Synthetic Cohomology in Homotopy Type Theory

Evan Cavallo

December 16, 2015

Contents

Introduction

This is a case study in *higher type theory*, the study of the infinite-dimensional structure which arises from the identity type of intensional Martin-Löf type theory [17]. For elements $x, y : A$, inhabitants of the identity type $x =_A y$ express *identifications* between x and y, or proofs that x and y are in some sense equal. In extensional Martin-Löf type theory [18], such types have at most one inhabitant, but this is not guaranteed in its intensional counterpart. In 1998, Hofmann and Streicher [10] made this precise by presenting a model interpreting types as groupoids, with $x =_A y$ interpreted as the discrete groupoid of morphisms from x to y in A. It was independently confirmed by van den Berg and Garner in 2008 [24] and Lumsdaine in 2009 [16] that types are equipped more generally with the structure of weak ω-groupoids. This opened the possibility of augmenting the theory in order to make such structure visible internally. The most notable addition is Voevodsky's univalence axiom, which unifies the concepts of *identifications between types* and *equivalences between types*. Another, which is crucial to this thesis, is the concept of higher inductive types: types generated not only by point constructors but also path constructors, thus exhibiting explicit higher structure. With these it becomes possible to define interesting ω-groupoids, such as spheres, within the theory. The intensional type theory created by augmenting Martin-Löf type theory with univalence and higher inductive types is what is now called *homotopy type theory* (or $HoTT$).

This thesis is an exploration of basic ideas in cohomology, which is a central tool from homotopy theory. It is an "informalization" of results developed in the HoTT-Agda library [11] over the past two years. It is intended to serve as documentation, as a lasting account in the inevitable event of deprecation, and as an exposition of the tools and methodology I have found useful. While cohomology is the guiding motivation, some of the proofs and results are neither strictly cohomological nor particularly efficient from the classical perspective. Rather, the focus is on techniques for "higher induction." My hope is to demonstrate the abilities and shortcomings of the current formulation of homotopy type theory in the broader context of higher type theories. Particularly relevant as a point of comparison is the recent work on *cubical type theories*, independently conducted by Coquand et al. [4], Brunerie and Licata [3], and Altenkirch and Kaposi [1]. While homotopy type theory is not natively cubical, it is possible to encode some notion of cubes using inductive types. This line of attack, initiated by Licata and Brunerie [14, 13], has been extremely fruitful as an organizing principle, though limited by the paucity of definitional equalities in vanilla homotopy type theory. I hope to elucidate further the flavor of the cubical approach by example.

The first chapter contains a brief introduction to the necessary prerequisites in homotopy type theory. The second chapter adds to these the square and cube types; the results in this chapter

are for the most part due to Brunerie and Licata and were previously formalized in [13]. Chapter 3 presents the definition of an Eilenberg-Steenrod cohomology theory and exhibits a class of examples. The construction of the Eilenberg-MacLane spectrum in this chapter is due to and originally formalized by Finster and Licata [15]. The proof that spectra satisfy the cohomology axioms was developed informally by a group at the Institute for Advanced Study during the Special Year on Univalent Foundations of Mathematics and described by Shulman [22], but the formalization is original. Chapter 4, which comprises the bulk of the thesis, proves a series of results which hold in all cohomology theories. None of these are new to classical algebraic topology, but the translation into type theory and formalization are my own.

Almost everything has been formalized in Agda; a few statements are less general in the formalization, but only for combinatorial reasons (for example, a lemma concerning a face of a cube might only be proven for the left face). Appendix A contains an index relating sections of this document with library modules.

I would like to thank my advisor, Robert Harper, for introducing me to type theory (homotopy and otherwise), welcoming me to Carnegie Mellon's HoTT research group, and guiding me through the process of this thesis. Special thanks go to Carlo Angiuli for steering me through my early adventures in type theory. This project would not have been possible without the efforts of contributors to the HoTT-Agda library, especially Guillaume Brunerie and Kuen-Bang Hou (Favonia). Several tools and theorems, particularly for cubical reasoning, originated in Dan Licata's Agda library, and my first encounter with cubes was at Dan's suggestion for the Mayer-Vietoris proof. I am grateful to Jeremy Avigad and Richard Statman, who served on my defense committee. Finally, I would like to thank the Department of Mathematical Sciences, the Department of Computer Science, and my friends and family, for their encouragement and for enduring my erratic progress through this project.

This research was sponsored in part by the National Science Foundation under grant numbers CCF-1116703 and CCF-1445995 (REU).

Chapter 1

Type Theory Preliminaries

In general, we assume familiarity with informal homotopy type theory in the style of *Homotopy Type Theory* [8]; the material of Chapters 1-2 is assumed, with 3-6 relevant to varying degrees. Beyond intensional Martin-Löf type theory, we take particular advantage of pushout types [8, §6.8] as well as truncations [8, §6.9] and more generally n-types [8, §7]. In order to fix notation, we give a brief overview.

1.1 Types and Paths

We write $\mathcal{U}$ for our universe of types. Technically (and in the formalization) we use a universe hierarchy $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \cdots$, but we will leave this implicit, as none of our results require attention to universe levels. For any type $A : \mathcal{U}$ and family (i.e. fibration) $B : A \to \mathcal{U}$, we have the *dependent function type* $\prod_{x:A} Bx : \mathcal{U}$ of maps which, given $a : A$, produce an element of Ba; if for every $a : A$ we have $b : Ba$ we write $\lambda a.b : \prod_{a:A} Ba$ for the corresponding function term. As a special case, for $A, B : \mathcal{U}$ we have the non-dependent function type $(A \to B) :\equiv \prod_{\_:A} B$. Likewise, we have the *dependent sum type* $\sum_{a:A} Ba : \mathcal{U}$ of pairs (a, b) where $a : A$ and $b : Ba$, with the cartesian product $A \times B$ as the non-dependent special case. Finally, for any $x, y : A$ we have the *identity type* $x =_A y : \mathcal{U}$ of identifications (or *paths*) between x and y, which is inductively generated by single constructors $\mathsf{refl}_x : x =_A x$ for $x : A$. We write $x = y$ when A is clear from context. The elimination rule for the identity states:

Definition 1.1 (Path Induction)**.** For any family $C : \prod_{x,y:A}(x = y) \to \mathcal{U}$, in order to construct a function $f : \prod_{x,y:A} \prod_{p:x=y} C(x, y, p)$, it suffices to give a term $d : \prod_{x:A} C(x, x, \mathsf{refl}_x)$. A function f defined in this way satisfies $f(x, x, \mathsf{refl}_x) \equiv d(x)$.

We use the symbol $\equiv$ for the external *definitional equality*, which we think of as syntactic equality modulo computation, that is, the congruence generated by β-reduction and η-expansion. This is strictly stronger than the internal notion of equality $=$ captured by the identity type (sometimes called *propositional equality*).

That $=$ is a congruence is derivable using the elimination rule. For any two paths $p : x = y$ and $q : y = z$ we have their concatenation $p \cdot q$ defined by $\mathsf{refl} \cdot q :\equiv q$ (we can define an alternate, equivalent composition by inducting instead on q). For any $p : x = y$, $q : y = z$, and $r : z = w$, there is an associativity path $\mathsf{assoc}\ p\ q\ r : (p \cdot q) \cdot r = p \cdot (q \cdot r)$ defined by $\mathsf{assoc}\ \mathsf{refl}\ \mathsf{refl}\ \mathsf{refl} :\equiv \mathsf{refl}$. For any path $p : x = y$ we have an inverse path $p^{-1} : y = x$ defined by $\mathsf{refl}^{-1} :\equiv \mathsf{refl}$, along with identities $\cdot\text{-}\mathsf{inv\text{-}l}\ p : p \cdot p^{-1} = \mathsf{refl}$ and $\cdot\text{-}\mathsf{inv\text{-}r}\ p : p^{-1} \cdot p = \mathsf{refl}$ defined by $\cdot\text{-}\mathsf{inv\text{-}l}\ \mathsf{refl} :\equiv \mathsf{refl}$ and $\cdot\text{-}\mathsf{inv\text{-}r}\ \mathsf{refl} :\equiv \mathsf{refl}$. Any function $f : A \to B$ induces a corresponding function on paths $\mathsf{ap}_f : (x =_A y) \to (fx =_B fy)$ defined by $\mathsf{ap}_f \mathsf{refl} :\equiv \mathsf{refl}$.

Between two types $A, B : \mathcal{U}$ we have a third notion of identity, that of *equivalence* $A \simeq B$. An equivalence $e : A \simeq B$ consists of

- maps $f : A \to B$ and $g : B \to A$ (for which we generally write e and e^{-1}),
- homotopies (i.e. families of paths) $p : \prod_{a:A} g(fa) = a$, $q : \prod_{b:B} f(gb) = b$,
- a two-dimensional homotopy $\prod_{a:A} \mathsf{ap}_f(pa) = q(fa)$.

For the technical considerations which motivate this definition, see [8, Chapter 4], which refers to this particular characterization as *half adjoint equivalence*. For our purposes we only need the following:

Assertion 1.2. To construct an equivalence $e : A \simeq B$, it suffices to give maps $f : A \to B$, $g : B \to A$ and homotopies $\prod_{a:A} g(fa) = a$, $\prod_{b:B} f(gb) = b$. The resulting equivalence has f and g as maps, but its inverse data may differ from those provided.

We can define a map $\mathsf{idtoeqv} : (A = B) \to (A \simeq B)$ by path induction, taking $\mathsf{idtoeqv}\ \mathsf{refl}$ to be the identity equivalence. The *univalence axiom*, a central feature distinguishing homotopy type theory from standard intensional type theory, states that the notions of equivalences and identifications between types correspond in the following sense:

Definition 1.3 (Univalence Axiom)**.** For any types $A, B : \mathcal{U}$, the map $\mathsf{idtoeqv} : (A = B) \to (A \simeq B)$ is an equivalence. We write $\mathsf{ua} : (A \simeq B) \to (A = B)$ for the given inverse.

We use the univalence axiom freely and implicitly to simplify arguments, typically to deduce that $F(A) \simeq F(B)$ from the fact that $A \simeq B$. We believe, however, that none of these applications are strictly necessary. In our estimation, the only essential use of univalence in this development is in the computation of $\pi_k(K(G, n))$ in §3.3.

For any $f, g : A \to B$, we write $f \sim g$ for the type $\prod_{a:A} fa = ga$ of homotopies from f to g. The univalence axiom implies *function extensionality*, as originally shown by Voevodsky:

Assertion 1.4 (Function Extensionality)**.** For any $f, g : A \to B$, the map $\mathsf{happly} : (f = g) \to (f \sim g)$ defined by $\mathsf{happly}\ \mathsf{refl} :\equiv \mathsf{id}$ is an equivalence with inverse $\mathsf{funext} : (f \sim g) \to (f = g)$.

Given a type $A : \mathcal{U}$ and family $B : A \to \mathcal{U}$, an identification $p : x =_A y$ induces a relationship between the fibers Bx and By. We have a map $\mathsf{transport}^B_p : Bx \to By$ defined by path induction

with $\mathsf{transport}^B_{\mathsf{refl}} :\equiv \mathsf{id}$. This map is easily seen to be an equivalence. We also have a notion of path lying over p in the total space:

Definition 1.5. Let $A : \mathcal{U}$, $B : A \to \mathcal{U}$, $x, y : A$, and $p : x = y$ be given. For any $u : Bx$ and $v : By$ we have the *dependent path type* (or *path-over type*) $u =^B_p v$, which is defined by induction on p with $(u =^B_{\mathsf{refl}} v) :\equiv (u = v)$.

Assertion 1.6. For any $A : \mathcal{U}$, $B : A \to \mathcal{U}$, $x, y : A$, $u : Bx$, and $v : By$, there is an equivalence between the types $(x, u) =_{\sum_{a:A} Ba} (y, v)$ and $\sum_{p:x=y}(u =^B_p v)$.

Analogous to the non-dependent ap, every dependent function $f : \prod_{a:A} Ba$ induces a map $\mathsf{apd}_f : \prod_{p:x=_A y} fx =^B_p fy$ defined by $\mathsf{apd}_f\mathsf{refl} :\equiv \mathsf{refl}$. Composition of dependent paths takes $\alpha : u =^B_p v$ and $\beta : u =^B_q v$ and outputs $\alpha \cdot^d \beta : v =^B_{p \cdot q} w$; it is defined by induction on p and q, taking $\alpha \cdot^d \beta :\equiv \alpha \cdot \beta$ when $p \equiv q \equiv \mathsf{refl}$. For sake of convenience, we use alternate definitions when composing non-dependent paths with dependent ones: for $\alpha : u = v$, $\beta : v =^B_q w$, $\gamma : w = x$ we define $\alpha \lhd \beta : u =^B_q w$ and $\beta \rhd \gamma : v =^B_q x$ by $\mathsf{refl} \lhd \beta :\equiv \beta$ and $\beta \rhd \mathsf{refl} :\equiv \beta$.

In certain cases, we can give an explicit characterization of the dependent path type. For example, when the fibration is a family of path types, we have:

Assertion 1.7. Let $f, g : A \to B$, $x, y : A$, and $p : x =_A y$. For any $u : fx =_B gx$ and $v : fy =_B gy$, the types $u =^{z.fz=gz}_p v$ and $u \cdot \mathsf{ap}_g(p) = \mathsf{ap}_f(p) \cdot v$ are equivalent.

When the fibration is a family of function types parameterized by domain or codomain and the base path is an equivalence:

Assertion 1.8. Let functions $f : A \to B$, $g : A \to C$ and an equivalence $e : B \simeq C$ be given. The type $f =^{D.A \to D}_{\mathsf{ua}\ e} g$ is equivalent to $e \circ f = g$.

Assertion 1.9. Let functions $f : A \to C$, $g : B \to C$ and an equivalence $A \simeq B$ be given. The type $f =^{D.D \to C}_{\mathsf{ua}\ e} g$ is equivalent to $f = g \circ e$.

Finally, a broadly useful gadget: for $A, B : \mathcal{U}$, $C : B \to \mathcal{U}$, $f : A \to B$, $p : x =_A y$, $u : C(fx)$, and $v : C(fy)$, we have a function $\mathsf{over\text{-}ap\text{-}in}_f : (u =^{C \circ f}_p v) \to (u =^C_{\mathsf{ap}_f p} v)$, which is defined by induction on p with $\mathsf{over\text{-}ap\text{-}in}_f :\equiv \mathsf{id}$ in the case $p :\equiv \mathsf{refl}$.

1.2 Pointed Types

We are often interested in types which are accompanied by a specified basepoint; thus we define the universe of *pointed types* $\mathcal{U}_* :\equiv \sum_{A:\mathcal{U}} A$. We generally use letters $X, Y, Z, \ldots$ for pointed types and write $x_0, y_0, z_0, \ldots$ for their respective basepoints, whereas ordinary types are named from $A, B, C, \ldots$. We overload most of our notation to apply to both non-pointed and pointed types – for example, for $X, Y : \mathcal{U}_*$ we write $X \to Y$ for the function type of the underlying types. For any

$X, Y : \mathcal{U}_*$, we also have the type $(X \cdot\!\to Y) :\equiv \sum_{f:X\to Y}(fx_0 = y_0)$ of *basepoint-preserving functions*, or more simply *pointed functions*. Composition is defined by

$$(g, q) \circ (f, p) :\equiv (g \circ f, \mathsf{ap}_g p \cdot q)$$

The identity map on X is $\mathsf{id}^\odot_X :\equiv (\mathsf{id}_X, \mathsf{refl})$ – in general, we use the superscript $^\odot$ where necessary to distinguish a pointed version of some previously defined concept. We have a modified form of function extensionality:

Assertion 1.10. Let (f, p) and (g, q) be basepoint-preserving functions from X to Y. Given a homotopy $h : \prod_{x:X} fx = gx$ and a path $\alpha : p = hx_0 \cdot q$, we can construct a path $(f, p) =_{X\cdot\to Y} (g, q)$.

Note that, in contrast to the unpointed case, two pointed functions may not be equal even if they agree on all arguments. We also have a pointed version of univalence, and a means of relating pointed functions over equivalences:

Assertion 1.11. Let $X, Y : \mathcal{U}_*$ and a basepoint-preserving equivalence $(e, q) : X \simeq Y$ be given (here $q : ex_0 = y_0$). Then there is a path $\mathsf{ua}^\odot : X =_{\mathcal{U}_*} Y$.

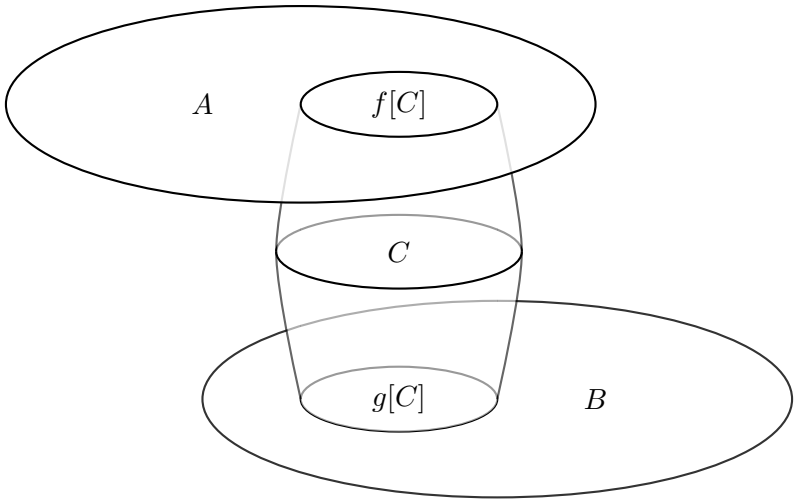
Assertion 1.12. Let a functions $(f, p) : X \cdot\!\to Y$, $(g, q) : X \cdot\!\to Z$ and a basepoint-preserving equivalence $(e, r) : Y \simeq Z$ be given. Then $(f, p) =^{W.X\cdot\to W}_{\mathsf{ua}^\odot(e,r)} (g, q)$ is equivalent to $(e, r) \circ (f, p) = (g, q)$.

Assertion 1.13. Let functions $(f, p) : X \cdot\!\to Z$, $(g, q) : Y \cdot\!\to Z$ and a basepoint-preserving equivalence $(e, r) : X \simeq Y$ be given. Then $(f, p) =^{W.W\cdot\to Z}_{\mathsf{ua}^\odot(e,r)} (g, q)$ is equivalent to $(f, p) = (g, q) \circ (e, r)$.

For any $X : \mathcal{U}_*$, the *loop space* ΩX is the pointed type $(x = x, \mathsf{refl})$. Composition of loops gives a pointed map $\mathsf{conc}^\odot : \Omega X \times \Omega X \cdot\!\to \Omega X$, where the basepoint of a product is of course the pair of basepoints.

1.3 Higher Inductive Types

As in standard intensional type theory, we may define inductive types such as the natural numbers $\mathbb{N}$ and the integers $\mathbb{Z}$. In addition to these, homotopy type theory introduces *higher inductive types*. Where an ordinary inductive type A is generated by a collection of constructors which introduce elements of A, a higher inductive type may also include a collection of *path constructors* which introduce identifications between elements. For our purposes, the main higher inductive type of interest is the *pushout type*. For any span of maps $A \overset{f}{\leftarrow} C \overset{g}{\to} B$, the pushout type $A \sqcup^C B$ (with functions left implicit) is generated by point constructors $\mathsf{left} : A \to A \sqcup^C B$ and $\mathsf{right} : B \to A \sqcup^C B$ along with a path constructor $\mathsf{glue} : \prod_{c:C} \mathsf{left}(fc) = \mathsf{right}(gc)$. Visually, we picture $A \sqcup^C B$ like so:



If we have a span of pointed spaces $X \overset{(f,p)}{\leftarrow\cdot} Z \overset{(g,q)}{\cdot\rightarrow} Y$, then we make $X \sqcup^Z Y$ a pointed type with basepoint $\mathsf{left}\ x_0$, and the maps left and right can be made pointed with basepoint-preservation paths refl and $\mathsf{ap}_{\mathsf{right}} q^{-1} \cdot \mathsf{glue}\ {z_0}^{-1} \cdot \mathsf{ap}_{\mathsf{left}} p$ respectively.

For any $P : A \sqcup^C B \to \mathcal{U}$, the induction principle for $A \sqcup^C B$ states that in order to construct a function $k : \prod_{\gamma : A \sqcup^C B} P(\gamma)$ it suffices to give

- a function $l : \prod_{a:A} P(\mathsf{left}\ a)$,
- a function $r : \prod_{b:B} P(\mathsf{right}\ b)$,
- a dependent homotopy $h : \prod_{c:C} l(fc) =^P_{\mathsf{glue}\ c} r(gc)$.

