsf{fib}_{f_i}, p_1)$ for all i. Using univalence this can be unpacked in an equivalence and a commuting triangle. In a type-valued exact sequence we just require two maps back and forth $A_{i+2} \leftrightarrow \mathsf{fib}_{f_i}$ such that the corresponding triangles commute, but we do not require that these maps are mutual inverses. In the text below we will have sequences that are not fiber sequences, so we require this additional generality.

Example 4.1.4. Some useful examples of successor structures are $(\mathbb{N}, \lambda n.\, n + 1)$ and $(\mathbb{Z}, \lambda n.\, n + 1)$. Sequences over these successor structures correspond to one-sided and two-sided infinite sequences. We can also mimic one-sided infinite sequences in the other direction using the successor structure $(\mathbb{N}, \lambda n.\, n - 1)$ (with the convention that $0 - 1 = 0$). This has the disadvantage that there is one extra map $A_0 \to A_0$. Whenever we use $\mathbb{N}$ as successor structure in this section, we use $\lambda n.\, n + 1$ as its successor.

Furthermore, if N is a successor structure and $k : \mathbb{N}$, then we define a successor structure on $N \times \mathsf{fin}_{k+1}$ by defining

$$S(n, i) :\equiv \begin{cases} (n + 1, 0) & \text{if } i = k \\ (n, i + 1) & \text{otherwise} \end{cases}$$

Note that $n + 1$ is addition in the successor structure N.

We now build the long exact sequence of homotopy groups in five steps. The order of these steps is somewhat arbitrary and can be altered. We perform the 0-truncation of the sequence as the last step, so that the intermediate sequences contain as much information as possible.

(1) First we define the fiber sequence of f.

(2) Then we show that this sequence is equivalent to a sequence involving iterated loop spaces.

(3) We fix some negation signs in the exact sequence.

(4) We index the sequence over $\mathbb{N} \times \mathsf{fin}_3$.

(5) We 0-truncate the sequence to obtain the sequence in Theorem .

We first need some lemmas about fibers.

Lemma 4.1.5. *Suppose given a pointed map $f : A \to^* B$. Let $p_1 : \mathsf{fib}_f \to^* A$ be the first projection. Then there is a pointed natural equivalence $e_f : \mathsf{fib}_{p_1} \simeq^* \Omega B$.*
   Furthermore, if $q_1 : \mathsf{fib}_{p_1} \to \mathsf{fib}_f$ is the first projection, we get a commuting square

$$
\begin{array}{ccc}
\Omega A & \xrightarrow{\ -\Omega f\ } & \Omega B \\
\big\uparrow{\scriptstyle e_{p_1}} & & \big\uparrow{\scriptstyle e_f} \\
\mathsf{fib}_{q_1} & \xrightarrow{\ r_1\ } & \mathsf{fib}_{p_1}
\end{array}
$$

where r_1 is (also) the first projection. We write $-\Omega f$ for the map $\Omega f \circ (-)^{-1}$.

Proof. The underlying equivalence is the following composite

$$
\begin{aligned}
\mathsf{fib}_{p_1} &\simeq ((a, p) : \mathsf{fib}_f) \times a = a_0 \\
&\simeq (a : A) \times a = a_0 \times f(a) = b_0 \\
&\simeq f(a_0) = b_0 \\
&\simeq b_0 = b_0 \equiv \Omega B
\end{aligned}
$$

This equivalence sends $((a, p), q) : \mathsf{fib}_{p_1}$ (with $p : fa = b_0$ and $q : a = a_0$) to $f_0^{-1} \cdot f(q) \cdot p$. So there is a path

$$
r(a, p, q) : e_f((a, p), q) = f_0^{-1} \cdot f(q^{-1}) \cdot p.
$$

This path satisfies $r(a_0, f_0, 1) = 1$ (equality is type correct since $e_f((a_0, f_0), q) \equiv f_0^{-1} \cdot q$). We also have $e_f^{-1}(p) = ((a_0, f_0 \cdot p), 1)$ for $p : \Omega B$.

Now e respects the basepoint, because

$$
e(a_0, f_0, 1) = f_0^{-1} \cdot f_0 = 1.
$$

We will not prove naturality here, since it is not required for the results in this section. For the commuting square, we will prove that

$$
h : e_f \circ r_1 \circ e_{p_1}^{-1} \sim^* -\Omega f
$$

For the underlying homotopy, we compute for $p : \Omega A$

$$
\begin{aligned}
e_f(r_1(e_{p_1}^{-1} p)) &= e_f(r_1(((a_0, f_0), 1 \cdot p), 1)) \\
&= e_f((a_0, f_0), p) \\
&= f_0^{-1} \cdot f(p^{-1}) \cdot f_0 \equiv -\Omega f(p).
\end{aligned}
$$

70

To show that h respects the basepoint, suppose that $p \equiv 1$. In that case, the first two steps of the above equation becomes definitional equalities. Since we know that $r(a_0, f_0, 1) = 1$, the last equality is also reflexivity. Since the maps $e_f \circ r \circ e_{p_1}^{-1}$ and $-\Omega f$ respect the basepoints using the same path, this shows that h is a pointed homotopy, which finishes the proof. $\qquad \square$

4.1.1.1 Step 1

Denote $\mathsf{arrow}^* :\equiv (X\ Y : \mathcal{U}_i^*) \times (X \to^* Y)$. We define $F : \mathsf{arrow}^* \to \mathsf{arrow}^*$ by $F(X, Y, f) :\equiv (\mathsf{fib}_f, X, p_1)$. Given a pointed map $f : X \to^* Y$, we define its fiber sequence $A : \mathbb{N} \to \mathcal{U}$ by $A_n :\equiv p_2(F^n(X, Y, f))$, and we define $f_n : A_{n+1} \to A_n$ by $p_3(F^n(X, Y, f))$ (which is well-typed, since $A_{n+1} \equiv p_1(F^n(X, Y, f))$ by unfolding the definition of F). It is easy to show that $(A_n, f_n)_n$ is a type-valued exact sequence, since A_{n+2} is (definitionally) the fiber of f_n.

Note that by Lemma 4.1.5 there is a pointed equivalence $e_f : A_3 \simeq^* \Omega Y$. We define the diagonal map $\delta :\equiv p_1 \circ e_f^{-1} : \Omega Y \to \mathsf{fib}_f$.

4.1.1.2 Step 2

Define the sequence $B : \mathbb{N} \to \mathcal{U}$ and $g_n : B_{n+1} \to B_n$ by

$$
\begin{aligned}
B_0 &:\equiv Y \\
B_1 &:\equiv X & g_0 &:\equiv f \\
B_2 &:\equiv \mathsf{fib}_f & g_1 &:\equiv p_1 \\
B_{n+3} &:\equiv \Omega B_n & g_2 &:\equiv \delta \\
& & g_{n+3} &:\equiv -\Omega g_n
\end{aligned}
$$

Note that g_2 has the correct type, since $A_3 \equiv B_3$.

Now we can show that (B, g) is a type-valued exact sequence by showing that it is equivalent to (A, f).

Lemma 4.1.6. *There is a natural equivalence $(A_n, f_n)_n \simeq (B_n, g_n)_n$. This means that there are pointed equivalences $\eta_n : A_n \simeq^* B_n$ such that for all $n : \mathbb{N}$ we have*

$$\eta_n \circ f_n \sim^* g_n \circ \eta_{n+1}.$$

Proof. We define the equivalence η_n by induction on n. Note that $A_k \equiv B_k$ for $k = 0, 1, 2$. Now suppose we have an equivalence $\eta_k : A_k \simeq B_k$. Then by Lemma 4.1.5 we have

$$A_{k+3} \equiv \mathsf{fib}_{f_{k+1}} \overset{e_{f_k}}{\simeq} \Omega A_k \overset{\Omega \eta_k}{\simeq} \Omega B_k \equiv B_{k+3}.$$

We also show the naturality by induction on n.
For $n \equiv 0$ we have $\mathsf{id}_Y \circ f \sim^* f \circ \mathsf{id}_X$.

For $n \equiv 1$ we have $\mathsf{id}_X \circ p_1 \sim^* p_1 \circ \mathsf{id}_{\mathsf{fib}_f}$.

For $n \equiv 2$ we have

$$\mathsf{id}_{\mathsf{fib}_f} \circ p_1 \equiv p_1 \sim^* (p_1 \circ e_f^{-1}) \circ e_f \sim^* \delta \circ (\Omega \mathsf{id}_Y \circ e_f).$$

Now suppose the naturality holds for k, then we get the following diagram.

$$
\begin{array}{ccc}
\Omega B_{k+1} & \xrightarrow{\ -\Omega g_k\ } & \Omega B_k \\[4pt]
\big\uparrow{\scriptstyle \Omega \eta_{k+1}} & & \big\uparrow{\scriptstyle \Omega \eta_k} \\[4pt]
\Omega A_{k+1} & \xrightarrow{\ -\Omega f_k\ } & \Omega A_k \\[4pt]
\big\uparrow{\scriptstyle e_{f_{k+1}}} & & \big\uparrow{\scriptstyle e_{f_k}} \\[4pt]
A_{k+4} & \xrightarrow{\ f_{k+3}\ } & A_{k+3}
\end{array}
$$

The bottom square can be filled by the second part of Lemma 4.1.5. The top square can be filled by applying the functor Ω to the naturality for k and then noticing that $(-)^{-1} \circ \Omega \eta_k \sim^* \Omega \eta_k \circ (-)^{-1}$, which is easily proven for an arbitrary pointed map. $\square$

4.1.1.3 Step 3

We now remove the inverses in our sequence. More precisely, we define a second sequence $h_n : B_{n+1} \to B_n$ by

$$h_0 :\equiv f \qquad h_1 :\equiv p_1 \qquad h_2 :\equiv \delta \qquad h_{n+3} :\equiv \Omega h_n.$$

To show that (B, h) is a type-valued exact sequence we use the following lemma.

Lemma 4.1.7. *Suppose N is a successor structure and (B, g) is a type-valued exact sequence over N. Suppose $h_n : B_{n+1} \to^* B_n$ is another sequence of maps, and suppose that there are pointed maps $e_n, \ell_n, r_n : B_n \to^* B_n$ such that e_n is an equivalence and the following diagrams commute as homotopies (not necessarily pointed):*

$$
\begin{array}{ccc}
B_{n+1} & & \\
{\scriptstyle e_{n+1}}\big\uparrow & \searrow^{h_n} & \\
B_{n+1} & \xrightarrow{\ g_n\ } & B_n
\end{array}
\qquad
\begin{array}{ccc}
B_{n+1} & \xrightarrow{\ h_n\ } & B_n \\
{\scriptstyle \ell_{n+1}}\big\downarrow & & \big\downarrow{\scriptstyle e_n} \\
B_{n+1} & \xrightarrow{\ h_n\ } & B_n
\end{array}
\qquad
\begin{array}{ccc}
B_{n+1} & \xrightarrow{\ h_n\ } & B_n \\
{\scriptstyle e_{n+1}}\big\downarrow & & \big\uparrow{\scriptstyle r_n} \\
B_{n+1} & \xrightarrow{\ h_n\ } & B_n
\end{array}
$$

Then (B, h) is a type-valued exact sequence over N.

Proof. First we need to show that for $x : B_{n+2}$ we have $h_n(h_{n+1}(x)) = b_0^n$. We compute

$$
\begin{aligned}
h_n(h_{n+1}(x)) &= r_n(h_n(e_{n+1}(h_{n+1}(x)))) \\
&= r_n(g_n(h_{n+1}(x))) \\
&= r_n(g_n(g_{n+1}(e_{n+2}^{-1}(x)))) \\
&= r_n(b_0^n) \\
&= b_0^n.
\end{aligned}
$$

For exactness, suppose that $y : B_{n+1}$ such that $h_n(y) = b_0^n$. Then $g_n(e_{n+1}^{-1}(y)) = h_n(y) = b_0^n$, therefore, by exactness of g there (purely) exists an $x : B_{n+2}$ such that $g_{n+1}(x) = e_{n+1}^{-1}(y)$. Now we compute

$$
\begin{aligned}
h_{n+1}(\ell_{n+2}(e_{n+2}(x))) &= e_{n+1}(h_{n+1}(e_{n+2}(x))) \\
&= e_{n+1}(g_{n+1}(x)) \\
&= e_{n+1}(e_{n+1}^{-1}(y)) \\
&= y.
\end{aligned}
$$

This finishes the proof. $\square$

Lemma 4.1.8. *The sequence (B, h) is a type-valued exact sequence.*

Proof. We first define for $k \geq 2$ we the pointed equivalence $\varepsilon_n^k : B_n \simeq^* B_n$ by induction on n. For $n \leq k$ $\varepsilon_n^k :\equiv \mathsf{id} : B_n \simeq^* B_n$ we define $\varepsilon_{n+3}^k :\equiv -\Omega\varepsilon_n^k : B_{n+3} \simeq^* B_{n+3}$ for $n + 3 > k$. Now define $e_n :\equiv \varepsilon_n^3$ and $\ell_n :\equiv \varepsilon_n^4$ and $r_n :\equiv \varepsilon_n^2$. We apply Lemma 4.1.7 using these equivalences to obtain the desired result. To do this we need to check three commuting triangles. We will check $h_n \circ e_{n+1} \sim g_n$, the other two proofs are similar. Apply induction on n. For $n = 0, 1, 2$ it is trivial, reducing to $g_n \circ \mathsf{id} \sim g_n$. Suppose the homotopy is true for $n = k$. Then

$$
h_{k+3} \circ e_{k+4} \equiv \Omega h_k \circ -\Omega e_{k+1} \sim -\Omega(h_k \circ e_{k+1}) \sim -\Omega g_k \equiv g_{k+3}.
$$

$\square$

4.1.1.4 Step 4

We now define a type-valued chain complex over $\mathbb{N} \times \mathsf{fin}_3$, which has a successor structure by Example 4.1.4. Let ρ_X be the equivalence $\Omega^{n+1}X \simeq^* \Omega^n(\Omega X)$. We now define the sequence $C : \mathbb{N} \times \mathsf{fin}_3$ and $k_n : C_{n+1} \to C_n$ by

$$
\begin{aligned}
C_{(n,0)} &:\equiv \Omega^n Y & k_{(n,0)} &:\equiv \Omega^n f \\
C_{(n,1)} &:\equiv \Omega^n X & k_{(n,1)} &:\equiv \Omega^n p_1 \\
C_{(n,2)} &:\equiv \Omega^n \mathsf{fib}_f & k_{(n,2)} &:\equiv \Omega^n \delta \circ \rho_X
\end{aligned}
$$

In a diagram, (C, k) looks like the following.

$$\vdots$$

$$
\begin{array}{ccc}
\Omega^2 F \xrightarrow{\;\Omega^2 p_1\;} \Omega^2 X \xrightarrow{\;\Omega^2 f\;} \Omega^2 Y \\
\end{array}
$$

$$\Omega\delta$$

$$
\begin{array}{ccc}
\Omega F \xrightarrow{\;\Omega p_1\;} \Omega X \xrightarrow{\;\Omega f\;} \Omega Y \\
\end{array}
$$

$$\delta$$

$$
\begin{array}{ccc}
F \xrightarrow{\;p_1\;} X \xrightarrow{\;f\;} Y \\
\end{array}
$$

There is an equivalence $e : \mathbb{N} \simeq \mathbb{N} \times \mathsf{fin}_3$ that sends n to its quotient and remainder when dividing n by 3. The proof of the following lemma is straightforward and omitted.

Lemma 4.1.9. *The sequence (B, h) is naturally equivalent to (C, k) over the equivalence e. Therefore, (C, k) is a type-valued exact sequence.*

4.1.1.5 Step 5

If we 0-truncate the sequences at step 4, we get the sequence $(D, \ell) :\equiv (\|C\|_0, \|k\|_0)$. This is exactly the sequence in Theorem 4.1.1. It is now easy to show that this is a long exact sequence.

Proof of Theorem 4.1.1. First note that it is a chain complex by the following computation:

$$\ell_n \circ \ell_{n+1} \sim \|k_n \circ k_{n+1}\|_0 \sim \|0\|_0 \sim 0.$$

To show that it is exact, suppose given $x : D_{n+1}$ and $p : \ell_n(x) = d_0^n$. We need to construct an element in a proposition, so we may assume by induction that $x \equiv |y|_0$. Now the type of p reduces to $|k_n(y)|_0 = |c_0^n|_0$, which is equivalent to $\|k_n(y) = c_0^n\|_{-1}$ by the characterization of the identity type in truncations. Therefore, the latter type is inhabited, and by induction, we may assume that we have a path $k_n(y) = c_0^n$. By exactness of (C, k) we get an element $z : C_{n+2}$ such that $q : k_{n+1}(z) = y$. Now we can find $|z|_0 : D_{n+2}$ and the path $\mathsf{ap}_{|-|_0}(q) : \ell_{n+1}(|z|_0) = x$, showing exactness. $\qquad\square$

4.1.2 Computation of homotopy groups

An important application of the long exact sequence of homotopy groups comes in combination with the Hopf fibration. Combining these tools, we can compute more homotopy groups of spheres. The Hopf fibration was constructed in [Uni13, Theorem 8.5.1] and has been formalized by Ulrik Buchholtz. We will not give the construction here.

Theorem 4.1.10 (Hopf Fibration)**.** *There is a pointed map $\mathbb{S}^3 \to \mathbb{S}^2$ with fiber $\mathbb{S}^1$.*

The quaternionic Hopf fibration has also been constructed in HoTT and formalized in Lean [BR16]. This gives a fibration $\mathbb{S}^7 \to \mathbb{S}^4$ with fiber $\mathbb{S}^3$.

Corollary 4.1.11. $\pi_2(\mathbb{S}^2) = \mathbb{Z}$ and $\pi_n(\mathbb{S}^3) = \pi_n(\mathbb{S}^2)$ for $n \geq 3$.

Proof. We know by the connectedness of spheres that $\pi_1(\mathbb{S}^3)$ and $\pi_2(\mathbb{S}^3)$ are trivial, and by the truncatedness of the circle that $\pi_k(\mathbb{S}^1)$ is trivial for $k > 1$ and $\mathbb{Z}$ for $k = 1$. We now get the following long exact sequence, from which the result immediately follows.

$$\vdots$$

$$
\begin{array}{ccccc}
0 & \longrightarrow & \pi_4(\mathbb{S}^3) & \twoheadrightarrow & \pi_4(\mathbb{S}^2) \\
0 & \xrightarrow{\;\;} & \pi_3(\mathbb{S}^3) & \twoheadrightarrow & \pi_3(\mathbb{S}^2) \\
0 & \xrightarrow{\;\;} & 0 & \longrightarrow & \pi_2(\mathbb{S}^2) \\
\mathbb{Z} & \xrightarrow{\;\;} & 0 & \longrightarrow & 0
\end{array}
$$

$\square$

The last ingredient we need is the Freudenthal Suspension Theorem. This has been formalized before by Dan Licata in Agda, and our formalization is a direct port of that proof to Lean. For the proof we refer to [Uni13, Section 8.6].

Theorem 4.1.12 (Freudenthal Suspension Theorem). *Suppose that X is n-connected. Then $\|X\|_{2n} \simeq \|\Omega\Sigma X\|_{2n}$.*

We can combine these results to compute the following homotopy groups.

Corollary 4.1.13. $\pi_n(\mathbb{S}^n) = \mathbb{Z}$ and $\pi_3(\mathbb{S}^2) = \mathbb{Z}$

Proof. Note that $\mathbb{S}^n$ is $(n-1)$-connected. Therefore, by the Freudenthal suspension theorem we have

$$\|\mathbb{S}^n\|_{2(n-1)} \simeq \|\Omega\mathbb{S}^{n+1}\|_{2(n-1)}.$$

For $n \geq 2$ we have $2(n-1) \geq n$, and therefore we also have

$$\|\mathbb{S}^n\|_n \simeq \|\Omega\mathbb{S}^{n+1}\|_n.$$

Taking the n-th homotopy group, we get

$$\pi_n(\mathbb{S}^n) \simeq \pi_{n+1}(\mathbb{S}^{n+1}).$$

Combining this with Corollary 4.1.11, we also get $\pi_3(\mathbb{S}^2) \simeq \mathbb{Z}$, as desired. $\square$

4.2 Eilenberg-MacLane Spaces

In this section we give an important equivalence between groups and Eilenberg-MacLane spaces [EM45].[19] Eilenberg-MacLane space are play an important role in homotopy theory, since they are spaces with simple homotopy groups. Therefore, they can be used to build up more complicated spaces with complicated homotopy groups. Also, they can be used to define homology and cohomology in HoTT, see Sections 5.4 and 5.5.

We prove in this section that the category of n-connected $(n+1)$-truncated pointed types is equivalent to the category of groups for $n = 0$ and the category of abelian groups for $n \geq 1$.

If G is a (pre-)groupoid, the groupoid quotient is a higher inductive type with constructors

HIT groupoid-quotient$(G) :=$
- $i : G_0 \to$ groupoid-quotient(G);
- $p : (x \ y : G_0) \to$ hom$(x, y) \to x = y$;
- $q : (x \ y \ z : G_0) \to (g :$ hom$(y, z)) \to (f :$ hom$(x, y)) \to p(g \circ f) = p(f) \cdot p(g)$;
- $\varepsilon :$ is-1-type(groupoid-quotient(G)).

The groupoid quotient can be constructed purely from homotopy pushouts. The untruncated version was constructed in Section 3.2. Then we can apply the 1-truncated afterwards, and we can also construct truncations from homotopy pushouts [Rij17].

In [LF14] the authors define Eilenberg-MacLane spaces. We use the same approach as in that paper. We first quickly review the results in that paper.

4.2.1 Construction of Eilenberg-MacLane spaces

If G is any group, the 1-dimensional Eilenberg-MacLane space $K(G, 1)$ can be defined by viewing G as a groupoid, and taking the groupoid quotient of G. It is not hard to see that $K(G, 1)$ is 0-connected and 1-truncated. Using an encode-decode proof, we can show that $\Omega K(G, 1) \simeq G$ and that this equivalence sends concatenation to multiplication. Hence the composite $\pi_1 K(G, 1) \simeq \|G\|_0 \simeq G$ is a group isomorphism.

If G is abelian, the higher Eilenberg-MacLane spaces can be defined recursively as

$$K(G, n + 1) :\equiv \|\Sigma K(G, n)\|_{n+1}$$

for $n \geq 1$. This definition is slightly different than the one given in [LF14], where $K(G, n + 1)$ was defined using the iterated suspension as $\|\Sigma^n K(G, 1)\|_{n+1}$. We chose to modify the definition, since a lot of properties of Eilenberg-MacLane spaces are proven by induction on n, so it is more convenient to have $K(G, n + 1)$ defined directly in terms of $K(G, n)$.

It is easy to show that $K(G, n)$ is $(n - 1)$-connected and n-truncated. It is trickier to show that $\Omega K(G, n + 1) \simeq K(G, n)$. This is done separately for $n = 1$ and for $n \geq 2$.

---

[19]Some of the contents of this section have been published in [BvDR18]. The work in this section is joint work with Ulrik Buchholtz and Egbert Rijke.

For $n = 1$ we need the result that for every type X with a coherent h-structure, the type $\|\Sigma X\|_2$ is a *delooping* of X, which means that $\Omega \|\Sigma X\|_2 \simeq X$. If G is abelian, then $K(G, 1)$ can be equipped with a coherent h-structure, showing that $\Omega K(G, 2) \simeq K(G, 1)$.

For $n \geq 2$, this can be done using the Freudenthal suspension theorem, Theorem 4.1.12. Then the equivalence follows from the following chain of equivalences:

$$\Omega K(G, n+1) \equiv \Omega \|\Sigma K(G, n)\|_{n+1} \simeq \|\Omega \Sigma K(G, n)\|_n \simeq \|K(G, n)\|_n \simeq K(G, n).$$

The Freudenthal Suspension Theorem is applied in the third step, which is allowed since $K(G, n)$ is $(n-1)$-connected and $n \leq 2(n-1)$ for $n \geq 2$.

This finishes the proof sketch that $\mathsf{K\text{-}loop}(G, n) : \Omega K(G, n+1) \simeq K(G, n)$. By induction, $\Omega^n K(G, n+1) \simeq K(G, 1)$, hence we get the following group isomorphism $\pi_{n+1}(K(G, n+1)) \simeq \pi_1(K(G, 1)) \simeq G$.

4.2.2 Uniqueness

In this section we prove that Eilenberg-MacLane spaces are unique, which means that if X and Y are both $(n-1)$-connected, n-truncated pointed types such that $\pi_n(X) \simeq \pi_n(Y)$, then $X \simeq Y$. Note that from these assumptions one can show that $\pi_k(X) \simeq 1 \simeq \pi_k(Y)$ for $k < n$ since X and Y are $(n-1)$-connected, but also for $k > n$ since X and Y are n-truncated. Hence from the assumptions we actually have that $\pi_k(X) \simeq \pi_k(Y)$ for all natural numbers k.

This is similar to Whitehead's Theorem, which states that if $f : X \to Y$ is a pointed map that induces an equivalence on all homotopy groups, then f is an equivalence. Whitehead's Theorem is not true in general, but it is true under the assumption that both X and Y are n-truncated for some n. For the special case that X and Y are both $(n-1)$-connected and n-truncated one does not need to find a map between X and Y to show that they are equivalent, as long as they have isomorphic homotopy groups.

We first give an elimination principle for $K(G, n)$.

Definition 4.2.1. Suppose that X is an n-truncated pointed type, and suppose that for some group G there is an map $\varphi : G \to \Omega^n X$ that sends multiplication to concatenation. Then there is a pointed map $\mathsf{K\text{-}elim}(\varphi, n) : K(G, n) \to X$.

