Prospectus

# On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory

Floris van Doorn

May 12, 2017

# Contents

# Chapter 1

# Introduction

Homotopy Type Theory (HoTT) is a dependent type theory with a homotopical interpretation [AW09, Voe06], which is inspired by the groupoid model [HS98]. In HoTT a type can be viewed as a space up to homotopy, and type-theoretical constructors correspond to homotopy-invariant constructions on spaces. In particular, the identity type corresponds to path spaces. We can define homotopy-invariant notions from algebraic topology in HoTT and prove theorems from algebraic topology synthetically internally in the language of HoTT. This gives a fully homotopy-invariant proof which is true in all models of HoTT. Just like extensional type theory can be interpreted in various categories, such as elementary toposes, HoTT can be interpreted in various higher toposes, such as simiplicial or cubical sets. This makes proofs in HoTT more general than the homotopy theoretic counterpart. Moreover, these proofs are constructive, and can be formally verified by a proof assistant in practice. There are various libraries for HoTT in Coq [BGL+16, VMA+17], Agda [BHC+] and the experimental proof assistant cubicaltt [CCHM]. In this prospectus we will describe the formalization of HoTT in a young proof assistant Lean [dMKA+15].

In HoTT, we can define a notion of *higher inductive types* (HITs) [Uni13] which generalize both inductive types in type theory and cell-complexes in homotopy theory. They can be interpreted as inductive types where we can specify not only point constructors, but also path and higher path constructors, or as cell-complexes where the points and paths can be recursive.

We will discuss some new results (or in some cases, new formalizations of existing results) in HoTT. The results can be divided as either research about higher inductive types in Chapter 3, and synthetic homotopy theory in Chap-

ter 4. The results on higher inductive types are focused on reducing many HITs to quotients internally in HoTT. This is important, since the semantics for general HITs is still an open problem. So if we can reduce a large class of HITs to quotients, we only have to interpret quotients in a model to get an interpretation of the large class of HITs. In fact, most of the HITs used in practice can be reduced to quotients. The results in synthetic homotopy theory are proving various properties known in homotopy theory in HoTT. We give a fully formalized proof of $\pi_3(\mathbb{S}^2) = \mathbb{Z}$ using the long exact sequence of the Hopf fibration. This was known before in HoTT, but no formalization has been given before. Furthermore, we prove various properties about Eilenberg-MacLane spaces and the smash product. These are needed to define homology and cohomology groups. Finally, we describe a construction of the Serre Spectral Sequence.
In this document I will describe the research I've done so far, and describe my plans for research for the next year. The document is written as an outline for my dissertation. The sections here correspond to the planned sections in my dissertation. Many proofs will be omitted or incomplete, and in the preliminaries chapter (Chapter 2) I will only describe what concepts I'll introduce. At the end of each section in Chapters 3 and 4 I will describe what parts are done, what parts are in progress, and what is planned for the next year.

# Chapter 2

# Preliminaries

This chapter gives a introduction to homotopy type theory and Lean.

## 2.1 Martin Löf Type Theory

We will describe MLTT, with $\Pi$, $\Sigma$, $\mathbb{N}$, $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, $=$, inductive types and families and universes. It will roughly correspond to chapter 1 and parts of chapter 5 of the HoTT-book. [Uni13].
We will use the following notation:

- We use $=$ for equality/identification/identity/"propositional" equality and $\equiv$ for "judgmental" or "definitional" equality. We use $x :\equiv t$ to define $x$ as $t$ and $f(p) := q$ (or $\mathsf{apd}_f(p) :\equiv q$) to define a function $f$ on a path constructor $p$ of a higher inductive type.

- We use "Agda-notation" for function types and pair types: $(a : A) \to B(a)$ and $(a : A) \times B(a)$ denote pi's and sigma's. $A \times B :\equiv (a : A) \times B$. Pairs in either type are denoted $(a, b)$. We will write $\{a : A\} \to B(a)$ if we treat $a$ as an implicit argument later

- If $f : A \to B$ and $g : C \to D$ we define $f \times g : A \times C \to B \times D$, $f + g : A + C \to B + D$.

- If $f : A \to B$ and $g : A \to C$ we define $(f, g) : A \to B \times C$.

- If $f : A \to C$ and $g : B \to C$ we define $\langle f, g\rangle : A + B \to C$.

- We denote the maps $\mathbf{0} \to A$ and $A \to \mathbf{1}$ by !.

## 2.2 Homotopy Type Theory

Here we will describe the basic notions of homotopy type theory.

### 2.2.1 Paths

Here we will describe path algebra including pathovers, squares and cubes. We will also describe ap, dependent ap and transport.

### 2.2.2 Equivalences

We will define equivalences, give some basic properties of equivalences. We will describe paths in certain types, including the univalence axiom and function extensionality.

### 2.2.3 Truncatedness

We will define truncatedness and connectedness, and the universes of truncated and connected types. This will also include or mention that suspension increases connectedness and that $\pi_k(\mathbb{S}^n) = 0$ if $k < n$.

### 2.2.4 Pointed Types

We will define pointed types, pointed maps, pointed homotopies, pointed equivalences and the fiber of a map.

### 2.2.5 Higher Inductive Types

We will introduce HITs using the interval as an example (not the circle, since it's easier to see for the interval why you need dependent paths in the induction principle).
Then we will define various other HITs.
We define the quotient (or graph quotient). Given $A : \mathcal{U}$, $R : A \to A \to \mathcal{U}$, the quotient is the following HIT.
`HIT` $\mathsf{quotient}_A(R) :\equiv$

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

The pushout is the following HIT.
`HIT` $\mathsf{pushout}(f,g) :\equiv$

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

We denote $\mathsf{pushout}(f,g)$ by $B +_A C$ if $f$ and $g$ are clear from the context.
Other higher inductive types can be defined in terms of these:

- 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 $\Sigma A$ 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 $\mathsf{N}$ and $\mathsf{S}$ and $\mathsf{glue}$ is called $\mathsf{merid}$.
- The wedge 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 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 $A + B \to A \times B$ which sends $\mathsf{inl}(a)$ to $(a, b_0)$ and $\mathsf{inr}(b)$ to $(a_0, b)$. We will discuss the smash product in Section 4.3.
- The $n$-sphere $\mathbb{S}^n$ is defined inductively for $n \geq -1$: $\mathbb{S}^{-1} :\equiv \mathbf{0}$ and $\mathbb{S}^{n+1} :\equiv \Sigma\mathbb{S}^n$. For $n \geq 0$ the $n$-sphere is pointed with point $\mathsf{N}$.

We define the $n$-truncation, given $n \geq -2$ and $A : \mathcal{U}$, as the following HIT.
`HIT` $\|A\|_n :\equiv$

- $|{-}| : A \to \|A\|_n$
- $\mathsf{is}\text{-}n\text{-}\mathsf{type}(A)$

Remark: this can be encoded more explicitly.
(sequential) colimit: given a sequence $(A, f)$ where $A : \mathbb{N} \to \mathcal{U}$ and $f : (n : \mathbb{N}) \times A(n) \to A(n+1)$
`HIT` $\mathsf{colim}(A,f) :\equiv$

- $i : (n : \mathbb{N}) \to A(n) \to \mathsf{colim}(A, f)$
- $\mathsf{glue} : (n : \mathbb{N}) \to (a : A(n)) \to i_{n+1}(f_n(a)) = i_n(a)$

## 2.3 Lean

*The contents of this section are from a paper on the Lean-HoTT library submitted to ITP 2017, written with Jakob von Raumer and Ulrik Buchholtz.*
Lean [dMKA+15] is an interactive theorem prover which is mainly developed at Microsoft Research and Carnegie Mellon University. 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 [dMERU17]. In this version, many components of Lean have been rewritten. Of note, the unification procedure has been restricted, since the full higher-order unification which 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 which requires axiom K [GMM06], the homotopy type theory mode is currently not supported in Lean 3. This has led to the situation that the homotopy type theory library is kept in the still maintained but not further developed Lean 2. In the future we hope that we will find a way to support a version of homotopy type theory in Lean 3 or a fork thereof.
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 $\beta$ and $\eta$ 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*, which 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` $a_1$ $a_2$ or $a_1$ `=` $a_2$ instead of `@eq A` $a_1$ $a_2$. 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 equivalences `f : 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))`.