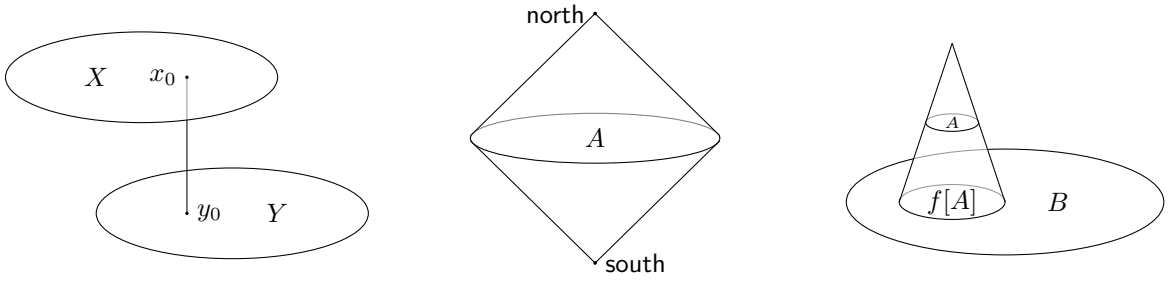
A function k defined in this way satisfies $k(\mathsf{left}\ a) \equiv l(a)$, $k(\mathsf{right}\ b) \equiv r(b)$, and $\mathsf{apd}_k(\mathsf{glue}\ c) = h(c)$. Note that we only assume the reduction for the path constructor holds propositionally – this is in part an artifact of limitations in the Agda formalization and in part due to uncertainty in the current understanding of the theory. In the non-dependent case where the target is a type D, the induction principle can be simplified to requiring functions $l : A \to D$, $r : B \to D$, and $h : \prod_{c:C} l(fc) = r(gc)$, and the resulting function $k : A \sqcup^C B \to D$ satisfies $\mathsf{ap}_k(\mathsf{glue}\ c) = h(c)$.

A few pushouts are of particular interest to us:

1. For $X, Y : \mathcal{U}_*$, the pushout of the span $X \overset{x_0}{\leftarrow\cdot} 1 \overset{y_0}{\cdot\rightarrow} Y$ is the *wedge* $X \vee Y$; in this case we write winl, winr, and $\mathsf{wglue} : \mathsf{winl}\ x_0 = \mathsf{winr}\ y_0$ for the three constructors.
2. For $A : \mathcal{U}$, the pushout of the span $1 \leftarrow A \to 1$ is the *suspension* ΣA; we write $\mathsf{north}, \mathsf{south} : \Sigma A$, and $\mathsf{merid} : A \to \mathsf{north} = \mathsf{south}$.
3. For any map $f : A \to B$, the pushout of the span $1 \leftarrow A \overset{f}{\to} B$ is the *cofiber type* $\mathsf{Cof}\ f$; we write $\mathsf{cfbase} : \mathsf{Cof}\ f$, $\mathsf{cfcod} : B \to \mathsf{Cof}\ f$, and $\mathsf{cfglue} : \prod_{a:A} \mathsf{cfbase} = \mathsf{cfcod}\ (fb)$.

4. For $A, B : \mathcal{U}$, the pushout of the span of projections $A \xleftarrow{\mathsf{fst}} A \times B \xrightarrow{\mathsf{snd}} B$ is the *join* $A * B$.

5. For $X, Y : \mathcal{U}_*$, the *smash product* $X \wedge Y$ is the cofiber type of the map $X \vee Y \to X \times Y$ which takes $\mathsf{winl}\ x$ to (x, y_0), $\mathsf{winr}\ y$ to (x_0, y), and wglue to $\mathsf{refl}_{(x_0,y_0)}$



(a) the wedge type $X \vee Y$ (b) the suspension type ΣA (c) the cofiber type $\mathsf{Cof}\ f$

Figure 1.1: Special cases of the pushout type

Finally, we define the 0-sphere S^0 to be the inductive type generated by two points true and false, and define the n-spheres inductively by $S^{n+1} :\equiv \Sigma S^n$.

1.4 n-Types and Truncation

For any type A, iterating the construction of identity types gives rise to an infinite hierarchy of structure: for $x, y : A$ we can consider the type of identifications $x =_A y$, for $p, q : x = y$ we can consider $p =_{x=y} q$, and so on. However, not every type has nontrivial structure at every level. A type A for which the structure cuts off at "level n" is called an n-type, defined precisely for $n \geq -2$ as follows:

- A is a (-2)-type if there exists $a : A$ and a homotopy $\prod_{x:A} a = x$. We say that A is *contractible*.
- For $n \geq -2$, A is an $(n+1)$-type if, for every $x, y : A$, the path type $x =_A y$ is an n-type.

We write $\mathsf{has\text{-}level}\ n\ A$ for the type of proofs that A is an n-type, that is, $\mathsf{has\text{-}level}\ (-2)\ A :\equiv \sum_{a:A} \prod_{x:A}(a = x)$ and $\mathsf{has\text{-}level}\ (n+1)\ A :\equiv \prod_{x,y:A} \mathsf{has\text{-}level}\ n\ (x = y)$. The unusual numbering scheme ensures that the 0-types are the *sets*, those types A for which any two paths $p, q : x =_A y$ can be identified – in particular, for which any path $p : x =_A x$ is equal to refl_x. The (-1)-types are the *mere propositions* or *truth values*, those types A for which any two points $x, y : A$ can be identified.

Given any type, we can obtain its best approximation as an n-type for some n via the *truncation modality* $\|-\|_n : \mathcal{U} \to \mathcal{U}$. The truncation modalities are definable as higher inductive types, but we will use a derivable characterization which is more convenient (see [8, §7.3]). For any type A, its n-truncation $\|A\|_n$ is generated by a constructor $|-|_n : A \to \|A\|_n$ as well as an element of $\mathsf{has\text{-}level}\ n\ \|A\|_n$. The accompanying induction principle is as follows:

Definition 1.14 (Truncation Induction)**.** Let $B : \|A\|_n \to \mathcal{U}$ be given. If Bt is an n-type for every $t : \|A\|_n$, then any map $f : \prod_{a:A} B|a|_n$ induces a map $f' : \prod_{t:\|A\|_n} Bt$ which satisfies $f'(|a|_n) \equiv f(a)$.

If $A : \mathcal{U}$ is an n-type, then $|-|_n : A \to \|A\|_n$ is an equivalence. Our main interest is in the *set truncation* $\|-\|_0$ and *propositional truncation* (or *squash*) $\|-\|_{-1}$. For a function type $A \to B$, the truncated type $\|A \to B\|_0$ is the set of homotopy classes of maps $A \to B$; this will be essential for exhibiting examples of cohomology theories. The (-1)-truncation is generally useful when we want to ignore the path structure of a particular type; we will see an example in the next section. We use the qualifier *mere* to describe (-1)-truncated types: for example, to inhabit $\|\sum_{a:A} B(a)\|_{-1}$ is to say there *merely exists* $a : A$ such that $B(a)$ holds. We will make use of the following lemma:

Assertion 1.15. For any $x, y : A$, the types $\|x = y\|_n$ and $|x|_{n+1} = |y|_{n+1}$ are equivalent.

1.5 Groups

A *group* consists of an underlying set (that is, a 0-type) G, a composition operation $(- \cdot -) : G \times G \to G$, an inverse operation $(-)^{-1} : G \to G$, and a unit $e : G$, along with paths expressing the group laws:

- for every $g : G$, paths $e \cdot g = g$ and $g \cdot e = g$,
- for every $g, h, k : G$, a path $(g \cdot h) \cdot k = g \cdot (h \cdot k)$,
- for every $g : G$, paths $g^{-1} \cdot g = e$ and $g \cdot g^{-1} = e$.

We abuse notation and write G for the group itself, specifying intention where ambiguous. We write Grp for the type of groups, and Ab for the type of abelian groups (which additionally satisfy $\prod_{g,h:G} g \cdot h = h \cdot g$). A *homomorphism* is a map between groups, which consists of a map $\varphi : G \to H$ of underlying sets along with an inhabitant of $\prod_{g_1,g_2:G} \varphi(g_1 \cdot g_2) = \varphi(g_1) \cdot \varphi(g_2)$. Since the underlying sets are required to be 0-types, two homomorphisms are equal precisely when their underlying functions are equal. Univalence guarantees that equality among groups is the expected notion of isomorphism:

Assertion 1.16. For any $G, H : \mathsf{Grp}$, the type $G =_{\mathsf{Grp}} H$ is equivalent to the type of homomorphisms $\varphi : G \to H$ whose underlying maps are equivalences.

For any homomorphism $\varphi : G \to H$, the *kernel* of φ is the set $\mathsf{Ker}\ \varphi :\equiv \Sigma_{g:G}(\varphi(g) = e)$ of elements of G which map to e. The *image* of φ is the set $\mathsf{Im}\ \varphi :\equiv \Sigma_{h:H}\|\Sigma_{g:G}(\varphi(g) = h)\|_{-1}$ of elements of H for which there merely exists $g : G$ with $\varphi(g) = h$. The use of mere existence guarantees that any element of H appears no more than once in the image, as opposed to once for every element of G which maps to it. A homomorphism φ is *injective* if $\varphi(g) = e$ implies $g = e$, equivalently if $\mathsf{Ker}\ \varphi = 1$. It is *surjective* if for any $h : H$ there merely exists $g : G$ with $\varphi(g) = h$.

Assertion 1.17. Let $\varphi : G \to H$ be a homomorphism. If φ is injective and surjective, then φ is an isomorphism.

As a sketch of the proof, recall that surjectivity gives for any element $h : H$ a term of type $\|\Sigma_{g:G}(\varphi(g) = h)\|_{-1}$. If φ is injective, then there is at most one $g : G$ such that $\varphi(g) = h$, which means that $\Sigma_{g:G}(\varphi(g) = h)$ is already a (-1)-type. Thus $\|\Sigma_{g:G}(\varphi(g) = h)\|_{-1} \simeq \Sigma_{g:G}(\varphi(g) = h)$ and we can transform each mere existence proof into a full existence proof, defining an inverse to φ by associating each h with its specified g.

Chapter 2

Squares and Cubes

Constructions involving higher inductive types commonly require us to build paths between paths. For example, consider the pushout $U \sqcup^{X \vee Y} V$ of a span $U \xleftarrow{f} X \vee Y \xrightarrow{g} V$. Say we wish to construct a function $h : U \sqcup^{X \vee Y} V \to C$. Per the induction principle, this requires in particular that we specify the value of $\mathsf{ap}_h(\mathsf{glue}\ w) : h\ (\mathsf{left}\ (f\ w)) = h\ (\mathsf{right}\ (g\ w))$. If we in turn specify the definition of $w \mapsto \mathsf{ap}_h(\mathsf{glue}\ w)$ by induction, we must relate the two paths $\mathsf{ap}_h(\mathsf{glue}\ (\mathsf{winl}\ x_0))$ and $\mathsf{ap}_h(\mathsf{glue}\ (\mathsf{winr}\ y_0))$ by giving the value of $\mathsf{apd}_{\mathsf{ap}_h \circ \mathsf{glue}}(\mathsf{wglue})$, which has type

$$\mathsf{ap}_h(\mathsf{glue}\ (\mathsf{winl}\ x_0)) =^{w.h(\mathsf{left}(fw))=h(\mathsf{right}(gw))}_{\mathsf{wglue}} \mathsf{ap}_h(\mathsf{glue}\ (\mathsf{winr}\ y_0))$$

Thus, constructing functions out of inductive types requires a means of dealing with dependent paths between paths. Assertion 1.7 tells us that the types $u =_p^{z.fz=gz} v$ and $u \cdot \mathsf{ap}_g(p) = \mathsf{ap}_f(p) \cdot v$ are equivalent. We can therefore think of a dependent path of type $u =_p^{z.fz=gz} v$ as expressing that the following square of paths commutes:

$$\begin{array}{ccc} fx & \xrightarrow{\mathsf{ap}_f p} & fy \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle v} \\ fy & \xrightarrow[\mathsf{ap}_g p]{} & fy \end{array}$$

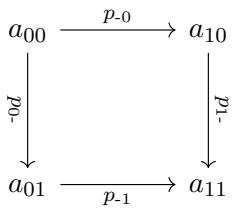
We can extend this perspective to higher dimensions:

dependent path in a family of path types	$\sim$	commutative square
dependent path in a family of square types	$\sim$	"commutative cube"

To accomplish this, we define inductive types which represent square and cube relationships directly, and which serve as higher-dimensional analogues of the identity type. These were first used in Agda by Licata and Brunerie in the library [13] and described on paper in [14].

2.1 The Square Type

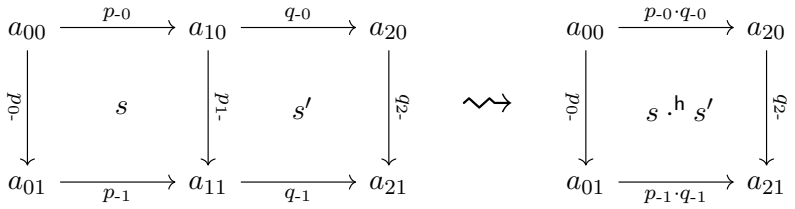
We define Square as an inductive family indexed by four points and four paths: a_{00}, a_{01}, a_{10}, $a_{11} : A$ along with $p_{0\text{-}} : a_{00} = a_{01}$, $p_{\text{-}0} : a_{00} = a_{10}$, $p_{\text{-}1} : a_{01} = a_{11}$, and $p_{1\text{-}} : a_{10} = a_{11}$.



We will write the fully applied type as $\mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ p_{1\text{-}}$, omitting the point arguments for brevity. The type is generated by

- For every $a : A$, the identity square at a, $\mathsf{srefl}_a : \mathsf{Square}\ \mathsf{refl}_a\ \mathsf{refl}_a\ \mathsf{refl}_a\ \mathsf{refl}_a$.

As with the identity type, this simple definition gives rise to a complex structure. To begin with, we have horizontal and vertical composition operations with corresponding identities. For example, the horizontal composition $s \cdot^{\mathsf{h}} s'$ as below can be defined by induction on s, with $\mathsf{srefl} \cdot^{\mathsf{h}} s' :\equiv s'$:



We can define a identity for horizontal composition $\mathsf{srefl\text{-}h}_p : \mathsf{Square}\ p\ \mathsf{refl}\ \mathsf{refl}\ p$ by taking $\mathsf{srefl\text{-}h}_{\mathsf{refl}} :\equiv \mathsf{srefl}$; we have $\mathsf{srefl\text{-}h} \cdot^{\mathsf{h}} s' = s'$ for any square s'. The fact that srefl-h is a right "identity," while true in some sense, is less easily expressed, since s and $s \cdot^{\mathsf{h}} \mathsf{srefl\text{-}h}$ may have different types. The vertical composition $\cdot^{\mathsf{v}}$ and identity srefl-v are analogously defined.

Using the horizontal and vertical identities, we can define "based" induction principles for squares, for example with a preselected left edge:

Lemma 2.1. Let $a_{00}, a_{01} : A$ and $p_{0\text{-}} : a_{00} = a_{01}$. Let a dependent family

$$P : \prod_{a_{10},a_{11}:A} \prod_{p_{\text{-}0}:a_{00}=a_{10}} \prod_{p_{\text{-}1}:a_{01}=a_{11}} \prod_{p_{1\text{-}}:a_{01}=a_{11}} (\mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ p_{1\text{-}} \to \mathcal{U})$$

be given (we will elide all but the square argument). In order to define a function

$$f : \prod_{a_{10},a_{11}:A} \prod_{p_{\text{-}0}:a_{00}=a_{10}} \prod_{p_{\text{-}1}:a_{01}=a_{11}} \prod_{p_{1\text{-}}:a_{01}=a_{11}} \prod_{s:\mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ p_{1\text{-}}} Ps$$

it suffices to give an element $d : P\ \mathsf{srefl\text{-}h}_{p_{0\text{-}}}$.

Proof. Let a square s : Square $p_{0\text{-}}$ $p_{\text{-}0}$ $p_{\text{-}1}$ $p_{1\text{-}}$ be given as above. By square induction we may assume $s \equiv$ srefl; we must construct an element of P srefl. Since $p_{0\text{-}} \equiv$ refl in this case, we have srefl-h$_{p_{0\text{-}}} \equiv$ srefl, and therefore $d : P$ srefl serves. □

In particular, if the left edge is refl, it suffices to show P srefl.

As with paths, functions act functorially on squares.

Lemma 2.2. Let a_{00}, a_{01}, a_{10}, $a_{11} : A$ and $p_{0\text{-}} : a_{00} = a_{01}$, $p_{\text{-}0} : a_{00} = a_{10}$, $p_{\text{-}1} : a_{01} = a_{11}$, $p_{1\text{-}} : a_{01} = a_{11}$. Let a square s : Square $p_{0\text{-}}$ $p_{\text{-}0}$ $p_{\text{-}1}$ $p_{1\text{-}}$ be given. For any function $f : A \to B$ we have a square ap-square s : Square (ap$_f p_{0\text{-}}$) (ap$_f p_{\text{-}0}$) (ap$_f p_{\text{-}1}$) (ap$_f p_{1\text{-}}$).

Proof. By square induction on s; when $s \equiv$ srefl we set ap-square $s :\equiv$ refl. □

In general, an element of a square type serves as a proof that its sides commute, and we can show that every square corresponds to a proof of commutativity.

Lemma 2.3. Let a_{00}, a_{01}, a_{10}, $a_{11} : A$ and $p_{0\text{-}} : a_{00} = a_{01}$, $p_{\text{-}0} : a_{00} = a_{10}$, $p_{\text{-}1} : a_{01} = a_{11}$, $p_{1\text{-}} : a_{01} = a_{11}$. The types $p_{0\text{-}} \cdot p_{\text{-}1} = p_{\text{-}0} \cdot p_{1\text{-}}$ and Square $p_{0\text{-}}$ $p_{\text{-}0}$ $p_{\text{-}1}$ $p_{1\text{-}}$ are equivalent.

Proof. We must define a function disc-to-square : $p_{0\text{-}} \cdot p_{\text{-}1} = p_{\text{-}0} \cdot p_{1\text{-}} \to$ Square $p_{0\text{-}}$ $p_{\text{-}0}$ $p_{\text{-}1}$ $p_{1\text{-}}$ and an inverse square-to-disc.

- For the forward direction, we are given $p_{0\text{-}}, p_{\text{-}0}, p_{\text{-}0}, p_{1\text{-}}$, and $\alpha : p_{0\text{-}} \cdot p_{\text{-}1} = p_{\text{-}0} \cdot p_{1\text{-}}$. By path induction, we assume that the first three paths are refl, which leaves α with type refl $= p_{1\text{-}}$. We can then induct on α, so that $p_{1\text{-}} \equiv$ refl as well. In this case, srefl gives the desired square.
- For the backward direction, we have $p_{0\text{-}}, p_{\text{-}0}, p_{\text{-}0}, p_{1\text{-}}$, and s : Square $p_{0\text{-}}$ $p_{\text{-}0}$ $p_{\text{-}1}$ $p_{1\text{-}}$. By square induction, we may assume $s \equiv$ srefl and therefore all four paths are refl. In this case, refl gives the desired path.

Showing that these constructions are mutually inverse is easily accomplished with the same pattern of inductions. □

Special cases of the previous lemma give us several ways of constructing squares from paths; for example, a path $\alpha : p =_{x=y} q$ gives rise to squares hsquare α : Square p refl refl q, vsquare α : refl p q refl and brsquare α : Square refl refl p q, and all of these constructions are equivalences. We also have an equivalence relating squares and dependent paths.

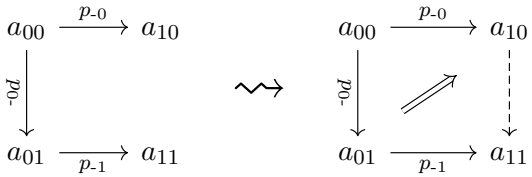
Lemma 2.4. Let $f, g : A \to B$, $x, y : A$, and $p : x =_A y$. For any $u : fx =_B gx$ and $v : fy =_B gy$, the types $u =_p^{z.fz=gz} v$ and Square u (ap$_f(p)$) (ap$_g(p)$) v are equivalent.

Proof. By induction we may assume $p \equiv$ refl, in which case hsquare gives the desired equivalence. □

Corollary 2.5. Let $f, g : A \to B$ be given along with $h : \prod_{a:A} fa = ga$. For any $x, y : A$ and $p : x = y$, we have a square $\mathsf{natural\text{-}sq}\ h\ p : \mathsf{Square}\ (hx)\ (\mathsf{ap}_f(p))\ (\mathsf{ap}_g(p))\ (hy)$. In particular, $\mathsf{natural\text{-}sq}\ h\ \mathsf{refl} = \mathsf{srefl\text{-}h}$.