Construction. We construct this by induction on n. For $n = 1$ this follows directly from the induction principle of $K(G, 1)$. For $n = k + 1 > 1$ we can define the group homomorphism $\tilde{\varphi}$ as the composite $G \xrightarrow{\varphi} \Omega^{k+1} X \simeq \Omega^k(\Omega X)$, and apply the induction hypothesis to get a map $\mathsf{K\text{-}elim}(\tilde{\varphi}, k) : K(G, k) \to^* \Omega X$. By the adjunction $\Sigma \dashv \Omega$ we get a pointed map $\Sigma K(G, k) \to^* X$, and by the elimination principle of the truncation we get a map $K(G, k + 1) \equiv \|\Sigma K(G, k)\|_{k+1} \to^* X$. $\qquad \square$

Lemma 4.2.2. *There is a pointed homotopy making the following diagram commute.*

$$K(G,n) \xrightarrow{\quad \sim \quad} \Omega K(G, n+1)$$

$$\text{K-elim}(\widetilde{\varphi}, n) \searrow \qquad \swarrow \Omega(\text{K-elim}(\varphi, n+1))$$

$$\Omega X$$

Proof. This follows by unwinding the definition of the function K-elim$(\varphi, n+1)$ in terms of K-elim(φ, n). □

Lemma 4.2.3. *The following diagram commutes.*

$$\Omega^n K(G,n) \xrightarrow{\quad \sim \quad} G$$

$$\Omega^n(\text{K-elim}(\varphi, n)) \searrow \qquad \swarrow \varphi$$

$$\Omega^n X$$

Proof. This follows by repeatedly applying Lemma 4.2.2. □

Theorem 4.2.4. *Suppose that X is an $(n-1)$-connected n-truncated pointed type, and suppose that for some group G there is an equivalence $\varphi : G \simeq \Omega^n X$ that sends multiplication to concatenation. Then the map K-elim$(\varphi, n) : K(G,n) \to X$ is an equivalence. In particular this means that if X is an $(n-1)$-connected n-truncated pointed type, and there is a group isomorphism $e : \pi_n(X) \simeq G$, then $X \simeq^* K(G,n)$.*

Proof. We apply Whitehead's principle for truncated types. This states that a *weak equivalence* (a map inducing an isomorphism on all homotopy groups) between truncated types is an equivalence. The proof can be found in [Uni13, Theorem 8.8.3]. Since both $K(G,n)$ and X are $(n-1)$-connected and n-truncated, the map K-elim(φ, n) trivially induces an isomorphism on all homotopy groups for all levels other than n. It also induces an isomorphism on level n by Lemma 4.2.3. This finishes the proof. □

Corollary 4.2.5. *The type of $(n-1)$-connected, n-truncated pointed types is equivalent to the type of groups for $n = 1$ and equivalent to the type of abelian groups for $n \geq 2$.*

Proof. The maps back and forth are $K(-, n)$ and π_n. The composites are homotopic to the identity map, since $\pi_n(K(G,n)) \simeq G$ and $K(\pi_n(X), n) \simeq^* X$ (the last equivalence comes from Theorem 4.2.4). □

4.2.3 Equivalence of categories

Definition 4.2.6. If $\varphi : G \to H$ is a homomorphism between groups, then there is a pointed map $K(\varphi, n) : K(G,n) \to K(H,n)$. This action is functorial, i.e. it respects composition and identity maps.

Construction. The functorial action comes from Definition 4.2.1. We omit the proof of the other properties. □

To show that we get the desired equivalence of categories, we need to fill the following naturality squares. We will omit the proofs here.

$$
\begin{array}{ccc}
\pi_n(K(G,n)) \xrightarrow{\pi_n(K(\varphi,n))} \pi_n(K(H,n)) & \qquad & X \xrightarrow{\quad f \quad} Y \\
\Big\uparrow{\sim} \qquad \qquad \Big\uparrow{\sim} & & \Big\uparrow{\sim} \qquad \qquad \Big\uparrow{\sim} \\
G \xrightarrow{\quad \varphi \quad} H & & K(\pi_n(X),n) \xrightarrow{K(\pi_n(f),n)} K(\pi_n(Y),n)
\end{array}
$$

These diagrams show the following result.

Theorem 4.2.7. $K(-,n)$ *is an equivalence from the category of $(n-1)$-connected n-truncated pointed types to the category of groups (for $n = 1$) or abelian groups (for $n \geq 2$).*

Remark 4.2.8. In particular this shows that the type of pointed maps between two $(n-1)$-connected n-truncated types is a set. This is a special case of the more general fact that the type of pointed maps from an n-connected type to a $(n+k+1)$-truncated type is k-truncated (for $n \geq -1$).

Remark 4.2.9. It would be interesting, but a lot more work, to do this one level up. In that case, it should be possible to show that crossed modules or 2-groups correspond to pointed connected 2-types. Furthermore, pointed $(n-2)$ connected n-types should correspond to braided 2-groups for $n = 3$ and to symmetric 2-groups for $n \geq 4$. A start of this project was given in [vR16].

4.3   The Smash Product

In this section we will discuss the smash product and its properties.[20] The smash product has many uses in homotopy theory. It can be used to define generalized homology theory (see Section 5.5) and it is used to define the cup product for cohomology [Bru16, Section 5.1].

The goal is to prove that the smash product defines a *1-coherent symmetric monoidal product on pointed types* [Bru16, Definition 4.1.1], which we repeat in Definition 4.3.3.

---

[20]The work in this section is joint work with Stefano Piceghello. Parts of this section are based on ideas from Robin Adams, Marc Bezem, Ulrik Buchholtz and Egbert Rijke.

Our proof strategy is to show that the smash product is left adjoint to pointed maps and then use a Yoneda-style argument to show that we get a 1-coherent symmetric monoidal product.

This proof is known in 1-category theory [EK66, Chapter 2, Theorem 5.3]. Suppose given a closed category[21] $\mathcal{C}$ with internal hom $[-, -] : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathcal{C}$. Moreover suppose that for every $A, B : \mathcal{C}$ the functor $[A, [B, -]] : \mathcal{C} \to \mathcal{C}$ is representable as a $\mathcal{C}$-enriched functor. This means that there is an object $A \otimes B : \mathcal{C}$ and a $\mathcal{C}$-enriched natural transformation $[A \otimes B, C] \cong [A, [B, C]]$. Then $\mathcal{C}$ is a monoidal closed category. We will spell out the precise formulation for pointed types in Definition 4.3.1, where we will call $\mathcal{U}^*$-enriched functors *pointed functors* and $\mathcal{U}^*$-enriched natural transformations *pointed natural transformations*.

In this section we will prove two main claims.

- We prove that $A \wedge B$ represents the functor $A \to^* B \to^* (-)$ on pointed types. In other words, that we have a natural equivalence

$$(A \wedge B \to^* C) \simeq^* (A \to^* B \to^* C).$$

- We prove that if we have a *pointed* natural equivalence

$$(A \wedge B \to^* C) \simeq^* (A \to^* B \to^* C),$$

  then the smash product forms a 1-coherent symmetric monoidal product on pointed types.

There is still a gap in this argument: we still need to show that the natural equivalence above is a pointed natural equivalence. We did not manage to do this, because of the high level of the path algebra involved, but we do not expect theoretical difficulties.

In this section, all types, maps, homotopies and equivalences are pointed, unless mentioned otherwise. We will denote pointed homotopies using equalities in diagrams. We will start with defining some categorical properties of pointed types. We will use the notation established in Section 2.2.5.

4.3.1 The Category of Pointed Types

Definition 4.3.1. Suppose we are given $F : \mathcal{U}^* \to \mathcal{U}^*$. We say that F is a *1-coherent functor* if

- F acts on pointed maps: given $f : A \to A'$, there is a pointed map $Ff : F(A) \to F(A')$;

- it respects identities: $F(\mathsf{id}_A) \sim \mathsf{id}_{FA}$;

---

[21]A *closed category* is a category with internal hom-objects. We can view pointed types as a higher closed category, where the internal hom-object is the type of pointed maps, pointed by the constant map.

- it respects composition: $F(f' \circ f) \sim Ff' \circ Ff$.

We will call a 1-coherent functor a *functor* for short.[22] We say that a functor F is a *pointed functor* if moreover $F\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ is the unit type (which is the zero object in pointed types). In this case we can show that $F(\mathbf{0}_{A,B}) = \mathbf{0}_{FA,FB}$, where $\mathbf{0}_{A,B}$ is the constant map.

Let F, G be functors of pointed types and suppose that θ is a family of pointed maps $(X : \mathcal{U}^*) \to F(X) \to G(X)$. We say that θ is a (1-coherent) *natural transformation* or *natural* if for every $f : A \to B$ there is a diagram:

$$
\begin{array}{ccc}
F(A) & \xrightarrow{F(f)} & F(B) \\
\theta_A \downarrow & & \downarrow \theta_B \\
G(A) & \xrightarrow[G(f)]{} & G(B)
\end{array}
$$

That is, a pointed homotopy

$$p_\theta(f) : \theta_B \circ F(f) \sim G(f) \circ \theta_A.$$

We say that θ is *pointed natural* if θ is natural and $p_\theta(\mathbf{0}) = (p_\theta)_0$, where

$$(p_\theta)_0 : G(\mathbf{0}) \circ \theta_A \sim \mathbf{0} \circ \theta_A \sim \mathbf{0} \sim \theta_B \circ \mathbf{0} \sim \theta_B \circ F(\mathbf{0})$$

is the canonical proof of the pointed homotopy $G(\mathbf{0}) \circ \theta_A \sim \theta_B \circ F(\mathbf{0})$.

For n-ary functions $F : \mathcal{U}^* \to \cdots \to \mathcal{U}^*$ we define functoriality similarly. We say that transformations between n-ary functors are natural if they are natural in all arguments.

Remark 4.3.2. We could define a notion of *weak naturality*, which is like naturality, but where the homotopy is not required to be pointed. However, this is generally ill-behaved. For example, if θ is weakly natural, neither $X \to \theta$ nor $\theta \to X$ needs to be weakly natural.

Definition 4.3.3. A 1-coherent symmetric monoidal product for pointed types is a binary operation $\otimes : \mathcal{U}^* \to \mathcal{U}^* \to \mathcal{U}^*$ that is functorial. Explicitly, this means that

- Given $f : A \to A'$ and $g : B \to B'$, there is a map $f \otimes g : A \otimes B \to A' \otimes B$.

- It respects identities: $\mathrm{id}_A \otimes \mathrm{id}_B \sim \mathrm{id}_{A \otimes B}$.

- It respects composition: $(f' \circ f) \otimes (g' \circ g) \sim (f' \otimes g') \circ (f \otimes g)$.

Furthermore, there is a pointed type I and natural equivalences

---

[22]While this is an abuse of terminology, it will not cause confusion in practice. Note that internally in the language of HoTT it is an open problem whether we can even formulate the type of fully coherent functors.

- $\alpha : (A \otimes B) \otimes C \simeq A \otimes (B \otimes C)$ (associativity of the smash product);

- $\lambda : I \otimes B \simeq B$ (left unitor for the smash product);

- $\gamma : A \otimes B \simeq B \otimes A$ (braiding for the smash product).

With pointed homotopies filling the following three diagrams.

We have a version of the Yoneda Lemma for pointed types.

Lemma 4.3.4 (Yoneda). *Let A, B be pointed types, and assume, for all pointed types X, a pointed equivalence $\varphi_X : (B \to X) \simeq (A \to X)$, natural in X, i.e. for all $f : X \to X'$ there is a homotopy*

$$p_\varphi(f) : (A \to f) \circ \varphi_X \sim \varphi'_X \circ (B \to f)$$

Then there exists a pointed equivalence $\psi_\varphi : A \simeq B$.

Proof. We define $\psi_\varphi :\equiv \varphi_B(\mathsf{id}_B) : A \to B$ and $\psi_\varphi^{-1} :\equiv \varphi_A^{-1}(\mathsf{id}_A)$. The given naturality square for $X :\equiv B$ and $g :\equiv \psi_\varphi^{-1}$ yields $\psi_\varphi^{-1} \circ \varphi_B(\mathsf{id}_B) \equiv \psi_\varphi^{-1} \circ \psi_\varphi \sim \varphi_A(\psi_\varphi^{-1} \circ \mathsf{id}_B) \equiv \varphi_A(\varphi_A^{-1}(\mathsf{id}_A)) \sim \mathsf{id}_A$, and similarly for the inverse composition. $\qquad\square$

Lemma 4.3.5. *Assume A, B, φ_X and p as in Lemma 4.3.4, and assume moreover that φ_X is pointed natural. Then there is a pointed homotopy $(\psi_\varphi \to X) \sim \varphi_X$.*

Proof. Let $f : B \to X$. The underlying homotopy is obtained by:

$$(\psi_\varphi \to X)(f) \equiv f \circ \psi_\varphi$$
$$\sim \varphi_X(f \circ \mathsf{id}) \qquad\qquad (\text{by } p_\varphi(f)(\mathsf{id}))$$
$$\sim \varphi_X(f) \qquad\qquad (\text{by } \mathsf{ap}_{\varphi_X}(\mathsf{u}_f))$$

To show that this is a pointed homotopy, we need to prove that the following diagram commutes:

$$(\psi_\varphi \to X)(\mathbf{0}) \xrightarrow{\; p_\varphi(\mathbf{0})(\mathsf{id}) \cdot \mathsf{ap}_{\varphi_X}(\mathsf{u_0}) \;} \varphi_X(\mathbf{0})$$

with $\mathsf{z}_{\psi_\varphi}$ and $(\varphi_X)_0$ paths into $\mathbf{0}$.

where the top-left expression is definitionally equal to $\mathbf{0} \circ \varphi_X(\mathsf{id})$, the horizontal path comes from the underlying homotopy and $(\varphi_X)_0$ is the canonical path from $\varphi_X(\mathbf{0})$ to $\mathbf{0}$. Since φ_X is pointed natural, we have that $p_{\varphi_X}(\mathbf{0})(\mathsf{id}) = (p_{\varphi_X})_0(\mathsf{id})$, which is the concatenation:

$$\mathbf{0} \circ \varphi_X(\mathsf{id}) = \mathbf{0} \qquad\qquad (\text{by } \mathsf{z}_{q_X(\mathsf{id})})$$
$$= \varphi_X(\mathbf{0}) \qquad\qquad (\text{by } (\varphi_X)_0^{-1})$$
$$= \varphi_X(\mathbf{0} \circ 1) \qquad\qquad (\text{by } (\mathsf{ap}_{\varphi_X}(\mathsf{z}_{\mathsf{id}}))^{-1})$$

The diagram then commutes by cancellation of inverses and using that $\mathsf{z}_{\mathsf{id}} = \mathsf{u_0}$. $\qquad \square$

4.3.2 Basic Properties of the Smash Product

Definition 4.3.6. The smash of A and B is the HIT generated by the point constructor (a, b) for $a : A$ and $b : B$ and two auxiliary points $\mathsf{auxl}, \mathsf{auxr} : A \wedge B$ and path constructors $\mathsf{gluel}_a : (a, b_0) = \mathsf{auxl}$ and $\mathsf{gluer}_b : (a_0, b) = \mathsf{auxr}$ (for $a : A$ and $b : B$). $A \wedge B$ is pointed with point (a_0, b_0).

Remark 4.3.7. This definition of $A \wedge B$ is basically the pushout of $\mathbf{2} \leftarrow A + B \to A \times B$. A more traditional definition of $A \wedge B$ is the pushout $\mathbf{1} \leftarrow A \vee B \to A \times B$; here $\vee$ denotes the wedge product, which can be equivalently described as either the pushout $A \leftarrow \mathbf{1} \to B$ or $\mathbf{1} \leftarrow \mathbf{2} \to A + B$. These two definitions of $A \wedge B$ are equivalent, because in the following diagram the top-left square and the top rectangle are pushout squares, hence the top-right square is a pushout square by applying the pushout lemma. Another application of the pushout lemma then states that the two definitions of $A \wedge B$ are equivalent.

$$
\begin{array}{ccccc}
\mathbf{2} & \longrightarrow & A + B & \longrightarrow & \mathbf{2} \\
\downarrow & & \downarrow & & \downarrow \\
\mathbf{1} & \longrightarrow & A \vee B & \longrightarrow & \mathbf{1} \\
& & \downarrow & & \downarrow \\
& & A \times B & \longrightarrow & A \wedge B
\end{array}
$$

Lemma 4.3.8. *The smash product is functorial: if $f : A \to A'$ and $g : B \to B'$, then $f \wedge g : A \wedge B \to A' \wedge B'$. We write $A \wedge g$ or $f \wedge B$ if one of the functions is the identity function. Moreover, if $p : f \sim f'$ and $q : g \sim g'$, then $p \wedge q : f \wedge g \sim f' \wedge g'$; this operation preserves reflexivities, symmetries and transitivies. We will write $p \wedge g$ or $f \wedge q$ if one of the homotopies is reflexivity.*

Lemma 4.3.9. *The smash product preserves composition, which gives rise to the interchange law:*
$$i : (f_2 \circ f_1) \wedge (g_2 \circ g_1) \sim f_2 \wedge g_2 \circ f_1 \wedge g_1$$
for maps $A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} A_3$ and $B_1 \xrightarrow{g_1} B_2 \xrightarrow{g_2} B_3$.

Proof. Let us denote the basepoints of A_i and B_i with a_i and b_i respectively. We first apply induction on the paths that all the maps in the statement respect the basepoint. We verify the underlying homotopy of i by induction on terms x of the domain $A_1 \wedge B_1$ of the two maps; this can be defined on point constructors (a, b), auxl and auxr to be the identity path. If x varies over gluel_a, we need to fill the following square:

$$
\begin{array}{ccc}
(f_2(f_1(a)), b_3) & \overset{1}{=\!=\!=} & (f_2(f_1(a)), b_3) \\
{\scriptstyle \mathsf{ap}_{(f_2 \circ f_1) \wedge (g_2 \circ g_1)}(\mathsf{gluel}_a)} \Big\| & & \Big\| {\scriptstyle \mathsf{ap}_{f_2 \wedge g_2 \circ f_1 \wedge g_1}(\mathsf{gluel}_a)} \\
\mathsf{auxl} & \underset{1}{=\!=\!=\!=\!=} & \mathsf{auxl}
\end{array}
\qquad (4.3.10)
$$

This reduces to proving that

$$\mathsf{ap}_{(f_2(f_1(a)),-)}(g_2 \circ g_1)_0 \cdot \mathsf{gluel}_{f_2(f_1(a))} = \mathsf{ap}_{(f_2(f_1(a)),-)}(\mathsf{ap}_{g_2}(g_1)_0 \cdot (g_2)_0) \cdot \mathsf{gluel}_{f_2(f_1(a))}$$

Since we assumed that $(g_1)_0$ and $(g_2)_0$ are the identity path, the claim is easily verified. The case for x varying over gluer_b is entirely analogous, giving the square:

$$
\begin{array}{ccc}
(a_3, g_2(g_1(b))) & \overset{1}{=\!=\!=} & (a_3, g_2(g_1(b))) \\
{\scriptstyle \mathsf{ap}_{(f_2 \circ f_1) \wedge (g_2 \circ g_1)}(\mathsf{gluer}_b)} \Big\| & & \Big\| {\scriptstyle \mathsf{ap}_{f_2 \wedge g_2 \circ f_1 \wedge g_1}(\mathsf{gluer}_b)} \\
\mathsf{auxr} & \underset{1}{=\!=\!=\!=\!=} & \mathsf{auxr}
\end{array}
\qquad (4.3.11)
$$

The resulting homotopy is pointed, as $i(a_1, b_1) \equiv 1$ and the proofs that the two maps respect the basepoint are assumed to be the identity path. $\qquad \square$

Lemma 4.3.12. *There are homotopies*
$$t_g : \mathbf{0} \wedge g \sim \mathbf{0} \qquad\qquad t'_f : f \wedge \mathbf{0} \sim \mathbf{0}$$
such that the following diagrams commute for given homotopies $p : g \sim g'$ and $q : f \sim f'$.

$$
\begin{array}{ccc}
\mathbf{0} \wedge g & \overset{1 \wedge p}{=\!=\!=\!=} & \mathbf{0} \wedge g' \\
& \underset{t_g \searrow \quad \swarrow t_{g'}}{} & \\
& \mathbf{0} &
\end{array}
\qquad\qquad
\begin{array}{ccc}
f \wedge \mathbf{0} & \overset{q \wedge 1}{=\!=\!=\!=} & f' \wedge \mathbf{0} \\
& \underset{t'_f \searrow \quad \swarrow t'_{f'}}{} & \\
& \mathbf{0} &
\end{array}
\qquad (4.3.13)
$$

Proof. We will define the homotopy $t_g : \mathbf{0} \wedge g$, with $\mathbf{0} : A_1 \to A_2$ and $g : B_1 \to B_2$ (with the notational convention for the basepoints as in Lemma 4.3.9); the definition for t'_f is analogous. First, we apply induction on the path that g respects the basepoint. The underlying homotopy of t_g is given by induction on terms $x : A_1 \wedge B_1$. On point constructors, we define:

$$
\begin{aligned}
t_g(a, b) &:\equiv \mathsf{gluer}_{g(b)} \cdot \mathsf{gluer}_{b_2}^{-1} && : (a_2, g(b)) = (a_2, b_2) \\
t_g(\mathsf{auxl}) &:\equiv \mathsf{gluel}_{a_2}^{-1} && : \mathsf{auxl} = (a_2, b_2) \\
t_g(\mathsf{auxr}) &:\equiv \mathsf{gluer}_{b_2}^{-1} && : \mathsf{auxr} = (a_2, b_2)
\end{aligned}
$$

If x varies over gluel_a, after some reductions, we need to fill the following square:

$$
\begin{array}{ccc}
(a_2, g(b_1)) & \overset{\mathsf{gluer}_{b_2} \cdot \mathsf{gluer}_{b_2}^{-1}}{=\!=\!=\!=\!=\!=\!=} & (a_2, b_2) \\
{\scriptstyle \mathsf{gluel}_{a_2}} \| & & \| {\scriptstyle 1} \\
\mathsf{auxl} & \underset{\mathsf{gluel}_{a_2}^{-1}}{=\!=\!=\!=\!=} & (a_2, b_2)
\end{array}
\tag{4.3.14}
$$

Similarly, if x varies over gluer_b, we need to fill the following square:

$$
\begin{array}{ccc}
(a_2, g(b)) & \overset{\mathsf{gluer}_{g(b)} \cdot \mathsf{gluer}_{b_2}^{-1}}{=\!=\!=\!=\!=\!=\!=} & (a_2, b_2) \\
{\scriptstyle \mathsf{gluer}_{g(b)}} \| & & \| {\scriptstyle 1} \\
\mathsf{auxr} & \underset{\mathsf{gluer}_{b_2}^{-1}}{=\!=\!=\!=\!=} & (a_2, b_2)
\end{array}
\tag{4.3.15}
$$

The squares in (4.3.14) and (4.3.15) can both be filled by simple path algebra. The resulting homotopy is pointed, as $t_g(a_1, b_1)$ is equal to the identity path and the proof that g respects the basepoint is also assumed to be the identity path. Finally, for $p : g \sim g'$, the diagram on the left in (4.3.13) commutes by induction on p. $\qquad\square$

Lemma 4.3.16. *Suppose that we have maps* $A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} A_3$ *and* $B_1 \xrightarrow{g_1} B_2 \xrightarrow{g_2} B_3$ *and suppose that either* f_1 *or* f_2 *is constant. Then there are two homotopies* $(f_2 \circ f_1) \wedge (g_2 \circ g_1) \sim \mathbf{0}$, *one of which uses the interchange law and one that does not. These two homotopies are equal. Specifically, the following two diagrams commute:*

$$
\begin{array}{ccc}
(f_2 \circ \mathbf{0}) \wedge (g_2 \circ g_1) & \overset{i}{=\!=\!=\!=} & (f_2 \wedge g_2) \circ (\mathbf{0} \wedge g_1) \\
\| & & \| {\scriptstyle (f_2 \wedge g_2) \circ t_{g_1}} \\
{\scriptstyle z' \wedge (g_2 \circ g_1)} \| & & (f_2 \wedge g_2) \circ \mathbf{0} \\
\| & & \| {\scriptstyle z'} \\
\mathbf{0} \wedge (g_2 \circ g_1) & \underset{t_{g_2 \circ g_1}}{=\!=\!=\!=\!=} & \mathbf{0}
\end{array}
$$

$$(\mathbf{0} \circ f_1) \wedge (g_2 \circ g_1) \overset{i}{=\!=\!=} (\mathbf{0} \wedge g_2) \circ (f_1 \wedge g_1)$$

$$\Big\| z \wedge (g_2 \circ g_1) \qquad\qquad \Big\| t_{g_2 \circ (f_1 \wedge g_1)}$$

$$\mathbf{0} \circ (f_1 \wedge g_1)$$

$$\Big\| z$$

$$\mathbf{0} \wedge (g_2 \circ g_1) \underset{t_{g_2 \circ g_1}}{=\!=\!=\!=\!=\!=\!=} \mathbf{0}$$