# Chapter 3

# Higher Inductive Types

In this chapter we will study properties of Higher Inductive Types (HITs), which we introduced in Section 2.2.5. There is no uniformly accepted scheme of which HITs are allowed, and the semantics of HITs is a topic of current research. Sematics of special cases of Higher Inductive Types are studied in different papers [LS12, BGvdW16, Cap14]. This is not what we study in this chapter. Instead, we will work internally in a type theory which 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, which is described in Section 2.2.5. We first note that we could also have started with pushouts, by the following lemma. We have chosen quotients as our starting point, because path constructors can be defined more easy with inductive families using quotients than with the functions using pushouts. For example, the definition of sequential colimit using quotients, given below, is straightforward, the type and relation can be read off directly from the point and path constructors. A direct definition of the colimit using pushouts is less straightforward.

**Lemma 3.0.1.** *The pushout and quotient are interdefinable in MLTT. Also, the colimit can be defined using quotients.*

*Proof.* We will only give the definitions. 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 $g :\equiv \langle \mathsf{id}, \mathsf{id} \rangle : T + T \to T$.
To define $\mathsf{colim}(A, f)$ using quotients, we define it 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 $\mathsf{mk} : (n : \mathbb{N}) \to (a : A(n)) \to R(f_n(a), a)$. □

One HIT from Section 2.2.5 which we have not yet defined using quotients is the $n$-truncation. In Section 3.1 we will define the propositional truncation using quotients. We will generalize this to $n$-truncations and more general constructions is Section 3.3. An alternative construction of the $n$-truncations is given by the join construction [Rij17]. This shows that we can define certain recursive 1-HITs using quotients.
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].

## 3.1 Propositional Truncation

*The contents of this section are have been published in [vD16].*
In this section we will construct the propositional truncation from quotients. This construction was given before in [vD16].
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. Then 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:

- $k(i_0(a)) :\equiv h(a)$
- $k(i_{n+1}(f_n(a))) :\equiv (g_n(b))_*^{-1}(k(i_n(b)))$

The definition $k\ (i_0\ a) :\equiv h\ a$ is also the judgmental computation rule for the 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 $\mathsf{ap}_g : x = x' \to g(x) = g(x')$ 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 $\mathsf{ap}_g$ 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,[1] 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 $g$.

[1]Of course, we do know that it is a mere proposition after we have finished the proof that $\{A\}_\infty$ is a mere proposition.

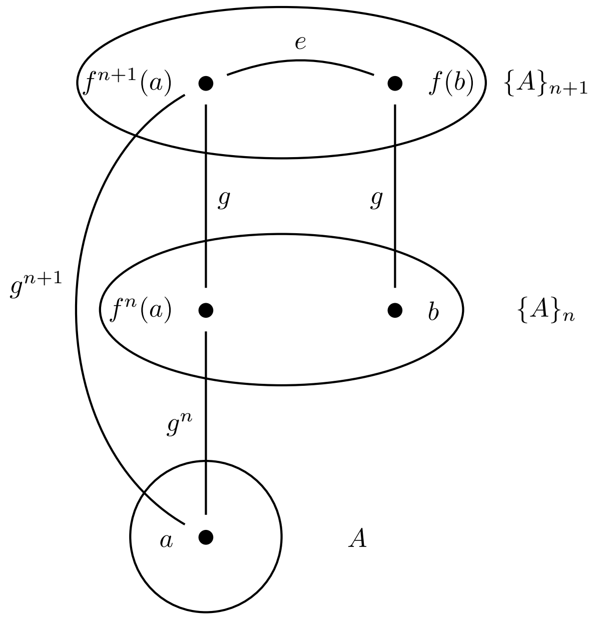

Figure 3.1: The definition of $p$. The applications of $i$ and the arguments of the paths are implicit.

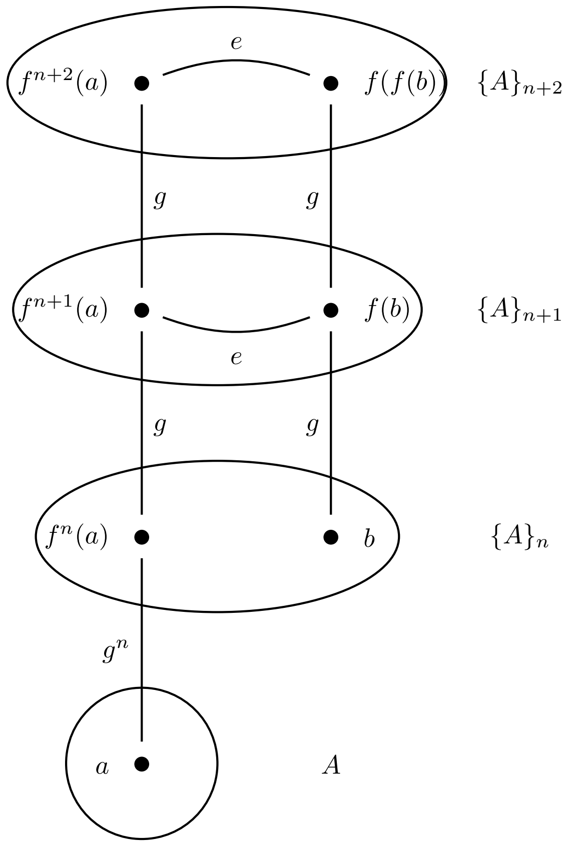

Figure 3.2: The coherence condition for $p$. The applications of $i$ and the arguments of the paths are implicit.

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

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

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

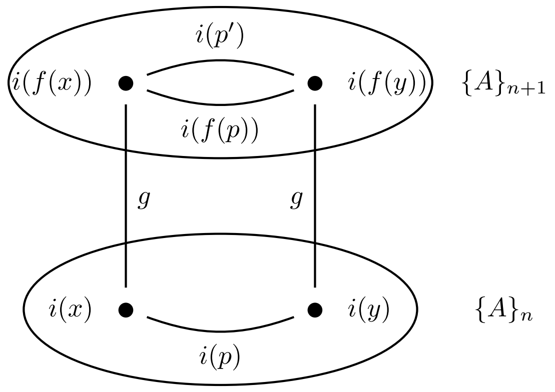

Figure 3.3: The situation in Lemma 3.1.7.

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.*

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

*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 $i_n(f^n(a)) = i_n(b)$ which 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. □

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.*

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

## 3.2 Non-recursive 2-HITs

*The contents of this section are from a paper on the Lean-HoTT library submitted to ITP 2017, written with Jakob von Raumer and Ulrik Buchholtz.*
We can also define HITs with 2-path constructors using quotients. This 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 = x` in type `A` we can define a map `f : S`$^1$ `→ A` with `ap f loop = p` by circle induction. Then we can prove the equivalence `(p = refl x)` $\simeq$ `Σ (x`$_0$ `: A), Π(z : S`$^1$`), 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 : 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 = refl x` by a new point constructor `x`$_0$ `: A` and a family of 1-path constructors `Π(z : S`$^1$`), f z = x`$_0$.
However, this does not quite define 2-HITs in terms of the quotient, since this family of path constructors refers to other path constructors (in the definition of `f`), which is not allowed in quotients. For this reason, we do the construction using two nested quotients. We first define a quotient with only the 1-paths (and the hubs), and then use another quotient to add the spokes.
More formally, we will define paths in `R` to be the following inductive family:

```
inductive paths {A} (R : A → A → Type) : A → A → Type :=
| of_rel  : Π{a a′ : A}, R a a′ → paths R a a′
| of_path : Π{a a′ : A}, a = a′ → paths R a a′
| symm    : Π{a a′ : A}, paths R a a′ → paths R a′ a
| trans   : Π{a a′ a″}, paths R a a′ → paths R a′ a″
                        → paths R a a″
```

A *specification for a (nonrecursive) 2-HIT* consists of a type `A` and two families `R : A → A → Type` and `Q : Π{a a′ : A}, paths R a a′ → paths R a a′ → Type`. Using this, we define the 2-HIT `two_quotient A R Q` with constructors

```
HIT two_quotient A R Q : Type :=
| i₀ : A → two_quotient A R Q
| i₁ : Π{a a′ : A}, R a a′ → i₀ a = i₀ a′
| i₂ : Π{a a′ : A} {r r′ : paths R a a′}, Q r r′ →
       extend i₁ r = extend i₁ r′
```

where `extend` $\texttt{i}_1$ `r` is the action of $\texttt{i}_1$ on paths in `R`, e.g. `extend` $\texttt{i}_1$ `(trans` $\texttt{r}_1$ $\texttt{r}_2$`) := extend` $\texttt{i}_1$ $\texttt{r}_1$ `·` `extend` $\texttt{i}_1$ $\texttt{r}_2$. We first define a special case with only reflexivities on the right hand side. We call this `simple_two_quotient A R Q′`, where `Q′` has type `Π(a : A), paths R a a → Type` and where