Proof. This follows from the previous lemma by converting $\mathsf{apd}_h(p) : hx =_p^{a.fa=ga} hy$ to a square. The second condition may be confirmed by tracing definitions and induction. □

A key property of squares is the existence of square *fillers*. Given a box – the frame of a square minus one side – there is a (propositionally) unique fourth side, the *composite*, for which the square type is inhabited. This inhabitant is called the filler.



Theorem 2.6 (Square Fillers)**.** Let $a_{00}, a_{01}, a_{10}, a_{11} : A$ be given along with $p_{0\text{-}} : a_{00} = a_{01}$, $p_{\text{-}0} : a_{00} = a_{10}$, and $p_{\text{-}1} : a_{01} = a_{11}$. Then there exists a path $\mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1} : a_{10} = a_{11}$ and a square of type $\mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ (\mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1})$. Moreover, for any $q : a_{10} = a_{11}$ such that $\mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ q$ is inhabited, we have a path $q = \mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}$.

Proof. By path induction, we may assume $p_{\text{-}0}$ and $p_{\text{-}1}$ are both refl. In this case, we define $\mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1} :\equiv p_{0\text{-}}$. The square $\mathsf{srefl\text{-}h}_{p_{0\text{-}}}$ then serves as the desired filler. For the uniqueness criterion, let $q : a_{10} = a_{11}$ and $s : \mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ q$ be given. By square induction, we may assume s is srefl. In this case, both q and $\mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}$ are refl, so that $q = \mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}$ by reflexivity. □

Of course, this theorem generalizes for fillers with any missing face, not only the right. For the two-dimensional case, there is a simple explicit definition of the filler: $\mathsf{square\text{-}comp\text{-}r}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}$ must be ${p_{\text{-}0}}^{-1} \cdot p_{0\text{-}} \cdot p_{\text{-}1}$, the propositionally unique q satisfying $p_{0\text{-}} \cdot p_{\text{-}1} = p_{\text{-}0} \cdot q$. For cubes, however, the situation is more complex.

Finally, just as we have a notion of dependent path in a fibration, we have a notion of a dependent square. In the same way that we can express a dependent path in a family of simple path types as a square, we can express a dependent path in certain families of dependent path types as a dependent square. For any frame as pictured below, we define the type $\mathsf{Square}^B_s\ q_{0\text{-}}\ q_{\text{-}0}\ q_{\text{-}1}\ q_{1\text{-}}$ by square induction on s, setting $\mathsf{Square}^B_{\mathsf{srefl}}\ q_{0\text{-}}\ q_{\text{-}0}\ q_{\text{-}1}\ q_{1\text{-}} :\equiv \mathsf{Square}\ q_{0\text{-}}\ q_{\text{-}0}\ q_{\text{-}1}\ q_{1\text{-}}$.

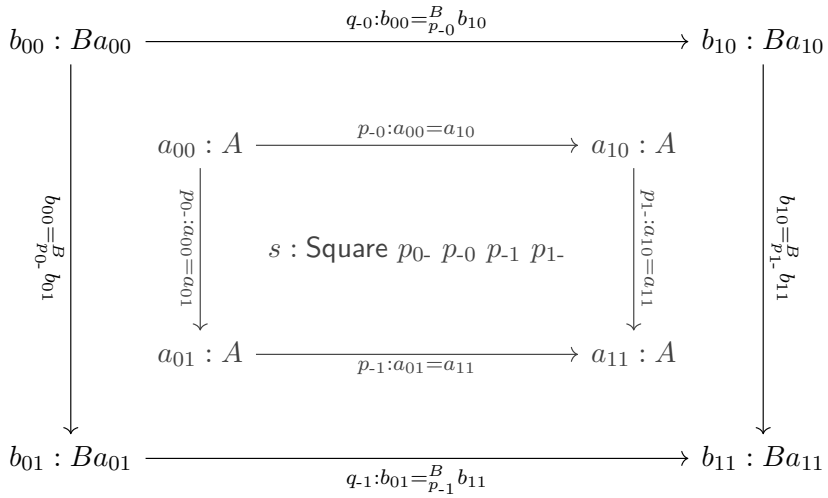


Figure 2.1: A dependent square. The inner square lies in the base space of the fibration $B : A \to \mathcal{U}$, with the outer square sitting above it in the total space.

Lemma 2.7. Let $A : \mathcal{U}$, $B : \mathcal{U}$, and $C : B \to \mathcal{U}$ along with $f, g : A \to B$, $q : \prod_{a:A} fa = ga$, $u : \prod_{a:A} C(fa)$, and $v : \prod_{a:A} C(ga)$ be given. For any $x, y : A$, $p : x = y$, $\alpha : ux =^{C}_{qx} vx$, and $\beta : uy =^{C}_{qy} vy$, the types $\alpha =^{z.uz =^{C}_{qz} vz}_{p} \beta$ and

$$\mathsf{Square}^{C}_{\mathsf{natural\text{-}sq}\ q\ p}\ \alpha\ (\mathsf{over\text{-}ap\text{-}in}_f(\mathsf{apd}_u p))\ (\mathsf{over\text{-}ap\text{-}in}_g(\mathsf{apd}_v p))\ \beta$$

are equivalent.

Proof. By path induction we may assume $p \equiv \mathsf{refl}$, so that $x \equiv y$. In this case our goal is to show $\alpha = \beta$ is equivalent to $\mathsf{Square}^{C}_{\mathsf{srefl\text{-}h}_{qx}}\ \alpha\ \mathsf{refl}\ \mathsf{refl}\ \beta$. By generalizing we may induct on qx, leaving us to prove that $\alpha = \beta$ is equivalent to square $\mathsf{Square}\ \alpha\ \mathsf{refl}\ \mathsf{refl}\ \beta$; in this case we have an equivalence given by $\mathsf{hsquare}$. □

2.2 The Cube Type

As with Square, the type family Cube is indexed by its frame: eight points, twelve paths, and six squares, which fit together in the appropriate way. We write $\mathsf{Cube}\ s_{left}\ s_{right}\ s_{back}\ s_{top}\ s_{bottom}\ s_{front}$ for the fully-applied type, leaving the other arguments implicit. The type is generated by

- For every $a : A$, the identity cube at a, $\mathsf{crefl}_a : \mathsf{Cube}\ \mathsf{srefl}_a\ \mathsf{srefl}_a\ \mathsf{srefl}_a\ \mathsf{srefl}_a\ \mathsf{srefl}_a\ \mathsf{srefl}_a$.

Just as squares can be used to represent dependent paths in a family of path types, cubes can be used to represent dependent paths in a family of square types, though the proof is more involved. First, let us observe some connections between paths, squares, and cubes: a square in a path type gives rise to a cube, as does a path in a square type.

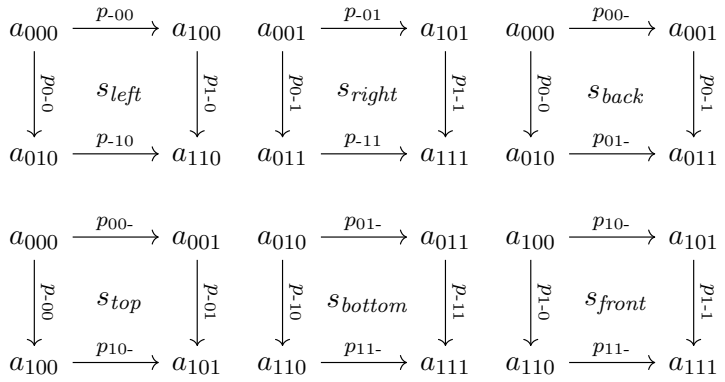


Figure 2.2: Faces of a cube.

Lemma 2.8. Let paths $p_{0\text{-}} : a_{00} = a_{01}$, $p_{\text{-}0} : a_{00} = a_{10}$, $p_{\text{-}1} : a_{01} = a_{11}$, and $p_{1\text{-}} : a_{10} = a_{11}$ be given. For any two squares $s, s' :$ Square $p_{0\text{-}}$ $p_{\text{-}0}$ $p_{\text{-}1}$ $p_{1\text{-}}$ with a path $\alpha : s = s'$, there is a cube of type Cube s s' srefl-h srefl-h srefl-h srefl-h.

Proof. By induction we may assume $s \equiv$ srefl and $\alpha \equiv$ refl, in which case we can take crefl. □

Lemma 2.9. Let paths $p_{00}, p_{01}, p_{10}, p_{11} : a_0 = a_1$ be given with two-dimensional paths $\alpha_{0\text{-}} : p_{00} = p_{01}$, $\alpha_{\text{-}0} : p_{00} = p_{10}$, $\alpha_{\text{-}1} : p_{01} = p_{11}$, and $\alpha_{1\text{-}} : p_{10} = p_{11}$. As a special case of Lemma 2.3, the two-dimensional paths correspond to squares $s_{0\text{-}} :$ Square refl p_{00} p_{01} refl, $s_{\text{-}0} :$ Square refl p_{00} p_{10} refl, $s_{\text{-}1} :$ Square refl p_{01} p_{11} refl, and $s_{1\text{-}} :$ Square refl p_{10} p_{11} refl. Given a square of type Square $\alpha_{0\text{-}}$ $\alpha_{\text{-}0}$ $\alpha_{\text{-}1}$ $\alpha_{1\text{-}}$, we can construct a cube of type Cube srefl srefl $s_{0\text{-}}$ $s_{\text{-}0}$ $s_{\text{-}1}$ $s_{1\text{-}}$.

Proof. Let a square of type Square $\alpha_{0\text{-}}$ $\alpha_{\text{-}0}$ $\alpha_{\text{-}1}$ $\alpha_{1\text{-}}$ be given. By square induction, we may assume this square is srefl. In this case, the two-dimensional paths are all refl, and per the proof of Lemma 2.3 the corresponding squares are all srefl. In this case, crefl gives the desired cube. □

We can prove the existence of cube fillers by reducing to the case of squares and using the square filling theorem.

Theorem 2.10 (Cube Fillers)**.** Let s_{left}, s_{back}, s_{top}, s_{bottom}, s_{front} be five faces of a cube, as in Figure 2.2. Then there is a propositionally unique composite square s' such that
Cube s_{left} s' s_{back} s_{top} s_{bottom} s_{front} is inhabited.

Proof. By square induction, we may assume s_{front} and s_{back} are srefl; these squares have no paths in common, so there is no problem of interference in inducting on both. In this case, the other squares have refl on two sides, and correspond by Lemma 2.3 to paths α_{left}, α_{top}, and α_{bottom}. By square filling (Theorem 2.6), there is a path α' and a square $t :$ Square α_{left} α_{top} α_{bottom} α'. The path α' itself corresponds to a square s'. By a variant of the previous lemma (taking back and front as the degenerate faces), t gives rise to a corresponding cube, of type Cube s_{left} s' srefl s_{top} s_{bottom} srefl. Thus s' is a composite for the given box.

For the uniqueness condition, let another right face r and a cube $c : \mathsf{Cube}\ s_{left}\ r\ s_{back}\ s_{top}\ s_{bottom}\ s_{front}$ be given. We go by cube induction on c. In the case $c \equiv \mathsf{crefl}$, all faces of c are srefl, and one can check that the composite defined above reduces to srefl in this case. Hence r is equal to the composite. □

Using the uniqueness of cube composites, we can give a converse to Lemma 2.8. This lemma in turn enables us to construct dependent paths in square types by inhabiting certain cubes.

Lemma 2.11. Let paths $p_{0\text{-}} : a_{00} = a_{01}$, $p_{\text{-}0} : a_{00} = a_{10}$, $p_{\text{-}1} : a_{01} = a_{11}$, and $p_{1\text{-}} : a_{10} = a_{11}$ be given. For any two squares $s, s' : \mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ p_{1\text{-}}$ and a cube $c : \mathsf{Cube}\ s\ s'\ \mathsf{srefl\text{-}h}\ \mathsf{srefl\text{-}h}\ \mathsf{srefl\text{-}h}\ \mathsf{srefl\text{-}h}$, there is a path $\alpha : s = s'$.

Proof. By induction we may assume $s \equiv \mathsf{srefl}$, in which case c has type $\mathsf{Cube}\ \mathsf{srefl}\ s'\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl}$. Since $\mathsf{Cube}\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl}$ is inhabited, namely by crefl, it follows by uniqueness of cube composites that $s' = \mathsf{srefl}$. □

Theorem 2.12 (Dependent Paths from Cubes)**.** Let $f_{00}, f_{01}, f_{10}, f_{11} : A \to B$ be given along with $p_{0\text{-}} : \prod_{a:A} f_{00}a = f_{01}a$, $p_{\text{-}0} : \prod_{a:A} f_{00}a = f_{10}a$, $p_{\text{-}1} : \prod_{a:A} f_{01}a = f_{11}a$, and $p_{1\text{-}} : \prod_{a:A} f_{10}a = f_{11}a$. For any $x, y : A$, $q : x = y$, and two squares $s_x : \mathsf{Square}\ (p_{0\text{-}}x)\ (p_{\text{-}0}x)\ (p_{\text{-}1}x)\ (p_{1\text{-}}x)$ and $s_y : \mathsf{Square}\ (p_{0\text{-}}y)\ (p_{\text{-}0}y)\ (p_{\text{-}1}y)\ (p_{1\text{-}}y)$, we can construct a term of type

$$s_x =_q^{z.\mathsf{Square}\ (p_{0\text{-}}z)\ (p_{\text{-}0}z)\ (p_{\text{-}1}z)\ (p_{1\text{-}}z)} s_y$$

from a term of type

$$\mathsf{Cube}\ s_x\ s_y\ (\mathsf{natural\text{-}sq}\ p_{0\text{-}}\ q)\ (\mathsf{natural\text{-}sq}\ p_{\text{-}0}\ q)\ (\mathsf{natural\text{-}sq}\ p_{\text{-}1}\ q)\ (\mathsf{natural\text{-}sq}\ p_{1\text{-}}\ q)$$

Proof. By path induction, we may assume $q \equiv \mathsf{refl}$. In this case the theorem follows from the previous lemma. □

This concludes the collection of basic facts we will need regarding squares and cubes.

Chapter 3

Cohomology Theories

For any type X and $n \geq 1$, we have the homotopy group $\pi_n(X) :\equiv \|S^n \cdot\!\!\to X\|_0$, the set of homotopy classes of n-dimensional loops in X with composition as the group operation. The homotopy groups of a type give an algebraic description of its structure at each dimension, which we can use to differentiate between types: for example, we know $S^1 \neq S^2$, since $\pi_1(S^1) = \mathbb{Z}$ and $\pi_1(S^2) = 1$. Unfortunately, it is often difficult to compute homotopy groups – the homotopy groups of spheres are extremely complex and many remain unknown.

Cohomology is an alternative tool. There are many notions of cohomology, but most are united under a common foundation: a family $(H^n)_{n:\mathbb{Z}}$ of contravariant functors, from pointed types to abelian groups, which satisfies the *Eilenberg-Steenrod axioms*. Such a collection is called a *cohomology theory*. As with homotopy groups, the cohomology groups of a type give us an algebraic description of its dimensional structure. Cohomology groups have the advantage of being more easily computable for many common types; we will give an example of a cohomology theory for which we can quickly calculate $H^n(S^m)$ for any n, m.

We will first define the axioms that a cohomology theory must satisfy. These were originally presented by Eilenberg and Steenrod in [6]. Their type-theoretic translation was developed at the Institute for Advanced Study during Special Year on Univalent Foundations of Mathematics, by a group including Brunerie, Finster, Lumsdaine, Licata, and Shulman [7, 22]. Next, we describe a method of constructing models, using the notion of *spectrum*. Finally, we outline the construction of the Eilenberg-MacLane spectra originally given in [15] and discuss the construction of other spectra.

3.1 Eilenberg-Steenrod Axioms

Let a collection of functors $(H^n)_{n:\mathbb{Z}}$ from pointed types to abelian groups be given. We abbreviate the functorial action $H^n(f, p)$ for $(f, p) : X \cdot\!\!\to Y$ as $(f, p)^*$ where unambiguous. We say that H is a *(generalized) cohomology theory* if it satisfies the following axioms:

(Suspension) For any $n : \mathbb{Z}$ and $X : \mathcal{U}_*$, there is an isomorphism $\mathsf{susp\text{-}iso}^n_X : H^{n+1}(\Sigma X) \simeq H^n(X)$. Moreover, $\mathsf{susp\text{-}iso}^n_X$ is natural in X, that is, for $(f, p) : X \cdot\!\!\to Y$ we have $\mathsf{susp\text{-}iso}^n_X \circ (\Sigma(f, p))^* = (f, p)^* \circ \mathsf{susp\text{-}iso}^n_Y$.

(Exactness) For any $n : \mathbb{Z}$, $X, Y : \mathcal{U}_*$, and $(f, p) : X \cdot\!\!\to Y$, the sequence of homomorphisms

$$H^n(\mathsf{Cof}(f, p)) \xrightarrow{\mathsf{cfcod}^{\odot *}} H^n(Y) \xrightarrow{(f,p)^*} H^n(X)$$

is *exact*. We say that a sequence of homomorphisms $\psi_n : G_n \to G_{n+1}$ (in a finite or infinite range of integers n) is exact if the kernel of each map is the image of the preceding map, in the sense that for $g : G_{n+1}$ we have $\psi_{n+1}(g) = e$ if and only if there merely exists $h : G_n$ such that $\psi_n(h) = g$.

We say that H is an *ordinary cohomology theory* if it also satisfies an additional axiom:

(Dimension) For any $n : \mathbb{Z}$ with $n \neq 0$, $H^n(S^0) \simeq 1$.

The cohomology groups of a large class of types can be expressed in terms of the cohomology groups of S^0. In an ordinary theory, they can be expressed in terms of the single group $H^0(S^0)$.

Classically, there is an additional axiom, which describes the cohomology of a general wedge sum. Given $I : \mathcal{U}$ and $X : I \to Ptd$, the general wedge sum $\bigvee_{i:I} X_i$ is generated by the following constructors:

- an element $\mathsf{wbase} : \bigvee_{i:I} X_i$,
- for $i : I$ and $x : X_i$, an element $\mathsf{win}_i x : \bigvee_{i:I} X_i$,
- for $i : I$, a path $\mathsf{wglue}_i : \mathsf{wbase} = \mathsf{win}_i x_0^i$, where x_0^i is the basepoint of X_i.

The axiom can then be stated like so:

(Additivity?) For any $I : \mathsf{Set}$ and $X : I \to \mathcal{U}_*$ the map $H^n(\bigvee_{i:I} X_i) \to \prod_{i:I} H^n(X_i)$ sending g to $\lambda i.\mathsf{win}_i{}^* g$ is an equivalence.

We will not assume this axiom, as the cohomology theories we present will not satisfy it. To prove that this axiom holds in typical theories requires a form of the axiom of choice which is not available by default in homotopy type theory. In §3.2 and §3.3, we will discuss the issues with verifying this axiom and the special cases in which it holds. We prove the case where $I = 2$ in §4.2 using only the Exactness Axiom.

3.2 Cohomology Theories from Spectra

Recall that we defined the homotopy groups of X as $\pi_n(X) :\equiv \|S^n \cdot\!\!\to X\|_0$. Since $S^{n+1} = \Sigma S^n$, we can take advantage of the fact that suspension is adjoint to loop space in the category of pointed

spaces. For example, this tells us that $\pi_n(X) = \|\Omega^n X\|_0$ and that $\pi_{n+1}(X) = \pi_n(\Omega X)$. For cohomology, we use this adjoint property in the opposite direction: rather than look at maps out of a sequence of spaces related by the suspension functor, we look at maps into a sequence of spaces related by the loop space functor. We say a family of pointed types $E : \mathbb{Z} \to \mathcal{U}_*$ is a *spectrum* if $E_n = \Omega E_{n+1}$ for every $n : \mathbb{Z}$. For such a family, we can define homotopy groups $\pi_n(E)$ for all $n : \mathbb{Z}$ by taking $\pi_n(E) :\equiv \pi_{n+k}(E_k)$, choosing k sufficiently large.

For any pointed types X and Y, we write $\mathsf{To\Omega Group}(X, Y)$ for the group on $\|X \cdot\to \Omega Y\|_0$ given by pointwise composition, which is to say

$$|f|_0 \otimes |g|_0 :\equiv |\mathsf{conc}^{\odot} \circ \langle f, g\rangle|_0$$

Given a spectrum $E : \mathbb{Z} \to \mathcal{U}_*$, we define a contravariant functor H by

$$\begin{aligned} H^n(X) &:\equiv \mathsf{To\Omega Group}(X, E_{n+1}) \quad (\equiv \|X \cdot\to \Omega E_{n+1}\|_0) \\ f^*(|g|_0) &:\equiv |g \circ f|_0 \end{aligned}$$

Note that $H^n(X)$ is equivalent as a type to $\|X \cdot\to E_n\|_0$. Composition in $H^n(X)$ is abelian: we have $\Omega E_{n+1} = \Omega^2 E_{n+2}$, and composition is commutative in higher loop spaces by the Eckmann-Hilton argument (see [8, Theorem 2.1.6]). Thus H is a contravariant functor from pointed types to abelian groups. We claim that H is a cohomology theory.