Proof. We start by filling the diagram on the left. First apply induction on the paths that f_2, g_1 and g_2 respect the basepoint. In this case $f_2 \circ \mathbf{0}$ is definitionally equal to $\mathbf{0}$, and the canonical proof that $f_2 \circ \mathbf{0} \sim \mathbf{0}$ is (definitionally) equal to reflexivity. This means that the homotopy $(f_2 \circ \mathbf{0}) \wedge (g_2 \circ g_1) \sim \mathbf{0} \wedge (g_2 \circ g_1)$ is also equal to reflexivity, and also the path that $f_2 \wedge g_2$ respects the basepoint is reflexivity, hence the homotopy $(f_2 \wedge g_2) \circ \mathbf{0} \sim \mathbf{0}$ is also reflexivity. This means we need to fill the following square:

$$(f_2 \circ \mathbf{0}) \wedge (g_2 \circ g_1) \overset{i}{=\!=\!=} (f_2 \wedge g_2) \circ (\mathbf{0} \wedge g_1)$$

$$\Big\| 1 \qquad\qquad\qquad \Big\| (f_2 \wedge g_2) \circ t_{g_1}$$

$$\mathbf{0} \wedge (g_2 \circ g_1) \underset{t_{g_1 \circ g_2}}{=\!=\!=\!=\!=\!=} \mathbf{0}$$

For the underlying homotopy, take $x : A_1 \wedge B_1$ and apply induction on x. Suppose $x \equiv (a, b)$ for $a : A_1$ and $b : B_1$. With the notational convention for basepoints as in Lemma 4.3.9, we have to fill the square (we use that the paths that the maps respect the basepoints are reflexivity):

$$(a_3, g_2(g_1(b))) \overset{1}{=\!=\!=\!=\!=} (a_3, g_2(g_1(b)))$$

$$\Big\| 1 \qquad\qquad\qquad \Big\| \mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluer}_{g_1(b)} \cdot \mathsf{gluer}_{b_2}^{-1}) \qquad\qquad (4.3.17)$$

$$(a_3, g_2(g_1(b))) \underset{\mathsf{gluer}_{g_2(g_1(b))} \cdot \mathsf{gluer}_{b_3}^{-1}}{=\!=\!=\!=\!=\!=\!=\!=} (a_3, b_3)$$

Now $\mathsf{ap}_{h \wedge k}(\mathsf{gluer}_z) = \mathsf{gluer}_{k(z)}$, so by general groupoid laws we see that the path on the bottom is equal to the path on the right, which means we can fill the square. For the other point constructors, the squares to fill are similar. If $x \equiv \mathsf{auxl}$, we have:

$$\mathsf{auxl} \overset{1}{=\!=\!=\!=\!=} \mathsf{auxl}$$

$$\Big\| 1 \qquad\qquad \Big\| \mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluel}_{a_2}^{-1}) \qquad\qquad (4.3.18)$$

$$\mathsf{auxl} \underset{\mathsf{gluel}_{a_3}^{-1}}{=\!=\!=\!=\!=} (a_3, b_3)$$

We can fill this square, as the path on the bottom is definitionally equal to $\mathsf{gluel}_{a_3}^{-1}$ (as we applied path induction on the path that f_2 respects the basepoint) and the path on

the right also reduces to $\mathsf{gluel}_{a_3}^{-1}$ using that $\mathsf{ap}_{h \wedge k}(\mathsf{gluel}_z) = \mathsf{gluel}_{h(z)}$. Similarly, we can fill the square for $x \equiv \mathsf{auxr}$, which is:

$$
\begin{array}{ccc}
\mathsf{auxr} & \xlongequal{\quad 1 \quad} & \mathsf{auxr} \\
{\scriptstyle 1}\Vert & & \Vert{\scriptstyle \mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluer}_{b_2}^{-1})} \\
\mathsf{auxr} & \xlongequal[\mathsf{gluer}_{b_3}^{-1}]{} & (a_3, b_3)
\end{array}
\tag{4.3.19}
$$

If x varies over gluel_a, after some reductions, we need to fill the following cube, where the front and the back are the squares in (4.3.17) for (a, b_1) and (4.3.18) respectively; the left square is degenerate; the other three sides are the squares in the definition of i and t to show that they respect gluel_a (given in (4.3.10) and (4.3.14) respectively), where we also apply $f_2 \wedge g_2$ to the square on the right. We suppress in the diagram the arguments of gluer in $\mathsf{gluer} \cdot \mathsf{gluer}^{-1}$ (which match, so the concatenation results equal to the identity path).

$$\tag{4.3.20}$$

Similarly, if x varies over gluer_b, we need to fill the cube below: the front and the back are the squares in (4.3.17) for (a_1, b) and (4.3.19) respectively; the left square is again degenerate; the other three sides come from the fact that i and t respect gluer_b (given in (4.3.11) and (4.3.15) respectively). Again, we omit the arguments of gluer in $\mathsf{gluer} \cdot \mathsf{gluer}^{-1}$ (in this case, not a priori judgmentally equal).

$$\tag{4.3.21}$$

In order to fill the cubes in (4.3.20) and (4.3.21), we generalize the paths and fill the cubes by path induction. The cube in (4.3.20) can be generalized to a cube:

for X and X' pointed types; a map $h : X \to X'$; terms $x, y\ z : X$; paths $p_l : x = y$, $p_r : x = z$, $q_l : h(x) = h(y)$, $q_r : h(x) = h(z)$; and 2-paths $s_l : \mathsf{ap}_h(p_l) = q_l$ (for the back and the top) and $s_r : \mathsf{ap}_h(p_r) = q_r$ (for the right side). This cube is filled by path induction on s_l, s_r, p_l and p_r. The cube in (4.3.21) can be generalized to a similar cube:

for paths $p_l : x = y$, $p_b : y = z$, $q_l : h(x) = h(y)$, $q_b : h(y) = h(z)$ and for 2-paths $s_l : \mathsf{ap}_h(p_l) = q_l$ (for the top) and $s_b : \mathsf{ap}_h(p_b) = q_b$ (for the back).

The diagram on the right is similar to the previous case. It is not hard to show that these homotopies are pointed.

□

Theorem 4.3.22. *Given pointed types A, B and C, the functorial action of the smash product induces a map*

$$(-) \wedge C : (A \to B) \to (A \wedge C \to B \wedge C)$$

that is natural in A and B and dinatural in C.

The naturality and dinaturality means that the following squares commute for $f : A' \to A\ g : B \to B'$ and $h : C \to C'$.

$$
\begin{array}{ccc}
(A \to B) & \xrightarrow{\;\;(-)\wedge C\;\;} & (A \wedge C \to B \wedge C) \\
\Big\downarrow{\scriptstyle f \to B} & & \Big\downarrow{\scriptstyle f\wedge C \to B\wedge C} \\
(A' \to B) & \xrightarrow{\;\;(-)\wedge C\;\;} & (A' \wedge C \to B \wedge C)
\end{array}
$$

$$
\begin{array}{ccc}
(A \to B) & \xrightarrow{\;\;(-)\wedge C\;\;} & (A \wedge C \to B \wedge C) \\
\Big\downarrow{\scriptstyle A \to g} & & \Big\downarrow{\scriptstyle A\wedge C \to g\wedge C} \\
(A \to B') & \xrightarrow{\;\;(-)\wedge C\;\;} & (A \wedge C \to B' \wedge C)
\end{array}
$$

$$
\begin{array}{ccc}
(A \to B) & \xrightarrow{\;\;(-)\wedge C\;\;} & (A \wedge C \to B \wedge C) \\
\Big\downarrow{\scriptstyle (-)\wedge C'} & & \Big\downarrow{\scriptstyle A\wedge C \to B\wedge h} \\
(A \wedge C' \to B \wedge C') & \xrightarrow{\;A\wedge h \to B\wedge C'\;} & (A \wedge C \to B \wedge C')
\end{array}
$$

Proof. First note that $\lambda f.\, f \wedge C$ preserves the basepoint so that the map is indeed pointed.

Let $k : A \to B$. Then as homotopy the naturality in A becomes $(k \circ f) \wedge C = k \wedge C \circ f \wedge C$. To prove an equality between pointed maps, we need to give a pointed homotopy, which is given by interchange. To show that this homotopy is pointed, we need to fill the following square (after reducing out the applications of function extensionality), which follows from Lemma 4.3.16.

$$
\begin{array}{ccc}
(\mathbf{0} \circ f) \wedge C & =\!\!=\!\!= & (\mathbf{0} \wedge C) \circ (f \wedge C) \\
\| & & \| \\
 & & \mathbf{0} \circ (f \wedge C) \\
\| & & \| \\
\mathbf{0} \wedge C & =\!\!=\!\!=\!\!=\!\!=\!\!= & \mathbf{0}
\end{array}
$$

The naturality in B is almost the same: for the underlying homotopy we need to show $i : (g \circ k) \wedge C = g \wedge C \circ k \wedge C$. For the pointedness we need to fill the following square, which follows from the left pentagon in Lemma 4.3.16.

$$
\begin{array}{ccc}
(g \circ \mathbf{0}) \wedge C & =\!\!=\!\!= & (g \wedge C) \circ (\mathbf{0} \wedge C) \\
\| & & \| \\
 & & (g \wedge C) \circ \mathbf{0} \\
\| & & \| \\
\mathbf{0} \wedge C & =\!\!=\!\!=\!\!=\!\!=\!\!= & \mathbf{0}
\end{array}
$$

The dinaturality in C is a bit harder. For the underlying homotopy we need to show $B \wedge h \circ k \wedge C = k \wedge C' \circ A \wedge h$. This follows from applying interchange twice:

$$
B \wedge h \circ k \wedge C \sim (\mathsf{id}_B \circ k) \wedge (h \circ \mathsf{id}_C) \sim (k \circ \mathsf{id}_A) \wedge (\mathsf{id}_{C'} \circ h) \sim k \wedge C' \circ A \wedge h.
$$

89

To show that this homotopy is pointed, we need to fill the following square:

$$B \wedge h \circ \mathbf{0} \wedge C \;=\!=\; (\mathsf{id}_B \circ \mathbf{0}) \wedge (h \circ \mathsf{id}_C) \;=\!=\; (\mathbf{0} \circ \mathsf{id}_A) \wedge (\mathsf{id}_{C'} \circ h) \;=\!=\; \mathbf{0} \wedge C' \circ A \wedge h$$

with the vertical identities, middle row

$$B \wedge h \circ \mathbf{0} \qquad \mathbf{0} \wedge (h \circ \mathsf{id}_C) \;=\!=\; \mathbf{0} \wedge (\mathsf{id}_{C'} \circ h) \qquad \mathbf{0} \circ A \wedge h$$

and bottom row

$$B \wedge h \circ \mathbf{0} \;=\!=\; \mathbf{0} \;=\!=\; \mathbf{0} \;=\!=\; \mathbf{0}$$

The left and the right squares are filled by Lemma 4.3.16. The squares in the middle are filled by (corollaries of) Lemma 4.3.8. $\qquad\square$

4.3.3 Adjunction

Lemma 4.3.23. *There is a unit $\eta_{A,B} \equiv \eta : A \to B \to A \wedge B$ natural in A and counit $\varepsilon_{B,C} \equiv \varepsilon : (B \to C) \wedge B \to C$ dinatural in B and natural in C. These maps satisfy the unit-counit laws:*

$$(A \to \varepsilon_{A,B}) \circ \eta_{A \to B, A} \sim \mathsf{id}_{A \to B} \qquad \varepsilon_{B, B \wedge C} \circ \eta_{A,B} \wedge B \sim \mathsf{id}_{A \wedge B}.$$

Note: η is also dinatural in B, but we do not need this.

Proof. We define $\eta a b = (a, b)$. We define the path that ηa respects the basepoint as

$$(\eta a)_0 :\equiv \mathsf{gluel}_a \cdot \mathsf{gluel}_{a_0}^{-1} : (a, b_0) = (a_0, b_0).$$

Also, η itself respects the basepoint. To show this, we need to give $\eta_0 : \eta(a_0) \sim \mathbf{0}$. The underlying maps are homotopic, by

$$\eta_0 b :\equiv \mathsf{gluer}_b \cdot \mathsf{gluer}_{b_0}^{-1} : (a_0, b) = (a_0, b_0).$$

To show that this homotopy is pointed, we need to show that the two given proofs of $(a_0, b_0) = (a_0, b_0)$ are equal, but they are both equal to reflexivity:

$$\eta_{00} : \mathsf{gluel}_{a_0} \cdot \mathsf{gluel}_{a_0}^{-1} = 1 = \mathsf{gluer}_{b_0} \cdot \mathsf{gluer}_{b_0}^{-1}.$$

This defines the unit. To show that it is natural in A, we need to give the following pointed homotopy $p_\eta(f)$ for $f : A \to A'$.

$$
\begin{array}{ccc}
A & \xrightarrow{\ \eta\ } & (B \to A \wedge B) \\
\downarrow{\scriptstyle f} & & \downarrow{\scriptstyle B \to f \wedge B} \\
A' & \xrightarrow{\ \eta\ } & (B \to A' \wedge B)
\end{array}
$$

90

We may assume that f_0 is reflexivity. For the underlying homotopy we need to define for $a : A$ that $p_\eta(f, a) : \eta(fa) \sim f \wedge B \circ \eta a$, which is another pointed homotopy. For $b : B$ we have $\eta(fa, b) \equiv (fa, b) \equiv (f \wedge B)(\eta ab)$. The homotopy $p_\eta(f, a)$ is pointed, since

$$(f \wedge B \circ \eta a)_0 = \mathsf{ap}_{f \wedge B}(\mathsf{gluel}_a \cdot \mathsf{gluel}_{a_0}^{-1}) = \mathsf{gluel}_{fa} \cdot \mathsf{gluel}_{a_0'}^{-1} = (\eta(fa))_0.$$

Now we need to show that $p_\eta(f)$ is pointed, for which we need to fill the following diagram.

$$
\begin{array}{ccc}
\eta(fa_0) & \xrightarrow{\;\;p_\eta(f, a_0)\;\;} & f \wedge B \circ \eta a_0 \\
& \eta_0 \qquad\qquad f \wedge B \circ \eta_0 & \\
& \mathbf{0}_{B, A' \wedge B} &
\end{array}
$$

These pointed homotopies have equal underlying homotopies, since for $b : B$ we have

$$p_\eta(f, a_0, b) \cdot \mathsf{ap}_{f \wedge B}(\eta_0 b) = 1 \cdot \mathsf{ap}_{f \wedge B}(\mathsf{gluer}_b \cdot \mathsf{gluer}_{b_0}^{-1}) = \mathsf{gluer}_b \cdot \mathsf{gluer}_{b_0}^{-1} = \eta_0 b.$$

We will skip the proof that these homotopies respect the point in the same way.

To define the counit, given $x : (B \to C) \wedge B$, we construct $\varepsilon(x) : C$ by induction on x. If $x \equiv (f, b)$, we set $\varepsilon(f, b) :\equiv f(b)$. If x is either auxl or auxr, then we set $\varepsilon(x) :\equiv c_0 : C$. If x varies over gluel_f, then we need to show that $f(b_0) = c_0$, which is true by f_0. If x varies over gluer_b, we need to show that $\mathbf{0}(b) = c_0$ which is true by reflexivity. Now $\varepsilon_0 :\equiv 1 : \varepsilon(\mathbf{0}_{B,C}, b_0) = c_0$ shows that ε is pointed.

We will skip the proof that the counit is dinatural in B and natural in C.

Finally, we need to show the unit-counit laws. For the underlying homotopy of the first one, let $f : A \to B$. We need to show that $p_f : \varepsilon \circ \eta f \sim f$. We define $p_f(a) = 1 : \varepsilon(f, a) = f(a)$. To show that p_f is a pointed homotopy, we need to show that $p_f(a_0) \cdot f_0 = \mathsf{ap}_\varepsilon(\eta f)_0 \cdot \varepsilon_0$, which reduces to $f_0 = \mathsf{ap}_\varepsilon(\mathsf{gluel}_f \cdot \mathsf{gluel}_0^{-1})$, but we can reduce the right hand side: (note: $\mathbf{0}_0$ denotes the proof that $\mathbf{0}(a_0) = b_0$, which is reflexivity)

$$\mathsf{ap}_\varepsilon(\mathsf{gluel}_f \cdot \mathsf{gluel}_0^{-1}) = \mathsf{ap}_\varepsilon(\mathsf{gluel}_f) \cdot (\mathsf{ap}_\varepsilon(\mathsf{gluel}_0))^{-1} = f_0 \cdot \mathbf{0}_0^{-1} = f_0.$$

Now we need to show that p itself respects the basepoint of $A \to B$, i.e. that the composite $\varepsilon \circ \eta(\mathbf{0}) \sim \varepsilon \circ \mathbf{0} \sim \mathbf{0}$ is equal to $p_{\mathbf{0}_{A,B}}$. The underlying homotopies are the same for $a : A$; on the one side we have $\mathsf{ap}_\varepsilon(\mathsf{gluer}_a \cdot \mathsf{gluer}_{a_0}^{-1})$ and on the other side we have reflexivity (note: this type checks since $\mathbf{0}_{A,B}a \equiv \mathbf{0}_{A,B}a_0$). These paths are equal, since

$$\mathsf{ap}_\varepsilon(\mathsf{gluer}_a \cdot \mathsf{gluer}_{a_0}^{-1}) = \mathsf{ap}_\varepsilon(\mathsf{gluer}_a) \cdot (\mathsf{ap}_\varepsilon(\mathsf{gluer}_{a_0}))^{-1} = 1 \cdot 1^{-1} \equiv 1.$$

Both pointed homotopies are pointed in the same way, which requires some path-algebra, and we skip the proof here.

For the underlying homotopy of the second unit-counit law, we need to show for $x : A \wedge B$ that $q(x) : \varepsilon((\eta \wedge B)x) = x$, which we prove by induction to x. If $x \equiv (a, b)$,

then we can define $q(a, b) :\equiv 1_{(a,b)}$. If x is auxl or auxr, then the left-hand side reduces to (a_0, b_0), so we can define $q(\mathsf{auxl}) :\equiv \mathsf{gluel}_{a_0}$ and $q(\mathsf{auxr}) :\equiv \mathsf{gluer}_{b_0}$. The following computation shows that q respects gluel_a:

$$\mathsf{ap}_{\varepsilon \circ \eta \wedge B}(\mathsf{gluel}_a) \cdot \mathsf{gluel}_{a_0} = \mathsf{ap}_\varepsilon(\mathsf{gluel}_{\eta a}) \cdot \mathsf{gluel}_{a_0} = (\eta a)_0 \cdot \mathsf{gluel}_{a_0} = \mathsf{gluel}_a \cdot \mathsf{gluel}_{a_0}^{-1} \cdot \mathsf{gluel}_{a_0}$$
$$= \mathsf{gluel}_a \, .$$

To show that it respects gluer_b we compute

$$\mathsf{ap}_{\varepsilon \circ \eta \wedge B}(\mathsf{gluer}_b) \cdot \mathsf{gluer}_{b_0} = \mathsf{ap}_{\varepsilon(-,b)}(\eta_0) \cdot \mathsf{ap}_\varepsilon(\mathsf{gluer}_b) \cdot \mathsf{gluer}_{b_0} = \mathsf{ap}_{\lambda f. \, fb}(\eta_0) \cdot \mathsf{gluer}_{b_0}$$
$$= \eta_0 b \cdot \mathsf{gluer}_{b_0} = \mathsf{gluer}_b \, .$$

To show that q is a pointed homotopy, we need to show that $(\varepsilon \circ \eta \wedge B)_0 = 1$, For this we compute

$$(\varepsilon \circ \eta \wedge B)_0 = \mathsf{ap}_{\varepsilon(-,b_0)}(\eta_0) = \eta_0 b_0 = \mathsf{gluer}_{b_0} \cdot \mathsf{gluer}_{b_0}^{-1} = 1.$$

$\square$

Definition 4.3.24. The function $e \equiv e_{A,B,C} : (A \to B \to C) \to (A \wedge B \to C)$ is defined as the composite

$$(A \to B \to C) \xrightarrow{(-)\wedge B} (A \wedge B \to (B \to C) \wedge B) \xrightarrow{A \wedge B \to \varepsilon} (A \wedge B \to C).$$

Lemma 4.3.25. *The function e is invertible, hence gives a pointed equivalence*

$$(A \to B \to C) \simeq (A \wedge B \to C).$$

Proof. Define

$$e_{A,B,C}^{-1} : (A \wedge B \to C) \xrightarrow{B \to (-)} ((B \to A \wedge B) \to (B \to C)) \xrightarrow{\eta \to (B \to C)} (A \to B \to C).$$

It is easy to show that e and e^{-1} are inverses as unpointed maps from the unit-counit laws (Lemma 4.3.23) and naturality of η and ε. $\square$

Lemma 4.3.26. *The function e is natural in A, B and C.*

Proof. **Naturality of e in A.** Suppose that $f : A' \to A$. Then the following diagram commutes. The left square commutes by naturality of $(-) \wedge B$ in the first argument and the right square commutes because composition on the left commutes with composition on the right.

$$
\begin{array}{ccccc}
(A \to B \to C) & \xrightarrow{(-)\wedge B} & (A \wedge B \to (B \to C) \wedge B) & \xrightarrow{A \wedge B \to \varepsilon} & (A \wedge B \to C) \\
\downarrow{\scriptstyle f \to B \to C} & & \downarrow{\scriptstyle f \wedge B \to \cdots} & & \downarrow{\scriptstyle f \wedge B \to C} \\
(A' \to B \to C) & \xrightarrow{(-)\wedge B} & (A' \wedge B \to (B \to C) \wedge B) & \xrightarrow{A \wedge B \to \varepsilon} & (A' \wedge B \to C)
\end{array}
$$

92

Naturality of e in C. Suppose that $f : C \to C'$. Then in the following diagram the left square commutes by naturality of $(-) \wedge B$ in the second argument (applied to $B \to f$) and the right square commutes by applying the functor $A \wedge B \to (-)$ to the naturality of ε in the second argument.

$$
\begin{array}{ccccc}
(A \to B \to C) & \longrightarrow & (A \wedge B \to (B \to C) \wedge B) & \longrightarrow & (A \wedge B \to C) \\
\downarrow & & \downarrow & & \downarrow \\
(A \to B \to C') & \longrightarrow & (A \wedge B \to (B \to C') \wedge B) & \longrightarrow & (A \wedge B \to C')
\end{array}
$$

Naturality of e in B. Suppose that $f : B' \to B$. Here the diagram is a bit more complicated, since $(-) \wedge B$ is dinatural (instead of natural) in B. Then we get the following diagram. The front square commutes by naturality of $(-) \wedge B$ in the second argument (applied to $f \to C$). The top square commutes by naturality of $(-) \wedge B$ in the third argument, the back square commutes because composition on the left commutes with composition on the right, and finally the right square commutes by applying the functor $A \wedge B' \to (-)$ to the naturality of ε in the first argument.

$\square$

Remark 4.3.27. Instead of showing that e is natural, we could show that e^{-1} is natural. In that case we need to show that the map $A \to (-) : (B \to C) \to (A \to B) \to (A \to C)$ is natural in A, B and C. This might actually be easier, since we do not need to work with any higher inductive type to prove that.

We have now obtained the following theorem

Theorem 4.3.28. *There is an equivalence*

$$(A \to B \to C) \simeq (A \wedge B \to C)$$

natural in A, B and C.

Remark 4.3.29. We can state Theorem 4.3.28 as an adjunction $(-) \wedge B \dashv B \to (-)$ or by saying that $A \wedge B$ represents the functor $A \to B \to (-)$.

In Section 4.3.4 we show that the smash product forms a 1-coherent symmetric monoidal product from the assumption that this adjunction is pointed in C. Explicitly,

this means that the naturality of e in C applied to the map $\mathbf{0}_{C,C'} : C \to C'$ is equal to the composite

$$(A \wedge B \to \mathbf{0}_{C,C'}) \circ e_{A,B,C} \sim \mathbf{0} \circ e_{A,B,C} \sim \mathbf{0} \sim e_{A,B,C'} \circ \mathbf{0} \sim e_{A,B,C'} \circ (A \to B \to \mathbf{0}_{C,C'}).$$

To prove this, we need that the counit ε is pointed natural in C. To prove that, we need to show that the map $(-) \wedge C$, defined in Theorem 4.3.22, is pointed natural in B. In order to prove that, we need to show that in the situation of Lemma 4.3.16, if both f_1 and f_2 are (judgmentally) the constant map, then the two pentagons stated in that lemma are equal (transported appropriately in order to make this equality type check). This can be formulated as a 3-path in a type of pointed maps, which is hard to fill.

4.3.4 Symmetric monoidal product

In this section we will prove that the smash product is a 1-coherent symmetric monoidal product Definition 4.3.3, from the assumption that the adjunction from Section 4.3.3 is pointed natural in C. We will need to following pointed equivalences. Without the proof that e is pointed natural, parts of this section are still true. In particular, the natural equivalences defined in Definition 4.3.33 do not require pointed naturality of e.

Definition 4.3.30. We define the pointed equivalences:

$$\mathsf{b} : (\mathbb{S}^0 \to X) \simeq X$$

where $\mathbb{S}^0$ is the type of booleans (pointed in $0_{\mathbf{2}}$) with underlying map defined with $\mathsf{b}(f) :\equiv f(1_{\mathbf{2}})$, and

$$\mathsf{tw} : (A \to B \to X) \simeq (B \to A \to X)$$

with underlying map defined with $\mathsf{tw}(f) :\equiv \lambda b.\, \lambda a.\, f(a)(b)$.

Using Lemma 4.3.4 (Yoneda) we can prove associativity, left and right unitality and braiding equivalences for the smash product, in the following way.