`i₂′ : Π{a} {r : paths R a a′}, Q r → extend i₁ r = refl (i₀ a)`

To define this, we first define

`X := quotient A R + Σ(a : A) (r : paths R a a), Q′ r`

We now define `simple_two_quotient A R Q′ := quotient X R′` where

```
inductive R′ : X → X → Type :=
| mk : Π{a : A} (r : paths R a a) (q : Q′ r) (x : S¹),
       R′ (f q x) (inr (a,q))
```

with `f q : S¹ → X` defined by induction so that `ap (f q) (loop) = extend (inl ∘ e) r` for `q : Q′ r`.
We now define the expected induction principle, (nondependent) recursion principle, and computation rules for this two-quotient. We did not manage to prove was the computation rule of the induction principle on 2-paths, but that is not necessary to define `simple_two_quotient A R Q′` up to equivalence. We then define the general version, `two_quotient A R Q`, to be equal to `simple_two_quotient A R Q′` where:

```
inductive Q′ : Π{a : A}, paths R a a → Type :=
| q₀ : Π{a a′ : A} {r r′ : paths R a a′},
       Q r r′ → Q′ (trans r (symm r′))
```

We then show that `two_quotient A R Q` and `trunc n (two_quotient A R Q)` have the right elimination principles and computation rules (it requires some work to show that the right computation rules of the truncated version from the untruncated version).

This allows us to define all nonrecursive HITs with point, 1-path and 2-path constructors. For example, we define the torus $\mathtt{T}^2$ `:= two_quotient unit R Q` where `R ⋆ ⋆ = bool` (giving two path constructors `p` and `q` from the basepoint to itself) and `Q` is generated by the constructor `q₀ : Q (trans ff tt) (trans ff tt)` (which gives a path `p · q = q · p`). We also define the *groupoid quotient*: For a groupoid `G` we define its quotient as `trunc 1 (two_quotient G (@hom G) Q)` where:

```
inductive Q :=
| q₀ : Π(a b c) (g : hom b c) (f : hom a b),
       Q (g ∘ f) (trans f 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`.

**Progress:** In my dissertation I will give a more detailed proof here. All results in this section have been fully formalized.

## 3.3 Colimits

*The work in this section is joint work with Egbert Rijke and Kristina Sojakova.*

We can ask whether we can use the construction of Section 3.1 can be generalized to construct other higher inductive types. 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 doesn't work in general: if a constructor is infinitary, there is no reason why the type after $\omega$ many steps is the desired type. In this section we describe that it does work for a general class of higher inductive types, the *$\omega$-compact localizations.*

Given a type $A$, families $P, Q : A \to \mathcal{U}$ and $F : \{a : A\} \to P(a) \to Q(a)$.

**Definition 3.3.1.** 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 $\ell_X$. $L_F X$ can be given as a higher inductive type with the following constructors:

```
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 $A_\infty$. 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 $\xi_X(\mathsf{glue}_n(f)) :\equiv \mathsf{glue}_n \circ f$.

**Definition 3.3.2.** A type $X$ is said to be *$\omega$-compact* if the map $\xi_X$ is an equivalence.

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

To prove this theorem, we need more machinery about sequences and sequential colimits.
Given a sequence $(A_n, f_n)$, where $A : \mathbb{N} \to \mathcal{U}$ and $f : (n : \mathbb{N}) \to A(n) \to A(n+1)$. We define $f_n^k : A(n) \to A(n+k)$ by repeatedly composing $f$'s, i.e. $f_n^0(a) :\equiv a$ and $f_n^{k+1}(a) :\equiv f_{n+k}(f_n^k(a))$. This type-checks if addition on the natural numbers is defined by induction on the second arguments, which we assume it is.

**Lemma 3.3.4.** *We have $f_{n+1}^k(f_n(x)) =^A_{\mathsf{succ\_add}(n,k)} f_n^{k+1}(x)$ for all $x : A(n)$ where $\mathsf{succ\_add}(n,k) : (n+1)+k = n+(k+1)$*

*Proof.* Trivial by induction on $k$. □

**Lemma 3.3.5.** *For a sequence* $(A_n, f_n)$*, we have* $\mathsf{shift} : A_\infty \simeq \mathsf{colim}(A_{n+1}, f_{n+1})_n$.

*Proof.* Straightforward. We will spell out the proof, because we will need underlying maps later. We define for $a : A(n)$ and $a' : A(n+1)$

$$\mathsf{shift}(i_n(a)) :\equiv i_n(f_n(a)) \qquad \mathsf{shift}(\mathsf{glue}_n(a)) := \mathsf{glue}_n(f_n(a)).$$

$$\mathsf{shift}^{-1}(i_n(a')) :\equiv i_{n+1}(a') \qquad \mathsf{shift}^{-1}(\mathsf{glue}_n(a')) := \mathsf{glue}_{n+1}(a').$$

To show that $p : (x : A_\infty) \to \mathsf{shift}^{-1}(\mathsf{shift}(x)) = x$ we define $p(i_n(a)) :\equiv \mathsf{glue}_n(a)$. If $x$ varies over $\mathsf{glue}_n(a)$ we need to fill the following square

$$\begin{tikzcd}
i_{n+2}(f^2(a)) \arrow[r, equal, "\mathsf{glue}_{n+1}(f(a))"] \arrow[d, equal, "\mathsf{shift}^{-1} \circ \mathsf{shift}(\mathsf{glue}_n(a))"'] & i_{n+1}(f(a)) \arrow[d, equal, "\mathsf{id}(\mathsf{glue}_n(a))"] \\
i_{n+1}(f(a)) \arrow[r, equal, "\mathsf{glue}_n(a)"] & i_n(a)
\end{tikzcd}$$

which is straightforward path algebra. To show that $q : (x : \mathsf{colim}(A_{n+1})_n) \to \mathsf{shift}(\mathsf{shift}^{-1}(x)) = x$ we define $q(i_n(a)) :\equiv \mathsf{glue}_n(a)$ and we have to fill a similar square which is also straightforward. $\square$

**Lemma 3.3.6.** *Given a natural transformation* $\alpha$ *of sequences* $(A_n, f_n) \to (B_n, g_n)$*, i.e. a family of maps* $\alpha_n : A_n \to B_n$ *such that the obvious squares commute. Then there is a map* $\mathsf{colim}(\alpha) : \mathsf{colim}(A) \to \mathsf{colim}(B)$*. Moreover, if* $\alpha$ *is a natural isomorphism (i.e.* $\alpha_n$ *is an equivalence for all* $n$*), then* $\mathsf{colim}(\alpha)$ *is an equivalence.*

*Proof.* Tedious, but standard proof that higher inductive types are functorial. $\square$

**Definition 3.3.7.** A *sequence* $(P_n, g_n)$ *over* $(A_n, f_n)$ consists of $P : \{n : \mathbb{N}\} \to A(n) \to \mathcal{U}$ and $g : \{n : \mathbb{N}\} \to (a : A(n)) \to P(a) \to P(f(a))$. We $(P_n, g_n)_n$ *equifibered* if $g_a$ is an equivalence for all $a : A(n)$.

Note that a equifibered sequence $(P_n, g_n)$ corresponds exactly to a family $A_\infty \to \mathcal{U}$.
Given a sequence $P \equiv (P_n, g_n)$ over $(A_n, f_n)$ we can define a new sequence

$$\Sigma_A P :\equiv ((a : A_n) \times P_n(a), (f_n, g_n))_n.$$

We can ask what the colimit of this sequence is. If $P$ is equifibered, the answer is quite easy: the sequence determines a family $A_\infty \to \mathcal{U}$, and the colimit of $\Sigma_A P$ is the total space of this family, by the flattening lemma. In the case that $P$ is not equifibered, it is more complicated, because we need to turn $P$ into an equifibered family first.
We can turn it $P$ into an equifibered sequence as follows using the *equifibrant replacement*, which we can define as

$$\mathsf{EqF}(P)_n(a) :\equiv \mathsf{colim}(P_{n+k}(f^k(a)), g_{n+k})_k.$$

Then there is an equivalence $e_n(a) : \mathsf{EqF}(P)_n(a) \simeq \mathsf{EqF}(P)_{n+1}(f(a))$ given by

$$\begin{aligned}\mathsf{colim}(P_{n+k}(f^k(a)), g_{n+k})_k &\simeq \mathsf{colim}(P_{n+(k+1)}(f^{k+1}(a)), g_{n+(k+1)})_k \\ &\simeq \mathsf{colim}(P_{(n+1)+k}(f^k(f(a))), g_{(n+1)+k})_k\end{aligned}$$

The first equivalence is given by Lemma 3.3.5 and the second equivalence is given by Lemma 3.3.6 applied to the natural isomorphism $P_{n+k+1}(f^{k+1}(a)) \simeq P_{n+1+k}(f^k(f(a)))$ which is given by transporting along the pathover of Lemma 3.3.4. Naturality is an instance of the general fact that transporting is natural. $\mathsf{EqF}(P)$ determines a family $P_\infty : A_\infty \to \mathcal{U}$ by $P_\infty(i_n(a)) :\equiv \mathsf{EqF}(P)_n(a)$ and $P_\infty$ respects the path constructor $\mathsf{glue}_n(a)$ by $e_n(a)$.
Now the total space of $P_\infty$ is the colimit of $\Sigma_A P$, as captured by the following theorem.