3.2.1 Suspension Axiom

It suffices to show that for any $X, Y : \mathcal{U}_*$ there is an group isomorphism $\mathsf{To\Omega Group}(\Sigma X, Y) \simeq \mathsf{To\Omega Group}(X, \Omega Y)$ and that this isomorphism is natural in X. To accomplish this, we first prove a counit-unit adjunction $\Sigma \dashv \Omega$, where we consider the two as functors $\mathcal{U}_* \to \mathcal{U}_*$.

Theorem 3.1 ($\Sigma \dashv \Omega$, Counit-Unit Adjunction)**.** There exist natural transformations $\eta : 1_{\mathcal{U}_*} \to \Omega\Sigma$ and $\epsilon : \Sigma\Omega \to 1_{\mathcal{U}_*}$ such that

1. for $X : \mathcal{U}_*$, $\epsilon_{\Sigma X} \circ \Sigma\eta_X = \mathsf{id}^{\odot}_{\Sigma X}$, and
2. for $X : \mathcal{U}_*$, $\mathsf{ap}^{\odot}_{\epsilon_X} \circ \eta_{\Omega X} = \mathsf{id}^{\odot}_{\Omega X}$.

Proof. For $X : \mathcal{U}_*$, we define $\eta_X x = \mathsf{merid}\ x \cdot (\mathsf{merid}\ x_0)^{-1}$, taking $\cdot\text{-}\mathsf{inv\text{-}r}\ (\mathsf{merid}\ x_0)$ as the proof that it preserves basepoint. We define ϵ_X by recursion on the suspension type:

$$\begin{aligned} &\epsilon_X : \Sigma\Omega X \to X \\ &\epsilon_X \mathsf{north} :\equiv x_0 \\ &\epsilon_X \mathsf{south} :\equiv x_0 \\ &\mathsf{ap}_{\epsilon_X}(\mathsf{merid}\ p) := p \end{aligned}$$

and take refl as the proof that it preserves basepoint. We confirm naturality, omitting for brevity the (straightforward) proofs that basepoint-preservation paths agree:

- For all $(f, p) : X \cdot\!\!\to Y$ and $x : X$, $\eta_Y(fx) = \mathsf{ap}_{\Sigma f}(\eta_X x)$:
  For any $x : X$, we have

$$\begin{aligned}\eta_Y(fx) &\equiv \mathsf{merid}\ (fx) \cdot \mathsf{merid}\ (fx_0)^{-1} \\ &= \mathsf{ap}_{\Sigma f}(\mathsf{merid}\ x \cdot (\mathsf{merid}\ x_0)^{-1}) \\ &\equiv \mathsf{ap}_{\Sigma f}(\eta_X x)\end{aligned}$$

- For all $(f, p) : X \cdot\!\!\to Y$ and $\sigma : \Sigma\Omega X$, $\epsilon_y(\Sigma(\mathsf{ap}_f)\sigma) = f(\epsilon_X \sigma)$:
  We may assume by path induction that $p \equiv \mathsf{refl}$, so that $fx_0 \equiv y_0$. We go by induction on $\sigma : \Sigma\Omega X$. We have definitionally that

$$\begin{aligned}\epsilon_Y((\Sigma\mathsf{ap}_f)\mathsf{north}) &\equiv \epsilon_Y \mathsf{north} \equiv y_0 \equiv fx_0 \equiv f(\epsilon_X \mathsf{north}) \\ \epsilon_Y((\Sigma\mathsf{ap}_f)\mathsf{south}) &\equiv \epsilon_Y \mathsf{south} \equiv y_0 \equiv fx_0 \equiv f(\epsilon_X \mathsf{south})\end{aligned}$$

  For the merid case, we must show for $q : \Omega X$ that $\mathsf{refl} =^{\sigma.\epsilon_y(\Sigma(\mathsf{ap}_f)\sigma)=f(\epsilon_X\sigma)}_{\mathsf{merid}\ q} \mathsf{refl}$. By Lemma 2.4, it suffices to give a square

$$\begin{array}{ccc} y_0 & \xrightarrow{\mathsf{ap}_{\epsilon_Y \circ \Sigma\mathsf{ap}_f}(\mathsf{merid}\ q)} & y_0 \\ {\scriptstyle \mathsf{refl}}\downarrow & & \downarrow{\scriptstyle \mathsf{refl}} \\ y_0 & \xrightarrow[\mathsf{ap}_{f\circ\epsilon_X}(\mathsf{merid}\ q)]{} & y_0 \end{array}$$

  Since $\mathsf{ap}_{\epsilon_Y \circ \Sigma\mathsf{ap}_f}(\mathsf{merid}\ q) = \mathsf{ap}_{\epsilon_Y}(\mathsf{merid}\ (\mathsf{ap}_f q)) = \mathsf{ap}_f q$ and $\mathsf{ap}_{f\circ\epsilon_X}(\mathsf{merid}\ q) = \mathsf{ap}_f q$, we see that the square is satisfied.

Now we show that the counit-unit laws hold, again omitting basepoint-preservation computations.

- for $X : \mathcal{U}_*$ and $\sigma : \Sigma X$, $\epsilon_{\Sigma X}((\Sigma\eta_X)\sigma) = \sigma$:
  We go by induction on $\sigma : \Sigma X$. We have

$$\begin{aligned}\epsilon_{\Sigma X}((\Sigma\eta_X)\mathsf{north}) &\equiv \epsilon_{\Sigma X}\mathsf{north} \equiv \mathsf{north} \\ \epsilon_{\Sigma X}((\Sigma\eta_X)\mathsf{south}) &\equiv \epsilon_{\Sigma X}\mathsf{south} \equiv \mathsf{north}\end{aligned}$$

  so we can take refl as our proof in the north case and $\mathsf{merid}\ x_0$ in the south case. For the merid case we must show then show that for $x : X$ we have $\mathsf{refl} =^{\sigma.\epsilon_{\Sigma X}((\Sigma\eta_X)\sigma)=\sigma}_{\mathsf{merid}\ x} \mathsf{merid}\ x_0$. By Lemma 2.4 this reduces to showing a square

$$\begin{array}{ccc} \mathsf{north} & \xrightarrow{\mathsf{ap}_{\epsilon_{\Sigma X}\circ\Sigma\eta_X}(\mathsf{merid}\ x)} & \mathsf{north} \\ {\scriptstyle \mathsf{refl}}\downarrow & & \downarrow{\scriptstyle \mathsf{merid}\ x_0} \\ \mathsf{north} & \xrightarrow{\mathsf{merid}\ x} & \mathsf{south} \end{array}$$

We have $\mathsf{ap}_{\epsilon_{\Sigma X} \circ \Sigma \eta_X}(\mathsf{merid}\ x) = \mathsf{ap}_{\epsilon_{\Sigma X}}(\mathsf{merid}\ (\eta_X x)) = \eta_X x \equiv \mathsf{merid}\ x \cdot (\mathsf{merid}\ x_0)^{-1}$, and so we see that the square commutes.

- for $X : \mathcal{U}_*$ and $p : \Omega X$, $\mathsf{ap}_{\epsilon_X}(\eta_{\Omega X} p) = p$:
  We have $\mathsf{ap}_{\epsilon_X}(\eta_{\Omega X} p) \equiv \mathsf{ap}_{\epsilon_X}(\mathsf{merid}\ p \cdot (\mathsf{merid}\ \mathsf{refl})^{-1}) = p \cdot \mathsf{refl}^{-1} = p$.

□

We now take advantage of two category-theoretic results. Since their proofs are no different here than in the standard setting, we leave them to the reader; one may refer to e.g. [2, p.235-236] for more detail.

Assertion 3.2. Let $F, G : \mathcal{U}_* \to \mathcal{U}_*$ be functors and $\eta : 1_{\mathcal{U}_*} \to GF$ and $\epsilon : FG \to 1_{\mathcal{U}_*}$ be natural transformations such that, for $X : \mathcal{U}_*$, $\epsilon_{FX} \circ F\eta_X = \mathsf{id}^{\odot}_{FX}$ and $G\epsilon_X \circ \eta_{GX} = \mathsf{id}^{\odot}_{GX}$. Then for $X, Y : \mathcal{U}_*$ we have an equivalence $e : (FX \cdot\!\to Y) \simeq (X \cdot\!\to GY)$, natural in both X and Y.

Assertion 3.3. Given the hypotheses of Assertion 3.2 and $X, Y : \mathcal{U}_*$, the map $\langle G\mathsf{fst}^{\odot}, G\mathsf{snd}^{\odot}\rangle : G(X \times Y) \cdot\!\to GX \times GY$ is an equivalence. For any $f : X \times Y \cdot\!\to Z$, we have a map $G_2 f : GX \times GY \cdot\!\to GZ$ defined by $G_2 f = Gf \circ \langle G\mathsf{fst}^{\odot}, G\mathsf{snd}^{\odot}\rangle^{-1}$. This map is natural with respect to e in the sense that, for $r_1 : FX \cdot\!\to Y$, $r_2 : FX \cdot\!\to Z$, and $f : Y \times Z \cdot\!\to W$, we have $e(f \circ \langle r_1, r_2\rangle) = G_2 f \circ \langle er_1, er_2\rangle$.

Observe that for $G = \Omega$, we have $G_2 f = \mathsf{ap2}^{\odot}_f$, since $\mathsf{ap2}^{\odot}_f \circ \langle \mathsf{ap}^{\odot}_{\mathsf{fst}^{\odot}}, \mathsf{ap}^{\odot}_{\mathsf{snd}^{\odot}}\rangle = \mathsf{ap}^{\odot}_f$. Using these two assertions, we can finish our proof of the Suspension Axiom. Let $X, Y : \mathcal{U}_*$ be given. By Assertion 3.2 we have an equivalence $e : (\Sigma X \cdot\!\to \Omega Y) \simeq (X \cdot\!\to \Omega^2 Y)$ which is natural in X and Y. To show that this gives rise to an group isomorphism $\mathsf{To}\Omega\mathsf{Group}(\Sigma X, Y) \simeq \mathsf{To}\Omega\mathsf{Group}(X, \Omega Y)$, it remains to show that this equivalence preserves composition, that is, for $f, g : \Sigma X \cdot\!\to \Omega Y$ we have $e(\mathsf{conc}^{\odot} \circ \langle f, g\rangle) = \mathsf{conc}^{\odot} \circ \langle ef, eg\rangle$. From Assertion 3.3 we have $e(\mathsf{conc}^{\odot} \circ \langle f, g\rangle) = \mathsf{ap2}^{\odot}_{\mathsf{conc}^{\odot}} \circ \langle ef, eg\rangle$, and the Eckmann-Hilton argument gives $\mathsf{ap2}^{\odot}_{\mathsf{conc}^{\odot}} = \mathsf{conc}^{\odot}$, thus completing the proof.

3.2.2 Exactness Axiom

Let $n : \mathbb{Z}$, $X, Y : \mathcal{U}_*$, and $(f, p) : X \cdot\!\to Y$ be given. We must show that the sequence of homomorphisms

$$\|\mathsf{Cof}(f,p) \cdot\!\to \Omega E_{n+1}\|_0 \xrightarrow{(-)\circ\mathsf{cfcod}^{\odot}} \|Y \cdot\!\to \Omega E_{n+1}\|_0 \xrightarrow{(-)\circ(f,p)} \|X \cdot\!\to \Omega E_{n+1}\|_0$$

is exact. Intuitively, this is true because a map $Y \cdot\!\to \Omega E_{n+1}$ factors as a map $\mathsf{Cof}(f,p) \cdot\!\to \Omega E_{n+1}$ composed with $\mathsf{cfcod}^{\odot}$ precisely when it takes all elements of X to the basepoint of ΩE_{n+1}.

First we show that any element of the image of $(-)\circ\mathsf{cfcod}^{\odot}$ is in the kernel of $(-)\circ(f,p)$. Since we are trying to inhabit path types in the set $\|X \cdot\!\to \Omega E_{n+1}\|_0$, our goal is a mere proposition, and we may therefore eliminate from 0- and (-1)-truncations in our proof. Let $|(g,q)|_0 : \|Y \cdot\!\to \Omega E_{n+1}\|_0$, and assume it is in the image of $(-) \circ \mathsf{cfcod}^{\odot}$. We thus have a pair $||(h,r)|_0, \alpha|_{-1}$ where $(h,r) : \mathsf{Cof} f \cdot\!\to$

ΩE_{n+1}, and α is a path $(h,r) \circ \mathsf{cfcod}^\odot = (g,q)$. We want to show that $(g,q) \circ (f,p) = (\lambda x.\mathsf{refl}, \mathsf{refl})$. We have

$$\begin{aligned}(g,q) \circ (f,p) &= (h,r) \circ \mathsf{cfcod}^\odot \circ (f,p) && \text{(by } \alpha\text{)} \\ &= (g,q) \circ (\lambda x.\mathsf{cfbase}, \mathsf{refl}) && \text{(by } \mathsf{cfglue}\text{)} \\ &= (\lambda x.\mathsf{refl}, \mathsf{refl})\end{aligned}$$

Second, we must show that any element of the kernel of $(-) \circ (f,p)$ is in the image of $(-)\mathsf{cfcod}^\odot$. Our goal is a (-1)-truncated type, so we may eliminate from 0- and (-1)-truncations in our proof. Let $|(g,q)|_0 : \|Y \cdot\!\!\to \Omega E_{n+1}\|_0$, $|(h,r)|_0 : \|\mathsf{Cof} f \cdot\!\!\to \Omega E_{n+1}\|_0$ be given, and assume that we have $|(g,q) \circ (f,p)|_0 = |(\lambda x.\mathsf{refl}, \mathsf{refl})|_0$. By Assertion 1.15, this path gives rise to a truncated path of type $\|(g,q) \circ (f,p) = (\lambda x.\mathsf{refl}, \mathsf{refl})\|_{-1}$, from which we can extract a path $\beta : (g,q) \circ (f,p) = (\lambda x.\mathsf{refl}, \mathsf{refl})$. We can define a map $k : \mathsf{Cof} f \cdot\!\!\to \Omega E_{n+1}$ by recursion on the cofiber type:

$$\begin{aligned}&k : \mathsf{Cof} f \to \Omega E_{n+1} \\ &k\ \mathsf{cfbase} :\equiv \mathsf{refl} \\ &k\ (\mathsf{cfcod}\ y) :\equiv g\, y \\ &\mathsf{ap}_k(\mathsf{cfglue}\ x) := \mathsf{ap}_{\lambda r.(\mathsf{fst}\, r)x}(\beta^{-1})\end{aligned}$$

We have $k \circ \mathsf{cfcod} = g$ definitionally. By Assertion 1.10, to complete the proof we must choose for k's basepoint-preservation proof a path s such that $\mathsf{ap}_k(\mathsf{ap}_{\mathsf{cfcod}}(p^{-1}) \cdot \mathsf{cfglue}\ x_0) \cdot s = q$. Thus we define $s :\equiv \mathsf{ap}_k(\mathsf{ap}_{\mathsf{cfcod}}(p^{-1}) \cdot \mathsf{cfglue}\ x_0)^{-1} \cdot q$.

3.2.3 Additivity Axiom

As noted in §3.1, we will not have the additivity axiom in general. For $I : \mathsf{Set}$ and $X : I \to \mathcal{U}_*$, we have $H^n(\bigvee_{i:I}(X_i)) \equiv \|(\bigvee_{i:I} X_i) \cdot\!\!\to \Omega E_{n+1}\|_0 = \|\prod_{i:I}(X_i \cdot\!\!\to \Omega E_{n+1})\|_0$, while $\prod_{i:I} H^n(\bigvee_{i:I} X_i) \equiv \prod_{i:I} \|X_i \cdot\!\!\to \Omega E_{n+1}\|_0$. In general, the existence of an equivalence

$$\left\|\prod_{i:I}(X_i \cdot\!\!\to \Omega E_{n+1})\right\|_0 \simeq \prod_{i:I} \|X_i \cdot\!\!\to \Omega E_{n+1}\|_0$$

requires some sort of choice axiom. While it is always true for finite I, it may fail even for $I = \mathbb{N}$ (see [22]). In [8, Exercise 7.8] is defined a family of choice axioms

$$\mathsf{AC}_{k,m} :\equiv \prod_{I:\mathsf{Set}} \prod_{A:I\to k\text{-}\mathsf{Type}} \left(\prod_{i:I} \|A(i)\|_m \to \left\|\prod_{i:I} A(i)\right\|_m\right)$$

The useful axioms in this case are strengthened versions of the axioms $\mathsf{AC}_{k,0}$, requiring the existence not only of a function but an equivalence (a natural formulation would require that the existing function from $\|\prod_{i:I} A(i)\|_m$ to $\prod_{i:I} \|A(i)\|_m$ is an equivalence). Depending on the spectrum E, we may also need to break the requirement that $A(i)$ is truncated; $X_i \cdot\!\!\to \Omega E_{n+1}$ is a k-type if and only if E_n is, and there are spectra for which E_n is not a k-type for any k (see §3.3).

3.2.4 Dimension Axiom

To see when the dimension axiom will hold, observe that

$$H^n(S^m) = \|S^m \cdot\!\to \Omega E_{n+1}\|_0 = \|S^m \cdot\!\to E_n\|_0 = \pi_m(E_n) = \pi_{m-n}(E)$$

Thus, the cohomology groups of the spheres are determined by the homotopy groups of the spectrum. The cohomology theory satisfies the dimension axiom if and only if $\pi_n(E) = 1$ for all $n \neq 0$.

3.3 Constructing Spectra

Now that have a means of building cohomology theories from spectra, we turn to constructing spectra themselves. The simplest class of spectra, which gives rise to ordinary cohomology theories, are the Eilenberg-MacLane spectra. For any abelian group G and $n : \mathbb{N}$, the *Eilenberg-MacLane space* $K(G, n)$ is an n-type which satisfies

$$\pi_k(K(G,n)) = \begin{cases} G, & \text{if } k = n \\ 1, & \text{otherwise} \end{cases}$$

(This is an equality of groups for $k > 0$, but of pointed sets for $k = 0$.) In addition, we have that $\Omega K(G, n+1) = K(G, n)$, thus giving rise to a spectrum E defined by

$$E_n :\equiv \begin{cases} K(G,n), & \text{if } n \geq 0 \\ 1, & \text{if } n < 0 \end{cases}$$

A construction of these types in homotopy type theory and verification of their properties, developed by Licata and Finster, is presented in detail in [15]; we only sketch it. The zeroth EM-space, $K(G, 0)$, is defined to be G's underlying pointed set. The first, $K(G, 1)$, is defined as the 1-truncation of a higher inductive type: it is generated by a single basepoint base, a loop $\mathsf{loop}\ g$ for every $g : G$, and paths enforcing that $\mathsf{loop}\ e = \mathsf{refl}$ and $\mathsf{loop}\ (gh) = \mathsf{loop}\ g \cdot \mathsf{loop}\ h$. The higher EM-spaces are defined by $K(G, n) :\equiv \|\Sigma^{n-1} K(G, 1)\|_n$ for $n \geq 2$.

The cohomology theory generated by the Eilenberg-MacLane spectrum $K(G, -)$ is an ordinary theory satisfying

$$H^n(S^0) \simeq \begin{cases} G, & \text{if } n = 0 \\ 1, & \text{otherwise} \end{cases}$$

Beyond the Eilenberg-MacLane spectra, we would like to construct for any family of abelian groups $G : \mathbb{Z} \to \mathsf{Ab}$ a spectrum E such that $\pi_n(E) = G_n$. Naïvely, we might think to set

$$E_n :\equiv \prod_{k:\mathbb{N}} K(G_{k-n}, k)$$

Indeed, function extensionality guarantees that $\Omega E_{n+1} \simeq E_n$, so we have a spectrum. However, we are unable to compute its homotopy groups in general: $\pi_n(E)$ is the 0-truncation of a dependent

product over $\mathbb{N}$, which we cannot simplify further without a choice axiom (as discussed in §3.2.3). We hope to sidestep the issue by using a different definition.

For any family of types $D : \mathbb{N} \to \mathcal{U}$ and maps $d : \prod_{n:\mathbb{N}} D_n \to D_{n+1}$, we define the *sequential colimit* $\operatorname{colim}_{n:\mathbb{N}} D_n$ as an inductive type generated by

- for $n : \mathbb{N}$ and $x : D_n$, a point $\mathsf{ncin}_n x : \operatorname{colim}_{n:\mathbb{N}} D_n$,
- for $n : \mathbb{N}$ and $x : D_n$, a path $\mathsf{ncglue}_n x : \mathsf{ncin}_n x = \mathsf{ncin}_{n+1}(d_n x)$.