Definition 4.3.31. The following pointed equivalences are defined for A, B, C and X pointed types:

- $\overline{\alpha}_X : (A \wedge (B \wedge C) \to X) \simeq ((A \wedge B) \wedge C \to X)$ as the composition of the equivalences:

$$
\begin{array}{ll}
A \wedge (B \wedge C) \to X \simeq A \to B \wedge C \to X & (e^{-1}) \\
\simeq A \to B \to C \to X & (A \to e^{-1}) \\
\simeq A \wedge B \to C \to X & (e) \\
\simeq (A \wedge B) \wedge C \to X. & (e)
\end{array}
$$

- $\overline{\lambda}_X : (B \to X) \simeq (\mathbb{S}^0 \wedge B \to X)$ as the composition of the equivalences:

$$
\begin{array}{ll}
B \to X \simeq \mathbb{S}^0 \to B \to X & (\mathsf{b}^{-1}) \\
\simeq \mathbb{S}^0 \wedge B \to X & (e)
\end{array}
$$

94

- $\bar{\rho}_X : (A \to X) \simeq (A \wedge \mathbb{S}^0 \to X)$ as the composition of the equivalences:

$$A \to X \simeq A \to \mathbb{S}^0 \to X \qquad\qquad (A \to \mathsf{b}^{-1})$$
$$\simeq A \wedge \mathbb{S}^0 \to X \qquad\qquad (e)$$

- $\bar{\gamma}_X : (B \wedge A \to X) \simeq (A \wedge B \to X)$ as the composition of the equivalences:

$$B \wedge A \to X \simeq B \to A \to X \qquad\qquad (e^{-1})$$
$$\simeq A \to B \to X \qquad\qquad (\mathsf{tw})$$
$$\simeq A \wedge B \to X \qquad\qquad (e)$$

Remark 4.3.32. The equivalences in Definition 4.3.31 are natural in all their arguments and from the assumption that e is pointed natural in C we can show that these maps are all pointed natural in X.

Definition 4.3.33. We define the following equivalences, natural in all their arguments, with inverses provided as in Lemma 4.3.4:

- $\alpha :\equiv \bar{\alpha}_{A \wedge (B \wedge C)}(\mathsf{id}) : (A \wedge B) \wedge C \simeq A \wedge (B \wedge C)$ (associativity of the smash product), with inverse $\alpha^{-1} :\equiv \bar{\alpha}^{-1}_{(A \wedge B) \wedge C}(\mathsf{id})$;

- $\lambda :\equiv \bar{\lambda}_B(\mathsf{id}) : \mathbb{S}^0 \wedge B \simeq B$ and $\rho :\equiv \bar{\rho}_A(\mathsf{id}) : A \wedge \mathbb{S}^0 \simeq A$ (left- and right unitors for the smash product), with inverses $\lambda^{-1} :\equiv \bar{\lambda}^{-1}_{\mathbb{S}^0 \wedge B}(\mathsf{id})$ and $\rho^{-1} :\equiv \bar{\rho}^{-1}_{A \wedge \mathbb{S}^0}(\mathsf{id})$, respectively;

- $\gamma :\equiv \bar{\gamma}_{B \wedge A}(\mathsf{id}) : A \wedge B \simeq B \wedge A$ (braiding for the smash product), with inverse $\gamma^{-1} :\equiv \bar{\gamma}^{-1}_{A \wedge B}(\mathsf{id})$.

α, λ, ρ and γ are natural in all their arguments, as $\bar{\alpha}$, $\bar{\lambda}$, $\bar{\rho}$ and $\bar{\gamma}$ are. Note that these definitions do *not* require pointed naturality of e.

Lemma 4.3.34. *There are pointed homotopies*

$$\bar{\alpha}_X \sim \alpha \to X \qquad\qquad \bar{\lambda}_X \sim \lambda \to X$$
$$\bar{\rho}_X \sim \rho \to X \qquad\qquad \bar{\gamma}_X \sim \gamma \to X$$

Proof. This follows directly from Lemma 4.3.5 and Remark 4.3.32 (this does require pointed naturality of e). $\qquad\square$

Theorem 4.3.35 (Associativity pentagon)**.** *For A, B, C and D pointed types, there is a homotopy*

$$\alpha \circ \alpha \sim (A \wedge \alpha) \circ \alpha \circ (\alpha \wedge D)$$

corresponding to the commutativity of the following diagram:

95

$$((A \wedge B) \wedge (C \wedge D))$$

$$(((A \wedge B) \wedge C) \wedge D) \qquad (A \wedge (B \wedge (C \wedge D)))$$

$$\alpha \wedge D \downarrow \qquad \qquad \uparrow A \wedge \alpha$$

$$((A \wedge (B \wedge C)) \wedge D) \xrightarrow{\quad\alpha\quad} (A \wedge ((B \wedge C) \wedge D))$$

Proof. We articulate the proof in several steps. A map homotopic to both sides of the sought homotopy will be constructed via the equivalence

$$\overline{\alpha}^4 : (A \wedge (B \wedge (C \wedge D)) \to X) \simeq (((A \wedge B) \wedge C) \wedge D \to X)$$

(natural in all its arguments), defined as the composite:

$$
\begin{aligned}
A \wedge (B \wedge (C \wedge D)) \to X &\simeq A \to B \wedge (C \wedge D) \to X & (e^{-1})\\
&\simeq A \to B \to C \wedge D \to X & (A \to e^{-1})\\
&\simeq A \to B \to C \to D \to X & (A \to B \to e^{-1})\\
&\simeq A \wedge B \to C \to D \to X & (e)\\
&\simeq (A \wedge B) \wedge C \to D \to X & (e)\\
&\simeq ((A \wedge B) \wedge C) \wedge D \to X & (e)
\end{aligned}
$$

giving $\overline{\alpha}^4(\mathsf{id}) : ((A \wedge B) \wedge C) \wedge D \simeq A \wedge (B \wedge (C \wedge D))$. Moreover, in order to simplify the expressions of $\alpha \wedge D$ and $A \wedge \alpha$, we also define:

$$\overline{\alpha}^R : ((A \wedge (B \wedge C)) \wedge D \to X) \simeq (((A \wedge B) \wedge C) \wedge D \to X)$$

as the composite:

$$
\begin{aligned}
(A \wedge (B \wedge C)) \wedge D \to X &\simeq A \wedge (B \wedge C) \to D \to X & (e^{-1})\\
&\simeq (A \wedge B) \wedge C \to D \to X & (\overline{\alpha})\\
&\simeq ((A \wedge B) \wedge C) \wedge D \to X & (e)
\end{aligned}
$$

and

$$\overline{\alpha}^L : (A \wedge (B \wedge (C \wedge D)) \to X) \simeq (A \wedge ((B \wedge C) \wedge D) \to X)$$

as the composite:

$$
\begin{aligned}
A \wedge (B \wedge (C \wedge D)) \to X &\simeq A \to B \wedge (C \wedge D) \to X & (e^{-1})\\
&\simeq A \to (B \wedge C) \wedge D \to X & (A \to \overline{\alpha})\\
&\simeq A \wedge ((B \wedge C) \wedge D) \to X & (e)
\end{aligned}
$$

also natural in their arguments. Evaluating these equivalences to the identity function, we get new arrows that fit in the original diagram:

96

$$((A \wedge B) \wedge (C \wedge D))$$

The theorem is then proved once we show the chain of homotopies:

$$\alpha \circ \alpha \sim \overline{\alpha}^4(\mathsf{id}) \sim \overline{\alpha}^L(\mathsf{id}) \circ \alpha \circ \overline{\alpha}^R(\mathsf{id}) \sim (A \wedge \alpha) \circ \alpha \circ (\alpha \wedge D) \qquad (4.3.36)$$

To verify the first homotopy in (4.3.36), we see that:

$$\alpha \circ \alpha \equiv \overline{\alpha}(\mathsf{id}) \circ \overline{\alpha}(\mathsf{id})$$
$$\sim (\overline{\alpha} \circ \overline{\alpha})(\mathsf{id}) \qquad \text{(naturality of } \overline{\alpha})$$
$$\equiv (e \circ e \circ (A \to e^{-1}) \circ e^{-1} \circ e \circ e \circ (A \to e^{-1}) \circ e^{-1})(\mathsf{id})$$
$$\sim (e \circ e \circ (A \to e^{-1}) \circ e \circ (A \to e^{-1}) \circ e^{-1})(\mathsf{id}) \qquad \text{(cancelling)}$$
$$\sim (e \circ e \circ e \circ (B \to A \to e^{-1}) \circ (A \to e^{-1}) \circ e^{-1})(\mathsf{id}) \qquad \text{(naturality of } e)$$
$$\equiv \overline{\alpha}^4(\mathsf{id})$$

The second homotopy in (4.3.36) is verified by (right-to-left):

$$\overline{\alpha}^L(\mathsf{id}) \circ \alpha \circ \overline{\alpha}^R(\mathsf{id}) \equiv \overline{\alpha}^L(\mathsf{id}) \circ \overline{\alpha}(\mathsf{id}) \circ \overline{\alpha}^R(\mathsf{id})$$
$$\sim (\overline{\alpha}^R \circ \overline{\alpha} \circ \overline{\alpha}^L)(\mathsf{id})$$
$$\text{(naturality of } \overline{\alpha} \text{ and } \overline{\alpha}^R)$$
$$\equiv (e \circ \overline{\alpha} \circ e^{-1} \circ e \circ e \circ (A \to e^{-1}) \circ e^{-1} \circ e \circ (A \to \overline{\alpha}) \circ e^{-1})(\mathsf{id})$$
$$\sim (e \circ \overline{\alpha} \circ e \circ (A \to e^{-1}) \circ (A \to \overline{\alpha}) \circ e^{-1})(\mathsf{id})$$
$$\text{(cancelling)}$$
$$\sim (e \circ \overline{\alpha} \circ e \circ (A \to (e^{-1} \circ \overline{\alpha})) \circ e^{-1})(\mathsf{id})$$
$$\text{(functoriality of } A \to -)$$
$$\equiv (e \circ e \circ e \circ (A \to e^{-1}) \circ e^{-1} \circ e$$
$$\circ (A \to (e^{-1} \circ e \circ e \circ (B \to e^{-1}) \circ e^{-1})) \circ e^{-1})(\mathsf{id})$$
$$\sim (e \circ e \circ e \circ (A \to ((B \to e^{-1}) \circ e^{-1})) \circ e^{-1})(\mathsf{id})$$
$$\text{(cancelling)}$$
$$\sim (e \circ e \circ e \circ (B \to A \to e^{-1}) \circ (A \to e^{-1}) \circ e^{-1})(\mathsf{id})$$
$$\text{(funct. of } A \to -)$$
$$\equiv \overline{\alpha}^4(\mathsf{id})$$

In order to prove the last homotopy in (4.3.36), it is sufficient to show that $\overline{\alpha}^R(\mathsf{id}) \sim$

$\alpha \wedge D$ and that $\overline{\alpha}^L(\mathsf{id}) \sim A \wedge \alpha$. We have:

$$\begin{aligned}
\overline{\alpha}^R(\mathsf{id}) &\equiv e(\overline{\alpha}(e^{-1}(\mathsf{id}))) \\
&\sim e(\overline{\alpha}(\eta)) \\
&\sim e(\eta \circ \overline{\alpha}(\mathsf{id})) && \text{(naturality of } \overline{\alpha}) \\
&\equiv \varepsilon \circ (\eta \circ \alpha) \wedge D \\
&\sim \varepsilon \circ (\eta \wedge D) \circ (\alpha \wedge D) && \text{(distrib. of } \wedge) \\
&\sim \alpha \wedge D && \text{(Lemma 4.3.23)}
\end{aligned}$$

and, lastly,

$$\begin{aligned}
\overline{\alpha}^L(\mathsf{id}) &\equiv e(\overline{\alpha} \circ e^{-1}(\mathsf{id})) \\
&\sim e(\overline{\alpha} \circ \eta) \\
&\sim e((\alpha \to A \wedge (B \wedge (C \wedge D))) \circ \eta) && \text{(Lemma 4.3.34)} \\
&\sim e((B \wedge (C \wedge D) \to A \wedge \alpha) \circ \eta) && \text{(dinaturality of } \eta) \\
&\sim (A \wedge \alpha) \circ e(\eta) && \text{(naturality of } e) \\
&\sim A \wedge \alpha && \text{(Lemma 4.3.23)}
\end{aligned}$$

thus proving the desired homotopy. $\qquad \square$

Theorem 4.3.37 (Unitors triangle). *For A and B pointed types, there is a homotopy*

$$(A \wedge \lambda) \circ \alpha \sim (\rho \wedge B)$$

corresponding to the commutativity of the following diagram:

$$((A \wedge \mathbb{S}^0) \wedge B) \xrightarrow{\quad\quad \alpha \quad\quad} (A \wedge (\mathbb{S}^0 \wedge B))$$

$$\underset{\rho \wedge B}{\searrow} \qquad \underset{A \wedge \lambda}{\swarrow}$$

$$(A \wedge B)$$

Proof. By an argument similar to the one for $\overline{\alpha}^L$ and $\overline{\alpha}^R$ in Theorem 4.3.35, one can verify the homotopies $A \wedge \lambda \sim (e \circ (A \to \overline{\lambda}) \circ e^{-1})(\mathsf{id})$ and $\rho \wedge B \sim (e \circ \overline{\rho} \circ e)(\mathsf{id})$, simplifying the expressions in the sought homotopy. Then:

$$\begin{aligned}
(A \wedge \lambda) \circ \alpha &\sim e(\overline{\lambda} \circ e^{-1}(\mathsf{id})) \circ \overline{\alpha}(\mathsf{id}) && \text{(simplification)} \\
&\sim \overline{\alpha}(e(\overline{\lambda} \circ e^{-1}(\mathsf{id}))) && \text{(naturality of } \overline{\alpha}) \\
&\equiv e(e(e^{-1} \circ e^{-1}(e(\overline{\lambda} \circ e^{-1}(\mathsf{id}))))) \\
&\sim e(e(e^{-1} \circ \overline{\lambda} \circ e^{-1}(\mathsf{id}))) && \text{(cancelling)} \\
&\equiv e(e(e^{-1} \circ e \circ \mathsf{b}^{-1} \circ e^{-1}(\mathsf{id}))) \\
&\sim e(e(\mathsf{b}^{-1} \circ e^{-1}(\mathsf{id}))) && \text{(cancelling)} \\
&\equiv (e \circ \overline{\rho} \circ e^{-1})(\mathsf{id}) \\
&\sim \rho \wedge B && \text{(simplification)}
\end{aligned}$$

gives the desired homotopy. $\qquad \square$

Theorem 4.3.38 (Braiding-unitors triangle)**.** *For a pointed type A, there is a homotopy*

$$\lambda \circ \gamma \sim \rho$$

corresponding to the commutativity of the following diagram:

$$
\begin{array}{ccc}
(A \wedge \mathbb{S}^0) & \xrightarrow{\quad\gamma\quad} & (\mathbb{S}^0 \wedge A) \\
& {\scriptstyle\rho}\searrow \quad A \quad \swarrow{\scriptstyle\lambda} &
\end{array}
$$

Proof. We have:

$$
\begin{aligned}
\lambda \circ \gamma &\equiv \overline{\lambda}(\mathsf{id}) \circ \overline{\gamma}(\mathsf{id}) \\
&\sim (\overline{\gamma} \circ \overline{\lambda})(\mathsf{id}) && \text{(naturality of } \overline{\gamma}) \\
&\equiv (e \circ \mathsf{tw} \circ e^{-1} \circ e \circ \mathsf{b}^{-1})(\mathsf{id}) \\
&\sim (e \circ \mathsf{tw} \circ \mathsf{b}^{-1})(\mathsf{id}) && \text{(cancelling)} \\
&\sim (e \circ (A \to \mathsf{b}^{-1}))(\mathsf{id}) \\
&\equiv \overline{\rho}(\mathsf{id}) \equiv \rho
\end{aligned}
$$

where the last homotopy is given by $(A \to c) \circ \mathsf{b} \sim \mathsf{tw} : (\mathbb{S}^0 \to A \to X) \to (A \to X)$. $\quad\square$

Lemma 4.3.39. *The following diagram commutes, for A, B, C and X pointed types:*

$$
\begin{array}{ccc}
(B \wedge C \to A \to X) & \xrightarrow{\quad e^{-1}\quad} & (B \to C \to A \to X) \\
\Big\downarrow{\scriptstyle\mathsf{tw}} & & \Big\downarrow{\scriptstyle B\to\mathsf{tw}} \\
& & (B \to A \to C \to X) \\
& & \Big\downarrow{\scriptstyle\mathsf{tw}} \\
(A \to B \wedge C \to X) & \xrightarrow{\quad A\to e^{-1}\quad} & (A \to B \to C \to X)
\end{array}
$$

Proof. Unfolding the definition of e^{-1}, we get the diagram:

$$
\begin{array}{ccccc}
& & & \xrightarrow{\quad e^{-1}\quad} & \\
(B \wedge C \to A \to X) & \xrightarrow{C\to-} & ((C \to B \wedge C) \to C \to A \to X) & \xrightarrow{\eta\to C\to A\to X} & (B \to C \to A \to X) \\
\Big\downarrow{\scriptstyle\mathsf{tw}} & & \Big\downarrow{\scriptstyle(C\to B\wedge C)\to\mathsf{tw}} & & \Big\downarrow{\scriptstyle B\to\mathsf{tw}} \\
& & ((C \to B \wedge C) \to A \to C \to X) & \xrightarrow{\eta\to A\to C\to X} & (B \to A \to C \to X) \\
& & \Big\downarrow{\scriptstyle\mathsf{tw}} & & \Big\downarrow{\scriptstyle\mathsf{tw}} \\
(A \to B \wedge C \to X) & \xrightarrow{A\to(C\to-)} & (A \to (C \to B \wedge C) \to C \to X) & \xrightarrow{A\to(\eta\to C\to X)} & (A \to B \to C \to X) \\
& & & \xrightarrow{\quad A\to e^{-1}\quad} &
\end{array}
$$

where the squares on the right are instances of naturality of tw, while the commutativity of the pentagon on the left follows easily from the definition of tw. $\quad\square$

Theorem 4.3.40 (Associativity-braiding hexagon). *For pointed types A, B and C, there is a homotopy*

$$\alpha \circ \gamma \circ \alpha \sim (B \wedge \gamma) \circ \alpha \circ (\gamma \wedge C)$$

corresponding to the commutativity of the following diagram:

$$
\begin{array}{ccccc}
((A \wedge B) \wedge C) & \xrightarrow{\ \alpha\ } & (A \wedge (B \wedge C)) & \xrightarrow{\ \gamma\ } & ((B \wedge C) \wedge A) \\
\ \downarrow{\scriptstyle \gamma \wedge C} & & & & \ \downarrow{\scriptstyle \alpha} \\
((B \wedge A) \wedge C)) & \xrightarrow[\ \alpha\ ]{} & (B \wedge (A \wedge C)) & \xrightarrow[\ B \wedge \gamma\ ]{} & (B \wedge (C \wedge A))
\end{array}
$$

Proof. The proof is structured similarly to the one for Theorem 4.3.35: the homotopies

$$
\begin{aligned}
B \wedge \gamma &\sim \overline{\gamma}^L(\mathsf{id}) && \text{with } \ \overline{\gamma}^L :\equiv e \circ (B \to \overline{\gamma}) \circ e^{-1} \\
\gamma \wedge C &\sim \overline{\gamma}^R(\mathsf{id}) && \text{with } \ \overline{\gamma}^R :\equiv e \circ \overline{\gamma} \circ e^{-1}
\end{aligned}
$$

can be proven in exactly the same way and, using these simplifications, we will show that both sides of the sought homotopy are homotopic to the same equivalence. Indeed we have:

$$
\begin{aligned}
\alpha \circ \gamma \circ \alpha &\equiv \overline{\alpha}(\mathsf{id}) \circ \overline{\gamma}(\mathsf{id}) \circ \overline{\alpha}(\mathsf{id}) \\
&\sim (\overline{\alpha} \circ \overline{\gamma} \circ \overline{\alpha})(\mathsf{id}) \\
&\qquad \text{(naturality of } \overline{\gamma} \text{ and } \overline{\alpha}) \\
&\equiv (e \circ e \circ (A \to e^{-1}) \circ e^{-1} \circ e \circ \mathsf{tw} \circ e^{-1} \circ e \circ e \circ (B \to e^{-1}) \circ e^{-1})(\mathsf{id}) \\
&\sim (e \circ e \circ (A \to e^{-1}) \circ \mathsf{tw} \circ e \circ (B \to e^{-1}) \circ e^{-1})(\mathsf{id}) \\
&\qquad \text{(cancelling)} \\
&\sim (e \circ e \circ \mathsf{tw} \circ (B \to \mathsf{tw}) \circ e^{-1} \circ e \circ (B \to e^{-1}) \circ e^{-1})(\mathsf{id}) \\
&\qquad \text{(Lemma 4.3.39)} \\
&\sim (e \circ e \circ \mathsf{tw} \circ (B \to \mathsf{tw}) \circ (B \to e^{-1}) \circ e^{-1})(\mathsf{id}) \\
&\qquad \text{(cancelling)}
\end{aligned}
$$

and

$$(B \wedge \gamma) \circ \alpha \circ (\gamma \wedge C) \sim \overline{\gamma}^L(\mathsf{id}) \circ \overline{\alpha} \circ \overline{\gamma}^R(\mathsf{id})$$

$$\text{(simplification)}$$

$$\sim (\overline{\gamma}^R \circ \overline{\alpha} \circ \overline{\gamma}^L)(\mathsf{id})$$

$$\text{(naturality of } \overline{\alpha} \text{ and } \overline{\gamma}^R)$$

$$\equiv (e \circ \overline{\gamma} \circ e^{-1} \circ e \circ e \circ (B \to e^{-1}) \circ e^{-1} \circ e \circ (B \to \overline{\gamma}) \circ e^{-1})(\mathsf{id})$$

$$\sim (e \circ \overline{\gamma} \circ e \circ (B \to e^{-1}) \circ (B \to \overline{\gamma}) \circ e^{-1})(\mathsf{id})$$

$$\text{(cancelling)}$$

$$\sim (e \circ \overline{\gamma} \circ e \circ (B \to (e^{-1} \circ \overline{\gamma})) \circ e^{-1})(\mathsf{id})$$

$$\text{(functoriality of } B \to -)$$

$$\equiv (e \circ e \circ \mathsf{tw} \circ e^{-1} \circ e \circ (B \to (e^{-1} \circ e \circ \mathsf{tw} \circ e^{-1})) \circ e^{-1})(\mathsf{id})$$

$$\sim (e \circ e \circ \mathsf{tw} \circ (B \to \mathsf{tw}) \circ (B \to e^{-1}) \circ e^{-1})(\mathsf{id})$$

$$\text{(cancelling)}$$

proving the commutativity of the diagram. $\qquad\square$

Theorem 4.3.41 (Double braiding). *For A and B pointed types, there is a homotopy*

$$\gamma \circ \gamma \sim \mathsf{id}$$

corresponding to the commutativity of the following diagram:

$$(A \wedge B) \xrightarrow{\ \gamma\ } (B \wedge A)$$
$$\Big\downarrow{\scriptstyle \gamma}$$
$$(A \wedge B)$$

Proof. Using that $\mathsf{tw} \circ \mathsf{tw} \sim \mathsf{id}$, we get:

$$\gamma \circ \gamma \equiv \overline{\gamma}(\mathsf{id}) \circ \overline{\gamma}(\mathsf{id})$$

$$\sim (\overline{\gamma} \circ \overline{\gamma})(\mathsf{id}) \qquad\qquad\qquad\qquad \text{(naturality of } \overline{\gamma})$$

$$\equiv (e \circ \mathsf{tw} \circ e^{-1} \circ e \circ \mathsf{tw} \circ e^{-1})(\mathsf{id})$$

$$\sim \mathsf{id} \qquad\qquad\qquad\qquad\qquad\qquad \text{(cancelling)}$$

as desired. $\qquad\square$

Finally we get the result of this section.

Theorem 4.3.42. *$\wedge$ is a 1-coherent symmetric monoidal product, assuming that e is pointed natural in C.*

Proof. This follows immediately from the theorems in this section. $\qquad\square$

Chapter 5

The Serre Spectral Sequence

Spectral sequences are important tools in algebraic topology.[23] They give a relationship between certain homotopy, homology and cohomology groups, in a way that generalizes long exact sequences. This generalization comes at a cost of being a lot more complicated than a long exact sequence.

In this chapter we will start the study of spectral sequences in homotopy type theory. We will introduce the notion of spectral sequences, and then construct the Atiyah-Hirzebruch and Serre spectral sequences for cohomology. We follow the construction due to Michael Shulman given in [Shu13]. We will also give a sketch on how to construct the analogues for homology, and look at some of the applications of these spectral sequences.

There are a couple of notable differences between spectral sequences in homotopy type theory compared to classical homotopy theory.

- As always, in HoTT all constructions have to be homotopy invariant, so we cannot use classical constructions that are not homotopy invariant. For example, the construction of the Serre spectral sequence for homology in [Hat04] uses CW-approximation of a space and the skeleton of the obtained CW-complex to construct the spectral sequence. These operations are not homotopy invariant, and therefore cannot be performed in HoTT.

- Another difference is that homology and cohomology are defined differently in HoTT than in classical homotopy theory. In classical homotopy theory (co)homology is defined using singular (co)homology. Since the intermediate steps in the construction of singular (co)homology is not homotopy invariant, we use a different definition of (co)homology (see Definition 5.4.2), which impacts the definition of spectral sequences involving (co)homology.

- The first page of a spectral sequence is often not homotopy invariant, and therefore cannot be constructed in HoTT. For this reason, we start counting the pages of

---

[23]The work in this chapter is joint work with Jeremy Avigad, Steve Awodey, Ulrik Buchholtz, Egbert Rijke and Mike Shulman.

spectral sequences at 2.

- HoTT offers a convenient language for formalizing proofs. Therefore, we have formalized all constructed spectral sequences in this chapter.