**Theorem 3.3.8.** *There is an equivalence*

$$\mathsf{colim}(\Sigma_A P) \simeq (x : A_\infty) \times P_\infty(x).$$

**Progress:** In my dissertation I will fill in the proof of both stated theorems. The formalization of the results in this section is ongoing. Large parts of Theorem 3.3.8 have been formalized, and Kristina Sojakova has a good trick to fill in the last steps and minimizing the amount of 2-path algebra required. The proof that we get the localization has been given on paper by Egbert Rijke, and needs to be formalized. Theorem 3.3.8 has also been formalized in cubicaltt by Rafaël Bocquet[2].

[2] `https://github.com/mortberg/cubicaltt/blob/674fdb7566ed83c865d2b80456a4ae9e186a020c/examples/seqcolim.ctt#L385-L386`

# 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 proven in HoTT 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. In order to prove more evidence for this, it is desirable to formalize all results proven interally in HoTT.
In this chapter we will look at various topics in homotopy theory and give proofs for them in HoTT which are fully checked by the proof assistant Lean. In Section 4.1 we will describe the formalization of the proof that $\pi_3(\mathbb{S}^2) = \mathbb{Z}$. This was already known 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 how 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. Finally, in Section 4.4 we construct the Serre Spectral Sequence.

## 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 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. There have been previous formalizations of parts of this result [AKL15, Voe15, VMA+17]; 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*

$$\begin{tikzcd}
 & \vdots & \\
\pi_2(F) \arrow[r, "\pi_2(p_1)"'] & \pi_2(X) \arrow[r, "\pi_2(f)"] & \pi_2(Y) \arrow[dll, "\pi_1(\delta)"'] \\
\pi_1(F) \arrow[r, "\pi_1(p_1)"'] & \pi_1(X) \arrow[r, "\pi_1(f)"] & \pi_1(Y) \arrow[dll, "\pi_0(\delta)"'] \\
\pi_0(F) \arrow[r, "\pi_0(p_1)"'] & \pi_0(X) \arrow[r, "\pi_0(f)"] & \pi_0(Y)
\end{tikzcd}$$

*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 doesn't 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 $\mathbb{N} \times 3$ 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* over 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* 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 not required to be a set (but a pointed type). A type-valued chain complex is exact if the above property holds without any propositional truncation, i.e. if

$$(i : I) \to (a : A_{i+1}) \to f_i(a) = a_0^i \to (a' : A_{i+2}) \times f_{i+1}(a') = a.$$

*Example* 4.1.3. Some useful examples of successor structures are $(\mathbb{N}, \lambda n.\, \lambda n + 1.\,)$, $(\mathbb{N}, \lambda n.\, \lambda n - 1.\,)$ (with the convention that $0 - 1 = 0$), $(\mathbb{Z}, \lambda n.\, \lambda n + 1.\,)$ and $(\mathbb{Z}, \lambda n.\, \lambda n - 1.\,)$.
Furthermore, if $N$ is a successor structure and $n : \mathbb{N}$, then we define a successor structure on $N \times \mathsf{fin}_{n+1}$ by defining

$$S(n, i) :\equiv \begin{cases} (n+1, 0) & \text{if } i = n \\ (n, i+1) & \text{otherwise} \end{cases}$$

Note that $n + 1$ is addition in the successor structure $N$.

We now build the long exact sequence of homotopy groups in four stages:

(1) First we define the fiber sequence of $f$, by iterating the fiber, and show that this is an exact type-valued chain complex.

(2) Then we define the following sequence. The differences with the sequence in Theorem 4.1.1 is that it's not 0-truncated, and that the maps on the odd levels are negated. We show that this sequence is equivalent to the sequence in part (1).

$$\begin{tikzcd}
 & \vdots & \\
\Omega^2 F \arrow[r, "\Omega^2 p_1"'] & \Omega^2 X \arrow[r, "\Omega^2 f"] & \Omega^2 Y \arrow[dll, "-\Omega\delta"'] \\
\Omega F \arrow[r, "-\Omega p_1"'] & \Omega X \arrow[r, "-\Omega f"] & \Omega Y \arrow[dll, "\delta"'] \\
F \arrow[r, "p_1"'] & X \arrow[r, "f"] & Y
\end{tikzcd}$$

(3) Now we negate the maps from step (2), and show that the result is still an exact type-valued chain complex

(4) Finally, we 0-truncate the sequence from step (3) to prove Theorem 4.1.1.

The long exact sequence of homotopy groups is useful if we have nontrivial pointed maps between types. The Hopf fibration gives rise to one such map. This has been formalized by Ulrik Buchholtz.

**Theorem 4.1.4** (Hopf Fibration)**.** *There is a pointed map $\mathbb{S}^3 \to \mathbb{S}^2$ with fiber $\mathbb{S}^1$.*

**Corollary 4.1.5.** *$\pi_2(\mathbb{S}^2) = \mathbb{Z}$ and $\pi_n(\mathbb{S}^3) = \pi_n(\mathbb{S}^2)$ for $n \geq 3$.*

The last ingredient to show that $\pi_3(\mathbb{S}^2) = \mathbb{Z}$ 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.

**Theorem 4.1.6** (Freudenthal Suspension Theorem)**.** *Suppose that $X$ is $n$-connected. Then $\|X\|_{2n} \simeq \|\Omega\Sigma X\|_{2n}$.*

**Corollary 4.1.7.** *$\pi_n(\mathbb{S}^n) = \mathbb{Z}$ and $\pi_3(\mathbb{S}^2) = \mathbb{Z}$*

**Progress:** All results in this section have been fully formalized.

## 4.2 Eilenberg-MacLane Spaces

*The work in this section is joint work with Ulrik Buchholtz and Egbert Rijke.*
In this section we prove some categorical properties of Eilenberg-MacLane spaces. In particular, we prove 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` $\mathsf{groupoid\text{-}quotient}(G) :=$
- $i : G_0 \to \mathsf{groupoid\text{-}quotient}(G)$;
- $p : (x\ y : G_0) \to \mathsf{hom}(x, y) \to x = y$;
- $q : (x\ y\ z : G_0) \to (g : \mathsf{hom}(y, z)) \to (f : \mathsf{hom}(x, y)) \to p(g \circ f) = p(f) \cdot p(g)$.
- $\varepsilon : \mathsf{is\text{-}1\text{-}type}(\mathsf{groupoid\text{-}quotient}(G))$.

As with the $n$-truncation, the fact that the groupoid quotient is 1-truncated can be encoded using hubs and spokes (or we could first define the HIT without truncation, and then truncate afterwards).
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 $G$ 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's easy to show that $K(G, n)$ is $(n-1)$-connected and $n$-truncated. Trickier

is to show that $\Omega K(G, n+1) \simeq K(G, n)$. This is done separately for $n = 1$ and for $n \geq 2$.
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 4.1.6. 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 $K\text{-}\mathsf{loop}(G, n) : \Omega K(G, n+1) \simeq K(G, n)$.
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 which 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 doesn't 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 $(n-1)$-connected $n$-truncated pointed type, and suppose that for some group $G$ there is an equivalence

$e : \Omega^n X \simeq G$ which sends concatenation to multiplication. Then there is a pointed map $K\text{-}\mathsf{elim}(e, n) : K(G, n) \to X$.