We now instead define

$$E_n :\equiv \operatorname*{colim}_{m:\mathbb{N}} \prod_{k \le m} K(G_{-n+k}, k)$$

where the maps of the colimit are the obvious inclusions. We can think of this as the "subtype" of our earlier definition consisting of tuples with finitely many "non-base" entries. To see that this construction has the desired homotopy groups, consider the following:

Lemma 3.4. For any $A : \mathbb{N} \to \mathcal{U}$, we have $\operatorname{colim}_{n:\mathbb{N}} \prod_{k \le n} A_k \simeq A_0 \times \operatorname{colim}_{n:\mathbb{N}} \prod_{k \le n} A_{k+1}$.

Proof. Straightforward equivalence proof. □

Lemma 3.5. We say that a type A is m-connected if $\|A\|_m$ is contractible. Let $D : \mathbb{N} \to \mathcal{U}$, $d : \prod_{n:\mathbb{N}} D_n \to D_{n+1}$, and $m : \mathbb{N}_{-2}$ be given. If D_n is m-connected for all $n : \mathbb{N}$, then $\operatorname{colim}_{n:\mathbb{N}} D_n$ is m-connected.

Proof. Let $x_n : \|D_n\|_m$ and $f_n : \prod_{v:\|D_n\|_m} x_n = y$ witness the connectedness of the individual D_n. We have an element $u : \|\operatorname{colim}_{n:\mathbb{N}} D_n\|_m$ defined by applying ncin_0 to x_0 under the the truncation. We must show for any $v : \|\operatorname{colim}_{n:\mathbb{N}} D_n\|_m$ we have $u = v$. Since our goal is an equality in an m-type, it is itself an m-type, and we may therefore assume $x_0 \equiv |x'_0|_m$ and $v \equiv |v'|_m$ (in which case $u \equiv |\mathsf{ncin}_0\, x'_0|_m$).

We go by induction on $v' : \operatorname{colim}_{n:\mathbb{N}} D_n$, with the goal $|\mathsf{ncin}_0\, x'_0|_m = |v'|_m$. First, given $n : \mathbb{N}$ and $z : D_n$, we must show $|\mathsf{ncin}_0\, x'_0|_m = |\mathsf{ncin}_n\, z|_m$. We write $d_0^n : D_0 \to D_n$ for $d_n \circ \cdots \circ d_0$ and $\mathsf{ncglue}_0^n : \mathsf{ncin}_0 \sim \mathsf{ncin}_n \circ d_0^n$ for the n-fold concatenation of the path constructors. We take

$$\mathsf{ap}_{|-|_m}(\mathsf{ncglue}_0^n x'_0) \cdot \mathsf{ap}_{\mathsf{trunc\text{-}map}_m(\mathsf{ncin}_n)}(f_n |d_0^n x'_0|_m{}^{-1} \cdot f_n |z|_m)$$

as our proof. Now, for $n : \mathbb{N}$ and $z : Dn$, we must construct per Lemma 2.4 a square

$$\begin{array}{ccc}
|\mathsf{ncin}_0\, x'_0|_m & \xrightarrow{\mathsf{ap}_{|-|_m}(\mathsf{ncglue}_0^n x'_0)\cdot \mathsf{ap}_{\mathsf{trunc\text{-}map}_m(\mathsf{ncin}_n)}(f_n |d_0^n x'_0|_m{}^{-1}\cdot f_n |z|_m)} & |\mathsf{ncin}_n\, z|_m \\
\Big\downarrow{\scriptstyle \mathsf{refl}} & & \Big\downarrow{\scriptstyle \mathsf{ap}_{|-|_m}(\mathsf{ncglue}_n z)} \\
|\mathsf{ncin}_0\, x'_0|_m & \xrightarrow{\mathsf{ap}_{|-|_m}(\mathsf{ncglue}_0^{n+1} x'_0)\cdot \mathsf{ap}_{\mathsf{trunc\text{-}map}_m(\mathsf{ncin}_{n+1})}(f_{n+1} |d_0^{n+1} x'_0|_m{}^{-1}\cdot f_{n+1} |z|_m)} & |\mathsf{ncin}_{n+1}(d_{n+1} z)|_m
\end{array}$$

(In the interest of saving space, we have flipped the axes from our usual layout.) Observe that $\mathsf{ap}_{|-|_m}(\mathsf{ncglue}_0^{n+1} x_0') = \mathsf{ap}_{|-|_m}(\mathsf{ncglue}_0^n x_0') \cdot \mathsf{ap}_{|-|_m}(\mathsf{ncglue}_n(d_0^n x_0'))$, so we may $\mathsf{ap}_{|-|_m}(\mathsf{ncglue}_0^n x_0')$ from the left of the top and bottom faces. If we additionally shift $\mathsf{ap}_{|-|_m}(\mathsf{ncglue}_n(d_0^n x_0'))$ from the bottom to the left face, we are left to prove

$$\begin{array}{ccc} |\mathsf{ncin}_0\, x_0'|_m & \xrightarrow{\mathsf{ap}_{\mathsf{trunc\text{-}map}_m(\mathsf{ncin}_n)}(f_n|d_0^n x_0'|_m{}^{-1} \cdot f_n|z|_m)} & |\mathsf{ncin}_n\, z|_m \\ \Big\downarrow{\scriptstyle \mathsf{ap}_{|-|_m}(\mathsf{ncglue}_n((0_x^0)_u^u))} & & \Big\downarrow{\scriptstyle \mathsf{ap}_{|-|_m}(\mathsf{ncglue}_n z)} \\ |\mathsf{ncin}_0\, x_0'|_m & \xrightarrow{\mathsf{ap}_{\mathsf{trunc\text{-}map}_m(\mathsf{ncin}_{n+1})}(f_{n+1}|d_0^{n+1} x_0'|_m{}^{-1} \cdot f_{n+1}|z|_m)} & |\mathsf{ncin}_{n+1}(d_{n+1} z)|_m \end{array}$$

Consider the homotopy $h : \prod_{t:|D_n|_m} \mathsf{trunc\text{-}map}_m\ \mathsf{ncin}_n\ t = \mathsf{trunc\text{-}map}_m\ \mathsf{ncin}_{n+1}\ (\mathsf{trunc\text{-}map}_m\ d_n\ t)$ defined by taking $h|w|_m :\equiv \mathsf{ap}_{|-|_m}(\mathsf{ncglue}_n\, w)$. We see that the left and right faces above match $h|d_0^n x_0'|_m$ and $h|z|_m$ respectively, leading us to consider the square $\mathsf{natural\text{-}sq}\ h\ (f_n|d_0^n x_0'|_m{}^{-1} \cdot f_n|z|_m)$. This square almost fulfills our requirements, but its bottom face is

$$\mathsf{ap}_{\mathsf{trunc\text{-}map}_m(\mathsf{ncin}_{n+1}) \circ \mathsf{trunc\text{-}map}_m d_n}((f_n|d_0^n x_0'|_m)^{-1} \cdot f_n|z|_m)$$

However, since $\|D_n\|_m$ is contractible, its path spaces are contractible, and so we have

$$\mathsf{ap}_{\mathsf{trunc\text{-}map}_m d_n}((f_n|d_0^n x_0'|_m)^{-1} \cdot f_n|z|_m) \;=\; (f_{n+1}|d_0^{n+1} x_0'|_m)^{-1} \cdot f_{n+1}|z|_m$$

as needed. □

Theorem 3.6. For any family $G : \mathbb{N} \to \mathsf{Ab}$ and $m : \mathbb{N}$, there is an equivalence $\|\mathrm{colim}_{n:\mathbb{N}} \prod_{k \le n} K(G_k, k)\|_m \simeq \prod_{k \le m} K(G_k, k)$.

Proof. By applying Lemma 3.4 and using the fact that $\|\cdot\|_m$ commutes with $\times$, we have

$$\begin{aligned} \left\|\operatorname*{colim}_{n:\mathbb{N}} \prod_{k \le n} K(G_k, k)\right\|_m &= \left\|\prod_{i \le m} K(G_i, i)\right\|_m \times \left\|\operatorname*{colim}_{n:\mathbb{N}} \prod_{k \le n} K(G_{k+m+1}, k+m+1)\right\|_m \\ &= \prod_{i \le m} K(G_i, i) \times \left\|\operatorname*{colim}_{n:\mathbb{N}} \prod_{k \le n} K(G_{k+m}, k+m)\right\|_m \end{aligned}$$

Since $K(G_{k+m+1}, k+m+1)$ is m-connected for any $k : \mathbb{N}$, we have by the previous lemma that $\mathrm{colim}_{n:\mathbb{N}} \prod_{k \le n} K(G_{k+m+1}, k+m+1)$ is m-connected and therefore that

$$\left\|\operatorname*{colim}_{n:\mathbb{N}} \prod_{k \le n} K(G_{k+m+1}, k+m+1)\right\|_m = 1$$

as needed. □

If we trace the group structure through this argument and use the properties of EM-spaces, we obtain:

Corollary 3.7. For any family $G : \mathbb{N} \to \mathsf{Ab}$ and $m : \mathbb{N}$, $\pi_m(\mathrm{colim}_{n:\mathbb{N}} \prod_{k\leq n} K(G_k, k)) \simeq G_m$.

Thus our construction has the desired homotopy groups. It remains to show that it forms a spectrum, for which we must essentially show that

$$\Omega\left(\operatorname*{colim}_{n:\mathbb{N}} \prod_{k\leq n} K(G_k, k)\right) \simeq \operatorname*{colim}_{n:\mathbb{N}} \prod_{k\leq n} \Omega K(G_k, k)$$

Unfortunately, we have no proof of this fact. We leave as conjecture the following:

Conjecture 3.8. For any family of types $X : \mathbb{N} \to \mathcal{U}_*$, there is an equivalence

$$\Omega\left(\operatorname*{colim}_{m:\mathbb{N}} \prod_{k\leq m} X_k\right) \simeq \operatorname*{colim}_{m:\mathbb{N}} \prod_{k\leq m} \Omega X_k$$

where the maps of the second colimit are those induced by Ω's functorial action $\mathsf{ap}^{\odot}$.

Attempts to formally prove this conjecture, or to prove more generally that colimit commutes with loop space, have suffered from an overabundance of complexity. This is partly due to iterated induction on colimits, which is presumably unavoidable. Also significant, however, is the accumulated cruft of transporting along arithmetic lemmas. Specifically, given a diagram

$$D_0 \xrightarrow{d_0} D_1 \xrightarrow{d_1} D_2 \xrightarrow{d_2} \cdots$$

computing its loop space by the standard encode-decode method (as elaborated in [8, §8.9]) requires computing more generally the path space $\mathsf{ncin}_0 x_0 = c$ for $c : \mathrm{colim}_{n:\mathbb{N}} D_n$, where x_0 is the specified basepoint. For this we define a family $\mathsf{Codes} : \mathrm{colim}_{n:\mathbb{N}} D_n \to \mathcal{U}$ by induction on the colimit, with the aim of proving $(\mathsf{ncin}_0 x_0 = c) \simeq \mathsf{Codes}\ c$. We want to set

$$\mathsf{Codes}\ (\mathsf{ncin}_n x) :\equiv \operatorname*{colim}_{m:\mathbb{N}} (d_0^{n+m} x_0 = d_n^m x)$$

where $d_n^m : D_n \to D_{n+m}$ is defined by induction with $d_n^0 :\equiv \mathsf{id}$ and $d_n^{m+1} :\equiv d_{n+m} \circ d_n^m$. Completing this definition requires showing that $\mathsf{Codes}\ (\mathsf{ncin}_n x) \simeq \mathsf{Codes}\ (\mathsf{ncin}_{n+1}(d_n x))$. Expanding, this is

$$\operatorname*{colim}_{m:\mathbb{N}} (d_0^{n+m} x_0 = d_n^m x) \simeq \operatorname*{colim}_{m:\mathbb{N}} (d_0^{(n+1)+m} x_0 = d_{n+1}^m (d_n x))$$

It is easy to show that for any diagram $A_0 \longrightarrow A_1 \longrightarrow A_2 \longrightarrow \cdots$ there is an equivalence $\mathrm{colim}_{m:\mathbb{N}} A_m \simeq \mathrm{colim}_{m:\mathbb{N}} A_{m+1}$. This leaves demonstrating that

$$\operatorname*{colim}_{m:\mathbb{N}} (d_0^{n+(m+1)} x_0 = d_n^{m+1} x) \simeq \operatorname*{colim}_{m:\mathbb{N}} (d_0^{(n+1)+m} x_0 = d_{n+1}^m (d_n x))$$

Herein lies the source of the arithmetical difficulties. While it is straightforward to give a proof $p : \prod_{n,m:\mathbb{N}} n + (m+1) = (n+m) + 1$ and show that $d_n^{m+1} =_{p(n,m)}^{k.D_k} d_{n+1}^m \circ d_n$, the proof that the

equivalence above holds amounts to several lines of code, and reasoning about these paths in the proof of the equivalence $(\mathsf{ncin}_0 x_0 = c) \simeq \mathsf{Codes}\ c$ quickly becomes involved. A potential solution, which we have yet to explore, would be to work in a system such as Voevodsky's Homotopy Type System [25] which has an internal notion of strict equality. Since the strict equality is substitutional (as is definitional equality) rather than transportational (as is propositional equality), one could prove strict arithmetical identities and apply them without having to reason about that application later on. Ideally, one could prove results in a system augmented with these identities and then expand the arguments into traditional homotopy type theory, but no such algorithm is known at present.

Chapter 4

Results in Eilenberg-Steenrod Cohomology

Having exhibited a class of models, we move on to prove results in the axiomatic framework. In §4.1 and §4.5 we develop two standard cohomological tools, the long exact cofiber sequence and Mayer-Vietoris sequence. The second of these takes advantage of a simplifying lemma developed in §4.3, which we can use to automatically handle three-dimensional coherence proofs in certain cases. Other applications of this lemma as well as a more complex variant are explored in §4.4. Finally, we compute in §4.6 the cohomology of finite products of spheres. The dominating computation is the construction of an equivalence witnessing associativity of the join (defined in §1.3). This is the first and only proof involving three-dimensional paths we give in which §4.3 does not apply, and serves as a somewhat disheartening demonstration of the state of cubical reasoning in vanilla homotopy type theory.

Before beginning the chapter proper, we observe a handy lemma:

Lemma 4.1 (Basepoint Independence)**.** Let $n : \mathbb{Z}$ be given. For any $A : \mathcal{U}$ and basepoints $a_1, a_2 : A$, we have $H^n(A, a_1) = H^n(A, a_2)$. In addition, for any $X, Y : \mathcal{U}_*$, $f : X \to Y$, and paths $p, q : fx_0 = y_0$, we have $(f, p)^* = (f, q)^*$.

Proof. For the first claim, we observe that by the Suspension Axiom we have

$$H^n(A, a_1) = H^n(\Sigma(A, a_1)) \equiv H^n(\Sigma A, \mathsf{north}) \equiv H^n(\Sigma(A, a_2)) = H^n(A, a_2)$$

For the second, we use naturality of the same. We have

$$\begin{aligned}(f, p)^* &= \mathsf{susp\text{-}iso}^n_X \circ (\Sigma(f, p))^* \circ (\mathsf{susp\text{-}iso}^n_Y)^{-1} \\ &\equiv \mathsf{susp\text{-}iso}^n_X \circ (\Sigma f, \mathsf{refl})^* \circ (\mathsf{susp\text{-}iso}^n_Y)^{-1} \\ &\equiv \mathsf{susp\text{-}iso}^n_X \circ \Sigma(f, q)^* \circ (\mathsf{susp\text{-}iso}^n_Y)^{-1} \\ &= (f, q)^*\end{aligned}$$

□

In light of this, we will henceforth omit basepoint path information when considering the action of the cohomology functor on maps.

4.1 Extending the Exactness Axiom

The Exactness Axiom tells us that each function $(f, p) : X \cdot\to Y$ gives rise to an exact sequence $H^n(\mathsf{Cof}(f,p)) \xrightarrow{\mathsf{cfcod}^*} H^n(Y) \xrightarrow{f^*} H^n(X)$. In fact, we have a long exact sequence

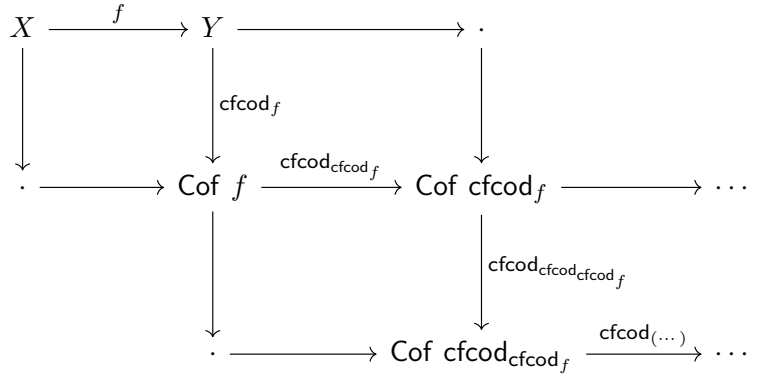
$$\cdots \xrightarrow{f^*} H^{n-1}(X) \xrightarrow{\partial} H^n(\mathsf{Cof}\ f) \xrightarrow{\mathsf{cfcod}^*} H^n(Y) \xrightarrow{f^*} H^n(X) \xrightarrow{\partial} \cdots$$

which extends infinitely in both directions. The homomorphisms ∂ are constructed using a map $\mathsf{ext\text{-}glue}$ which we define for any span $U \xleftarrow{g} \cdot \xrightarrow{h} V$ as

$$\begin{aligned}
&\mathsf{ext\text{-}glue} : U \sqcup^W V \to \Sigma W \\
&\mathsf{ext\text{-}glue}\ (\mathsf{left}\ u) :\equiv \mathsf{north} \\
&\mathsf{ext\text{-}glue}\ (\mathsf{right}\ v) :\equiv \mathsf{south} \\
&\mathsf{ap}_{\mathsf{ext\text{-}glue}}(\mathsf{glue}\ w) := \mathsf{merid}\ w
\end{aligned}$$

This map definitionally preserves basepoint. Here, we use $\mathsf{ext\text{-}glue} : \mathsf{Cof}\ f \to \Sigma X$ to define $\partial :\equiv \mathsf{susp\text{-}iso}^n_X \circ \mathsf{ext\text{-}glue}^*$.

We can obtain the long exact cofiber sequence by iterating the cofiber type construction. Starting with the function $(f, p) : X \cdot\to Y$, we can construct first the cofiber type $\mathsf{Cof}\ f$, then the type $\mathsf{Cof}\ \mathsf{cfcod}_f$, and so on. We thus assemble a diagram of pushouts:



Let us write $\mathsf{Cof}^n f$ for the nth iterated cofiber type and cfcod^n_f for the accompanying map $\mathsf{Cof}^{n-1} f \to \mathsf{Cof}^n f$. Courtesy of the two pushouts lemma, we observe that $\mathsf{Cof}^2 f$ is the pushout of the span $\cdot \leftarrow X \to \cdot$; in other words, it is the suspension of X. Similarly, we will have $\mathsf{Cof}^3 f \simeq \Sigma Y$, and $\mathsf{Cof}^4 f \simeq \Sigma(\mathsf{Cof}\ f)$. To show this in detail, we first prove the following:

Lemma 4.2 (Symmetry of Pushouts)**.** Let a span $X \xleftarrow{(f,p)} Z \xrightarrow{(g,q)} Y$ be given. There is a equivalence $\mathsf{flip} : X \sqcup^Z Y \simeq Y \sqcup^Z X$ which preserves basepoint (we write $\mathsf{flip}^\odot$ for the pointed equivalence). Moreover, we have $\mathsf{left} =^{W.X\to W}_{\mathsf{flip}^\odot} \mathsf{right}$ and $\mathsf{right} =^{W.Y\to W}_{\mathsf{flip}^\odot} \mathsf{left}$.

Proof. One can easily confirm that the map $\mathsf{flip} : X \sqcup^Z Y \to Y \sqcup^Z X$ sending $\mathsf{left}\ x$ to $\mathsf{right}\ x$, $\mathsf{right}\ y$ to $\mathsf{left}\ y$, and $\mathsf{glue}\ z$ to $(\mathsf{glue}\ z)^{-1}$ is an equivalence. The path

$$\mathsf{ap}_{\mathsf{right}}(p^{-1}) \cdot \mathsf{glue}\ {z_0}^{-1} \cdot \mathsf{ap}_{\mathsf{left}}(q)\ :\ \mathsf{right}\ x_0 = \mathsf{left}\ y_0$$

proves that flip preserves basepoint. The action of the equivalence on left and right follows straightforwardly from Assertion 1.8. □

Theorem 4.3 (Twice-Iterated Cofiber Type)**.** Let $X, Y : \mathcal{U}_*$ and $(f, p) : X \cdot\!\to Y$ be given. There is an equivalence $\mathsf{cof^2\text{-}equiv} : \mathsf{Cof}^2 f \simeq \Sigma X$ which preserves basepoint (with $\mathsf{cof^2\text{-}equiv}^\odot$ denoting the pointed equivalence). Additionally, we have paths (1) $\mathsf{cfcod}^2_f =^{V.\mathsf{Cof}(f,p)\to V}_{\mathsf{cof^2\text{-}equiv}^\odot} \mathsf{ext\text{-}glue}$ and (2) $\mathsf{ext\text{-}glue} =^{U.U\to\Sigma Y}_{\mathsf{cof^2\text{-}equiv}^\odot} \mathsf{flip} \circ \Sigma f$.