The spectral sequences we construct are not the most general version of these spectral sequences. The spectral sequences we construct are still more general than the formulation of the Serre spectral sequence in many textbooks (we give a version of generalized and parametrized cohomology), but there exist more general versions. There are two places where we compromised on generality for the sake of making the formalization easier. The first compromise is that we only formalized exact couples for graded R-modules for a ring R (which is not graded). More generally we could do this for any abelian category, which would require building up the theory of abelian categories (this is done in UniMath [VAG$^+$]). Furthermore, we did not look at convergence of spectral sequences in the most general sense, since that can get quite complicated and subtle. Instead, we only look at spectral sequences that are eventually constant pointwise, so the ∞-page is just the eventual value. This restriction adds the condition to the spectral sequences we construct that the coefficients are only in truncated spectra.

5.1 Spectral Sequences

A spectral sequence consists of a sequence of pages, each of them containing a two-dimensional grid of abelian groups. There are maps between these groups, called *differentials*. These differentials form (co)chain complexes, and the (co)homology of these complexes determine the groups on the next page. In 5.1 we show an example of two pages of a spectral sequence, where each dot represents an abelian group. In this figure only the two first quadrants are shown, because in simple applications all other groups are trivial, though that need not be the case in general.

Before we start, we define the notion of a graded abelian group. We will give a nonstandard definition that is equivalent to the standard one.

Definition 5.1.1. For an abelian group G, an G-*graded abelian group* is a family of abelian groups indexed over G. If M and M' are G-graded abelian groups, the type of *graded abelian group homomorphism from M to M'* is a triple consisting of a *degree* $e : G \simeq G$ (this is an equivalence of types, not a group isomorphism), a proof of $(g : G) \to e(g) = g + e(0)$ and a term of type

$$\{x\ y : I\} \to (p : e(x) = y) \to M_x \to M'_y.$$

We will denote the type of homomorphisms as $M \to M'$. For $\varphi : M \to M'$ we write $\deg_\varphi$ for the first projection. We will often call $\deg_\varphi(0)$ the degree of φ. For $x : I$ we will write

$$\varphi_x :\equiv \varphi_{\mathsf{refl}_x} : M_x \to M'_{\deg_\varphi(x)}$$

(a) The page $E_2^{p,q}$ (b) $E_3^{p,q}$
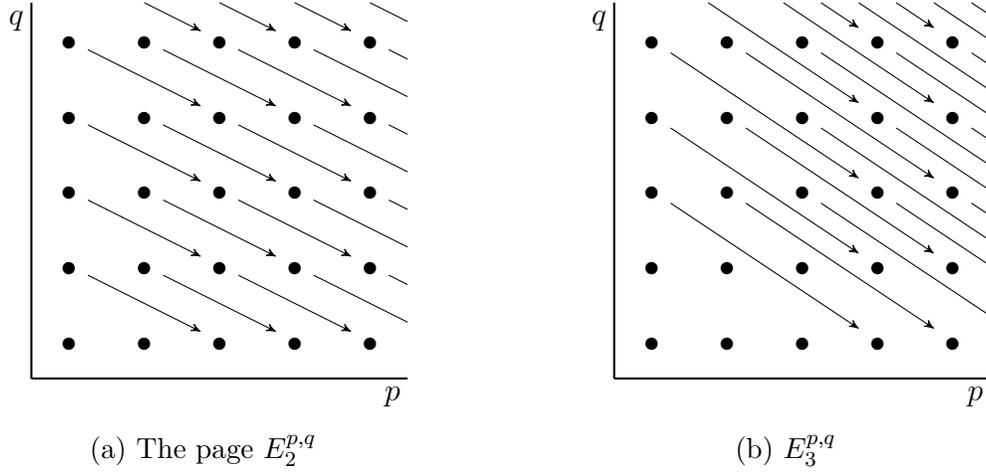
Figure 5.1: Two pages of a spectral sequence.

and

$$\varphi_{[x]} :\equiv \varphi_{p_x} : M_{\mathsf{deg}_\varphi^{-1}(x)} \to M_x'$$

where $p_x : \mathsf{deg}_\varphi(\mathsf{deg}_\varphi^{-1}(x)) = x$ is the proof obtained from the equivalence deg_φ.

Remark 5.1.2. This definition looks a bit cumbersome, since the condition on e forces e to be homotopic to the function $\lambda g.\, g + h$ for some group element h. Furthermore, the type of φ is equivalently $(x : I) \to M_x \to M_{x+h}'$. We will now discuss why we made these choices.

To see why this is more convenient, we consider the composition of two graded homomorphisms. Suppose we have two graded homomorphisms $\varphi : M \to M'$ and $\psi : M' \to M''$ of degrees $h : G$ and $k : G$, respectively. Then the pointwise composition $\lambda(g : G).\, \lambda(m : M_g).\, \psi_{g+h}(\varphi_g(m))$ has type $(g : G) \to M_g \to M_{(g+h)+k}''$. So to get a graded homomorphism of degree $h + k$, with the more straightforward representation, we would need to transport along the equality $(g + h) + k = g + (h + k)$. Since compositions are ubiquitous, this would happen all over the place. However, in our setting, the composite of two graded homomorphisms of degree e and e' will have degree $e' \circ e$, without using any transports.

We eliminated a transport to define composition, but there are other places where we cannot get rid of them so easily. For example, given morphisms $\varphi : M \to M'$ and $\psi : M' \to M''$ with $\psi \circ \varphi = 0$ (the graded map that is constantly 0), we are interested in the homology of φ and ψ. This is the kernel of ψ quotiented by the image of φ in M_x'. However, if φ_x has type $M_x \to M_{\mathsf{deg}_\varphi(x)}'$, there is no map that (without transports) lands in M_x'. We would need to transport along the path $p_x : \mathsf{deg}_\varphi^{-1}(\mathsf{deg}_\varphi(x)) = x$ and take the image of this composite:

$$M_{\mathsf{deg}_\varphi^{-1}(x)} \xrightarrow{\varphi_{\mathsf{deg}_\varphi^{-1}(x)}} M_{\mathsf{deg}_\varphi(\mathsf{deg}_\varphi^{-1}(x))}' \xrightarrow{\sim} M_x'.$$

104

For this reason, we allow graded homomorphisms to be applied to paths, so that we have a "built-in" transport. Then we can define the homology as $H_x :\equiv \ker(\psi_x)/\operatorname{im}(\varphi_{[x]})$, or diagramatically

$$M_{\deg_\varphi^{-1}(x)} \xrightarrow{\varphi_{[x]}} M'_x \xrightarrow{\psi_x} M''_{\deg_\psi(x)}.$$

For the construction of spectral sequences, we do not actually need the second component of a graded homomorphism: all constructions also work if the degrees are arbitrary equivalences of type $I \simeq I$, where I is an arbitrary set. This is the definition used in the formalization. In this document we add this condition, so that our definition is equivalent to the usual definition of graded morphism.

Definition 5.1.3. A *spectral sequence* consists of the following data.

- A sequence E_r of abelian groups graded over $\mathbb{Z} \times \mathbb{Z}$ for $r \geq 2$. E_r is called the *r-page* of the spectral sequence;

- *differentials*, which are graded morphisms $d_r : E_r \to E_r$ such that $d_r \circ d_r = 0$;

- isomorphisms $\alpha_r^{p,q} : H^{p,q}(E_r) \simeq E_{r+1}^{p,q}$ where $H^{p,q}(E_r) = \ker(d_r^{p,q})/\operatorname{im}(d_r^{[p,q]})$ is the cohomology of the cochain complex determined by d_r.

We use the notation for cohomologically indexed spectral sequences, since we will construct spectral sequences in cohomology in this chapter. For the spectral sequences in this chapter, the degree of d_r will be $(r, 1-r)$, which signifies a cohomologically indexed spectral sequence.

As mentioned before, we start counting the pages at 2, since the first page of the spectral sequences we construct will not be homotopy invariant. In the formalization we start counting at 0 for convenience. Also, in the formalization, we assume that the grading of E_r is over some set I instead of fixing it to $\mathbb{Z} \times \mathbb{Z}$. It is not clear whether this extra generality is useful. Instead of abelian groups, we could take objects of an arbitrary abelian category, but for concreteness and to simplify things, we choose to develop the theory only for abelian groups. In the formalization we developed the theory for graded R-module for a (non-graded) ring R, but we have only applied it to abelian groups so far.

Note that (E_r, d_r) determines E_{r+1} but *not* d_{r+1}. Furthermore, E_r is a subquotient (subgroup of a quotient) of E_2, so if $E_2^{p,q}$ is trivial, then $E_r^{p,q}$ is trivial for all r.

In many cases, the spectral sequence will *converge*. That means that for a fixed $(p, q) : \mathbb{Z} \times \mathbb{Z}$ the sequence $E_r^{p,q}$ will be constant for r large enough. For example, suppose that the degree of d_r is $(r, r-1)$, and E_2 is limited to the first quadrant. Now for any (p, q) all differentials in or out of $E_r^{p,q}$ will go out the first quadrant for sufficiently large r. This means that the image of $d_r^{[p,q]}$ is trivial, and the kernel of $d_r^{p,q}$ is the full group. This implies that $E_{r+1}^{p,q} \simeq E_r^{p,q}$, so the spectral sequence converges.

Whenever a spectral sequence converges, we write $E_\infty^{p,q}$ for the eventual value of $E_r^{p,q}$ for r large enough. Now the power of spectral sequences is that there is often a relation between $E_2^{p,q}$ and $E_\infty^{p,q}$. This relation does not specify $E_\infty^{p,q}$ exactly, but specifies that $E_\infty^{p,q}$ build up some group D^n for the diagonals where $p + q = n$.

Definition 5.1.4. Suppose given an abelian group D and a finite sequence of abelian groups $(E^n)_n$. We say that D *is built from* $(E^n)_n$ if there is a sequence of abelian groups $(D^n)_n$ and short exact sequences

$$E^0 \to D \to D^1$$

$$\vdots$$

$$E^k \to D^k \to D^{k+1}$$
$$E^{k+1} \to D^{k+1} \to D^{k+2}$$

$$\vdots$$

$$E^m \to D^m \to 0$$

The sequence $(D^n)_n$ is called a *cofiltration* of D, they are successive quotients of D.

Definition 5.1.5. Given a graded abelian group D^n and a bigraded abelian group $C^{p,q}$, we write
$$E_2^{p,q} = C^{p,q} \Rightarrow D^{p+q}$$
if there is a spectral sequence E such that

- $E_2^{p,q} = C^{p,q}$;

- E converges to E_∞;

- D^n is built from $E_\infty^{p,q}$ where $p + q = n$.

Remark 5.1.6. This definition implicitly requires that for $p + q = n$ only finitely many $E_\infty^{p,q}$ are nontrivial. This is sufficient for the spectral sequences we consider in this chapter, but this condition can be relaxed in more general constructions of spectral sequences.

5.2   Exact Couples

As we said before, the pair (E_r, d_r) in a spectral sequence specifies E_{r+1}, but not d_{r+1}. If we have some more information about page r, then we can construct page $r + 1$ and the extra information for page $r + 1$. Now we can iterate this construction and obtain a spectral sequence by forgetting about the extra information.

An *exact couple* exactly gives this extra information [Mas52]. From it, we can compute the *derived exact couple*, which gives us the information next page of the spectral sequence.

Definition 5.2.1. An *exact couple* is a pair (D, E) of $\mathbb{Z} \times \mathbb{Z}$-graded abelian groups with graded homomorphisms

$$D \xrightarrow{\ i\ } D$$
$$k \nwarrow \quad \swarrow j$$
$$E$$

that is exact in all three vertices. This means that for all $p : \deg_j(x) =_I y$ and $q : \deg_k(y) = z$ that $\ker(k_q) = \operatorname{im}(j_p)$, and similarly for the other two pairs of maps.

For an exact couple we will write $\iota :\equiv \deg_i$ and $\eta :\equiv \deg_j$ and $\kappa :\equiv \deg_k$ for the degrees.

Lemma 5.2.2. *Given an exact couple (D, E, i, j, k), we can define a* derived exact couple *(D', E', i', j', k') where E' is the homology of $d :\equiv j \circ k : E \to E$. The degrees of the derived maps are $\deg_{i'} \equiv \iota$, $\deg_{k'} \equiv \kappa$ and $\deg_{j'} \equiv \eta \circ \iota^{-1}$.*

$$D' \xrightarrow{\ i'\ } D'$$
$$k' \nwarrow \quad \swarrow j'$$
$$E'$$

Proof. In this proof we will be explicit about the grading of D and E, which is a lot trickier (at least in intensional type theory) than a proof without the grading. For a proof that does not take the grading into account, see for example [Hat04, Lemma 1.1]. We define for $x : \mathbb{Z} \times \mathbb{Z}$ the graded abelian groups D' and E' by $D'_x = \operatorname{im} i_{[x]}$ and $E'_x = \ker d_x / \operatorname{im} d_{[x]}$. Now $i'_x : D'_x \to D'_{\iota x}$ is defined as the composite

$$D'_x \hookrightarrow D_x \xrightarrow{i} D'_{\iota x}.$$

This is sufficient to define i' on all paths $\iota x = y$ as a function $D_x \to D_y$.
We first define $j'_{px} : D'_{\iota x} \to E'_{\eta x}$ for the canonical path $px : \eta(\iota^{-1}(\iota x)) = \eta x$, which is sufficient to define j' in general. Note that $D'_{\iota x} \equiv \operatorname{im} i_{[\iota x]} \simeq \operatorname{im} i_x$, so to define j'_{px} it is sufficient to define $\tilde{\jmath} : D_x \to E'_{\eta x}$ such that $(a : D_x) \to i_x(a) = 0 \to \tilde{\jmath}(a) = 0$. We define $\tilde{\jmath}(a) :\equiv [j_x a]$. This is well-defined, since $j_x a \in \ker d_{\eta x}$,[24] because

$$d_{\eta x}(j_x a) = j_{\kappa(\eta x)}(k_{\eta x}(j_x a)) = j_{\kappa(\eta x)}\mathbf{0} = \mathbf{0}.$$

Now suppose that $i_x(a) = 0$. Without loss of generality we may assume that $x \equiv \kappa y$. By exactness, this means that $a \in \operatorname{im} k_y$, so there is $b : E_y$ such that $k_y(b) = a$. Now $j_x a = j_x(k_y b) \equiv d_y b$, so

$$j_x a \in \operatorname{im} d_y \simeq \operatorname{im} d_{[\deg_d y]} \equiv \operatorname{im} d_{[\eta x]}.$$

---

[24]We use the set-theoretical notation $g \in H$ to say that a group element $g : G$ is in subgroup H. Formally, a subgroup H is an element of $G \to \mathsf{Prop}$ (containing 0, and closed under addition and negation) and $g \in H$ is defined as $H(g)$. Note that $(g : G) \times H(g)$ can be endowed with a group structure, which is H viewed as a group.

This shows that $\tilde{j}(a) = 0$, completing the definition of j'. Note that $j'_{px}(i_x a) = [j_x a]$. To define $k'_x : E'_x \to D'_{\kappa x}$, first note that if $a \in \ker d_x$, then $k_x a \in \ker j_{\kappa x} = \operatorname{im} i_{[\kappa x]}$ by exactness. Now we need to show that if $a \in \operatorname{im} d_{[x]}$, then $k_x a = 0$. By assumption, we have $b : E_{\eta^{-1}\kappa^{-1}x}$ such that $d_{[x]}(b) = a$. Now we compute (using $k_x j_{[x]} = \mathbf{0}$)

$$k_x a = k_x(d_{[x]} b) = k_x(j_{[x]}(k_{[\eta^{-1}x]} b)) = 0.$$

This defines k.

Showing exactness of the derived couple involves some diagram chasing. To show that $j'i' = \mathbf{0}$ it is sufficient to show that for all $a : D'_{\iota x}$ we have $j'_{p(\iota x)}(i'_{\iota x} a) = 0$. Since $a \in \operatorname{im} i_{[\iota x]} \simeq i_x$ we know that $a = i_x b$ for some $b : D_x$. We compute

$$j'_{p(\iota x)}(i'_{\iota x} a) = j'_{p(\iota x)}(i_{\iota x} a) = [j_{\iota x} a] = [j_{\iota x}(i_x b)] = [0] = 0.$$

To show that $\ker j' \subseteq \operatorname{im} i'$, it is sufficient to show that $\ker j'_{p(\kappa x)} \subseteq \operatorname{im} i'_{\kappa x}$. Suppose $a : D'_{\iota(\kappa x)}$ such that $j'_{p(\kappa x)}(a) = 0$, we know that $a = i_{\kappa x}(b)$ for some b. Now

$$0 = j'_{p(\kappa x)}(a) = j'_{p(\kappa x)}(i_{\kappa x}(b)) = [j_{\kappa x}(b)],$$

which means that $j_{\kappa x}(b) \in \operatorname{im} d_{[\kappa(\eta x)]} \simeq \operatorname{im} d_x$. This means that for some $c : E_x$ we have $j_{\kappa x}(b) = d_x(c) = j_{\kappa x}(k_x c)$. This means that $j_{\kappa x}(b - k_x c) = 0$, hence $b - k_x c \in \ker j_{\kappa x} = \operatorname{im} i_{[\kappa x]}$. This means that we can define $b - k_x c : D'_{\kappa x}$. Now we compute

$$i'_{\kappa x}(b - k_x c) = i_{\kappa x} b - i_{\kappa x}(k_x c) = a - 0 = a,$$

which means $a \in \operatorname{im} i'_{\kappa x}$, as desired.

We will omit the other cases, which are similar but easier. $\qquad\square$

Repeating the process of deriving exact couples, we get a sequence of exact couples $(D_r, E_r, i_r, j_r, k_r)$.[25] We get a spectral sequence (E_r, d_r) where $d_r :\equiv j_r \circ k_r$. Note that

$$\deg_{d_r} = \deg_{j_r} \circ \deg_{k_r} = \eta \circ \iota^r \circ \kappa$$

Given some extra conditions on the exact couple, we can show that this spectral sequence converges.

Definition 5.2.3. We call an exact couple *bounded* if for every $x : \mathbb{Z} \times \mathbb{Z}$ there is are bounds $B_x : \mathbb{N}$ such that for all $s \geq B_x$ we have

$$E^{\iota^{-s}(x)} = 0 \qquad \text{and} \qquad D^{\iota^s(x)} = 0$$

Remark 5.2.4. The condition on D also shows that if you go sufficiently far in the ι-direction, then E is trivial, since $D \xrightarrow{j} E \xrightarrow{k} D$ is exact and the occurrences of D will be trivial. Converse, the condition on E shows that if you go sufficiently far in the ι^{-1} direction, i will be an equivalence, by the following exact sequence.

$$E \xrightarrow{k} D \xrightarrow{i} D \xrightarrow{j} E$$

We call $x : \mathbb{Z} \times \mathbb{Z}$ a *stable index* whenever $i^{[\iota^{-s}x]}$ is surjective for *all* $s \geq 0$.

---

[25]We will now put the grading of D, E and the maps as superscript, so that we can put the page as subscript.

Given a bounded exact couple, the pages stabilize pointwise, which is the content of the next lemma.

Lemma 5.2.5. *For a bounded exact couple (D, E, i, j, k) we have for all sufficiently large r that $D_{r+1}^x = D_r^x$ and $E_{r+1}^x = E_r^x$.*

Proof. Note that $E_{r+1}^x = \ker d_r^x / \operatorname{im} d_r^{[x]}$. Since d_r has degree $\eta \circ \iota^r \circ \kappa$, and because $\mathbb{Z} \times \mathbb{Z}$ is an abelian group, the degrees commute.[26] The codomain of d_r^x is $E_r^{\eta(\iota^r(\kappa x))} = E_r^{\iota^r(\eta(\kappa x))}$, which is trivial for sufficiently large r by Remark 5.2.4. Also, the domain of $d_r^{[x]}$ is $E_r^{\kappa^{-1}(\iota^{-r}(\eta^{-1}x))} = E_r^{\iota^{-r}(\kappa^{-1}(\eta^{-1}x))}$, which is trivial for sufficiently large r by the definition of boundedness.

To show that D stabilizes, first note that if $i_r^{[\iota^{-1}x]}$ is surjective, then $i_{r+1}^{[x]}$ is surjective. The reason is that

$$i_{r+1} : D_{r+1}^{\iota^{-1}x} \xrightarrow{\sim} D_r^{\iota^{-1}x} \to D_{r+1}^x$$

is now a composite of two surjective maps. This means that if the maps $i_{r_0}^{[\iota^{-s}x]}$ are surjections for all $s \geq B + 1$, then the maps $i_{r_0+1}^{[\iota^{-s}x]}$ will be surjections for all $s \geq B$. In this case, for $r \geq r_0 + B$ we have that $i_r^{[x]}$ is a surjection, hence that $D_{r+1}^x = D_r^x$. Since $i_0^{[\iota^{-s}x]}$ are surjections for sufficiently large s by Remark 5.2.4, we finish the proof. $\square$

By the proof of Lemma 5.2.5 we get explicit bounds B_x^D and B_x^E such that $D_r^x = D_{B_x^D}^x$ and $E_{r'}^x = E_{B_x^E}^x$ for all $r \geq B_x^D$ and $r' \geq B_x^E$. We define $D_\infty^x := D_{B_x^D}^x$ and $E_\infty^x := E_{B_x^E}^x$. Both B_x^D and B_x^E will be the maximum of B_y for some sequence of indices y.

Theorem 5.2.6 (Convergence Theorem). *Let (D, E, i, j, k) be a bounded exact couple and let x be a stable index. Then $D^{\kappa x}$ is built from $(E_\infty^{\iota^n(x)})_{0 \leq n < B_{\kappa x}}$.*

Proof. Define $C^n = D_\infty^{\kappa(\iota^n x)}$. Let $n : \mathbb{N}$ be arbitrary, then for sufficiently large r the following is a short exact sequence

$$0 \xrightarrow{j_r} E_r^{\iota^n x} \xrightarrow{k_r} D_r^{\kappa(\iota^n x)} \xrightarrow{i_r} D_r^{\iota(\kappa(\iota^n x))} \xrightarrow{j_r} 0.$$

This is the case, because for sufficiently large r the domain of $j_r^{[\iota^n x]}$ and the codomain of $j_r \iota(\kappa(\iota^n x))$ are contractible. Now (possibly by increasing r) these groups are in the stable range, so we get a short exact sequence

$$0 \to E_\infty^{\iota^n x} \to C^n \to C^{n+1} \to 0.$$

Moreover, we have $C^0 \equiv D_\infty^{\kappa x} = D^{\kappa x}$ because x is a stable index. Lastly, for $s \geq B_{\kappa x}$ we know that C^s is trivial, because $D^{\kappa(\iota^n)}$ is trivial by the condition of being bounded. This shows that $D^{\kappa x}$ is built from $(E_\infty^{\iota^n(x)})_{0 \leq n < B_{\kappa x}}$. $\square$

---

[26]In the formalization, we do not assume that the degrees are shifts by a group element, and we explicitly assume that $\kappa \iota = \iota \kappa$ and $\iota \eta = \eta \iota$.

5.3 Spectra

We have not yet discussed how to get an exact couple in the first place. Recall that from a pointed map we get a long exact sequence of homotopy groups. For a *sequence* of pointed maps we get a sequence of long exact sequences. However, we do not want to do this for pointed maps, but for maps between *spectra.*

You can think of a spectrum as a generalized space with negative dimensions. Suppose we are given a pointed type X and a chosen delooping Y of X. That is, Y is a pointed type such that $\Omega Y \simeq^* X$. Now the $(n+1)$-th homotopy group of Y is equal to the n-th homotopy group of X. The 0-th homotopy group of Y is new information, and we can think of it as the (-1)-th homotopy group of X. Spectra go further on this idea: it is a pointed type with infinitely many deloopings.

Definition 5.3.1. A *prespectrum* is a pair consisting of a sequence of pointed types $Y : \mathbb{Z} \to \mathcal{U}^*$ and a sequence of pointed maps $e : (n : \mathbb{Z}) \to Y_n \to^* \Omega Y_{n+1}$. An Ω-*spectrum* or *spectrum* is a prespectrum (Y, e) where e_n is a pointed equivalence for all n. We will often just write Y for the pair (Y, e), and we denote the type of (pre)spectra by Prespectrum and Spectrum.
A map between (pre)spectra $(Y, e) \to (Y', e')$ is a pair consisting of $f : (n : \mathbb{Z}) \to Y_n \to Y_n'$ and $p : (n : \mathbb{N}) \to e_n' \circ f_n \sim^* \Omega f_{n+1} \circ e_n$.

Remark 5.3.2. Usually a (pre)spectrum is indexed over $\mathbb{N}$ and not over $\mathbb{Z}$. We index it over $\mathbb{Z}$ so that we do not have to do a case split in — for example — the definition of homotopy group of a spectrum, see Definition 5.3.4.

Example 5.3.3.

- If A is an abelian group, we have $HA :$ Spectrum where $(HA)_n = K(A, n)$ for $n \geq 0$ and $(HA)_n = \mathbf{1}$ for $n < 0$.

- Given $Y :$ Spectrum and $k : \mathbb{Z}$, we can define two new spectra $\Omega^k Y$ and $\Sigma^k Y :$ Spectrum with

$$(\Omega^k Y)_n :\equiv Y_{n-k} \qquad\qquad (\Sigma^k Y)_n :\equiv Y_{n+k}$$

- Given a spectrum map $f : X \to Y$, we have a spectrum $\mathsf{fib}_f :$ Spectrum with $(\mathsf{fib}_f)_n :\equiv \mathsf{fib}_{f_n}$. Furthermore we have a spectrum map $p_1 : \mathsf{fib}_f \to X$. This follows from the following two facts about fibers (which we will not prove here).

  (i) Given a pointed map $g : A \to B$, there is a pointed equivalence $e_1 : \Omega \mathsf{fib}_g \simeq^*$ with a pointed homotopy

110

(ii) fib is a functor from pointed maps to pointed types and p_1 is a natural transformation. This means the following. Suppose we are given a square of pointed maps and a homotopy filling the following square.