**Lemma 4.2.2.** *There is a pointed homotopy making the following diagram commute.*

$$\begin{tikzcd}
K(G,n) \arrow[rr, "\sim"] \arrow[dr, "K\text{-}\mathsf{elim}(\hat{e},n)"'] & & \Omega K(G,n+1) \arrow[dl, "\Omega(K\text{-}\mathsf{elim}(e,n+1))"] \\
& \Omega X &
\end{tikzcd}$$

**Lemma 4.2.3.** *The following diagram commutes.*

$$\begin{tikzcd}
\Omega^n K(G,n) \arrow[rr, "\sim"] \arrow[dr, "\Omega^n(K\text{-}\mathsf{elim}(e,n))"'] & & G \\
& \Omega^n X \arrow[ur, "e"'] &
\end{tikzcd}$$

**Theorem 4.2.4.** $K\text{-}\mathsf{elim}(e, n)$ *is an equivalence.*

**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* $n \geq 2$.

### 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.

**Lemma 4.2.7.** *The following diagram commutes.*

$$\begin{tikzcd}
\pi_n(K(G,n)) \arrow[r, "\pi_n(K(\varphi,n))"] & \pi_n(K(H,n)) \\
G \arrow[u, "\sim"] \arrow[r, "\varphi"] & H \arrow[u, "\sim"']
\end{tikzcd}$$

**Lemma 4.2.8.** *The following diagram commutes.*

$$\begin{tikzcd}
X \arrow[r, "f"] & Y \\
K(\pi_n(X),n) \arrow[u, "\sim"] \arrow[r, "K(\pi_n(f),n)"] & K(\pi_n(Y),n) \arrow[u, "\sim"']
\end{tikzcd}$$

**Theorem 4.2.9.** *$K(-,n)$ 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 $n \geq 2$).*

**Progress:** All results in this section (so far) have been fully formalized. Future work in this section might include characterizing higher groups. For example, (symmetric/braided) 2-groups should correspond to $(n-2)$-connected $n$-truncated pointed types.

## 4.3 The Smash Product

*The work in this section is joint work with Robin Adams, Marc Bezem, Ulrik Buchholtz, Stefano Piceghello and Egbert Rijke..*

In this section we will discuss the smash product and its properties. The smash product has many uses in homotopy theory. It can be used to define generalized homology theory, and can be used to compute $\pi_4(\mathbb{S}^3)$ [Bru16].

The goal is to prove that the smash product defines a *1-coherent symmetric monoidal product on pointed types* [Bru16, Definition 4.1.1]. In this section, all types, maps, homotopies and equivalences are pointed, unless mentioned otherwise. We will denote pointed homotopies using equalities in diagrams, but we really mean pointed homotopies.
First, we need a property of pointed maps.

**Lemma 4.3.1.** *Given maps $f : A' \to A$ and $g : B \to B'$. Then there are maps $(f \to C) : (A \to C) \to (A' \to C)$ and $(C \to g) : (C \to B) \to (C \to B')$ given by precomposition with $f$, resp. postcomposition with $g$. The map $\lambda g.\, C \to g$ preserves the basepoint, giving rise to a map*

$$(C \to (-)) : (B \to B') \to (C \to B) \to (C \to B').$$

*Also, the following square commutes*

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

## 4.3.1 Naturality of the Smash Product

**Definition 4.3.2.** The smash of $A$ and $B$ is the HIT generated by the point constructor $(a, b)$ for $a : A$ and $b : B$ and two auxilliary 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* 4.3.3. 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 now states that the two definitions of $A \wedge B$ are equivalent.

$$\begin{tikzcd}
\mathbf{2} \arrow[r] \arrow[d] & A+B \arrow[r] \arrow[d] & \mathbf{2} \arrow[d] \\
\mathbf{1} \arrow[r] & A \vee B \arrow[r] \arrow[d] & \mathbf{1} \arrow[d] \\
 & A \times B \arrow[r] & A \wedge B
\end{tikzcd}$$

**Lemma 4.3.4.**

- *The smash 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.*
- *Smash preserves composition, which gives rise to the interchange law: $i : (f' \circ f) \wedge (g' \circ g) \sim f' \wedge g' \circ f \wedge g$*
- *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.*
- *There are homotopies $f \wedge 0 \sim 0$ and $0 \wedge g \sim 0$ such that the following diagrams commute for given homotopies $p : f \sim f'$ and $q : g \sim g'$.*

$$\begin{tikzcd}
f \wedge 0 \arrow[rr, equal, "p\wedge 1"] \arrow[dr, equal] & & f' \wedge 0 \arrow[dl, equal] & 0 \wedge g \arrow[rr, equal, "1\wedge q"] \arrow[dr, equal] & & 0 \wedge g' \arrow[dl, equal] \\
 & 0 & & & 0 &
\end{tikzcd}$$

**Lemma 4.3.5.** *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 $(f_2 \circ f_1) \wedge (g_2 \circ g_1) \sim 0$, one which uses interchange and one which doesn't. These two homotopies are equal. Specifically, the following two diagrams commute:*

$$\begin{tikzcd}
(f_2 \circ 0) \wedge (g_2 \circ g_1) \arrow[r, equal] \arrow[dd, equal] & (f_2 \wedge g_2) \circ (0 \wedge g_1) \arrow[d, equal] \\
 & (f_2 \wedge g_2) \circ 0 \arrow[d, equal] \\
0 \wedge (g_2 \circ g_1) \arrow[r, equal] & 0
\end{tikzcd}$$

$$\begin{tikzcd}
(0 \circ f_1) \wedge (g_2 \circ g_1) \arrow[r, equal] \arrow[dd, equal] & (0 \wedge g_2) \circ (f_1 \wedge g_1) \arrow[d, equal] \\
 & 0 \circ (f_1 \wedge g_1) \arrow[d, equal] \\
0 \wedge (g_2 \circ g_1) \arrow[r, equal] & 0
\end{tikzcd}$$

*Proof.* We will only do the case where $f_1 \equiv 0$, i.e. fill the diagram on the left. The other case is similar (and slightly easier). First apply induction on the paths that $f_2$, $g_1$ and $g_2$ respect the basepoint. In this case $f_2 \circ 0$ is definitionally equal to 0, and the canonical proof that $f_2 \circ 0 \sim 0$ is (definitionally) equal to reflexivity. This means that the homotopy $(f_2 \circ 0) \wedge (g_2 \circ g_1) \sim 0 \wedge (g_2 \circ g_1)$ is also equal to reflexivity, and also the path that $f_2 \wedge g_2$ respects the basepoint is reflexivity, hence the homotopy $(f_2 \wedge g_2) \circ 0 \sim 0$ is also reflexivity. This means we need to fill the following square, where $q$ is the proof that $0 \wedge f \sim 0$.

$$\begin{tikzcd}
(f_2 \circ 0) \wedge (g_2 \circ g_1) \arrow[r, equal, "i"] \arrow[d, equal, "1"] & (f_2 \wedge g_2) \circ (0 \wedge g_1) \arrow[d, equal, "(f_2 \wedge g_2) \circ q"] \\
0 \wedge (g_2 \circ g_1) \arrow[r, equal, "q"] & 0
\end{tikzcd}$$

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 $b : B_1$. Then we have to fill the square (denote the basepoints of $A_i$ and $B_i$ by $a_i$ and $b_i$ and we suppress the arguments of $\mathsf{gluer}$). 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.

$$\begin{tikzcd}
(f_2(a_2), g_2(g_1(b))) \arrow[r, equal, "1"] \arrow[d, equal, "1"] & (f_2(a_2), g_2(g_1(b))) \arrow[d, equal, "\mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluer} \cdot \mathsf{gluer}^{-1})"] \\
(a_3, g_2(g_1(b))) \arrow[r, equal, "\mathsf{gluer} \cdot \mathsf{gluer}^{-1}"] & (a_3, b_3)
\end{tikzcd}$$

If $x$ is either $\mathsf{auxl}$ or $\mathsf{auxr}$ it is similar but easier. For completeness, we will write down the square we have to fill in the case that $x$ is $\mathsf{auxr}$.

$$\begin{tikzcd}
\mathsf{auxr} \arrow[r, equal, "1"] \arrow[d, equal, "1"] & \mathsf{auxr} \arrow[d, equal, "\mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluer}_{b_2}^{-1})"] \\
\mathsf{auxr} \arrow[r, equal, "\mathsf{gluer}_{g_2(b_2)}^{-1}"] & (a_3, b_3)
\end{tikzcd}$$