Proof.

Defining the Equivalence

For the equivalence maps we set

$$\begin{aligned}
&\mathsf{into} : \mathsf{Cof}^2 f \to \Sigma X && \mathsf{out} : \Sigma X \to \mathsf{Cof}^2 f\\
&\mathsf{into}\ \mathsf{cfbase} :\equiv \mathsf{south} && \mathsf{out}\ \mathsf{north} :\equiv \mathsf{cfcod}\ \mathsf{cfbase}\\
&\mathsf{into}\ (\mathsf{cfcod}\ c) :\equiv \mathsf{ext\text{-}glue}\ c && \mathsf{out}\ \mathsf{south} :\equiv \mathsf{cfbase}\\
&\mathsf{ap}_{\mathsf{into}}(\mathsf{cfglue}\ y) := \mathsf{refl} && \mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ x) := \mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x) \cdot (\mathsf{cfglue}\ (fx))^{-1}
\end{aligned}$$

We take $(\mathsf{merid}\ x_0)^{-1}$ as the proof that into preserves basepoint, and write $\mathsf{into}^\odot$ for the pointed function. Let us first show that for every $\sigma : \Sigma X$ we have $\mathsf{into}\ (\mathsf{out}\ \sigma) = \sigma$. In the north case, we have $\mathsf{refl} : \mathsf{into}\ (\mathsf{out}\ \mathsf{north}) = \mathsf{north}$, and in the south case $\mathsf{refl} : \mathsf{into}\ (\mathsf{out}\ \mathsf{south}) = \mathsf{south}$. For the merid case, we must show for all $x : X$ that $\mathsf{refl} =^{\sigma.\mathsf{into}(\mathsf{out}\sigma)=\sigma}_{\mathsf{merid}\ x} \mathsf{refl}$. By Lemma 2.4, it suffices to give a square

$$\begin{array}{ccc}
\mathsf{north} & \xrightarrow{\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}}(\mathsf{merid}\ x)} & \mathsf{south}\\
\Big\downarrow{\scriptstyle \mathsf{refl}} & & \Big\downarrow{\scriptstyle \mathsf{refl}}\\
\mathsf{north} & \xrightarrow{\mathsf{merid}\ x} & \mathsf{south}
\end{array}$$

We have

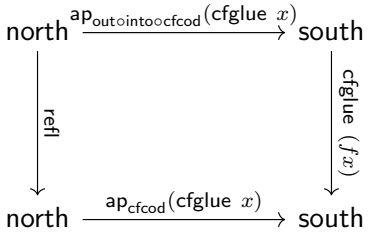
$$\begin{aligned}
\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}}(\mathsf{merid}\ x) &= \mathsf{ap}_{\mathsf{into}}(\mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x) \cdot (\mathsf{cfglue}\ f(x))^{-1})\\
&= \mathsf{ap}_{\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{cfglue}\ x) \cdot (\mathsf{ap}_{\mathsf{into}}(\mathsf{cfglue}\ f(x)))^{-1}\\
&= \mathsf{ap}_{\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{cfglue}\ x)\\
&= \mathsf{ap}_{\mathsf{ext\text{-}glue}}(\mathsf{cfglue}\ x)\\
&= \mathsf{merid}\ x
\end{aligned}$$

and so the square commutes.

Now we show that for $\kappa : \mathsf{Cof}^2 f$ we have $\mathsf{out}\ (\mathsf{into}\ \kappa) = \kappa$. For the cfbase case, we have $\mathsf{refl} : \mathsf{out}\ (\mathsf{into}\ \mathsf{cfbase}) = \mathsf{cfbase}$. For the cfcod case, we define $\mathsf{out\text{-}into\text{-}cod} : \prod_{c:\mathsf{Cof}\, f} : \mathsf{out}\ (\mathsf{into}\ (\mathsf{cfcod}\ c)) = \mathsf{cfcod}\ c$ by induction on the cofiber type. For the point cases, we set

$$\begin{aligned}\mathsf{out\text{-}into\text{-}cod}\ \mathsf{cfbase} &:\equiv \mathsf{refl}\\ \mathsf{out\text{-}into\text{-}cod}\ (\mathsf{cfcod}\ y) &:\equiv \mathsf{cfglue}\ y\end{aligned}$$

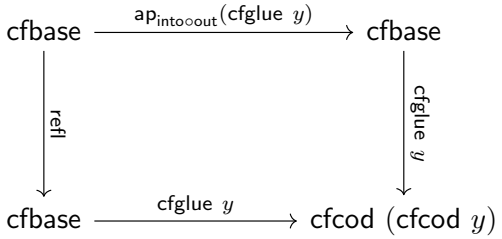
For the cfglue case we need for $x : X$ a path $\mathsf{refl} =^{c.\mathsf{out}\ (\mathsf{into}\ (\mathsf{cfcod}\ c))=\mathsf{cfcod}\ c}_{\mathsf{cfglue}\ x} \mathsf{cfglue}\ (fx)$; by Lemma 2.4 it suffices to give a square



We compute

$$\begin{aligned}\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{cfglue}\ x) &= \mathsf{ap}_{\mathsf{out}\circ\mathsf{ext\text{-}glue}}(\mathsf{cfglue}\ x)\\ &= \mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ x)\\ &= \mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x) \cdot (\mathsf{cfglue}\ (fx))^{-1}\end{aligned}$$

and see that the square commutes. This completes the definition of $\mathsf{out\text{-}into\text{-}cod}$. To finish we must prove the cfglue case of the outer induction, which requires we construct for every $y : Y$ a path $\mathsf{refl} =^{\kappa.\mathsf{into}\ (\mathsf{out}\ \kappa)=\kappa}_{\mathsf{cfglue}\ y} \mathsf{out\text{-}into\text{-}cod}\ (\mathsf{cfcod}\ y)$. By Lemma 2.4, and since $\mathsf{out\text{-}into\text{-}cod}\ (\mathsf{cfcod}\ y) \equiv \mathsf{cfglue}\ y$, it suffices to give a square



We have $\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}}(\mathsf{cfglue}\ y) = \mathsf{ap}_{\mathsf{out}}(\mathsf{refl}) \equiv \mathsf{refl}$, so the square commutes.

Constructing the Paths (1) and (2)

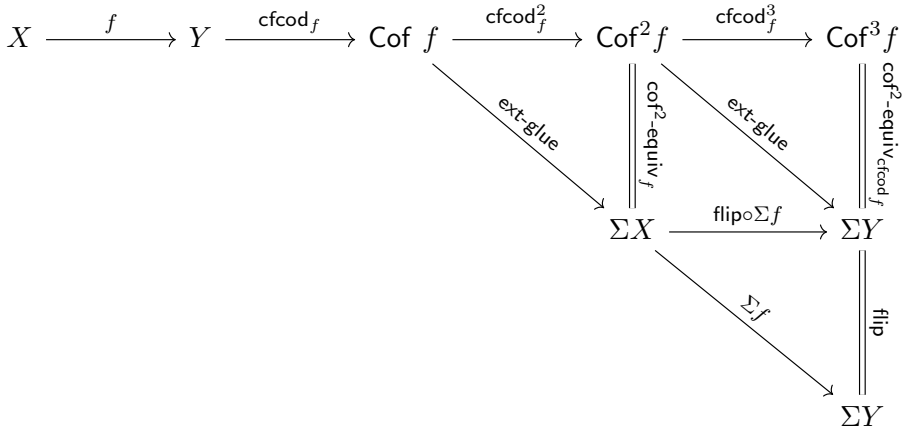
By Assertion 1.8, we have $\mathsf{cfcod}^2_f =^{V.\mathsf{Cof}(f,p)\to V}_{\mathsf{cof}^2\text{-}\mathsf{equiv}^{\odot}} \mathsf{into} \circ \mathsf{cfcod}_f$, and $\mathsf{into}\circ\mathsf{cfcod} \equiv \mathsf{ext\text{-}glue}$ by definition, so we have (1). For (2), Assertion 1.9 gives $\mathsf{flip} \circ \Sigma f \circ \mathsf{into} =^{U.U\to\Sigma Y}_{\mathsf{cof}^2\text{-}\mathsf{equiv}^{\odot}} \mathsf{flip} \circ \Sigma f$, so with function

extensionality it suffices to show $\prod_{\kappa:\mathsf{Cof}^2(f,p)} \mathsf{ext\text{-}glue}\ \kappa = \mathsf{flip}\ (\Sigma f\ (\mathsf{into}\ \kappa))$. This is a straightforward proof by induction, which we leave to the reader. □

Now we can establish the existence of the long exact sequence. We have the following equalities at the type $\sum_{(U,V):\mathcal{U}_*\times\mathcal{U}_*}(\mathsf{Cof}\ f \to U) \times (U \to V)$:

$$\begin{aligned}
(\mathsf{Cof}^2 f, \mathsf{Cof}^3 f, \mathsf{cfcod}^2_f, \mathsf{cfcod}^3_f) &= (\mathsf{Cof}^2 f, \Sigma Y, \mathsf{cfcod}^2_f, \mathsf{ext\text{-}glue}) && \text{(by 4.3 with } \mathsf{cfcod}_f\text{)} \\
&= (\Sigma X, \Sigma Y, \mathsf{ext\text{-}glue}, \mathsf{flip} \circ \Sigma f) && \text{(by 4.3 with } f\text{)} \\
&= (\Sigma X, \Sigma Y, \mathsf{ext\text{-}glue}, \Sigma f) && \text{(by 4.2 with } \Sigma Y\text{)}
\end{aligned}$$

Diagrammatically, this amounts to the following:



Using this equality, we can transport the proof that

$$H^n(\mathsf{Cof}^3 f) \xrightarrow{\mathsf{cfcod}^{3\,*}_f} H^n(\mathsf{Cof}^2 f) \xrightarrow{\mathsf{cfcod}^{2\,*}_f} H^n(\mathsf{Cof}\ f) \xrightarrow{\mathsf{cfcod}_f{}^*} H^n(Y) \xrightarrow{f^*} H^n(X)$$

is exact, given by the Exactness Axiom, to a proof that

$$H^n(\Sigma Y) \xrightarrow{\Sigma f^*} H^n(\Sigma X) \xrightarrow{\mathsf{ext\text{-}glue}^*} H^n(\mathsf{Cof}(f,p)) \xrightarrow{\mathsf{cfcod} f^*} H^n(Y) \xrightarrow{f^*} H^n(X)$$

is exact. Finally, applying the Suspension Axiom gives the desired result.

4.2 Cohomology of a Binary Wedge

In classical cohomology, we have that $H^n(X \vee Y) \simeq H^n(X) \times H^n(Y)$ by the Additivity Axiom. Although we do not assume general Additivity, we can prove the finite case using only the Exactness Axiom. This proof is adapted from [6, p.32-33]. We first have an algebraic lemma:

Lemma 4.4. Let groups G, H_1, H_2 be given. Given maps

$$H_1 \underset{j_1}{\overset{i_1}{\rightleftarrows}} G \underset{j_2}{\overset{i_2}{\leftrightarrows}} H_2$$

such that $j_1 \circ i_1 \sim \mathsf{id}_{H_1}$, $j_2 \circ i_2 \sim \mathsf{id}_{H_2}$, and the sequences $H_1 \overset{i_1}{\to} G \overset{j_2}{\to} H_2$ and $H_2 \overset{i_2}{\to} G \overset{j_1}{\to} H_1$ are exact, then $\langle j_1, j_2 \rangle : G \to H_1 \times H_2$ is an isomorphism. Moreover, i_k corresponds to the injection $H_k \to H_1 \times H_2$ and j_k to the projection $H_1 \times H_2 \to H_k$.

Proof. Let $\varphi = \langle j_1, j_2 \rangle$. By Lemma 1.17, we can show φ is an isomorphism by showing it is injective and surjective.

To show φ is injective we must that for any $g : G$, if $\varphi(g) = e$, then $g = e$. Let $g : G$ be given such that $\varphi(g) = e$, that is, such that $j_1 g = e$ and $j_2 g = e$. Since $H_1 \overset{i_1}{\to} G \overset{j_2}{\to} H_2$ is exact, we have an element of $\| \sum_{h_1 : H_1} i_1 h_1 = g \|_{-1}$. As our goal, $g = e$, is a (-1)-type, we can extract h_1 and the proof $i_1 h_1 = g$. Then $g = i_1 h_1 = i_1 (j_1 i_1 h_1) = i_1 j_1 g = i_1 e = e$.

To show it is surjective, it suffices to give for $h_1 : H_1$ and $h_2 : H_2$ some $g : G$ such that $j_1 g = h_1$ and $j_2 g = h_2$. Take $g = i_1 h_1 \cdot i_2 h_2$. Then $j_1 g = j_1 (i_1 h_1 \cdot i_2 h_2) = j_1 i_1 h_1 \cdot j_1 i_2 h_2 = h_1 \cdot e = h_1$. Likewise $j_2 g = h_2$.

We have $(i_1 \circ \varphi)(h_1) = \langle i_1 j_1 h_1, i_1 j_2 h_1 \rangle = \langle h_1, e \rangle$ and analogously $(i_2 \circ \varphi)(h_2) = \langle h_2, e \rangle$; thus i_1 and i_2 correspond to the injections over φ. We have $\pi_1 \circ \varphi = j_1$ and $\pi_2 \circ \varphi = j_2$ by definition, so j_1 and j_2 correspond to the projections. □

We combine this with a topological lemma:

Lemma 4.5. Let $X, Y : \mathcal{U}_*$. Then where winl denotes the map $X \to X \vee Y$, we have a pointed equivalence $(e, p) : \mathsf{Cof}(\mathsf{winl}) \simeq Y$. Moreover, if we define

$$\begin{aligned}
&\mathsf{projl} : X \vee Y \to X \\
&\mathsf{projl}\ (\mathsf{winl}\ x) :\equiv x \\
&\mathsf{projl}\ (\mathsf{winr}\ y) :\equiv x_0 \\
&\mathsf{ap}_{\mathsf{projl}}\mathsf{wglue} := \mathsf{refl}
\end{aligned}$$

we have $\mathsf{cfcod}_{\mathsf{winl}} =^{U.X \vee Y \to U}_{\mathsf{ua}^{\odot}(e,p)} \mathsf{projl}$. The corresponding result holds for winr and the analogously-defined projr.

Proof. Define $f : \mathsf{Cof}(\mathsf{winl}) \to Y$ by

$$\begin{aligned}
&f\ \mathsf{cfbase} :\equiv y_0 \\
&f\ (\mathsf{cfcod}\ w) :\equiv \mathsf{projr}\ w \\
&\mathsf{ap}_f(\mathsf{cfglue}\ x) := \mathsf{refl}
\end{aligned}$$

and $g : Y \to \mathsf{Cof}(\mathsf{winl})$ by $g(y) = \mathsf{cfcod}\ (\mathsf{winr}\ y)$. We have $f \circ g \equiv \mathsf{id}_Y$ definitionally. For the other inverse, we go by induction on the cofiber space. For cfbase, we have $g(f\ \mathsf{cfbase}) :\equiv$

$\mathsf{cfcod}\ (\mathsf{winr}\ y_0)$, so $(\mathsf{cfglue}\ x_0 \cdot \mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue})^{-1}$ is a proof $g(f\ \mathsf{cfbase}) = \mathsf{cfbase}$. For cfcod, we define $q : \prod_{w:X\vee Y} g(f(\mathsf{cfcod}\ w)) = \mathsf{cfcod}\ w$ by induction on the wedge, giving the point cases as follows:

$$q\ (\mathsf{winl}\ x) :\equiv (\mathsf{cfglue}\ x_0 \cdot \mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue})^{-1} \cdot \mathsf{cfglue}\ x$$
$$q\ (\mathsf{winr}\ y) :\equiv \mathsf{refl}$$

For the wglue case we must give a proof $q\ (\mathsf{winl}\ x_0) =^{w.g(f(\mathsf{cfcod}\ w))=\mathsf{cfcod}\ w}_{\mathsf{wglue}} q\ (\mathsf{winr}\ y_0)$; by Lemma 2.4 it suffices to give a square

$$\begin{array}{ccc}
\mathsf{cfbase} & \xrightarrow{\mathsf{ap}_{g\circ\mathsf{projl}}\mathsf{wglue}} & \mathsf{cfcod}\ (\mathsf{winr}\ y) \\
{\scriptstyle (\mathsf{cfglue}\ x_0 \cdot \mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue})^{-1}\cdot\mathsf{cfglue}\ x_0}\downarrow & & \downarrow{\scriptstyle \mathsf{refl}} \\
\mathsf{cfcod}\ (\mathsf{winl}\ x) & \xrightarrow[\mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue}]{} & \mathsf{cfcod}\ (\mathsf{winr}\ y)
\end{array}$$

As $\mathsf{ap}_{g\circ\mathsf{projl}}\mathsf{wglue} = \mathsf{ap}_g(\mathsf{ap}_{\mathsf{projl}}\mathsf{wglue}) = \mathsf{ap}_g\mathsf{refl} \equiv \mathsf{refl}$, we see this square can be satisfied. This completes the cfcod case. For the cfglue case, we must give for $x : X$ a path

$$(\mathsf{cfglue}\ x_0 \cdot \mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue})^{-1} =^{c.g(fc)=c}_{\mathsf{cfglue}\ x} q\ (\mathsf{winl}\ x)$$

Again by Lemma 2.4, it suffices to give a square

$$\begin{array}{ccc}
\mathsf{cfcod}\ (\mathsf{winr}\ y_0) & \xrightarrow{\mathsf{ap}_{g\circ f}(\mathsf{cfglue}\ x)} & \mathsf{cfcod}\ (\mathsf{winr}\ y_0) \\
{\scriptstyle (\mathsf{cfglue}\ x_0 \cdot \mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue})^{-1}}\downarrow & & \downarrow{\scriptstyle (\mathsf{cfglue}\ x_0 \cdot \mathsf{ap}_{\mathsf{cfcod}}\mathsf{wglue})^{-1}\cdot\mathsf{cfglue}\ x} \\
\mathsf{cfbase} & \xrightarrow{\mathsf{cfglue}\ x} & \mathsf{cfcod}(\mathsf{winl}\ x)
\end{array}$$

Since $\mathsf{ap}_{g\circ f}\mathsf{cfglue} = \mathsf{ap}_g\mathsf{refl} \equiv \mathsf{refl}$, this square commutes. □

By combining these two results, we obtain the following theorem:

Theorem 4.6. There is an isomorphism $H^n(X \vee Y) \simeq H^n(X) \times H^n(Y)$. Over this isomorphism, winl^* and winr^* correspond to the two projections and projl^* and projr^* to the two injections.

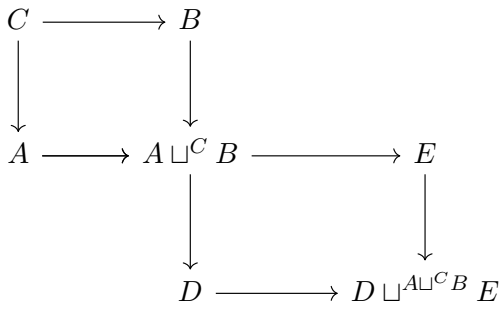
Proof. By applying the Exactness Axiom together with Lemma 4.5 (and using Lemma 4.1 to handle basepoint paths), we have exact sequences

$$H^n(Y) \overset{\mathsf{projr}^*}{\to} H^n(X \vee Y) \overset{\mathsf{winl}^*}{\to} H^n(X)$$
$$H^n(X) \overset{\mathsf{projl}^*}{\to} H^n(X \vee Y) \overset{\mathsf{winr}^*}{\to} H^n(Y)$$

Since $\mathsf{winl}^* \circ \mathsf{projl}^* = (\mathsf{projl} \circ \mathsf{winl})^* = \mathsf{id}^*$ and likewise for winr and projr, we can apply Lemma 4.4 to obtain the desired result. □

4.3 A Coherence Lemma

From this point forward, the results we prove become more complex. Since cohomology deals naturally in suspensions and cofiber types, we are often interested in iterated pushout types, as pictured:



Proving that such a type is equivalent to something else presents a significant challenge, because it requires satisfying three-dimensional coherences: when we eliminate into a path type (as is required for an equivalence proof), the path cases of the outer pushout require squares, and the path cases of the inner pushout require cubes. Luckily, we will rarely need to work with cubes explicitly, because our pushouts will come in certain forms which satisfy the coherence conditions automatically. The only result we prove which cannot be simplified in this way is Theorem 4.21, the associativity of the join, which is far and away the most technically involved result we present.