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

Then there is a pointed map $e_2 : \mathsf{fib}_f \to \mathsf{fib}_g$, functorial in (h, h'). In particular this means that if h and h' are equivalences, then e_2 is. The naturality of p_1 means that we have the following pointed homotopy.

$$
\begin{array}{ccc}
\mathsf{fib}_f & \xrightarrow{\ p_1\ } & A \\
\downarrow{\scriptstyle e_2} & & \downarrow{\scriptstyle h} \\
\mathsf{fib}_g & \xrightarrow[\ p_1\ ]{} & B
\end{array}
$$

Given an Ω-spectrum Y and $n : \mathbb{Z}$, we define can the n-th homotopy group of Y to be

$$\pi_n(Y) := \pi_{n+k}(Y_k) : \mathsf{AbGroup}$$

for any k such that $n + k \geq 0$. This is independent of k, because

$$\pi_{n+(k+1)}(Y_{k+1}) \simeq \pi_{n+k}(\Omega Y_{k+1}) \simeq \pi_{n+k}(Y_k).$$

For concreteness, in the following definition we pick $k = 2 - n$. We make this choice so that $\pi_n(Y)$ directly carries the structure of an abelian group.

The homotopy group of a prespectrum Y is a bit different, since $\pi_{n+k}(Y_k)$ is not independent of k. In this case, it is the colimit as $k \to \infty$. We make the substitution $\ell = n + k - 2$ to make the index of the homotopy group always positive.

Definition 5.3.4. Given an Ω-spectrum Y and $n : \mathbb{Z}$, we define the *n-th homotopy group of Y* as

$$\pi_n(Y) :\equiv \pi_2(Y_{2-n}).$$

For a prespectrum Y we define

$$\pi_n(Y) :\equiv \mathsf{colim}_{\ell \to \infty}(\pi_{\ell+2}(Y_{\ell+2-n})).$$

Note that the homotopy group of a prespectrum is a set by Corollary 3.3.30(i), and the colimit can be equipped with a group structure, making $\pi_n(Y)$ an abelian group for a prespectrum Y.

The long exact sequence of homotopy groups for pointed types, constructed in Section 4.1.1, induces one on spectra.

Theorem 5.3.5. *Given a spectrum map $f : X \to Y$ with fiber F, we get the following long exact sequence of homotopy groups indexed over $\mathbb{Z} \times \mathsf{fin}_3$.*

$$\vdots$$

$$\pi_{k+2}(F) \xrightarrow[\pi_{k+2}(p_1)]{} \pi_{k+2}(X) \xrightarrow{\pi_{k+2}(f)} \pi_{k+2}(Y)$$

$$\pi_{k+1}(F) \xrightarrow[\pi_{k+1}(p_1)]{} \pi_{k+1}(X) \xrightarrow{\pi_{k+1}(f)} \pi_{k+1}(Y)$$

$$\pi_k(F) \xrightarrow[\pi_k(p_1)]{} \pi_k(X) \xrightarrow{\pi_k(f)} \pi_k(Y)$$

$$\vdots$$

We will use the following lemma. Recall the definition of successor structure from Definition 4.1.2.

Lemma 5.3.6. *Suppose given two successor structures N and M, and for each $n : N$ let G^n be a long exact sequence index by M. Let $m : M$ and $k \geq 2$. Suppose that*

- *for all $n : N$, $G_m^{n+1} \simeq G_{m+k}^n$ and $G_{m+1}^{n+1} \simeq G_{m+k+1}^n$*

- *for all $n : N$ the following diagram commutes.*

$$
\begin{array}{ccc}
G_{m+1}^{n+1} & \longrightarrow & G_m^{n+1} \\
\downarrow {\scriptstyle \wr} & & \downarrow {\scriptstyle \wr} \\
G_{m+k+1}^n & \longrightarrow & G_{m+k}^n
\end{array}
$$

Then there is a long exact sequence $H : N \times \mathsf{fin}_{k-1} \to \mathsf{Set}^*$ with $H_{(n,\ell)} :\equiv G_{m+\ell}^n$

For $k = 3$ the hypotheses can be represented in the diagram below.

112

$$\vdots$$

Proof (Lemma 5.3.6). The map $H_{(n,\ell+1)} \to H_{(n,\ell)}$ is defined to be the given map $G^n_{m+\ell+1} \to G^n_{m+\ell}$. The map $H_{(n+1,0)} \to H_{(n,k-1)}$ is defined to be the composite

$$G^{n+1}_m \xrightarrow{\sim} G^n_{m+k} \to G^n_{m+k-1}.$$

It is easy to check that this is a long exact sequence from the conditions. $\qquad\square$

Proof (Theorem 5.3.5). For each $n : \mathbb{Z}$ we get a long exact sequence of homotopy groups for f_{2-n} by Theorem 4.1.1. We splice them together using Lemma 5.3.6 with $N = (\mathbb{Z}, \lambda n.\, n+1)$ and $M = (\mathbb{N}, \lambda n.\, n+1)$ and with $k = 3$ and $m = (2, 0)$. This means that the resulting sequence is

$$\vdots$$

We still need to check the conditions for the Lemma. The first isomorphism is given by the following composition

$$\pi_2(Y_{2-(n+1)}) \simeq \pi_2(\Omega Y_{2-(n+1)+1}) \simeq \pi_2(\Omega Y_{2-n}) \equiv \pi_3(Y_{2-n}),$$

113

The second isomorphism is the same, replacing Y by X. The square commutes because the two isomorphisms are both natural in Y. □

Suppose given a sequence of spectra A and a sequence of spectrum maps

$$\cdots \to A_s \xrightarrow{f_s} A_{s-1} \xrightarrow{f_{s-1}} A_{s-2} \to \cdots$$

Let $B_s :\equiv \mathsf{fib}_{f_s}$. Then $D^{n,s} :\equiv \pi_n(A_s)$ and $E^{n,s} :\equiv \pi_n(B_s)$ are graded abelian groups and the maps of the long exact sequences become graded homomorphisms. This gives exactly the data of an exact couple.

For cohomology, it is customary to reindex the pages of the spectral sequence with the base change $(p, q) = (s - n, -s)$, or equivalently $(n, s) = (-(p + q), -q)$.

Theorem 5.3.7. *Given a sequence of spectra*

$$\cdots \to A_s \xrightarrow{f_s} A_{s-1} \xrightarrow{f_{s-1}} A_{s-2} \to \cdots$$

with fibers $B_s :\equiv \mathsf{fib}_{f_s}$, suppose for all n there is a β_n such that for all $s \leq \beta_n$ we have $\pi_n(A_s) = 0$ and suppose that for all n there is a γ_n such that for all $s > \gamma_n$ the map $\pi_n(f_s)$ is an isomorphism. Then the exact couple constructed from this sequence is bounded. This spectral sequence gives

$$E_2^{p,q} = \pi_{-(p+q)}(B_{-q}) \Rightarrow \pi_{-(p+q)}(A_{\gamma_{-(p+q)}}).$$

Proof. Note that for this spectral sequence we have $\iota(n, s) \equiv \mathsf{deg}_i(n, s) \equiv (n, s - 1)$ and $\kappa = \mathsf{id}$. This means that we need to show that for all $(n, s) : \mathbb{Z} \times \mathbb{Z}$ there is a bound $\beta'_{n,s}$ such that for all $t \geq \beta'_{n,s}$ we have

$$E^{n,s+t} \equiv \pi_n(B_{s+t}) = 0 \quad \text{and} \quad D^{n,s-t} \equiv \pi_n(A_{s-t}) = 0.$$

Note that the right equation holds if $s - t \leq \beta_n$, i.e. if $t \geq s - \beta_n$. By the long exact sequence of homotopy groups we know that if $f_s : A_s \to A_{s-1}$ induces an equivalence on both π_n and π_{n+1}, then $\pi_n(B_{s+t}) = 0$. So if we define

$$\beta'_{n,s} :\equiv \mathsf{max}(s - \beta_n, \gamma_n - s, \gamma_{n+1} - s),$$

we know that the exact couple is bounded with bound β'.

Now note that $x = (n, \gamma_n)$ is a stable index, because $\pi_n(f_{\gamma_n+t})$ is surjective for all $t \geq 0$. Therefore, by Theorem 5.2.6 we know that D^{n,γ_n} is built from $(E_\infty^{n,\gamma_n-s})_{0 \leq s \leq \beta'_{n,\gamma_n}}$. If we apply the reindexing $(p, q) = (s - n, -s)$, we get the desired relation

$$E_2^{p,q} = \pi_{-(p+q)}(B_{-q}) \Rightarrow \pi_{-(p+q)}(A_{\gamma_{-(p+q)}}).$$

□

5.4 Spectral Sequences for Cohomology

Cohomology groups are algebraic invariants of types. They are often easier to compute than homotopy groups, but they can also be used to compute certain homotopy groups, often via the universal coefficient theorem and the Hurewicz theorem (neither of which have been proven in HoTT yet).

The intermediate steps of most classical constructions of the singular cohomology are not homotopy invariant. *Cellular cohomology* is only defined for cell complexes and not for arbitrary spaces, but it can be defined in HoTT [BH18]. *Singular cohomology* is defined as a quotient of a large abelian group that is not homotopy invariant, which makes this definition impossible in HoTT. However, classically, Eilenberg-MacLane spaces represent cohomology, and we can use this fact as the *definition* of cohomology in HoTT [Cav15].

Normally cohomology groups have coefficients in an abelian group, but more generally they can have coefficients in a spectrum, or even a family of spectra. In this section we will define cohomology groups and construct the Atiyah-Hirzebruch spectral sequence for cohomology. This is a generalization of the spectral sequence defined in [AH61] in the special case of topological K-theory. From the Atiyah-Hirzebruch spectral sequence we can construct the Serre spectral sequence, sometimes also called the Leray-Serre spectral sequence.

Definition 5.4.1. Suppose given $X : \mathcal{U}^*$ and $Y : X \to \mathsf{Spectrum}$. We define $(x : X) \to^* Yx : \mathsf{Spectrum}$ such that $((x : X) \to^* Yx)_n :\equiv (x : X) \to^* (Yx)_n$. If Y does not depend on X, we write $X \to^* Y$.

For an unpointed type $X : \mathcal{U}$ and $Y : X \to \mathsf{Spectrum}$, we similarly define $(x : X) \to Yx : \mathsf{Spectrum}$ such that $((x : X) \to Yx)_n :\equiv (x : X) \to (Yx)_n$ (this has as basepoint the constant map into the basepoint of $(Yx)_n$), and abbreviate this to $X \to Y$ if Y does not depend on X.

These spectra are well-defined, since we have

$$\Omega((a : A) \to^* (Ba)) \simeq (a : A) \to^* \Omega(Ba)$$

and

$$\Omega((a : A) \to (Ba)) \simeq (a : A) \to \Omega(Ba).$$

Moreover, they satisfy the expected properties of dependent product. In particular, if $X : \mathcal{U}^*$ and $Y, Z : X \to \mathsf{Spectrum}$ and moreover if we have a fiberwise spectrum map $f : (x : X) \to Yx \to Zx$, this induces a map on the dependent products

$$\Pi_f : ((x : X) \to Yx) \to ((x : X) \to Zx).$$

Definition 5.4.2. Suppose given $X : \mathcal{U}^*$, $Y : X \to \mathsf{Spectrum}$ and $n : \mathbb{Z}$. We define the *geneneralized, parametrized, reduced cohomology* of X with coefficients in Y as[27]

$$\tilde{H}^n(X; \lambda x. Yx) :\equiv \pi_{-n}((x : X) \to^* Yx) \simeq \|(x : X) \to^* (Yx)_n\|_0.$$

---

[27]We will write $\lambda x. Yx$ in η-expanded form to remember that this is parametrized cohomology.

If Y does not depend on X, we have the *unparametrized cohomology* as

$$\tilde{H}^n(X; Y) :\equiv \pi_{-n}(X \to^* Yx) \simeq \|X \to^* Y_n\|_0.$$

If $X : \mathcal{U}$ is an arbitrary type, we define the *unreduced cohomology* as

$$H^n(X; \lambda x. Yx) :\equiv \pi_{-n}((x : X) \to Yx) \simeq \|(x : X) \to (Yx)_n\|_0 \simeq \tilde{H}^n(X_+; \lambda x. Y_+x).$$

Here $X_+ :\equiv X + 1 : \mathcal{U}^*$ and $Y_+ : X_+ \to \mathsf{Spectrum}$ is defined as $Y_+(\mathsf{inl}(x)) :\equiv Yx$ and $Y_+(\mathsf{inr}(\star)) :\equiv 1$. If $X : \mathcal{U}^*$ and $A : X \to \mathsf{AbGroup}$, we define the *ordinary cohomology* as

$$\tilde{H}^n(X; \lambda x. Ax) :\equiv \tilde{H}^n(X; \lambda x. H(Ax)).$$

We can combine the attributes ordinary/generalized, parametrized/unparametrized and reduced/unreduced for cohomology however we want, leading to eight different notions.

We define

$$\tilde{H}^n(X) :\equiv \tilde{H}^n(X; \mathbb{Z})$$

and similarly for unreduced cohomology.

Unparametrized cohomology satisfies the Eilenberg-Steenrod axioms for cohomology. Although we will not use this fact in this chapter, for completeness we will state it here.

To give the definition we need to introduce one more concept.

Definition 5.4.3. A type X has n-choice for $n \geq -2$ if for all $P : X \to \mathcal{U}$ the canonical map

$$\|(x : X) \to Px\|_n \to ((x : X) \to \|Px\|_n)$$

is an equivalence.

Note that in particular fin_k has n-choice for all k, n.

Definition 5.4.4. A *unparametrized reduced cohomology theory* is a contravariant functor $\tilde{E}^n : \mathcal{U}^* \to \mathsf{AbGroup}$ for every $n : \mathbb{Z}$ satisfying the *Eilenberg-Steenrod axioms*. Functoriality means that for a pointed map $f : X \to^* Y$ there is a map $\tilde{E}^n(f) : \tilde{E}^n(Y) \to \tilde{E}^n(X)$ such that $\tilde{E}(\mathsf{id}) \sim^* \mathsf{id}$ and $\tilde{E}(g \circ f) \sim^* \tilde{E}(f) \circ \tilde{E}g$. The Eilenberg-Steenrod axioms are

- *(Suspension axiom)* There is a natural transformation $\tilde{E}^{n+1}(\Sigma X) \simeq \tilde{E}^n(X)$.

- *(Exactness)* Given a cofiber sequence $X \xrightarrow{f} Y \xrightarrow{g} Z$, the sequence

$$\tilde{E}^n(Z) \xrightarrow{\tilde{E}^n(g)} \tilde{E}^n Y \xrightarrow{\tilde{E}^n(f)} \tilde{E}^n(X)$$

is exact at $\tilde{E}^n(Y)$.

- *(Additivity)* Suppose given a type I satisfying 0-choice and $X : I \to \mathcal{U}^*$. Then the canonical homomorphism

$$\tilde{E}^n\Big(\bigvee_i Xi\Big) \to ((i : I) \to \tilde{E}^n(Xi))$$

  is an isomorphism.

A cohomology theory is called *ordinary* if it also satisfies the following axiom.

- *(Dimension)* If $n \neq 0$, then $\tilde{E}^n(\mathbb{S}^0)$ is trivial.

The following theorem has been proven in [Cav15]. We will not repeat the proof here.

Theorem 5.4.5. *Unparametrized generalized reduced cohomology is a cohomology theory. Ordinary cohomology also satisfies the dimension axiom.*

We will not use Theorem 5.4.5 in the remainder of this chapter.

To construct the Atiyah-Hirzebruch spectral sequence, we need the Postnikov tower of a spectrum.

Definition 5.4.6. We say that for $k : \mathbb{Z}$ a spectrum Y is *k-truncated* if Y_n is $(k+n)$-truncated for all $n : \mathbb{Z}$ (using the convention that any type is ℓ-truncated for $\ell \leq -2$).

The *k-truncation* of a spectrum Y, written $\|Y\|_k$, is defined as $(\|Y\|_k)_n :\equiv \|Y_n\|_{k+n}$ where we define $\|A\|_\ell = \mathbf{1}$ for $\ell \leq -2$.

Lemma 5.4.7. *The usual properties of truncations also hold for spectra. In particular we will use that there is a spectrum map $|-|_k : Y \to \|Y\|_k$ and that if Z is k-truncated, then a spectrum map $f : Y \to Z$ induces a spectrum map $\|Y\|_k \to Z$.*

Proof. The underlying maps are the corresponding facts for pointed maps. The fact that these maps are spectrum maps comes from the fact that these operations commute with taking loop spaces. We omit the details here. $\square$

Lemma 5.4.8 (Postnikov Tower for spectra). *For $s : \mathbb{Z}$ and $Y : \mathsf{Spectrum}$ there is a spectrum map $f^s : \|Y\|_s \to \|Y\|_{s-1}$ that levelwise has fiber $\Sigma^n(H\pi_s(Y))$. That is,*

$$(\mathsf{fib}_{f^s})_k \simeq^* \left(\Sigma^s(H\pi_s(Y))\right)_k.$$

We should be able to extend this equivalence to a spectrum equivalence, but we do not need this strengthening for the remainder of the proof.

Proof. Note that $\|Y\|_{s-1}$ is $(s-1)$-truncated, and therefore s-truncated. By the elimination of spectrum truncation in Lemma 5.4.7 we get a spectrum map $f^s : \|Y\|_s \to \|Y\|_{s-1}$. For the levelwise pointed equivalence, we need to show that

$$\mathsf{fib}_{f^s_k} \simeq^* K(\pi_s(Y), s + k)$$

117

To show this, by Theorem 4.2.4 we need to show that $\mathsf{fib}_{f_k^s}$ is $(s+k)$-truncated, $(s+k-1)$-connected and $\pi_{s+k}(\mathsf{fib}_{f_k^s}) \simeq \pi_s(Y)$.

Note that $f_k^s : \|Y_k\|_{s+k} \to \|Y_k\|_{s+k-1}$, so the truncatedness follows because the domain and codomain of f_k^s are both $(s+k)$-truncated. For the connectedness, we know that $|-|_{s+k-1} : Y_k \to \|Y_k\|_{s+k-1}$ is $(s+k-1)$-connected, and the elimination principle for truncations preserve connectedness, therefore f_k^s is $(s+k-1)$-connected. To compute the homotopy group, we look at a piece of the long exact sequence for homotopy groups for f_k^s at level $s+k$ and $s+k+1$.

$$\begin{array}{ccccc}
\bullet & \longrightarrow & 0 & \longrightarrow & 0 \\
& & & \nearrow & \\
\pi_{s+k}(\mathsf{fib}_{f_k^s}) & \longrightarrow & \pi_{s+k}(Y) & \longrightarrow & 0
\end{array}$$

Since we have the exact sequence $0 \to \pi_{s+k}(\mathsf{fib}_{f_k^s}) \to \pi_{s+k}(Y) \to 0$, the middle map must be an equivalence, which finishes the proof. $\qquad\square$

For a spectrum Y, we get the Postnikov tower

$$\cdot \to \|Y\|_s \to \|Y\|_{s-1} \to \|Y\|_{s-2} \to \cdots$$

This satisfies the conditions of Theorem 5.3.7, but unfortunately the spectral sequence constructed from this is trivial. We need another ingredient to get an interesting spectral sequence.

Lemma 5.4.9. *Suppose given $X : \mathcal{U}^*$ and two family of spectra $Y, Z : X \to \mathsf{Spectrum}$. A family of spectrum maps*

$$f : (x : X) \to Yx \to Zx$$

induces a spectrum map between the spectra of sections for Y and Z:

$$f \circ (-) : ((x : X) \to Yx) \to ((x : X) \to Zx).$$

Moreover, the fiber of this spectrum map is levelwise $(x : X) \to \mathsf{fib}_{fx}$, that is

$$(\mathsf{fib}_{f \circ (-)})_n \simeq^* ((x : X) \to \mathsf{fib}_{fx})_n.$$

The levelwise equivalence should be extendable to a spectrum equivalence, but we do not need that in this chapter.

Proof. We define (see Lemma 2.2.7.(vi))

$$(f \circ (-))_n :\equiv f_n \circ (-) : ((x : X) \to^* (Yx)_n) \to^* ((x : X) \to^* (Zx)_n).$$

This is a spectrum map because of the pointed function extensionality mentioned in Lemma 2.2.7.(viii).

By Lemma 2.2.7.(vi) the fiber of this map is levelwise $(x : X) \to \mathsf{fib}_{fx}$. $\qquad\square$

We now have all the ingredients of the Atiyah-Hirzebruch spectral sequence.

Theorem 5.4.10 (Atiyah-Hirzebruch spectral sequence for reduced cohomology)**.** *If $X : \mathcal{U}^*$ is a pointed type and $Y : X \to k\text{-}\mathsf{Spectrum}$ is a family of k-truncated spectra over X, then we get a spectral sequence with*

$$E_2^{p,q} = \tilde{H}^p(X; \lambda x.\, \pi_{-q}(Yx)) \Rightarrow \tilde{H}^{p+q}(X; \lambda x.\, Yx).$$

Proof. Define $A_s :\equiv ((x : X) \to^* \|Yx\|_s)$ and consider the sequence of spectra

$$\cdots \to A_s \xrightarrow{f_s} A_{s-1} \xrightarrow{f_{s-1}} A_{s-2} \to \cdots$$

where f_s is the map induced by the Postnikov tower. By Lemma 5.4.9 and Lemma 5.4.8 f_s levelwise has fiber $B_s :\equiv (x : X) \to^* \Sigma^s H\pi_s(Yx)$. We want to apply Theorem 5.3.7, so we need to check the conditions of that theorem. For $n : \mathbb{Z}$ we define $\beta_n :\equiv n - 1$. Notice that A_s is s-truncated, and thus for $s \leq B_n$ we have

$$\pi_n(A_s) :\equiv \pi_n((x : X) \to^* \|Yx\|_s) = 0.$$

For $n : \mathbb{Z}$ define $\gamma_n :\equiv k$. Then for $s \geq \gamma_n$ the spectrum A_s is levelwise equivalent to $(x : X) \to^* Yx$, so for $s > \gamma_n$ the map $A_s \to A_{s-1}$ becomes levelwise the identity map under that equivalence. This means that f_s is an equivalence, so in particular $\pi_n(f_s)$ is an isomorphism. By Theorem 5.3.7 we now get the spectral sequence

$$E_2^{p,q} = \pi_{-(p+q)}(B_{-q}) \Rightarrow \pi_{-(p+q)}(A_k).$$

We now compute

$$\begin{aligned}
\pi_{-(p+q)}(B_{-q}) &\simeq \pi_{-(p+q)}((x : X) \to^* \Sigma^{-q} H\pi_{-q}(Yx)) \\
&\simeq \tilde{H}^{p+q}(X; \lambda x.\, \Sigma^{-q} H\pi_{-q}(Yx)) \\
&\simeq \tilde{H}^p(X; \lambda x.\, \pi_{-q}(Yx))
\end{aligned}$$

and

$$\begin{aligned}
\pi_{-(p+q)}(A_k) &\simeq \pi_{-(p+q)}((x : X) \to^* \|Yx\|_k) \\
&\simeq \pi_{-(p+q)}((x : X) \to^* Yx) \\
&\simeq \tilde{H}^{p+q}(X; \lambda x.\, Yx).
\end{aligned}$$

$\square$

We also have the corresponding spectral sequence for unreduced cohomology.

Corollary 5.4.11 (Atiyah-Hirzebruch spectral sequence for unreduced cohomology)**.** *If $X : \mathcal{U}$ is any type and $Y : X \to k\text{-}\mathsf{Spectrum}$ is a family of k-truncated spectra over X, then*

$$E_2^{p,q} = H^p(X; \lambda x.\, \pi_{-q}(Yx)) \Rightarrow H^{p+q}(X; \lambda x.\, Yx).$$

Proof. Apply Theorem 5.4.10 to X_+ and Y_+ (defined in Definition 5.4.2). $\qquad\square$

From the Atiyah-Hirzebruch spectral sequence we can construct the Serre spectral sequence.

Theorem 5.4.12 (Serre spectral sequence for cohomology). *Suppose given $B : \mathcal{U}$, a family of types $F : B \to \mathcal{U}$ and a spectrum $Y :$ Spectrum that is k-truncated. Then*

$$E_2^{p,q} = H^p(B; \lambda b.\, H^q(Fb; Y)) \Rightarrow H^{p+q}((b : B) \times Fb; Y).$$

Proof. Apply Corollary 5.4.11 to the type B and and the family of spectra $\lambda b.\, Fb \to Y$, which is k-truncated. Then we get

$$E_2^{p,q} = H^p(B; \lambda b.\, \pi_{-q}(Fb \to Y)) \Rightarrow H^{p+q}(B; \lambda b.\, Fb \to Y).$$

Note that $\pi_{-q}(Fb \to Y) \simeq H^q(Fb; Y)$, so the second page is the desired group, and for the ∞-page we compute

$$
\begin{aligned}
H^{p+q}(B; \lambda b.\, Fb \to Y) &= \pi_{-(p+q)}((b : B) \to Fb \to Y) \\
&= \pi_{-(p+q)}(((b : B) \times Fb) \to Y) \\
&= H^{p+q}((b : B) \times Fb; Y).
\end{aligned}
$$

$\qquad\square$

Equivalent to the data given in Theorem 5.4.12 is a map $X \to B$ and a k-truncated spectrum Y. In that case we get the spectral sequence

$$E_2^{p,q} = H^p(B; \lambda b.\, H^q(\mathsf{fib}_f(b); Y)) \Rightarrow H^{p+q}(X; Y).$$

Analogous to the proof of Theorem 5.4.12 we also have a version when Y is parametrized over B. In that case we get

$$E_2^{p,q} = H^p(B; \lambda b.\, H^q(\mathsf{fib}_f(b); Y)) \Rightarrow H^{p+q}(X; \lambda x.\, Y(fx)).$$

We get a useful special case of the Serre spectral sequence when the family $\lambda b.\, H^q(Fb; Y)$ is constant. This happens in particular when B is simply connected.