If $x$ varies over $\mathsf{gluer}_b$, we need to fill the cube below. The front and the back are the squares we just filled, the left square is a degenerate square, and the other three squares are the squares in the definition of $q$ and $i$ to show that they respect $\mathsf{gluer}_b$ (and on the right we apply $f_2 \wedge g_2$ to that square).

$$\begin{tikzcd}
& \mathsf{auxr} \arrow[rr, equal, "1"] \arrow[dd, equal, "1"] \arrow[dl, equal] & & \mathsf{auxr} \arrow[dd, equal, "\mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluer}_{b_2}^{-1})"] \arrow[dl, equal] \\
(f_2(a_2), g_2(g_1(b))) \arrow[rr, equal, "1"] \arrow[dd, equal, "1"] & & (f_2(a_2), g_2(g_1(b))) \arrow[dd, equal, "\mathsf{ap}_{f_2 \wedge g_2}(\mathsf{gluer} \cdot \mathsf{gluer}^{-1})"] & \\
& \mathsf{auxr} \arrow[rr, equal, "\mathsf{gluer}_{g_2(b_2)}^{-1}"] \arrow[dl, equal] & & (a_3, b_3) \arrow[dl, equal] \\
(a_3, g_2(g_1(b))) \arrow[rr, equal, "\mathsf{gluer} \cdot \mathsf{gluer}^{-1}"] & & (a_3, b_3) &
\end{tikzcd}$$

After canceling applications of $\mathsf{ap}_{h \wedge k}(\mathsf{gluer}_z) = \mathsf{gluer}_{k(z)}$ on various sides of the squares (TODO).
If $x$ varies over $\mathsf{gluel}_a$ the proof is very similar. Only in the end we need to fill the following cube instead (TODO).
To show that this homotopy is pointed, (TODO)

□

**Theorem 4.3.6.** *Given pointed types $A$, $B$ and $C$, the functorial action of the smash product induces a map*

$$(-) \wedge C : (A \to B) \to (A \wedge C \to B \wedge C)$$

*which is natural in $A$, $B$ and dinatural in $C$.*

The naturality and dinaturality means that the following squares commute for $f : A' \to A$ $g : B \to B'$ and $h : C \to C'$.

$$\begin{tikzcd}
(A \to B) \arrow[r, "(-)\wedge C"] \arrow[d, "f\to B"] & (A \wedge C \to B \wedge C) \arrow[d, "f\wedge C\to B\wedge C"] & (A \to B) \arrow[r, "(-)\wedge C"] \arrow[d, "A\to g"] & (A \wedge C \to B \wedge C) \arrow[d, "A\wedge C\to g\wedge C"] \\
(A' \to B) \arrow[r, "(-)\wedge C"] & (A' \wedge C \to B \wedge C) & (A \to B') \arrow[r, "(-)\wedge C"] & (A \wedge C \to B' \wedge C)
\end{tikzcd}$$

$$\begin{tikzcd}
(A \to B) \arrow[r, "(-)\wedge C"] \arrow[d, "(-)\wedge C'"] & (A \wedge C \to B \wedge C) \arrow[d, "A\wedge C\to B\wedge h"] \\
(A \wedge C' \to B \wedge C') \arrow[r, "A\wedge h\to B\wedge C'"] & (A \wedge C \to B \wedge C')
\end{tikzcd}$$

*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 extensinality), which follows from Lemma 4.3.5.

$$\begin{tikzcd}
(0 \circ f) \wedge C \arrow[r, equal] \arrow[dd, equal] & (0 \wedge C) \circ (f \wedge C) \arrow[d, equal] \\
& 0 \circ (f \wedge C) \arrow[d, equal] \\
0 \wedge C \arrow[r, equal] & 0
\end{tikzcd}$$

The naturality in $B$ is almost the same: for the underlying homotopy we need to show $(g \circ k) \wedge C = g \wedge C \circ k \wedge C$. For the pointedness we need to fill the following square, which follows from Lemma 4.3.5.

$$\begin{tikzcd}
(g \circ 0) \wedge C \arrow[r, equal] \arrow[dd, equal] & (g \wedge C) \circ (0 \wedge C) \arrow[d, equal] \\
& (g \wedge C) \circ 0 \arrow[d, equal] \\
0 \wedge C \arrow[r, equal] & 0
\end{tikzcd}$$

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 by applying interchange twice:

$$B \wedge h \circ k \wedge C \sim (\mathsf{id}_B \circ k) \wedge (h \circ \mathsf{id}_C) \sim (k \circ \mathsf{id}_A) \wedge (\mathsf{id}_{C'} \circ h) \sim k \wedge C' \circ A \wedge h.$$

To show that this homotopy is pointed, we need to fill the following square:

$$\begin{tikzcd}
B \wedge h \circ 0 \wedge C \arrow[r, equal] \arrow[d, equal] & (\mathsf{id}_B \circ 0) \wedge (h \circ \mathsf{id}_C) \arrow[r, equal] \arrow[d, equal] & (0 \circ \mathsf{id}_A) \wedge (\mathsf{id}_{C'} \circ h) \arrow[r, equal] \arrow[d, equal] & 0 \wedge C' \circ A \wedge h \arrow[d, equal] \\
B \wedge h \circ 0 \arrow[d, equal] & 0 \wedge (h \circ \mathsf{id}_C) \arrow[r, equal] \arrow[d, equal] & 0 \wedge (\mathsf{id}_{C'} \circ h) \arrow[d, equal] & 0 \circ A \wedge h \arrow[d, equal] \\
B \wedge h \circ 0 \arrow[r, equal] & 0 \arrow[r, equal] & 0 \arrow[r, equal] & 0
\end{tikzcd}$$

The left and the right squares are filled by Lemma 4.3.5. The squares in the middle are filled by (corollaries of) Lemma 4.3.4. □

### 4.3.2 The smash-pointed map adjunction

**Lemma 4.3.7.** *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: $\eta$ is also dinatural in $B$, but we don't need this.

*Proof.* We define $\eta ab = (a, b)$. Then $\eta a$ respects the basepoint because $(a, b_0) = (a_0, b_0)$. Also, $\eta$ itself respects the basepoint. To show this, we need to show that $\eta a_0 \sim 0$. The underlying maps are homotopic, since $(a_0, b) = (a_0, b_0)$. 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:

$$\mathsf{gluel}_{a_0} \cdot \mathsf{gluel}_{a_0}^{-1} = 1 = \mathsf{gluer}_{b_0} \cdot \mathsf{gluer}_{b_0}^{-1}.$$

This defines the unit. To 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 $\mathsf{auxl}$ or $\mathsf{auxr}$ then we set $\varepsilon x :\equiv c_0 : C$. If $x$ varies over $\mathsf{gluel}_f$ then we need to show that $f(b_0) = c_0$, which is true by $f_0$. If $x$ varies over $\mathsf{gluer}_b$

we need to show that $0(b) = c_0$ which is true by reflexivity. $\varepsilon$ is trivially a pointed map, which defines the counit.
Now we need to show that the unit and counit are (di)natural. (TODO).
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 $p : \varepsilon \circ \eta f \sim f$. For the underlying homotopy of $p$, let $a : A$, and we need to show that $\varepsilon(f, a) = f(a)$, which is true by reflexivity. To show that $p$ is a pointed homotopy, we need to show that $p(a_0) \cdot f_0 = \mathsf{ap}_\varepsilon(\eta f)_0 \cdot \varepsilon_0$, 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: $0_0$ denotes the proof that $0(a_0) = b_0$, which is reflexivity)

$$\mathsf{ap}_\varepsilon(\mathsf{gluel}_f \cdot \mathsf{gluel}_0^{-1}) = \mathsf{ap}_\varepsilon(\mathsf{gluel}_f) \cdot (\mathsf{ap}_\varepsilon(\mathsf{gluel}_0))^{-1} = f_0 \cdot 0_0^{-1} = f_0.$$

Now we need to show that $p$ itself respects the basepoint of $A \to B$, i.e. that the composite $\varepsilon \circ \eta 0 \sim \varepsilon \circ 0 \sim 0$ is equal to $p$ for $f \equiv 0_{A,B}$. 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 typechecks, since $0_{A,B}a \equiv 0_{A,B}a_0$). 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 one, we need to show for $x : A \wedge B$ that $\varepsilon(\eta \wedge B(x)) = x$, which we prove by induction to $x$. (TODO).