Most cases will fall under the umbrella of the following lemma:

Theorem 4.7. Let a span $A \overset{f}{\leftarrow} C \overset{g}{\to} B$ and a map $h : A \sqcup^C B \to D$ be given such that f is a section, which is to say there is a map $r : C \to B$ such that $\prod_{c:C} r(fc) = c$. For any $P : \mathsf{Cof}\ h \to \mathcal{U}$, in order to construct a term of type $\prod_{\kappa:\mathsf{Cof}\, h} P(\kappa)$, it suffices to give:

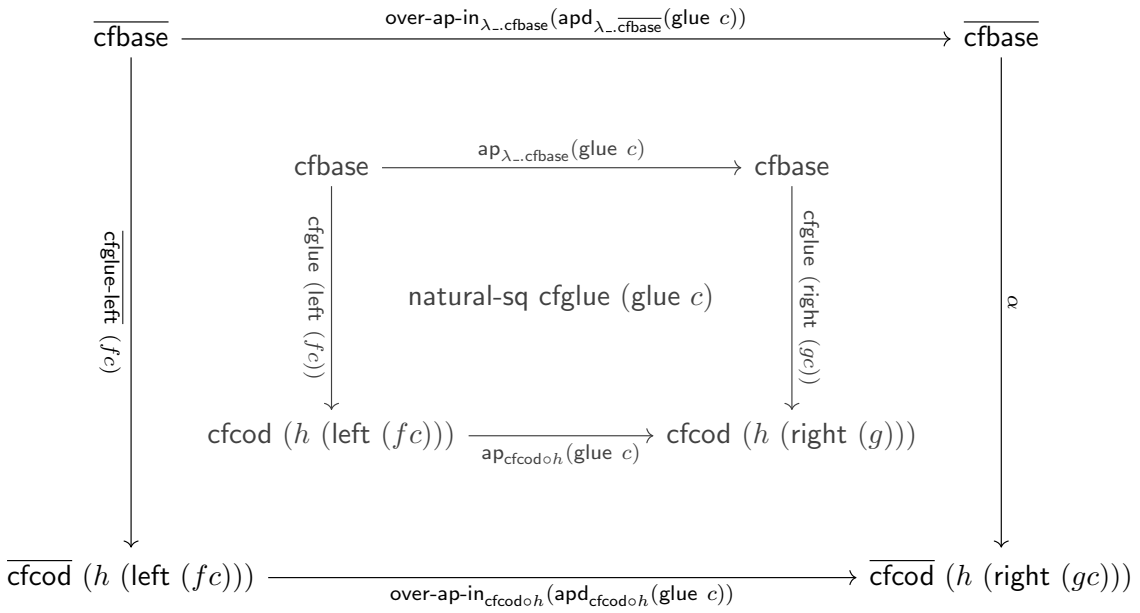
- an element $\overline{\mathsf{cfbase}} : P\ \mathsf{cfbase}$,
- for $d : D$, an element $\overline{\mathsf{cfcod}}\ d : P\ (\mathsf{cfcod}\ d)$,
- for $a : A$, a path $\overline{\mathsf{cfglue\text{-}left}}\ a : \overline{\mathsf{cfbase}} =^{P}_{\mathsf{cfglue}\ (\mathsf{left}\ a)} \overline{\mathsf{cfcod}}\ (h\ (\mathsf{left}\ a))$,
- for $b : B$, a path $\overline{\mathsf{cfglue\text{-}right}}\ b : \overline{\mathsf{cfbase}} =^{P}_{\mathsf{cfglue}\ (\mathsf{right}\ b)} \overline{\mathsf{cfcod}}\ (h\ (\mathsf{right}\ b))$.

No relationship between $\overline{\mathsf{cfglue\text{-}left}}$ and $\overline{\mathsf{cfglue\text{-}right}}$ is required. The $k : \prod_{\kappa:\mathsf{Cof}\ h} P(\kappa)$ we construct in the proof satisfies $k\ \mathsf{cfbase} \equiv \overline{\mathsf{cfbase}}$, $k\ (\mathsf{cfcod}\ d) \equiv \overline{\mathsf{cfcod}}\ d$, and $\mathsf{apd}_k(\mathsf{cfglue}\ (\mathsf{left}\ a)) = \overline{\mathsf{cfglue\text{-}left}}\ a$.

Proof. Let the above be given. We show $\prod_{\kappa:\mathsf{Cof}\ h} P(\kappa)$ by induction on κ. For the cfbase and cfcod cases we take the given $\overline{\mathsf{cfbase}}$ and $\overline{\mathsf{cfcod}}$. For the cfglue case, we must give for $\gamma : A \sqcup^{C} B$ a path $\overline{\mathsf{cfbase}} =^{P}_{\mathsf{cfglue}\ \gamma} \overline{\mathsf{cfcod}}\ (h\gamma)$. We proceed by induction on $\gamma : A \sqcup^{C} B$. For the left case, we take for $a : A$ the given $\overline{\mathsf{cfglue\text{-}left}}\ a$. In the right case, however, we make an adjustment, defining an alternate $\overline{\mathsf{cfglue\text{-}right}}'$. Recall that for the glue case we must give for $c : C$ a term of type

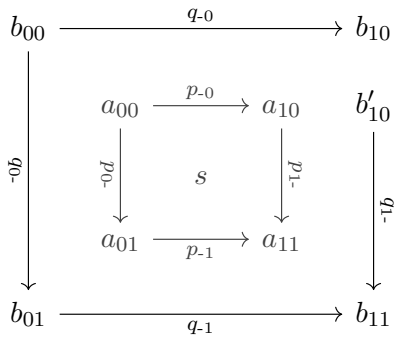
$$\overline{\mathsf{cfglue\text{-}left}}(fc) =^{\gamma.\overline{\mathsf{cfbase}}=^{P}_{\mathsf{cfglue}\ \gamma}\overline{\mathsf{cfcod}}\ (h\gamma)}_{\mathsf{glue}\ c} \overline{\mathsf{cfglue\text{-}right}}'(gc)$$

By Lemma 2.7 it suffices to inhabit the following dependent square over $\mathsf{natural\text{-}sq}\ \mathsf{cfglue}\ (\mathsf{glue}\ c)$, with $\overline{\mathsf{cfglue\text{-}right}}'(gc)$ substituted for α:



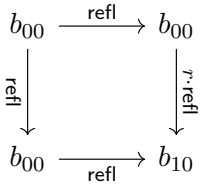
Our plan, drawing on the existence of cube fillers, is to choose for $c : C$ a path $pc : \overline{\mathsf{cfbase}} = \overline{\mathsf{cfbase}}$ so that the above cube is satisfied when we substitute $pc \lhd \overline{\mathsf{cfglue\text{-}right}}(gc)$ for α. If we have such a p, we can complete the theorem by defining $\overline{\mathsf{cfglue\text{-}right}}'(b) :\equiv p(rb) \lhd \overline{\mathsf{cfglue\text{-}right}}\ b$; since $r(gc) = c$, we would have $\overline{\mathsf{cfglue\text{-}right}}'(gc) = pc \lhd \overline{\mathsf{cfglue\text{-}right}}(gc)$ and therefore satisfy the cube. To define such a p, we go by the following lemma:

Lemma 4.8. Let the following collection of paths be given, depicting a partial frame for a dependent square in some fibration P:



Here b_{10} and b'_{10} both lie in the fiber Pa_{10}. Then there exists a propositionally unique path $r : b_{10} = b'_{10}$ such that the completed dependent square, taking $r \lhd q_{1-}$ as the right side, is inhabited.

Proof. By square induction, we may assume $s \equiv \mathsf{srefl}$. In this case, the dependent square reduces to a simple square. We may then also assume the three paths q_{0-}, q_{-0}, and q_{-1} are definitionally refl. In this case, we need a propositionally unique path r such that the square



Clearly setting $r :\equiv \mathsf{refl}$ inhabits the square, and it follows easily from uniqueness of square fillers (Theorem 2.6) that this choice is propositionally unique. □

With this lemma in hand we can, as desired, pick for $c : C$ a path $pc : \overline{\mathsf{cfbase}} = \overline{\mathsf{cfbase}}$ such that the cube is satisfied when we substitute $pc \lhd \overline{\mathsf{cfglue\text{-}right}}\ c$ for α. As described earlier, this is enough to complete the theorem. □

This is certainly not the most general theorem of this sort we can prove, and our choice of statement is somewhat arbitrary – one could, for example, extend from cofiber types to some more general class of pushout. However, the case we have proved is simple and general enough for our purposes, so we leave such generalizations to the reader.

4.4 Sections

Following the investigations of the previous section, we now consider the cohomology of types Y for which we are given a section $f : X \to Y$. Ultimately, our result is the following:

Theorem 4.9. Let a section $f : X \to Y$ be given. Then $H^n(Y) = H^n(X) \times H^n(\mathsf{Cof}\, f)$.

We will prove this theorem in two ways. First, we will give a simple proof using the Exactness Axiom and a bit of homological algebra. Second, we will prove it through Suspension and Additivity, by proving that $\Sigma Y \simeq \Sigma X \vee \Sigma(\mathsf{Cof}\, f)$. The second of these, while more involved, has a more concrete feel: whereas passing to cohomology leaves only the information left after any number of suspensions, we only need to suspend once for something which looks like Theorem 4.9 to appear.

For the first proof, we will need a few simple lemmas.

Lemma 4.10 (Cohomology of Unit)**.** For any $n : \mathbb{Z}$, $H^n(1) = 1$.

Proof. We can easily check that the cofiber type of the identity map $1 \to 1$ is itself 1. Across this equivalence $\mathsf{cfcod} : 1 \to \mathsf{Cof}\ \mathsf{id}$ must of course correspond to id_1. By the Exactness Axiom we have an exact sequence $H^n(1) \xrightarrow{\mathsf{id}^*} H^n(1) \xrightarrow{\mathsf{id}^*} H^n(1)$. Since ${\mathsf{id}_1}^* = \mathsf{id}_{H^n(1)}$, we see that the image of $\mathsf{id}_{H^n(1)}$ is equal to its kernel, which can only be true if $H^n(1) = 1$. □

Lemma 4.11 (Action on Constant Maps)**.** The action ${\mathsf{cst}_{X,Y}}^*$ of a cohomology functor on the constant map $\mathsf{cst}_{X,Y} : X \cdot\!\to Y$ is the constant homomorphism.

Proof. The constant map $\mathsf{cst}_{X,Y}$ factors as the composition $\mathsf{cst}_{1,Y} \circ \mathsf{cst}_{X,1}$. By functoriality, ${\mathsf{cst}_{X,Y}}^* = {\mathsf{cst}_{X,1}}^* \circ {\mathsf{cst}_{1,Y}}^*$. Since $H^n(1) = 1$, ${\mathsf{cst}_{X,Y}}^*$ factors through the trivial group and must be constant. □

We also need a case of the splitting lemma, which is a standard result in homological algebra. The statement and proof is no different from the classical version, so we refer the reader to e.g. [9, p.147].

Assertion 4.12 (Right Splitting Lemma)**.** Let a short exact sequence of abelian groups

$$1 \to G \xrightarrow{\varphi} H \xrightarrow{\psi} K \to 1$$

be given. If ψ has a right inverse χ, then $H \simeq G \times K$. Over this isomorphism, φ corresponds to the natural injection and ψ to the natural projection.

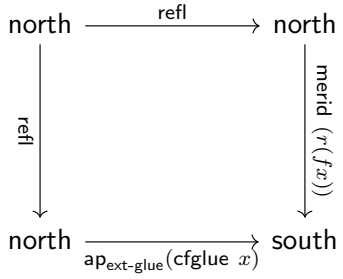
This is enough to prove the following:

Theorem 4.9 (First Proof)**.** Let a section $f : X \cdot\!\to Y$ be given. Then $H^n(Y) \simeq H^n(X) \times H^n(\mathsf{Cof}\, f)$.

Proof. Let $r : Y \to X$ be f's left inverse. Per §4.1, we have an exact sequence

$$H^n(\Sigma X) \xrightarrow{\mathsf{ext\text{-}glue}^*} H^n(\mathsf{Cof}\ f) \xrightarrow{\mathsf{cfcod}^*} H^n(Y) \xrightarrow{f^*} H^n(X)$$

First, we show that $\mathsf{ext\text{-}glue} : \mathsf{Cof}\ f \to \Sigma X$ is constant, which by Lemma 4.11 implies that $\mathsf{ext\text{-}glue}^*$ is constant as well. We define $p : \prod_{\kappa : \mathsf{Cof}\ f} \mathsf{north} = \mathsf{ext\text{-}glue}\ \kappa$ by induction; any basepoint considerations can be handled by Lemma 4.1. For the point cases we set $p\ \mathsf{cfbase} :\equiv \mathsf{refl}$ and $p\ (\mathsf{cfcod}\ y) = \mathsf{merid}\ (ry)$. For the cfglue case we must give for $x : X$ a path $\mathsf{refl} =^{\kappa.\mathsf{north}=\mathsf{ext\text{-}glue}\ \kappa}_{\mathsf{cfglue}\ x} \mathsf{merid}\ (ry)$, equivalent by Lemma 2.4 to giving a square



Since $\mathsf{ap}_{\mathsf{ext\text{-}glue}}(\mathsf{cfglue}\ x) = \mathsf{merid}\ x$ and $r(fx) = x$, this square commutes.

Thus $\mathsf{ext\text{-}glue}^*$ is constant, which by exactness implies cfcod^* is injective. Since f has a left inverse, f^* has a right inverse and is therefore surjective. Hence we have an improved exact sequence:

$$1 \longrightarrow H^n(\mathsf{Cof}\ f) \xrightarrow{\mathsf{cfcod}^*} H^n(Y) \xrightarrow{f^*} H^n(X) \longrightarrow 1$$

and can apply the splitting lemma to conclude that $H^n(Y) \simeq H^n(X) \times H^n(\mathsf{Cof}\ f)$.

□

We proceed to the second proof. The main content is the following, from which we derive our result by the Suspension Axiom and Additivity.

Theorem 4.13. Let a section $f : X \cdot\!\!\to Y$ be given. Then there is a pointed equivalence $\Sigma Y \simeq \Sigma X \vee \Sigma(\mathsf{Cof}\ f)$.

Proof. Let r be f's left inverse. First we define $\mathsf{into} : \Sigma Y \to \Sigma X \vee \Sigma(\mathsf{Cof}\ f)$. Given the proliferation of suspensions, we will leave the reader to infer values on points, and give the merid case:

$$\mathsf{ap}_{\mathsf{into}}(\mathsf{merid}\ y) = (\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (ry)))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y))$$

This map is evidently basepoint-preserving.

For the inverse, we define $\mathsf{out} : \Sigma X \vee \Sigma(\mathsf{Cof}\ f) \to \Sigma Y$, by defining $\mathsf{out\text{-}winl} : \Sigma X \to \Sigma Y$ and $\mathsf{out\text{-}winr} : \Sigma \mathsf{Cof}\ f \to \Sigma Y$ and showing they agree on basepoint. For $\mathsf{out\text{-}winl}$ we set

$$\mathsf{ap}_{\mathsf{out\text{-}winl}}(\mathsf{merid}\ x) = (\mathsf{merid}\ (fx))^{-1} \cdot \mathsf{merid}\ y_0$$

For $\mathsf{out\text{-}winr}$ we define its action on $\mathsf{merid}\ \kappa$ by induction. For the point cases we set

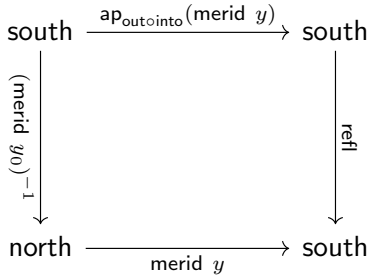
$$\mathsf{ap}_{\mathsf{out\text{-}winr}}(\mathsf{merid}\ \mathsf{cfbase}) = \mathsf{refl}$$
$$\mathsf{ap}_{\mathsf{out\text{-}winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y)) = (\mathsf{merid}\ (f(ry)))^{-1} \cdot \mathsf{merid}\ y$$

Since $r(fx) = x$, we see that $\mathsf{out\text{-}winr}$'s action on $\mathsf{merid}\ \mathsf{cfbase}$ agrees with its action on on $\mathsf{merid}\ (\mathsf{cfcod}\ (fx))$ as needed. Finally, we take refl as the path $\mathsf{out\text{-}winl}\ \mathsf{north} = \mathsf{out\text{-}winr}\ \mathsf{north}$ required for the wglue case. The intuition behind the definition of out as an inverse is that $\mathsf{out\text{-}winr}$ recovers y modulo an adjustment $f(ry)$ necessary to satisfy the cfglue coherence, and $\mathsf{out\text{-}winl}$ serves to correct for that adjustment.

Following this logic we show that $\mathsf{out} \circ \mathsf{into} \sim \mathsf{id}$, by induction on ΣY. For the point cases we give

$$\begin{aligned}
&\mathsf{out\text{-}into\ north} :\equiv (\mathsf{merid}\ y_0)^{-1} \\
&\mathsf{out\text{-}into\ south} :\equiv \mathsf{refl}
\end{aligned}$$

By Lemma 2.4 it suffices for the merid case to give for $y : Y$ a square



Since

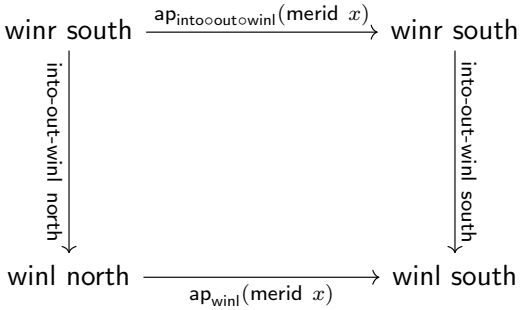
$$\begin{aligned}
\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}}(\mathsf{merid}\ y) &= \mathsf{ap}_{\mathsf{out}}((\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (ry)))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y))) \\
&= \mathsf{ap}_{\mathsf{out}\circ\mathsf{winl}}(\mathsf{merid}\ (ry))^{-1} \cdot \mathsf{ap}_{\mathsf{out}}\mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{out}\circ\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y)) \\
&= (\mathsf{merid}\ (f(ry))^{-1} \cdot \mathsf{merid}\ y_0)^{-1} \cdot \mathsf{refl} \cdot ((\mathsf{merid}\ (f(ry)))^{-1} \cdot \mathsf{merid}\ y) \\
&= (\mathsf{merid}\ y_0)^{-1} \cdot \mathsf{merid}\ y
\end{aligned}$$

the square commutes.