Corollary 5.4.13. *Suppose given a simply connected pointed type $B : \mathcal{U}^*$, a family of types $F : B \to \mathcal{U}$ and a spectrum $Y :$ Spectrum that is k-truncated. Then*

$$E_2^{p,q} = H^p(B; H^q(Fb_0; Y)) \Rightarrow H^{p+q}((b : B) \times Fb; Y).$$

Proof. Apply Theorem 5.4.12. The family $\lambda b.\, H^q(Fb; Y) : B \to$ AbGroup is a family of sets. Since B is simply connected, every such family is constant, so all fibers are equal to $H^q(Fb_0; Y)$. $\qquad\square$

In the spectral sequences constructed we assumed that the spectra were truncated. The reason we need this assumption is that the notion of convergence we used for spectral sequence is the eventual value of the sequence. If we had a stronger notion of convergence, we might be able to relax the truncatedness condition. However, there is another reason why the spectral sequence can become (pointwise) eventually constant. Instead of assuming that the spectra are truncated, we can pose a restriction on the base space.

Definition 5.4.14. We say that $X : \mathcal{U}^*$ satisfies *weak pointed choice* if there is a natural number n such that for all families $Y : X \to \mathcal{U}^*$ of n-connected types the type of dependent pointed maps $(x : X) \to^* Y(x)$ is 0-connected.

Example 5.4.15.

- The spheres $\mathbb{S}^n$ satisfy weak pointed choice. The proof is easy for $n = 0$, which we will skip. For $\mathbb{S}^{n+1}$, note that

$$((x : S^{n+1}) \to^* Y(x)) \simeq \star =^{\Omega^n Y(-)}_{\text{surf}} \star$$

  where $\text{surf} : \Omega^{n+1}\mathbb{S}^{n+1}$ is the surface of $\mathbb{S}^{n+1}$ and $\star : \Omega^n Y(\text{base})$ is the basepoint. Now if Y is a family of $(n+1)$-connected types, then $\Omega^n Y(-)$ is a family of 1-connected types, and a pathover in that family is 0-connected, as desired.

- Suppose I is a type that satisfies 0-choice (see Definition 5.4.3). Then the collection of types that satisfy weak pointed choice are closed under I-indexed wedges. This follows from the dependent universal property of the wedge.

$$\left(\bigvee_{i:I} X_i \to^* P(x) \right) \simeq (i : I) \to (x : X_i) \to^* P(\text{in}_i(x)).$$

Theorem 5.4.16. *If $X : \mathcal{U}^*$ satisfies weak pointed choice and $Y : X \to \mathsf{Spectrum}$ is any family of spectra, we get the spectral sequence in Theorem 5.4.10:*

$$E_2^{p,q} = \tilde{H}^p(X; \lambda x.\, \pi_{-q}(Yx)) \Rightarrow \tilde{H}^{p+q}(X; \lambda x.\, Yx).$$

Proof. The proof is the mostly the same as for Theorem 5.4.10. The only difference is in showing that the sequence stabilizes on homotopy groups when s is large. Suppose X satisfies choice with respect to k-connected families. For $n : \mathbb{Z}$ define $\gamma_n :\equiv n + k$. Then for $s > \gamma_n$ we know that the fiber of f_s has as k-th homotopy group

$$\pi_\ell(B_s) = \pi_0(\Omega^\ell B_s) = \|(x : X) \to^* K(\pi_s(Yx), s - \ell)\|_0.$$

This is a product into a family of $(s - \ell - 1)$-connected types, which for $\ell = n$ and $\ell = n - 1$ is a family of at least k-connected types. By the weak choice principle on X this type is 0-connected, so these homotopy groups are trivial. Now by the long exact sequence of homotopy groups for f_s the map $\pi_n(f_s)$ is an isomorphism, as required. $\qquad \square$

5.5 Spectral Sequences for Homology

Homology theory has not been developed as much as cohomology theory in HoTT. It is known that the homology given by a prespectrum forms a homology theory [Gra17]. Lemma 18 in that paper was not proven carefully, but it follows from the results in Section 4.3.

In this section, we sketch the construction of the Atiyah-Hirzebruch and Serre spectral sequences for homology [Ser51]. The results in the section are not proven in HoTT, and are therefore stated as remarks without proof.

If $X : \mathcal{U}^*$ and $Y : \mathsf{Prespectrum}$, we can define $X \wedge Y : \mathsf{Prespectrum}$ with

$$(X \wedge Y)_n :\equiv X \wedge Y_n.$$

To show that it is a prespectrum, recall the adjunction between the suspension and the loop. For pointed types X and Y we have a natural equivalence

$$(\Sigma X \to^* Y) \simeq (X \to^* \Sigma Y).$$

Therefore, to characterize a prespectrum, it is sufficient to give a map $f_n : \Sigma Y_n \to Y_{n+1}$. This is given for the smash prespectrum as the composite

$$\Sigma(X \wedge Y_n) \xrightarrow{\sim} X \wedge \Sigma Y_n \xrightarrow{X \wedge f_n} X \wedge Y_{n+1}.$$

We can define reduced homology

$$\tilde{H}_n(X; Y) :\equiv \pi_n(X \wedge Y).$$

For the construction of parametrized homology we need to generalize the smash product.

Definition 5.5.1. Given $A : \mathcal{U}^*$ and $B : A \to \mathcal{U}^*$, we define the parametrized smash

$$(x : A) \wedge B(x)$$

to be the pushout

Remark 5.5.2. The strategy for constructing the spectral sequences for homology is as follows.

- The parametrized smash is (should be) left adjoint to pointed dependent maps. That means that there is a natural equivalence

$$(((x : A) \wedge Bx) \to^* C) \simeq (x : A) \to^* Bx \to^* C.$$

- From this we get (natural) equivalences

$$\Sigma((x : A) \wedge Bx) \simeq ((x : A) \wedge \Sigma(Bx));$$

$$((x : A) \wedge Bx) \wedge C \simeq (x : A) \wedge (Bx \wedge C);$$

$$(x : A_+) \wedge B_+x) \wedge C \simeq (x : A) \times Bx.$$

The proofs of these properties should be similar to the proofs in Section 4.3.4.

- Therefore, for $X : \mathcal{U}^*$ and $Y : X \to \mathsf{Prespectrum}$ we have a prespectrum $(x : X) \wedge Yx$. The maps are given by the above equivalence.

- We can now define parametrized (reduced, generalized) homology as

$$H_n(X; \lambda x.\, Yx) :\equiv \pi_n((x : X) \wedge Yx).$$

We can define unreduced homology by adding a point to X, in the same way as for cohomology.

- As before, given $X : \mathcal{U}^*$ and $Y : X \to \mathsf{Spectrum}$, we can again form the Postnikov tower of Yx for any $x : X$. We now want to take the parametrized smash over X, but there is no hope to compute the fiber of this spectrum.

- However, we should be able to do it when we work in spectra. The forgetful functor $\mathsf{Spectrum} \to \mathsf{Prespectrum}$ has a left adjoint, called *spectrification*. The spectrification LY of a prespectrum Y can be constructed either as a higher inductive family of types [S+11] or as the colimit

$$(LY)_n :\equiv \mathsf{colim}_{k \to \infty}\, \Omega^k Y_{n+k}.$$

For neither definition a careful proof of the adjunction has been given.

- We can now define the parametrized smash of spectra as the spectrification of the parametrized smash for prespectra. This should preserve cofiber sequences of spectra, in the sense that if

$$Ax \to Bx \to Cx$$

is a family of cofiber sequences of spectra indexed by $x : \mathcal{U}^*$, the following sequence is also a cofiber sequence of spectra

$$((x : X) \wedge Ax) \to ((x : X) \wedge Bx) \to ((x : X) \wedge Cx)$$

123

- A sequence of spectra should be a fiber sequence of spectra if and only if it is a cofiber sequence of spectra. This is true classically, and should also hold in HoTT.

- Assuming that all the above properties have been proven, we can get the Atiyah-Hirzebruch spectral sequence for reduced homology. Suppose given a pointed type X and $Y : X \to \mathsf{Spectrum}$ a family of spectra. We can apply Theorem 5.3.7 to the iterated fiber sequence

$$((x : X) \wedge \Sigma^n H) \to ((x : X) \wedge \|Yx\|_s) \to ((x : X) \wedge \|Yx\|_{s-1}).$$

To satisfy the conditions for that theorem we need to assume some conditions on X and/or Y. In particular it is sufficient if Y is a family of truncated and connected spectra, but weaker conditions might also suffice. Using homological indexing (where p and q have their sign reversed) we get

$$E_{p,q}^2 = \pi_{p+q}(B_q) \Rightarrow \pi_{p+q}(A_{\gamma_{p+q}}).$$

Now we compute

$$\begin{aligned}
\pi_{p+q}(B_q) &\simeq \pi_{p+q}((x : X) \wedge \Sigma^q H \pi_q(Yx)) \\
&\simeq \tilde{H}_{p+q}(X; \lambda x.\, \Sigma^q H \pi_q(Yx)) \\
&\simeq \tilde{H}_p(X; \lambda x.\, \pi_q(Yx))
\end{aligned}$$

and

$$\begin{aligned}
\pi_{p+q}(A_k) &\simeq \pi_{p+q}((x : X) \wedge \|Yx\|_k) \\
&\simeq \pi_{p+q}((x : X) \to Yx) \\
&\simeq \tilde{H}_{p+q}(X; \lambda x.\, Yx).
\end{aligned}$$

This gives the desired spectral sequence:

$$E_{p,q}^2 = \tilde{H}_p(X; \lambda x.\, \pi_q(Yx)) \Rightarrow \tilde{H}_{p+q}(X; \lambda x.\, Yx).$$

- We get the Atiyah-Hirzebruch spectral sequence for *unreduced* homology in the same way as for cohomology, by applying the version for reduced homology to X_+ and Y_+.

- We get the Serre spectral sequence for homology also in the same way. Suppose given $B : \mathcal{U}$ and $F : B \to \mathcal{U}$ and a truncated spectrum Y. Applying the Atiyah-Hirzebruch spectral sequence for unreduced homology to the type B and the spectrum $\lambda b.\, Fb \wedge Y$ we get

$$E_{p,q}^2 = H_p(B; \lambda b.\, \pi_q(Fb \wedge Y)) \Rightarrow H_{p+q}(B; \lambda b.\, Fb \wedge Y).$$

The second page is what we want. For the ∞-page we compute

$$\begin{aligned}
H_{p+q}(B; \lambda b.\, Fb \to Y) &= \pi_{p+q}((b : B_+) \wedge (F_+ b \wedge Y)) \\
&= \pi_{p+q}(((b : B_+) \wedge F_+ b) \wedge Y) \\
&= \pi_{p+q}(((b : B) \times Fb) \wedge Y) \\
&= H_{p+q}((b : B) \times Fb; Y).
\end{aligned}$$

This gives the Serre spectral sequence for homology:

$$E^2_{p,q} = H_p(B; \lambda b.\, H_q(Fb; Y)) \Rightarrow H_{p+q}((b : B) \times Fb; Y).$$

Remark 5.5.3. We can also use the parametrized smash to get a spectral sequence for reduced homology and reduced cohomology. Suppose given $B : \mathcal{U}^*$ and a family of types $F : B \to \mathcal{U}^*$ and a spectrum $Y : \mathsf{Spectrum}$ that is k-truncated. Then we get the following two spectral sequences

$$E_2^{p,q} = \tilde{H}^p(B; \lambda b.\, \tilde{H}^q(Fb; Y)) \Rightarrow \tilde{H}^{p+q}((b : B) \wedge Fb; Y);$$

$$E^2_{p,q} = \tilde{H}_p(B; \lambda b.\, \tilde{H}_q(Fb; Y)) \Rightarrow \tilde{H}_{p+q}((b : B) \wedge Fb; Y).$$

For homology, the proof is the same as above. For cohomology, we apply the Atiyah-Hirzebruch spectral sequence for reduced cohomology to the pointed type B and the family of spectra $\lambda b.\, Fb \to^* Y$. We get the desired spectral sequence by the adjunction between parametrized smash and dependent pointed maps.

These spectral sequences generalize Theorem 5.4.12 and the corresponding version for homology: we get those versions back when we add a point to B and F. Whether this extra generality is useful is unknown.

5.6 Applications of Spectral Sequences

Classically, there are many applications of the Serre and Atiyah-Hirzebruch spectral sequences. Here we will list some of these applications, and give thoughts on how to translate these results in HoTT. The results in this section have not been formalized. Before we start, we compute the cohomology of spheres.

Lemma 5.6.1. *If $n \geq 1$, then*

$$H^k(\mathbb{S}^n; A) = \begin{cases} A & \text{if } k \in \{0, n\} \\ 0 & \text{otherwise.} \end{cases} \tag{5.6.2}$$

This is a special case of the universal coefficient theorem, which we do not have yet in HoTT. However, we can prove these equalities directly from the definition of cohomology.

Proof. For $k = 0$ we have

$$H^0(\mathbb{S}^n; A) = \|\mathbb{S}^n \to A\|_0 = (\mathbb{S}^n \to A) = A,$$

where we use that $\mathbb{S}^n$ is 0-connected. For $k \neq 0$ we have

$$H^k(\mathbb{S}^n; A) = \tilde{H}^k(\mathbb{S}^n + 1; A) = \tilde{H}^k(\mathbb{S}^n; A) = \|\mathbb{S}^n \to^* K(A, k)\|_0 = \|\Omega^n K(A, k)\|_0.$$

Now for $n < k$ the type $\Omega^n K(A, k)$ is 0-connected, hence the result is contractible. For $n = k$ the result is A, and for $n > k$ the type $\Omega^n K(A, k)$ itself is contractible. $\qquad \square$

The first application is the path fibration. Suppose given a simply connected pointed type B we have a map $\mathbf{1} \to B$ that has fiber ΩB.[28] In other words, we have the fiber sequence

$$\Omega B \to \mathbf{1} \to B.$$

Now the Serre spectral sequence for cohomology gives (say, with integer coefficients)

$$E_2^{p,q} = H^p(B; H^q(\Omega B)) \Rightarrow H^{p+q}(\mathbf{1}).$$

Note that the ∞-page vanishes, except when $p + q = 0$, when the coefficient is $\mathbb{Z}$. For ordinary cohomology H^n is trivial for $n < 0$, which means that the second page is only nontrivial in the first quadrant of the plane, hence this is true for all pages, including the ∞-page. Therefore, the ∞-page has one group $\mathbb{Z}$ at the origin, and trivial groups everywhere else, as shown in Figure 5.2.

q					
2	0	0	0	0	0
1	0	0	0	0	0
0	$\mathbb{Z}$	0	0	0	0
	0	1	2	3	4 $\quad p$
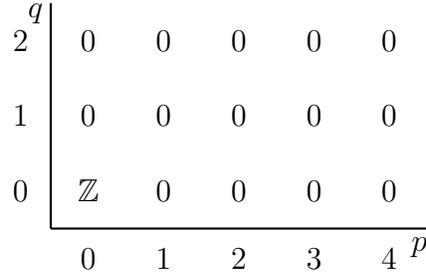
Figure 5.2: $E_\infty^{p,q}$ for the path fibration.

This gives a relation between the cohomology of B and the cohomology of ΩB. If we know the cohomology for one of the spaces one of them, then we can sometimes compute the cohomology from the other using this. Using the Serre spectral sequence for homology, we have the same relationship between the homology of B and the homology of ΩB. The computations in the next example will work exactly the same for homology.

---

[28]It is called the *path fibration* because classically to get a Serre fibration we need to take the path space PB instead of $\mathbf{1}$.

Example 5.6.3. As an example, we can compute the cohomology groups of $B = K(\mathbb{Z}, 2)$ (which is the complex projective space $\mathbf{CP}^\infty$). Its loop space is $\Omega K(\mathbb{Z}, 2) = K(\mathbb{Z}, 1) = \mathbb{S}^1$, and by Lemma 5.6.1 we have

$$H^n(\mathbb{S}^1) = \begin{cases} \mathbb{Z} & \text{if } n = 0, 1 \\ 0 & \text{otherwise.} \end{cases}$$

The resulting second page of the spectral sequence is shown in Figure 5.3, all other groups on the second page are trivial. Note that the shown differentials are the only
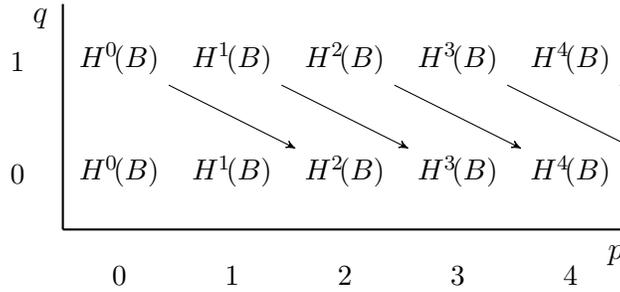
Figure 5.3: $E_2^{p,q}$ for the path fibration of $K(\mathbb{Z}, 2)$.

nontrivial differentials on the second page, and all differentials on all later pages are also trivial. This means that $E_3 = E_\infty$, depicted in Figure 5.2. Note that there are no nontrivial differentials going in or out of the $H^0(B)$ and $H^1(B)$ in the bottom line. This means that

$$H^0(B) = E_\infty^{0,0} = \mathbb{Z}$$

and

$$H^1(B) = E_\infty^{1,0}.$$

All other groups displayed on the second page vanish on the ∞-page. Therefore, all shown differentials must be isomorphisms. This means that $H^{n+2}(B) = H^n(B)$, which shows that $H^n(B)$ is $\mathbb{Z}$ for even n and 0 for odd n.

Another simple application of the Serre spectral sequence is to compute the homology and cohomology groups of $\Omega\mathbb{S}^n$, given in [Hat04, Example 1.5]. In this case, we know the (co)homology of the base space $\mathbb{S}^n$, and from it we can deduce the (co)homology of the loop space $\Omega\mathbb{S}^n$. We will do the computation here for cohomology.

Example 5.6.4. If we take the Serre spectral sequence for the path fibration of $B = S^n$ for $n \geq 2$, then the second page has entries

$$E_2^{p,q} = H^p(S^n; H^q(\Omega S^n)) = \begin{cases} H^q(\Omega S^n) & \text{if } p = 0, n \\ 0 & \text{otherwise.} \end{cases}$$
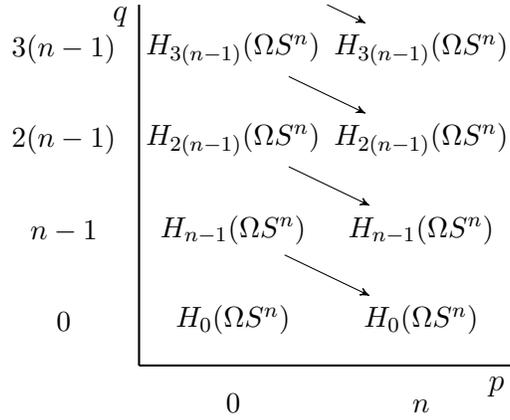
Figure 5.4: $E_n^{p,q}$ for the path fibration of S^n.

using Lemma 5.6.1. Therefore, the only nontrivial groups are in the columns $p = 0$ and $p = n$. This means that by looking at the degree of the differentials, the only nonzero differentials can occur in page n, as shown in Figure 5.4. Because all later differentials are trivial, $E_{n+1} = E_\infty$, which is depicted in Figure 5.2. This means that all differentials on page n from the $p = 0$ column to the $p = n$ column must be isomorphisms, except for the differential from $(0,0)$ to $(n, -(n-1))$. Hence we can conclude by induction that

$$H^k(\Omega\mathbb{S}^n) \begin{cases} \mathbb{Z} & \text{if } n-1 \mid k \\ 0 & \text{otherwise.} \end{cases} \tag{5.6.5}$$

As a generalization of Example 5.6.3, we can construct the *Gysin sequence* from the Serre spectral sequence [HP13, Theorem 3.3.3]. The Gysin sequence for homology states that if $f : E \to B$ is a pointed map with fiber $\mathbb{S}^{n-1}$ for $n \geq 2$ and if B is simply connected, then there exists a long exact sequence

$$\cdots \to H_i(E) \to H_i(B) \to H_{i-n}(B) \to H_{i-1}(E) \to \cdots .$$

There is also an analogue for cohomology, which states that under the same assumptions there exists a long exact sequence of cohomology groups

$$\cdots \to H^{i-1}(E) \to H^{i-n}(B) \to H^i(B) \to H^i(E) \to \cdots .$$

The proof given in [HP13, Theorem 3.3.3] works the same in HoTT. An alternative construction of the Gysin sequence in HoTT is given in [Bru16, Section 6.1], which was used as a main ingredient to compute $\pi_4(\mathbb{S}^3)$.

We can also generalize Example 5.6.4 to get the *Wang sequence*. For homology this states that if $E \to \mathbb{S}^n$ is a pointed map for $n \geq 2$ with fiber F, then there exists a long exact sequence

$$\cdots \to H_i(F) \to H_i(E) \to H_{i-n}(F) \to H_{i-1}(F) \to \cdots .$$

128

Again, a similar long exact sequence holds for cohomology, and the proof given in [HP13, Theorem 3.3.6] works the same in HoTT.

As another application, we can prove the Hurewicz theorem from the Serre spectral sequence [HP13]. The Hurewicz theorem only holds for homology, and requires the Serre spectral sequence for homology. The theorem states that if X is a simply connected pointed type, $n \geq 2$ and $\pi_q(X)$ is trivial for $q < n$, then $H_q(X) = 0$ for $q < n$ and $H_n(X) = \pi_n(X)$. For $n = 1$ the Hurewicz theorem states that for a 0-connected pointed type X, the first homology group $H_1(X)$ is the abelianization of $\pi_1(X)$. In the proof given in the aforementioned reference, the case for $n = 1$ needs to be proven separately, but then the case for $n \geq 2$ follows from that using the Serre spectral sequence. Since the case for $n = 1$ seems easier than the general case, this should be very helpful to prove the Hurewicz theorem in HoTT.

An application for the Atiyah-Hirzebruch spectral sequence would be to compute cohomology groups of generalized cohomology theories. One such generalized cohomology theory is K-theory. Although K-theory has not been precisely defined yet in HoTT, one possible idea by Ulrik Buchholtz is to define it using Snaith's theorem [Sna81]. If we have defined K-theory, we could try to compute its cohomology groups using the Atiyah-Hirzebruch spectral sequence. With the current machinery, we can compute the cohomology groups of all types that satisfy weak pointed choice (cf. Definition 5.4.14), which probably includes all finite CW-complexes.

An application that is probably trickier in HoTT is the Serre class theorem. A *Serre class* is a class $\mathcal{C}$ of abelian groups such that for every short exact sequence $0 \to A \to B \to C \to 0$ of abelian groups we have $B \in \mathcal{C}$ iff $A, C \in \mathcal{C}$. In particular, any Serre class is closed under taking subgroups and quotient groups. Classical examples of Serre classes include

- finite abelian groups;

- finitely generated abelian groups;

- torsion abelian groups.

However, constructively, the first two classes are not closed under either taking subgroups or quotient groups (torsion abelian groups do form a Serre class constructively).

The Serre class theorem is a theorem about certain Serre classes that satisfy some extra properties. This include the three examples mentioned above. If $\mathcal{C}$ is such a Serre class and if X is a simply connected type,[29] then the theorem states that $\pi_n(X) \in \mathcal{C}$ for all n iff $H_n(X) \in \mathcal{C}$ for all n. More general is the Hurewicz theorem modulo a Serre class, which states that if $\pi_i(X) \in \mathcal{C}$ for all $i < n$, then the kernel and the cokernel of the Hurewicz homomorphism $h : \pi_n(X) \to H_n(X)$ belong to $\mathcal{C}$.

As a corollary of the Serre class theorem, we know that the homotopy groups of the spheres are finitely generated, since their homology groups are finitely generated.

---

[29]or path-connected and *abelian*. A space X is abelian if the action of $\pi_1(X)$ on $\pi_n(X)$ is trivial for all $n \geq 1$.

Moreover, the homotopy groups of simply connected finite CW-complexes are also finitely generated, using cellular cohomology [BH18]. A classical proof of the Serre class theorem can be found in [Hat04, Section 1.1].

It is not straightforward to adapt the proof of the Serre class theorem to a proof in HoTT. One difficulty is that the classical proof uses the *universal coefficient theorem* for homology. This theorem is not yet proven in HoTT. The universal coefficient theorem relates the homology group $H_n(X; A)$ with coefficients in any abelian group A to the the homology group $H_n(X)$ with integer coefficients. There is also a dual universal coefficient theorem for cohomology that relates the cohomology group $H^n(X; A)$ with the homology group $H_n(X)$. It is not clear how to prove or even formulate the universal coefficient theorem in HoTT. The universal coefficient theorem for homology uses the Tor functor, whose definition requires projective resolutions. Similarly, the universal coefficient theorem for cohomology uses the Ext functor, whose definition requires injective resolutions. Basic properties of projective and injective resolutions are classically proven with the axiom of choice [Bla79], so it is not clear whether we can prove the universal coefficient theorem without the axiom of choice. Another problem with proving the universal coefficient theorem is that classically it is proven algebraically for chain complexes. Since homology and cohomology groups of spaces are defined as the (co)homology of a chain complex, the universal coefficient theorem then applies to spaces. Even if we could solve the issues with the axiom of choice, and we could prove the universal coefficient theorem for chain complexes in HoTT, it does not directly follow that it is true for spaces, since these groups are *not* defined as the (co)homology of chain complexes.[30] Therefore, in HoTT, it seems fruitful to prove the universal coefficient theorem directly for spaces, using the definition in terms of Eilenberg-MacLane spaces, but this is an open problem as of now. A good first step might be to try to prove a special case of the universal coefficient theorem where the Tor and Ext functors vanish, although that will not be sufficient to prove the Serre class theorem.