□

**Definition 4.3.8.** 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 4.3.9.** *$e$ is invertible, hence gives a pointed equivalence*

$$(A \to B \to C) \simeq (A \wedge B \to C).$$

*Proof.* Define

$${}^{-1}e_{A,B,C} : (A \wedge B \to C) \xrightarrow{B \to (-)} ((B \to A \wedge B) \to (B \to C)) \xrightarrow{\eta \to (B \to C)} (A \to B \to C).$$

It is easy to show that $e$ and ${}^{-1}e$ are inverses as unpointed maps from the unit-counit laws and naturality of $\eta$ and $\varepsilon$.

□

**Lemma 4.3.10.** *$e$ is natural in $A$, $B$ and $C$.*

*Remark* 4.3.11. Instead of showing that $e$ is natural, we could instead 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 don't need to work with any higher inductive type to prove that.

*Proof.* $e$ **is natural 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.

$$\begin{tikzcd}
(A \to B \to C) \arrow[r, "(-)\wedge B"] \arrow[d, "f \to B \to C"] & (A \wedge B \to (B \to C) \wedge B) \arrow[r, "A \wedge B \to \varepsilon"] \arrow[d, "f \wedge B \to \cdots"] & (A \wedge B \to C) \arrow[d, "f \wedge B \to C"] \\
(A' \to B \to C) \arrow[r, "(-)\wedge B"] & (A' \wedge B \to (B \to C) \wedge B) \arrow[r, "A \wedge B \to \varepsilon"] & (A' \wedge B \to C)
\end{tikzcd}$$

$e$ **is natural 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 $\varepsilon$ in the second argument.

$$\begin{tikzcd}
(A \to B \to C) \arrow[r] \arrow[d] & (A \wedge B \to (B \to C) \wedge B) \arrow[r] \arrow[d] & (A \wedge B \to C) \arrow[d] \\
(A \to B \to C') \arrow[r] & (A \wedge B \to (B \to C') \wedge B) \arrow[r] & (A \wedge B \to C')
\end{tikzcd}$$

$e$ **is natural 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 $\varepsilon$ in the first argument.

$$\begin{tikzcd}
 & (A \wedge B \to (B \to C) \wedge B) \arrow[rr] \arrow[dd] & & (A \wedge B' \to (B \to C) \wedge B) \arrow[dd] \\
(A \to B \to C) \arrow[ur] \arrow[rr] \arrow[dd] & & (A \wedge B' \to (B \to C) \wedge B') \arrow[ur] \arrow[dd] & \\
 & (A \wedge B \to C) \arrow[rr] & & (A \wedge B' \to C) \\
(A \to B' \to C) \arrow[rr] & & (A \wedge B' \to (B' \to C) \wedge B') \arrow[ur] &
\end{tikzcd}$$

□

**Theorem 4.3.12.** *The smash product is associative: there is an equivalence* $f : A \wedge (B \wedge C) \simeq (A \wedge B) \wedge C$ *which is natural in* $A$*,* $B$ *and* $C$*.*

*Proof.* Let $\varphi_X$ be the composite of the following equivalences:

$$\begin{aligned}
A \wedge (B \wedge C) \to X &\simeq A \to B \wedge C \to X \\
&\simeq A \to B \to C \to X \\
&\simeq A \wedge B \to C \to X \\
&\simeq (A \wedge B) \wedge C \to X.
\end{aligned}$$

$\varphi_X$ is natural in $A, B, C, X$ by repeatedly applying Lemma 4.3.10. Let $f :\equiv \varphi_{A \wedge (B \wedge C)}(\mathsf{id})$ and $f^{-1} :\equiv \varphi^{-1}_{(A \wedge B) \wedge C}(\mathsf{id})$. Now these maps are inverses by naturality of $\varphi$ in $X$:

$$f^{-1} \circ f \equiv f^{-1} \circ \varphi(\mathsf{id}) \sim \varphi(f^{-1} \circ \mathsf{id}) \sim \varphi(\varphi^{-1}(\mathsf{id})) \sim \mathsf{id}.$$

The other composition is the identity by a similar argument. Lastly, $f$ is natural in $A$, $B$ and $C$, since $\varphi_X$ is. □

**Progress:** The adjunction between the smash product and pointed maps has been fully formalized. From this, we formalized the associativity of the smash product, including all naturality. The coherences (such as the pentagon and hexagon) have not been formalized yet.

## 4.4 The Serre Spectral Sequence

*The work in this section is joint work with Jeremy Avigad, Steve Awodey, Ulrik Buchholtz, Egbert Rijke and Mike Shulman.*

Spectral sequences are important tools for algebraic topology. They have a wide variety of applications, such as giving connections between homotopy groups, homology groups and cohomology groups, and for computing homotopy groups of spheres. In this section we will construct various spectral sequences, most notably the Serre spectral sequence, and look at their applications. The goal is to fully formalize the blog posts by Mike Shulman on this topic [Shu13]. We will adapt some standard definitions to make formalization easier.

**Definition 4.4.1.** For a set $I$ and a ring $R$, an *$I$-graded $R$-module* is a family of $R$-modules indexed over $I$. If $e : I \simeq I$ and $M$ and $M'$ are $I$-graded $R$-modules, a *graded $R$-module homomorphism of degree $e$ from $M$ to $M'$* is a term $\varphi$ of type

$$\{x\ y : I\} \to (p : e(x) = y) \to M_x \to M'_y.$$

We will denote this as $\varphi : M \to M'$ and we define $\deg_\varphi :\equiv e$. For $x : I$ and $m : M_x$ we will write $\varphi_x(m) :\equiv \varphi_{\mathsf{refl}_x}(m) : M'_{\deg_\varphi(x)}$.

*Remark* 4.4.2. First, note that the type of $\varphi$ is equivalently $(x : I) \to M_x \to M'_{e(x)}$. We will see below why the above type is easier to work with.
Traditionally, an $R$-module is graded over a group $G$, and the degree of a graded homomorphism is just a group element, and a map of degree $h$ has type $(g : G) \to M_g \to M'_{g+h}$. The definition given here is more general, because we can define such a map as having degree $\lambda g.\, \lambda g+h.\ : G \simeq G$. Furthermore, our definition is more convenient. Suppose classically we have two graded homomorphisms $\varphi : M \to M'$ and $\psi : M' \to M''$ of degrees $h$ and $k$, respectively. Then the pointwise composition $\lambda(g : G).\, \lambda(m : M_g).\, \lambda\psi_{g+h}(\varphi_g(m)).$ has type $(g : G) \to M_g \to M''_{(g+h)+k}$. So to get a graded homomorphism of degree $h+k$, we need to transport along the equality $(g+h)+k = g+(h+k)$. Since compositions are ubiquitous, this will be an obstacle. 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 can't 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 constantly map which is constantly 0), we are interested in the homology of $\varphi$ and $\psi$. This is the kernel of $\psi$ quotiented by the image of $\varphi$ in $M'_x$. However, if $\varphi_x$ has type $M_x \to M'_{\deg_\varphi(x)}$, there is no map which (without transports) lands in

$M'_x$. We would need to transport along the path $p_x : \deg_\varphi^{-1}(\deg_\varphi(x)) = x$ and take the image of this composite:

$$M_{\deg_\varphi^{-1}(x)} \xrightarrow{\varphi_{\deg_\varphi^{-1}(x)}} M'_{\deg_\varphi(\deg_\varphi^{-1}(x))} \xrightarrow{\sim} M'_x.$$

For this reason, we allow graded homomorphisms to be applied to paths, which are "built-in" tranports, so that we can define the homology as $H_x :\equiv \ker(\psi_x)/\mathsf{im}(\varphi_{p_x})$, or diagramatically

$$M_{\deg_\varphi^{-1}(x)} \xrightarrow{\varphi_{p_x}} M'_x \xrightarrow{\psi_x} M''_{\deg_\psi(x)}.$$

**Definition 4.4.3.** A (homologically indexed) *spectral sequence* consists of

- A family $(E^r)_{r:\mathbb{N}}$ of graded $R$-modules. For a fixed $r$ this gives the *$r$-page* of the spectral sequence.
- graded $R$-module homomorphisms $d^r : E^r \to E^r$ which are called *differentials*.
- isomorphisms $\alpha^r : H(E^r) \simeq E^{r+1}$ where $H(E^r) = \ker(d^r)/\mathsf{im}(d^r)$ is the homology of $d$.

*Remark* 4.4.4. Normally, $r$ ranges over the natural numbers which are at least 1 or 2, but for formalization it adds unnecessary complexity to start counting at a number other than 0. The grading of $E^r$ is usually over $\mathbb{Z} \times \mathbb{Z}$, but we won't require that in general. Instead of $R$-modules, we could take objects of an arbitrary abelian category, but for concreteness and to simplify things, we choose to develop the theory only for $R$-modules, which is general enough for most applications.