Proving the Serre class theorem in HoTT will be tricky, and it might be necessary to reformulate or weaken some notions to get a usable result in HoTT. If we manage to prove these results in HoTT, we can get a lot of information about the homotopy groups of spheres. One additional ingredient that is required is the fact that the cup product structure of the cohomology groups respect the Serre spectral sequence. From these ingredients we can classically show the following:

- The groups $\pi_i(\mathbb{S}^n)$ are finite for $i > n$, except for $\pi_{4k-1}(\mathbb{S}^{2k})$, which are the direct sum of $\mathbb{Z}$ and a finite group [Hat04, Theorem 1.21].

- For a prime p the p-torsion subgroup of $\pi_i(\mathbb{S}^3)$ is 0 for $i < 2p$ and $\mathbb{Z}_p$ for $i = 2p$ [Hat04, Example 1.18].

- From the two above results we can immediately conclude that $\pi_4(\mathbb{S}^3) = \mathbb{Z}_2$.

---

[30]Cellular (co)homology [BH18] is defined as the (co)homology of a chain complex, and therefore we could prove it for finite CW complexes, but it would not follow for arbitrary types.

- Using additionally the localization of a space at a prime, we can show that for p a prime, the p-torsion subgroup of $\pi_i(\mathbb{S}^{n+3})$ is 0 for $i < n + 2p$ and $\mathbb{Z}_p$ for $i = n + 2p$ [Hat04, Theorem 1.28].

- We can compute more homotopy groups of spheres using significantly more machinery. For this we need the EHP sequence, Steenrod squares and Serre's theorem, which computes the cohomology rings of $K(\mathbb{Z}_2, n)$ and $K(\mathbb{Z}, n)$ and $K(\mathbb{Z}_{2^k}, n)$ with coefficients in $\mathbb{Z}_2$. If we have all these results, we can compute $\pi_{n+i}(\mathbb{S}^n)$ for all n and $i \leq 3$ [Hat04, Theorem 1.40].

Conclusion

In this dissertation I have shown that homotopy type theory is a practical language to prove involved theorems in homotopy theory, most notably the construction of two important spectral sequences: the Atiyah-Hirzebruch and the Serre spectral sequences for cohomology. The discovery of these spectral sequences in classical homotopy theory was an important milestone, and we expect that the corresponding proof in HoTT will lead to many useful corollaries in synthetic homotopy theory. That said, many applications of these spectral sequences require more machinery, such as the universal coefficient theorem, Serre classes and the Hurewicz theorem. The first two of these three results might be problematic to prove in HoTT, because of their dependence on the axiom of choice. I hope that an adapted or weaker version of these theorems can be found, which avoids the use of the axiom of choice, or alternatively, that when looking at their applications, we can avoid the use of choice. For example, we cannot prove constructively that finitely generated abelian groups form a Serre class, but it is conceivable that we can still prove that all homotopy groups of spheres are finitely generated without resorting to the axiom of choice. That said, we could also assume the axiom of choice and continue proving results in synthetic homotopy theory using it. However, then the resulting theorem would not hold anymore in all models of HoTT.

It would be interesting to see other spectral sequences proven in HoTT, such as the Adams spectral sequence and the Eilenberg-Moore spectral sequences [Hat04].

Thoughts on formalization

This dissertation also shows that homotopy type theory provides a good language for the computer formalization of results in homotopy theory.

Through the formal methods community there is a strong desire that formal methods will be adopted in a large scale by general mathematicians. The main bottlenecks for this adoption are

(i) the necessary expertise of formalization in the proof assistant of choice;

(ii) the vast number of proof assistant in existence;

(iii) the amount of work it takes to formalize mathematics compared to writing it on paper.

It definitely takes time to learn a proof assistant, familiarize oneself with the library and get enough practice to use a proof assistant efficiently. Moreover, in my experience, learning to use a proof assistant takes longer than learning to use other programs, like LaTeX or Mathematica. Still, I do not think this is the main bottleneck to the adoption of proof assistants. Various courses that integrate the use of proof assistants have been taught, and students taking those courses will get a level of proficiency of using that proof assistant.

The second concern is the number of proof assistants in existence, each with a separate library and the near-impossibility to translate theorems and proofs between two proof assistants. There are translation procedures between some proof assistants, such as [McL06], but such translations are often incomplete, and only specific to two proof assistants.

However, I think the main bottleneck is the amount of extra time it takes to formalize mathematics compared to writing a paper proof. Rough estimates for the formalization time is about one week to formalize a page of a mathematical paper or textbook [AC10]. My experience with formalizing synthetic homotopy theory specifically is a little different. Paper proofs given in synthetic homotopy theory are often quite detailed, and the techniques used are often very close to the underlying type theory. I would argue that this is necessary; we do not have much experience with proving theorems in synthetic homotopy theory yet, and it is not always clear which results are hard to prove. Some results turn out more difficult to prove than initially thought. For example, the "basic property" of the smash product that it forms a 1-coherent symmetric monoidal product (see Section 4.3) was assumed with a vague proof sketch in [Bru16] to prove $\pi_4(\mathbb{S}^3) = \mathbb{Z}_2$, but this result is still open as of now. Another example is Theorem 3.3.26, which was originally thought to be a basic result about colimits by Egbert Rijke and me, but the proof turned out to be much harder than expected.

Because paper proofs in synthetic homotopy theory are often proven with many details, in my experience, giving a fully formal proof is not much more work. In cases where the formal proof is a lot more work, the paper proof sometimes omitted showing the case of the path constructor when inducting over a HIT, which is the hardest — but least enlightening — part of the proof. In the paper proofs of this dissertation, I have also sometimes omitted these steps, because they are tedious to work through and not very enlightening. However, the formal proofs (of course) contain all the details. For some theorems the formalization did take substantially more work. For the formalization of spectral sequences, a substantial algebra library had to be developed, consisting of basic group theory, ring theory, modules over a ring and graded modules. This took many man-hours of work, which would have no counterpart in a paper proof.

Another reason why formalization is more work, is that necessarily such proofs have to be encoded in the corresponding logic, intensional type theory. Most of the time, this is straightforward, but in some cases it takes a bit more work. Especially when dealing with sequences of types, in intensional type theory one has to work explicitly with transports (or its relatives, like pathovers or heterogenous equality), which is especially laborious in the proof-relevant setting of HoTT. Sometimes an "encoding trick" is useful

when dealing with these dependent types. In this dissertation some of these tricks have been given. In Section 4.1.1 we defined a chain complex over an arbitrary successor structure, because we wanted to not only index chain complexes over $\mathbb{N}$ or $\mathbb{Z}$, but also over $\mathbb{N} \times \mathsf{fin}_3$ or similar types, to get a more convenient computational content. These successor structures also turned out to be useful for spectra in Section 5.3, so that we can apply the same notion to spectra indexed over $\mathbb{N}$ and spectra indexed over $\mathbb{Z}$. The reason that spectra indexed over $\mathbb{Z}$ are useful (traditionally they are only indexed over $\mathbb{N}$) is that for certain definitions, such as the homotopy group of a spectrum, no case-splits are required when they are indexed over $\mathbb{Z}$. Another encoding trick was given in the definition of graded morphisms. In order to define the composition of graded morphisms more easily, we defined the degree of a graded morphism to be an automorphism of the indexing set. In order to avoid dealing with transports everywhere, we defined a graded morphism to act on a path in the indexing set, see Section 5.1.

Formalizing in Lean is a fun activity, and Lean is a good language for formalization. In Lean 2 one of the main annoyances when formalizing was the unpredictability of the elaborator, which was greatly improved in Lean 3. Another issue was the ability to simplify expressions. There was a tactic `esimp` that simplified by evaluation, but it was quite slow, and would sometimes use 99% of the elaboration time of a proof. I do not have enough experience with `dsimp` in Lean 3 to see whether it has similar issues.

In 2016 Leonardo de Moura decided that he would stop supporting homotopy type theory in Lean 3. It was quite devastating to hear this. I am glad that Gabriel Ebner has found a method to do homotopy type theory in Lean 3 safely, by avoiding the use of `Prop`. Since then, I have been slowly working on porting the Lean 2 HoTT library to Lean 3, although the progress has been slow. The main reasons for this are:

- The elaborator in Lean 3 is weaker to make it more robust, which causes many proofs to break.

- Some tactics do not work without using `Prop` or the `Prop`-valued equality. Gabriel Ebner has modified the `simp` and `rewrite` tactic to work in HoTT. I have written an `induction`-tactic, since the default induction tactic does not allow custom induction principles to eliminate to only non-`Prop` sorts.

- The notation ! has been removed in Lean 3. This was used in Lean 2 to turn (some) explicit arguments into implicit ones.

- There are many small differences in Lean 2 and Lean 3 in syntax for tactics, proof styles, attributes, universe levels and declarations. None of these issues take much time to fix, but the sheer number of them add up.

Despite this, a significant part of the library has been ported, and I am planning to continue this so that Lean 3 (and later Lean 4) can be used to formalize results in homotopy type theory.

The current implementation in Lean is probably not the ultimate proof assistant for HoTT in the long-term. Many cubical type theories have been developed over the last

few years, and a few proof assistants have been developed using a cubical type theory as their underlying logic. Cubical type theory offers many advantages when reasoning about higher inductive types and when doing higher path algebra, since more relations hold strictly. For example, the computation rule of the induction principle for a higher inductive type holds judgmentally in cubical type theory. This is very convenient when working with HITs, especially HITs with higher path constructors. It is conceivable that a cubical type theory can be implemented in Lean, although it will require some hacking in the `C++` code, and many features of Lean will need to be modified to work well with the cubical structure. This will be a big project, and it is probably smart not do this project until the different variants of cubical type theory have been studied more. In particular, current versions of cubical type theory do not satisfy regularity, which states that the induction principle for paths has judgmental computation rules.[31] Some constructions in HoTT are done by doing a long string of path inductions, and such proofs will be harder to reason with in cubical type theory. That said, it would be interesting to perform some constructions of this dissertation in one of the cubical type theories to see whether the proof would significantly simplify. In particular the proofs in Section 3.2 would simplify when the induction principle of higher inductive types reduces definitionally when applied to path constructors.

---

[31]It is possible to have two notions of paths: a path type with all the cubical structure, and an identity type with an induction principle and a judgmental computation rule. However, in current cubical type theories these cannot be the same type.

Acknowledgements

First and foremost I would like to thank my advisor Jeremy Avigad, who was always ready to give useful feedback, proofread drafts of all my written work and provide support. Futhermore, I would like to thank Steve Awodey for always being ready to answer any questions I have about HoTT or category theory. I would like to thank Mike Shulman for many helpful remarks and insights whenever I show my work. I also want to thank Ulrik Buchholtz, Egbert Rijke, Jakob von Raumer, Stefano Piceghello and Kristina Sojakova for the collaborations and discussions. I am grateful towards Leonardo de Moura for all his help with getting me up to speed with Lean, and answering all my stupid questions and ideas I brought up early in the development of Lean. I would like to thank Marc Bezem and Dan Christensen to invite me for academic visits. More generally, I would like to thank everyone in the HoTT community for maintaining such a good research community. It is very nice to be part of such a friendly and collaborative research community, where it is normal to have unfinished projects on Github or discuss half-baked ideas on a mailing list.

For moral support, I would like to thank my parents, Peter van Doorn and Judith van Wakeren, for supporting me during times when I was struggling. Dank jullie wel! Lastly I would like to thank Cecilia Hornberger for the moral support over the last months.

Bibliography

[AB04] Steve Awodey and Andrej Bauer, *Propositions as [Types]*, Journal of Logic and Computation **14** (2004), no. 4, 447–471.

[AC10] Andrea Asperti and Claudio Sacerdoti Coen, *Some considerations on the usability of interactive provers*, International Conference on Intelligent Computer Mathematics, Springer, 2010, pp. 147–156.

[ACD⁺16] Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg, *Quotient inductive-inductive types*, ArXiv e-prints (2016), `arXiv:1612.02346`.

[AH61] Michael F Atiyah and Friedrich Hirzebruch, *Vector bundles and homogeneous spaces*, Differential geometry, Proceedings of Symposia in Pure Mathematics, no. 3, 1961, pp. 7–38.

[AHW17] Carlo Angiuli, Robert Harper, and Todd Wilson, *Computational higher-dimensional type theory*, POPL '17: Proceedings of the 44th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, 2017, `doi:10.1145/3009837.3009861`.

[AKL15] Jeremy Avigad, Chris Kapulkin, and Peter LeFanu Lumsdaine, *Homotopy limits in type theory*, Mathematical Structures in Computer Science **25** (2015), no. 05, 1040–1070.

[AW09] Steve Awodey and Michael A. Warren, *Homotopy theoretic models of identity types*, Math. Proc. Camb. Phil. Soc., vol. 146, Cambridge Univ Press, 2009, pp. 45–55.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber, *A model of type theory in cubical sets*, 19th International Conference on Types for Proofs and Programs (TYPES 2013), vol. 26, 2014, pp. 107–128.

[BGLL⁺16] Andrej Bauer, Jason Gross, Peter LeFanu LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters, *The HoTT Library: A formalization of homotopy type theory in Coq*, ArXiv e-prints (2016), `arXiv:1610.04591`.

[BH18]   Ulrik Buchholtz and Kuen-Bang Hou (Favonia), *Cellular Cohomology in Homotopy Type Theory*, ArXiv e-prints (2018), `arXiv:1802.02191`.

[BHC⁺]   Guillaume Brunerie, Kuen-Bang Hou (Favonia), Evan Cavallo, Eric Finster, Jesper Cockx, Christian Sattler, Chris Jeris, Michael Shulman, et al., *Homotopy type theory in Agda*, `https://github.com/HoTT/HoTT-Agda`.

[Bla79]   Andreas Blass, *Injectivity, projectivity, and the axiom of choice*, Transactions of the American Mathematical Society **255** (1979), 31–59.

[BR16]   Ulrik Buchholtz and Egbert Rijke, *The Cayley-Dickson construction in Homotopy Type Theory*, ArXiv e-prints (2016), `arXiv:1610.01134`.

[Bru16]   Guillaume Brunerie, *On the homotopy groups of spheres in homotopy type theory*, Ph.D. thesis, University of Nice Sophia Antipolis, 2016, `https://arxiv.org/abs/1606.05916`.

[BvDR18] Ulrik Buchholtz, Floris van Doorn, and Egbert Rijke, *Higher Groups in Homotopy Type Theory*, ArXiv e-prints (2018), `arXiv:1802.04315`.

[Car18]   Mario Carneiro, *The type theory of Lean*, 2018, online, `https://github.com/digama0/lean-type-theory/releases`.

[Cav15]   Evan Cavallo, *Synthetic cohomology in homotopy type theory*, Master's thesis, Carnegie Mellon University, 2015, `http://www.cs.cmu.edu/~ecavallo/works/thesis.pdf`.

[CCHM]   Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg, *Cubical type theory*, code library, `https://github.com/mortberg/cubicaltt`.

[CCHM16] _____, *Cubical type theory: a constructive interpretation of the univalence axiom*, November 2016, `arXiv:1611.02108`.

[CF58]   Haskell B. Curry and Robert Feys, *Combinatory logic vol. i*.

[Cis14]   Denis-Charles Cisinski, *Univalent universes for elegant models of homotopy types*, ArXiv preprint arXiv:1406.0058 (2014).

[dMKA⁺15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer, *The Lean Theorem Prover (system description)*, CADE-25 (2015), 378–388.

[Dyb94]   Peter Dybjer, *Inductive families*, Formal aspects of computing **6** (1994), no. 4, 440–465.

[EK66]   Samuel Eilenberg and G. Max Kelly, *Closed categories*, Proceedings of the Conference on Categorical Algebra, Springer, 1966, pp. 421–562.

[EM45]   Samuel Eilenberg and Saunders MacLane, *Relations between homology and homotopy groups of spaces*, Annals of mathematics (1945), 480–509.

[EUR⁺17] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura, *A metaprogramming framework for formal verification*, Proc. ACM Program. Lang. **1** (2017), no. ICFP, 34:1–34:29, `doi:10.1145/3110278`.

[G⁺13]   Georges Gonthier et al., *A machine-checked proof of the odd order theorem*, pp. 163–179, Springer, 2013, `doi:10.1007/978-3-642-39634-2_14`.

[Gir72]   Jean-Yves Girard, *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*, Ph.D. thesis, Université Paris Diderot, 1972.

[GMM06]   Healfdene Goguen, Conor McBride, and James McKinna, *Eliminating dependent pattern matching*, Algebra, Meaning, and Computation (2006), 521–540.

[Gon05]   Georges Gonthier, *A computer-checked proof of the four colour theorem*.

[Gra17]   Robert Graham, *Synthetic Homology in Homotopy Type Theory*, ArXiv e-print 1706.01540 (2017), `https://arxiv.org/abs/1706.01540`.

[H⁺]   John Harrison et al., *The hol light theorem prover*, `https://github.com/jrh13/hol-light`.

[H⁺17]   Thomas Hales et al., *A formal proof of the Kepler conjecture*, Forum of Mathematics, Pi **5** (2017), `doi:10.1017/fmp.2017.1`.

[Hat04]   Allen Hatcher, *Spectral sequences in algebraic topology*, Unpublished book, 2004, `https://www.math.cornell.edu/~hatcher/SSAT/SSATpage.html`.

[HFLL16] Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter Le-Fanu Lumsdaine, *A mechanization of the Blakers-Massey connectivity theorem in Homotopy Type Theory*, Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, ACM, 2016, pp. 565–574.

[Hou17]   Kuen-Bang Hou (Favonia), *Higher-dimensional types in the mechanization of homotopy theory*, Ph.D. thesis, Carnegie Mellon University, 2017.

[How80]   William A. Howard, *The formulae-as-types notion of construction*, To H.B. Curry: essays on combinatory logic, lambda calculus and formalism **44** (1980), 479–490.

[HP13] Maximilien Holmberg-Péroux, *The serre spectral sequence*, preprint (2013), http://homepages.math.uic.edu/~mholmb2/serre.pdf.

[HS98] Martin Hofmann and Thomas Streicher, *The groupoid interpretation of type theory*, Twenty-five years of constructive type theory (Venice, 1995), Oxford Logic Guides, vol. 36, Oxford Univ. Press, New York, 1998, pp. 83–111.

[HS16] Kuen-Bang Hou (Favonia) and Michael Shulman, *The Seifert-van Kampen theorem in homotopy type theory*, 25th EACSL Annual Conference on Computer Science Logic (CSL 2016), Leibniz International Proceedings in Informatics (LIPIcs), vol. 62, 2016, pp. 22:1–22:16, doi:10.4230/LIPIcs.CSL.2016.22.

[KECA14] Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch, *Notions of anonymous existence in Martin-Löf type theory*, Submitted to the special issue of TLCA'13 (2014).

[KL12] Chris Kapulkin and Peter LeFanu Lumsdaine, *The Simplicial Model of Univalent Foundations (after Voevodsky)*, ArXiv e-prints (2012), arXiv:1211.2851.

[Kra15] Nicolai Kraus, *The general universal property of the propositional truncation*, 20th International Conference on Types for Proofs and Programs (TYPES 2014), Leibniz International Proceedings in Informatics (LIPIcs), vol. 39, 2015, pp. 111–145, doi:10.4230/LIPIcs.TYPES.2014.111.

[Kra16] _____, *Constructions with non-recursive higher inductive types*, Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, ACM, 2016, pp. 595–604.

[LF14] Daniel R. Licata and Eric Finster, *Eilenberg-MacLane spaces in homotopy type theory*, Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), ACM, 2014, p. 66.

[Lic11] Daniel R. Licata, *Running circles around (in) your proof assistant; or, quotients that compute*, blog post, April 2011, http://homotopytypetheory.org/2011/04/23/running-circles-around-in-your-proof-assistant/.

[LS13] Daniel R. Licata and Michael Shulman, *Calculating the fundamental group of the circle in homotopy type theory*, 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013), IEEE Computer Soc., Los Alamitos, CA, 2013, pp. 223–232.

[LS17] Peter LeFanu Lumsdaine and Michael Shulman, *Semantics of higher inductive types*, ArXiv e-prints (2017), `arXiv:1705.07088`.

[Lum11] Peter LeFanu Lumsdaine, *Higher inductive types: a tour of the menagerie*, blog post, April 2011, `https://homotopytypetheory.org/2011/04/24/higher-inductive-types-a-tour-of-the-menagerie/`.

[Luo12] Zhaohui Luo, *Notes on universes in type theory*, preprint, 2012, `http://www.cs.rhul.ac.uk/home/zhaohui/universes.pdf`.

[Mas52] William S Massey, *Exact couples in algebraic topology (parts i and ii)*, Annals of Mathematics **56** (1952), no. 2, 363–396.

[McL06] Sean McLaughlin, *An interpretation of Isabelle/HOL in HOL Light*, International Joint Conference on Automated Reasoning, Springer, 2006, pp. 192–204.

[ML75] Per Martin-Löf, *An intuitionistic theory of types: Predicative part*, Studies in Logic and the Foundations of Mathematics, vol. 80, Elsevier, 1975, pp. 73–118.

[ML84] _____, *Intuitionistic type theory*, Bibliopolis, 1984, Notes by Giovanni Sambin of a series of lectures given in Padova.

[Rez14] Charles Rezk, *Proof of the blakers-massey theorem*, 2014, `http://www.math.uiuc.edu/~rezk/freudenthal-and-blakers-massey.pdf`.

[Rij17] Egbert Rijke, *The join construction*, ArXiv (2017), `arXiv:1701.07538`.

[RSS17] Egbert Rijke, Michael Shulman, and Bas Spitters, *Modalities in homotopy type theory*, ArXiv e-prints (2017), `arXiv:1706.07526`.

[S+11] Michael Shulman et al., *higher inductive type*, 2011, nLab article, `https://ncatlab.org/nlab/revision/higher+inductive+type/31`.

[Ser51] Jean-Pierre Serre, *Homologie singulière des espaces fibrés*, Annals of Mathematics (1951), 425–505.

[Shu11a] Michael Shulman, *A formal proof that $\pi_1(s^1) = \mathbb{Z}$*, blog post, April 2011, `https://homotopytypetheory.org/2011/04/29/a-formal-proof-that-pi1s1-is-z/`.

[Shu11b] _____, *Homotopy type theory, vi*, forum post, April 2011, `https://golem.ph.utexas.edu/category/2011/04/homotopy_type_theory_vi.html`.

[Shu11c] _____, *An interval type implies function extensionality*, blog post, April 2011, `https://homotopytypetheory.org/2011/04/04/an-interval-type-implies-function-extensionality/`.

[Shu13] _____, *Spectral sequences in HoTT*, blog posts, August 2013, `https://ncatlab.org/homotopytypetheory/revision/spectral+sequences/5`.

[Shu17] _____, *Elementary $(\infty, 1)$-topoi*, blog post, April 2017, `https://golem.ph.utexas.edu/category/2017/04/elementary_1topoi.html`.

[Sna81] Victor Snaith, *Localized stable homotopy of some classifying spaces*, Mathematical Proceedings of the Cambridge Philosophical Society, vol. 89, Cambridge University Press, 1981, pp. 325–330.

[Str14] Thomas Streicher, *A model of type theory in simplicial sets: A brief introduction to Voevodsky's homotopy type theory*, Journal of Applied Logic **12** (2014), no. 1, 45 – 49, Logic Categories Semantics, `doi:https://doi.org/10.1016/j.jal.2013.04.001`.

[The18] The RedPRL Development Team, *RedPRL – the People's Refinement Logic*, 2018, `http://www.redprl.org/`.

[Uni13] The Univalent Foundations Program, *Homotopy type theory: Univalent foundations of mathematics*, `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[VAG$^+$] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al., *UniMath — Univalent Mathematics*, code library, `https://github.com/UniMath`.

[vD16] Floris van Doorn, *Constructing the propositional truncation using non-recursive hits*, Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, ACM, 2016, pp. 122–129.

[vDvRB17] Floris van Doorn, Jakob von Raumer, and Ulrik Buchholtz, *Homotopy type theory in lean*, pp. 479–495, Springer, 2017, `doi:10.1007/978-3-319-66107-0_30`.

[vG12] Benno van den Berg and Richard Garner, *Topological and simplicial models of identity types*, ACM transactions on computational logic (TOCL) **13** (2012), no. 1, 3.

[Voe06] Vladimir Voevodsky, *A very short note on the homotopy λ-calculus*, online, 2006, `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/Hlambda_short_current.pdf`.

[Voe09] _____, *Notes on type systems*, online, 2009, `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/expressions_current_1.pdf`.

[Voe14] _____, *The equivalence axiom and univalent models of type theory. (Talk at CMU on February 4, 2010)*, `arXiv:1402.5556`.

[Voe15] ______, *Oxford lectures on UniMath*, filmed by Kohei Kishida, available at https://www.math.ias.edu/vladimir/Lectures, 2015.

[vR16] Jakob von Raumer, *Formalizing double groupoids and cross modules in the lean theorem prover*, Mathematical Software – ICMS 2016, Springer International Publishing, 2016, pp. 28–33.

[Zha17] Bohua Zhan, *Formalization of the fundamental group in untyped set theory using auto2*, International Conference on Interactive Theorem Proving, Springer, Springer, 2017, pp. 514–530, doi:10.1007/978-3-319-66107-0_32.