Note that $(E^r, d^r)$ determine $E^{r+1}$ but *not* $d^{r+1}$. However, if at page $r$ we have extra information, such as an *exact couple*, then we can compute a *derived exact couple*, which gives us all the information for the next page (and hence for all higher pages).

**Definition 4.4.5.** An *exact couple* is a pair $(D, E)$ of $I$-graded $R$-modules with morphisms

$$\begin{tikzcd} D \arrow[rr, "i"] & & D \arrow[dl, "j"] \\ & E \arrow[ul, "k"] & \end{tikzcd}$$

which is exact in all three vertices. This means that for all $p : \deg_j(x) =_I y$ and $q : \deg_k(y) = z$ that $\ker(k_q) = \mathsf{im}(j_p)$, and similarly for the other two pairs of maps.

**Lemma 4.4.6.** *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 are as follows: $\deg_{i'} \equiv \deg_i$, $\deg_{k'} \equiv \deg_k$ and $\deg_{j'} \equiv \deg_j \circ \deg_i^{-1}$.*

Repeating this process, we get a sequence of exact couples $(D^r, E^r, i^r, j^r, k^r)$. We get a spectral sequence $(E^r, d^r)$ where $d^r :\equiv j^r \circ k^r$. Note that

$$\deg_{d^r} = \deg_{j^r} \circ \deg_{k^r} = \deg_j \circ \deg_i^r \circ \deg_k \equiv: f_r$$

Spectral sequences converge under some boundedness assumptions of the initial exact couple. In [Shu13] the boundedness conditions is formulated in terms of the iterated fibration sequence from which we construct an exact couple. However, it should be possible to formulate it directly in terms of an exact couple. The following is a first attempt, which might not work exactly. The conditions and/or theorem statements might need to be changed.

**Definition 4.4.7.** We call an exact couple *bounded* if for every $x : I$ there is are bounds $B_x : \mathbb{N}$ such that for all $s \geq B_x$ we have

$$E_{f_s(x)} = 0, \qquad E_{f_s^{-1}(x)} = 0, \qquad D_{f_s(x)} = 0 \quad \text{and} \quad i_{\deg_i^{-s}(x)} \text{ is an equivalence.}$$

Given an bounded exact couple, the pages stabilize pointwise. Namely for all $s \geq B_x$ we have $E_x^{s+1} = E_x^s$ and $D_x^{s+1} = D_x^s$, so we can define $E_x^\infty :\equiv E_x^{B_x}$ and $D_x^\infty :\equiv D_x^{B_x}$.

**Theorem 4.4.8** (Convergence Theorem)**.** *Let $(D, E, i, j, k)$ be a bounded exact couple. Suppose that $i_{\deg_i^{-s}(x)}$ for* all *$s \geq 1$ (and probably some contractibility condition for $E$ in the $\deg_i^{-1}$-direction of $x$). Then there are $R$-modules $B_{x,s}$ with $B_{x,0} = D_x$ such that we have the following short exact* *sequences:*

$$\begin{aligned}
E^\infty_x &\to B_{x,0} \to B_{x,1} \\
&\vdots \\
E^\infty_{\deg_i^s(x)} &\to B_{x,s} \to B_{x,s+1} \\
E^\infty_{\deg_i^{s+1}(x)} &\to B_{x,s+1} \to B_{x,s+2} \\
&\vdots \\
E^\infty_{\deg_i^{B_x}(x)} &\to B_{x,B_x} \to 0
\end{aligned}$$

*This is denoted*

$$E^0_x \Rightarrow D_x.$$

Note that the condition on $i$ implies that $D^\infty_x = D_x$.
We haven't discussed yet 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 can turn this into an exact couple. We will assume that either we have a map between spectra, or that the pointed types are all simply connected, so that all terms in the long exact sequence of homotopy groups are abelian groups.
Given a sequence of maps

$$\cdots \to Y_{s+2} \xrightarrow{f_{s+1}} Y_{s+1} \xrightarrow{f_s} Y_s \to \cdots$$

Let $X_{s+1} :\equiv \mathsf{fib}_{f_s}$. Then $D :\equiv (\pi_n(Y_s))_{n,s}$ and $E :\equiv (\pi_n(X_s))_{n,s}$ are graded abelian groups ($\mathbb{Z}$-modules) and the maps of the long exact sequences become graded homomorphisms. It is customary to reindex the spectral sequence using $(p,q) = (n-s,s)$, or equivalently $(n,s) = (p+q,q)$

**Lemma 4.4.9.** *Given a sequence*

$$Y_T \xrightarrow{f_{s+1}} Y_{s+1} \xrightarrow{f_s} Y_s \to \cdots$$

*with fibers $X_{s+1} :\equiv \mathsf{fib}_{f_s}$ such that for all $n$ there is a $B$ such that for all $s \le B$ we have $\pi_n(Y_s) = \pi_n(X_s) = 0$. Then the exact couple constructed from this sequence is bounded. In particular*

$$\pi_{p+q}(X_q) \Rightarrow \pi_{p+q}(Y_T).$$

We can apply this machinery to get the Serre Spectral Sequence.

**Theorem 4.4.10** (Serre Spectral Theorem)**.** *Given a pointed map $f : X \to B$ with fiber $F$ where $B$ is simply connected. For a spectrum $Y$ we get*

$$H^p(B; H^q(F; Y)) \Rightarrow H^{p+q}(X; Y).$$

Using the Serre Spectral sequence, it is possible to show many properties about homotopy groups and (co)homology groups. One goal to try is to prove that $\pi_4(\mathbb{S}^3) = \mathbb{Z}_2$. This also depends on other machinery, like Hurewicz Theorem.

**Progress:** A lot of work has been done to construct the Serre spectral sequence by the HoTT-group at CMU. We have a basic theory for spectra (by Mike Shulman), cohomology theory, many algebraic constructions for groups and $R$-modules. Currently in progress is defining the derived exact couple and proving the convergence theorem. To construct the Serre spectral sequence, we will need other results, such as the spectrification and cohomology with local coefficients. For most applications of the Serre spectral sequence, we will need other tools, such as Hurewicz Theorem.

# Bibliography

[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.

[BGL$^+$16] A. Bauer, J. Gross, P. LeFanu Lumsdaine, M. Shulman, M. Sozeau, and B. Spitters, *The HoTT Library: A formalization of homotopy type theory in Coq*, ArXiv e-prints (2016), `arXiv:1610.04591`.

[BGvdW16] Henning Basold, Herman Geuvers, and Niels van der Weide, *Higher inductive types in programming*, Preprint (2016).

[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`.

[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`.

[Cap14] Paolo Capriotti, *Mutual and higher inductive types in homotopy type theory*, Preprint (2014).

[CCHM] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg, *Cubical type theory*, code library, `https://github.com/mortberg/cubicaltt`.

[dMERU17] Leonardo de Moura, Gabriel Ebner, Jared Roesch, and Sebastian Ullrich, *The Lean theorem prover*, slides, January 2017, https://leanprover.github.io/presentations/20170116_POPL.

[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.

[GMM06] Healfdene Goguen, Conor McBride, and James McKinna, *Eliminating dependent pattern matching*, Algebra, Meaning, and Computation (2006), 521–540.

[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.

[Kra16] Nicolai Kraus, *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.

[LS12] Peter LeFanu Lumsdaine and Michael Shulman, *Semantics of higher inductive types*, Preprint (2012).

[Rij17] Egbert Rijke, *The join construction*, ArXiv (2017), arXiv:1701.07538.

[Shu13] Michael Shulman, *Spectral sequences in HoTT*, blog posts, merged on ncatlab, August 2013, https://ncatlab.org/homotopytypetheory/revision/spectral+sequences/5.

[Uni13] The Univalent Foundations Program, *Homotopy type theory: Univalent foundations of mathematics*, http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[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.

[VMA+17] Vladimir Voevodsky, Anders Mörtberg, Benedikt Ahrens, Catherine Lelay, Tomi Pannila, and Ralph Matthes, *UniMath:* Univalent Mathematics, code library, 2017, https://github.com/UniMath.

[Voe06] Vladimir Voevodsky, *A very short note on the homotopy $\lambda$-calculus*, online, 2006, http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/Hlambda_short_current.pdf.

[Voe15] ———, *Oxford lectures on UniMath*, filmed by Kohei Kishida, available at https://www.math.ias.edu/vladimir/Lectures, 2015.