We now show that $\mathsf{into} \circ \mathsf{out} \sim \mathsf{id}$, by induction on $\Sigma X \vee \Sigma(\mathsf{Cof}\ f)$. We define auxiliary functions into-out-winl and into-out-winr. For the first we give as point cases

$$\begin{aligned}
&\mathsf{into\text{-}out\text{-}winl} : \mathsf{into} \circ \mathsf{out} \circ \mathsf{winl} \sim \mathsf{winl} \\
&\mathsf{into\text{-}out\text{-}winl\ north} :\equiv (\mathsf{ap}_{\mathsf{winr}}(\mathsf{merid\ cfbase}))^{-1} \cdot \mathsf{wglue}^{-1} \\
&\mathsf{into\text{-}out\text{-}winl\ south} :\equiv (\mathsf{ap}_{\mathsf{winr}}(\mathsf{merid\ cfbase}))^{-1} \cdot \mathsf{wglue}^{-1} \cdot \mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (ry_0))
\end{aligned}$$

For the merid case we must give for $x : X$ a square



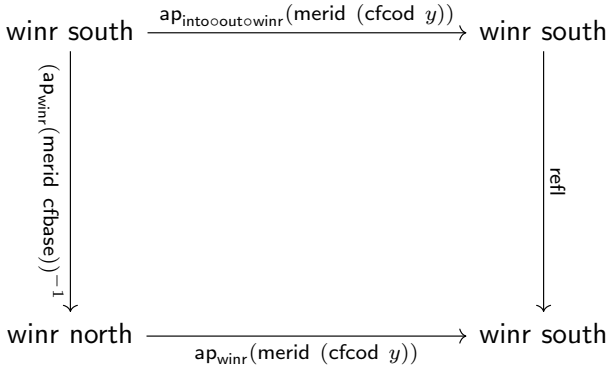
We have

$$\begin{aligned}
\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}\circ\mathsf{winl}}(\mathsf{merid}\ x) &= \mathsf{ap}_{\mathsf{into}}((\mathsf{merid}\ (fx))^{-1} \cdot \mathsf{merid}\ y_0) \\
&= \mathsf{ap}_{\mathsf{into}}(\mathsf{merid}\ (fx))^{-1} \cdot \mathsf{ap}_{\mathsf{into}}(\mathsf{merid}\ y_0) \\
&= (\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (r(fx)))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ (fx))))^{-1} \\
&= (\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (r(fx)))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ (fx))))^{-1} \\
&\quad \cdot ((\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (ry_0)))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y_0)))
\end{aligned}$$

We leave the reader to check that this makes the square commute. We proceed to into-out-winr. For this we take advantage of Theorem 4.7, which allows us to skip the cfglue case. For the point cases, we set

$$\begin{aligned}
&\mathsf{into\text{-}out\text{-}winr} : \mathsf{into} \circ \mathsf{out} \circ \mathsf{winr} \sim \mathsf{winr} \\
&\mathsf{into\text{-}out\text{-}winr}\ \mathsf{north} :\equiv (\mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ \mathsf{cfbase}))^{-1} \\
&\mathsf{into\text{-}out\text{-}winr}\ \mathsf{south} :\equiv \mathsf{refl}
\end{aligned}$$

For the merid ∘ cfcod case, we must give for $y : Y$ a square



We have

$$\begin{aligned}
\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}\circ\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y)) &= \mathsf{ap}_{\mathsf{into}}((\mathsf{merid}\ (f(ry)))^{-1} \cdot \mathsf{merid}\ y) \\
&= \mathsf{ap}_{\mathsf{into}}(\mathsf{merid}\ (f(ry)))^{-1} \cdot \mathsf{ap}_{\mathsf{into}}(\mathsf{merid}\ y) \\
&= (\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (r(f(ry))))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ (f(ry)))))^{-1} \\
&\quad \cdot (\mathsf{ap}_{\mathsf{winl}}(\mathsf{merid}\ (ry)))^{-1} \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ (\mathsf{cfcod}\ y))
\end{aligned}$$

Again, we leave the reader to check this is as needed. We can then give the merid cfbase case by transporting our merid (cfcod (fx_0)) proof along cfglue x_0. □

As an easy corollary, we have

Corollary 4.14. For any $X : \mathcal{U}_*$ and $Y : X \to \mathcal{U}_*$, $\Sigma(\sum_{x:X} Y(x)) \simeq \Sigma X \vee \Sigma(\bigvee_{x:X} Y(x))$, and therefore $H^n(\sum_{x:X} Y(x)) \simeq H^n(X) \times H^n(\bigvee_{x:X} Y(x))$.

In particular, for $X, Y : \mathcal{U}_*$, $H^n(X + Y) \simeq H^n(X) \times H^n(Y) \times H^n(S^0)$.

With a bit more work, we can also show that $\Sigma(X \times Y) \simeq \Sigma X \vee \Sigma Y \vee \Sigma(X \wedge Y)$, where $\wedge$ is the smash product definited in §1.3. It follows that $H^n(X \times Y) \simeq H^n(X) \times H^n(Y) \times H^n(X \wedge Y)$, which we will use in §4.6. We achieve this by applying Theorem 4.9 twice, once to extract X and once to extract Y. We only need the following lemma:

Lemma 4.15. Let $(f, p) : X \cdot\!\!\to Z$ and $(g, q) : Y \cdot\!\!\to Z$ be given. There is a pointed equivalence $\mathsf{Cof}\ (\mathsf{cfcod}_f \circ g) \simeq \mathsf{Cof}\ [f, g]$, where $[f, g] : X \vee Y \to Z$ is defined by

$$\begin{aligned}
&[f, g](\mathsf{winl}\ x) :\equiv fx \\
&[f, g](\mathsf{winr}\ y) :\equiv gy \\
&\mathsf{ap}_{[f,g]}\mathsf{wglue} := p \cdot q^{-1}
\end{aligned}$$

Proof. We first define a map $\mathsf{into} : \mathsf{Cof}\ (\mathsf{cfcod}_f \circ g) \to \mathsf{Cof}\ [f, g]$ by

$$\begin{aligned}
&\mathsf{into}\ \mathsf{cfbase} :\equiv \mathsf{cfbase} \\
&\mathsf{into}\ (\mathsf{cfcod}\ \mathsf{cfbase}) :\equiv \mathsf{cfbase} \\
&\mathsf{into}\ (\mathsf{cfcod}\ (\mathsf{cfcod}\ z)) = \mathsf{cfcod}\ z \\
&\mathsf{ap}_{\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{cfglue}\ x) = \mathsf{cfglue}\ (\mathsf{winl}\ x) \\
&\mathsf{ap}_{\mathsf{into}}(\mathsf{cfglue}\ y) = \mathsf{cfglue}\ (\mathsf{winr}\ y)
\end{aligned}$$

The point and one-dimensional cases for the inverse out are given by

$$\begin{aligned}
&\mathsf{out}\ \mathsf{cfbase} :\equiv \mathsf{cfbase} \\
&\mathsf{out}\ (\mathsf{cfcod}\ z) :\equiv \mathsf{cfcod}\ (\mathsf{cfcod}\ z) \\
&\mathsf{ap}_{\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winl}\ x)) := \mathsf{cfglue}\ y_0 \cdot (\mathsf{ap}_{\mathsf{cfcod}\circ\mathsf{cfcod}}(p \cdot q^{-1}))^{-1} \cdot (\mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x_0))^{-1} \cdot \mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x) \\
&\mathsf{ap}_{\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winr}\ y)) := \mathsf{cfglue}\ y
\end{aligned}$$

For the wglue case, it suffices by Lemma 2.4 to show

$$\begin{array}{ccc}
\mathsf{cfbase} & \xrightarrow{\quad\mathsf{refl}\quad} & \mathsf{cfbase} \\
{\scriptstyle \mathsf{ap}_{\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winl}\ x_0))}\downarrow & & \downarrow{\scriptstyle \mathsf{ap}_{\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winr}\ y_0))} \\
\mathsf{cfcod}\ (\mathsf{cfcod}\ (\mathsf{winl}\ x_0)) & \xrightarrow[\mathsf{ap}_{\mathsf{cfcod}\circ\mathsf{cfcod}\circ[f,g]}\mathsf{wglue}]{} & \mathsf{cfcod}\ (\mathsf{cfcod}\ (\mathsf{winr}\ y_0))
\end{array}$$

This follows by inspection once we observe that $\mathsf{ap}_{\mathsf{cfcod}\circ\mathsf{cfcod}\circ[f,g]}\mathsf{wglue} = \mathsf{ap}_{\mathsf{cfcod}\circ\mathsf{cfcod}}(p \cdot q^{-1})$.

We show that $\mathsf{out} \circ \mathsf{into} \sim \mathsf{id}$ by induction on $\mathsf{Cof}(\mathsf{cfcod}_f \circ g)$. First we define $\mathsf{out\text{-}into\text{-}cod} : \mathsf{out} \circ \mathsf{into} \circ \mathsf{cfcod} \sim \mathsf{cfcod}$ by induction on $\mathsf{Cof}\ f$, setting for point cases

$$\mathsf{out\text{-}into\text{-}cod}\ \mathsf{cfbase} :\equiv \mathsf{cfglue}\ y_0 \cdot (\mathsf{ap}_{\mathsf{cfcod}\circ\mathsf{cfcod}}(p \cdot q^{-1}))^{-1} \cdot (\mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x_0))^{-1}$$
$$\mathsf{out\text{-}into\text{-}cod}\ (\mathsf{cfcod}\ z) :\equiv \mathsf{refl}$$

For the cfglue case, we must give for $x : X$ a square

$$\begin{array}{ccc}
\mathsf{cfbase} & \xrightarrow{\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{cfglue}\ x)} & \mathsf{cfcod}\ (\mathsf{cfcod}\ (fx)) \\
{\scriptstyle \mathsf{out\text{-}into\text{-}cod}\ \mathsf{cfbase}}\downarrow & & \downarrow{\scriptstyle \mathsf{out\text{-}into\text{-}cod}\ (\mathsf{cfcod}\ z)} \\
\mathsf{cfcod}\ \mathsf{cfbase} & \xrightarrow[\mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x)]{} & \mathsf{cfcod}\ (\mathsf{cfcod}\ (fx))
\end{array}$$

Given that

$$\begin{aligned}
\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{cfglue}\ x) &= \mathsf{ap}_{\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winl}\ x)) \\
&= \mathsf{cfglue}\ y_0 \cdot (\mathsf{ap}_{\mathsf{cfcod}\circ\mathsf{cfcod}}(p \cdot q^{-1}))^{-1} \cdot (\mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x_0))^{-1} \cdot \mathsf{ap}_{\mathsf{cfcod}}(\mathsf{cfglue}\ x)
\end{aligned}$$

we see the square commutes.

To show $\mathsf{into} \circ \mathsf{out} \sim \mathsf{id}$, we go by induction on $\mathsf{Cof}\ [f, g]$. First, observe that $[f, g] : X \vee Y \to Z$ is a map out of a pushout, and the left map of that pushout has a left inverse. We may therefore apply Theorem 4.7 and skip the highest coherence. The point cases are simple:

$$\mathsf{into\text{-}out} : \mathsf{into} \circ \mathsf{out} \sim \mathsf{id}$$
$$\mathsf{into\text{-}out}\ \mathsf{cfbase} :\equiv \mathsf{refl}$$
$$\mathsf{into\text{-}out}\ (\mathsf{cfcod}\ z) :\equiv \mathsf{refl}$$

Within the cfglue case, we must show the winl and winr cases, that is, inhabit two squares. Per the above definition, each of these squares has refl on both vertical sides, so in truth we simply need to give paths $\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winl}\ x)) = \mathsf{cfglue}\ (\mathsf{winl}\ x)$ and $\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}}(\mathsf{cfglue}\ (\mathsf{winr}\ y)) = \mathsf{cfglue}\ (\mathsf{winr}\ y)$. We leave the reader to check these, which are straightforward if tedious computations. □

We may now conclude:

Corollary 4.16. For any $X, Y : \mathcal{U}_*$, there is a pointed equivalence $\Sigma(X \times Y) \simeq \Sigma X \vee \Sigma Y \vee \Sigma(X \wedge Y)$, and thus an isomorphism $H^n(X \times Y) \simeq H^n(X) \times H^n(Y) \times H^n(X \wedge Y)$.

Proof. By the previous lemma, $X \wedge Y \equiv \mathsf{Cof}\ [\lambda x.(x, y_0), \lambda y.(x_0, y)] \simeq \mathsf{Cof}(\lambda y.\mathsf{cfcod}_{\lambda x.(x,y_0)}(x_0, y))$. Since $x \mapsto (x, y_0)$ and $y \mapsto (x_0, y)$ each have obvious left inverses, we can apply Theorem 4.13 twice to achieve the desired result. □

4.5 The Mayer-Vietoris Sequence

Given a span $X \overset{(f,p)}{\leftarrow\cdot} Z \overset{(g,q)}{\cdot\rightarrow} Y$, the *Mayer-Vietoris sequence* is a long exact sequence of the form

$$\cdots \longrightarrow H^{n-1}(Z) \overset{\mathsf{ext\text{-}glue}^*}{\longrightarrow} H^n(X \sqcup^Z Y) \overset{\langle \mathsf{left}^*, \mathsf{right}^* \rangle}{\longrightarrow} H^n(X) \times H^n(Y) \overset{f^* - g^*}{\longrightarrow} H^n(Z) \longrightarrow \cdots$$

The formulation as an exact sequence originates in [6, p.37]. Using this sequence we can, at least in some cases, determine the cohomology groups of $X \sqcup^Z Y$ in terms of those of X,Y, and Z. We can also think of this as a generalization of the long exact cofiber sequence, which we obtain (modulo some equivalences) as the special case where $Y = 1$. It is, however, derivable from the cofiber sequence. The result is typically proven algebraically, as in [6] or more recently in [19, p.110]. Though in principle we could repeat that proof in this setting, we instead show that a particular sequence of maps is equivalent to a cofiber sequence, as is suggested in [5], and apply the cohomology functor to derive the result.

Theorem 4.17 (Mayer-Vietoris)**.** For any span $X \overset{(f,p)}{\leftarrow\cdot} Z \overset{(g,q)}{\cdot\rightarrow} Y$, define the map

$$\begin{aligned}
&\mathsf{reglue} : X \vee Y \to X \sqcup^Z Y \\
&\mathsf{reglue}\ (\mathsf{winl}\ x) :\equiv \mathsf{left}\ x \\
&\mathsf{reglue}\ (\mathsf{winr}\ y) :\equiv \mathsf{right}\ y \\
&\mathsf{ap}_{\mathsf{reglue}}\mathsf{wglue} := \mathsf{ap}_{\mathsf{left}}(p^{-1}) \cdot \mathsf{glue}\ z_0 \cdot \mathsf{ap}_{\mathsf{right}}(q^{-1})
\end{aligned}$$

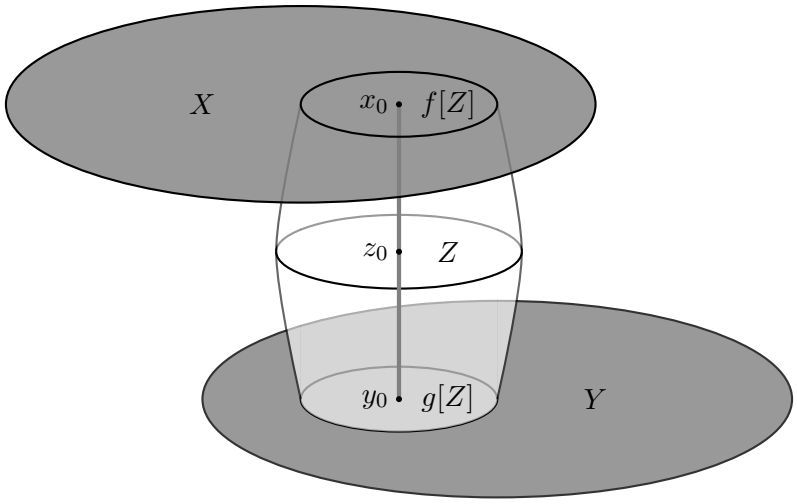
There is a pointed equivalence $\mathsf{mv\text{-}equiv}^\odot : \mathsf{Cof}\ \mathsf{reglue} \simeq \Sigma Z$. We have (1) $\mathsf{cfcod} =^{V.X \sqcup^Z Y \to V}_{\mathsf{ua}^\odot \mathsf{mv\text{-}equiv}^\odot} \mathsf{ext\text{-}glue}$ and (2) $\mathsf{ext\text{-}glue} =^{U.U \to \Sigma(X \vee Y)}_{\mathsf{ua}^\odot \mathsf{mv\text{-}equiv}^\odot} \mathsf{diff}$, where $\mathsf{diff} : \Sigma Z \to \Sigma(X \vee Y)$ is defined by

$$\begin{aligned}
&\mathsf{diff}\ \mathsf{north} :\equiv \mathsf{north} \\
&\mathsf{diff}\ \mathsf{south} :\equiv \mathsf{north} \\
&\mathsf{ap}_{\mathsf{diff}}(\mathsf{merid}\ z) := \mathsf{merid}\ (fz) \cdot (\mathsf{merid}\ (gz))^{-1}
\end{aligned}$$

Proof. For the proof, we assume by path induction that $p \equiv \mathsf{refl}$ and $q \equiv \mathsf{refl}$ (and thus that $fz_0 \equiv x_0$ and $fz_0 \equiv y_0$).

Defining the Equivalence Maps

As an intuitive argument we consider a visualization of the pushout $X \sqcup^Z Y$, with the image of $X \vee Y$ shaded:



To construct the cofiber type Cof reglue is essentially to contract this shaded part to a point. With X and Y contracted to poles (and leaving the already contractible path connecting them intact), the resulting space is seen to be ΣZ.

We define a map into : Cof reglue $\to \Sigma Z$ as suggested by the picture. For the point cases, we set

$$\begin{aligned}\mathsf{into}\ \mathsf{cfbase} &:\equiv \mathsf{north}\\ \mathsf{into}\ (\mathsf{cfcod}\ \gamma) &:\equiv \mathsf{ext\text{-}glue}\ \gamma\end{aligned}$$

For the coherence we will define into-glue : $\prod_{w:X\vee Y} \mathsf{cfbase} = \mathsf{ext\text{-}glue}\ (\mathsf{reglue}\ w)$, setting $\mathsf{ap}_{\mathsf{into}}(\mathsf{cfglue}\ w) :=$ into-glue w. For the point cases,

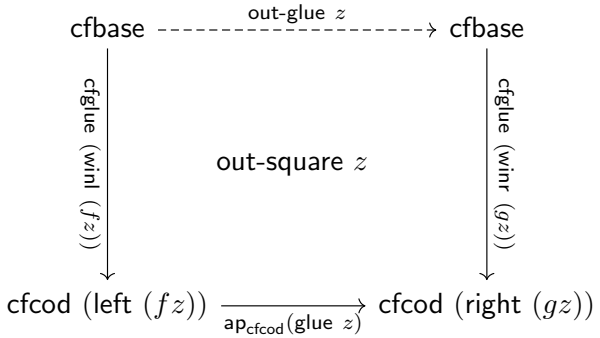
$$\begin{aligned}\mathsf{into\text{-}glue}\ (\mathsf{winl}\ x) &:\equiv \mathsf{refl}\\ \mathsf{into\text{-}glue}\ (\mathsf{winr}\ y) &:\equiv \mathsf{merid}\ z_0\end{aligned}$$

To complete this definition, we need to give a path $\mathsf{refl} =^{w.\mathsf{north}=\mathsf{ext\text{-}glue}\ (\mathsf{reglue}\ w)}_{\mathsf{wglue}} \mathsf{merid}\ z_0$. By Lemma 2.4 this is equivalent to giving a square

$$\begin{array}{ccc}\mathsf{north} & \xrightarrow{\mathsf{refl}} & \mathsf{north}\\ {\scriptstyle \mathsf{refl}}\downarrow & & \downarrow{\scriptstyle \mathsf{merid}\ z_0}\\ \mathsf{north} & \xrightarrow[\mathsf{ap}_{\mathsf{ext\text{-}glue}\circ\mathsf{reglue}}\mathsf{wglue}]{} & \mathsf{south}\end{array}$$

We have $\mathsf{ap}_{\mathsf{ext\text{-}glue}\circ\mathsf{reglue}}\mathsf{wglue} = \mathsf{ap}_{\mathsf{ext\text{-}glue}}(\mathsf{glue}\ z_0) = \mathsf{merid}\ z_0$ and so the square commutes.

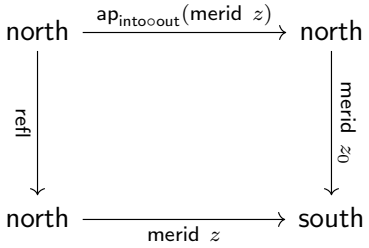
We define out : $\Sigma Z \to$ Cof reglue by suspension recursion. We want $\mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ z)$ to correspond to glue z in the pushout, but we need a path with constant endpoints, so we take $\mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ z) =$ out-glue as given by the following square filler (per Theorem 2.6):



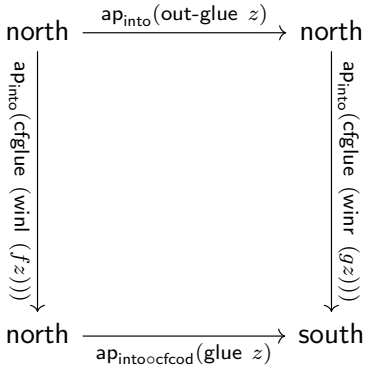
This forces out north :≡ out south :≡ cfbase.

Right Inverse

We show into ∘ out ∼ $\mathsf{id}_{\Sigma Z}$ by induction on the suspension. For the point cases we have refl : into (out north) = north and merid z_0 : into (out south) = south. For the merid case we need to give for $z : Z$ a path refl $=^{\sigma.\mathsf{into}\ (\mathsf{out}\ \sigma)=\sigma}_{\mathsf{merid}\ z}$ merid z_0. By Lemma 2.4 it suffices to give a square



Note that $\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}}(\mathsf{merid}\ z) = \mathsf{ap}_{\mathsf{into}}(\mathsf{out\text{-}glue}\ z)$. We observe that ap-square into (out-square z) has type



Since $\mathsf{ap}_{\mathsf{into}}(\mathsf{cfglue}\ (\mathsf{winl}\ (fz))) = \mathsf{refl}$, $\mathsf{ap}_{\mathsf{into}}(\mathsf{cfglue}\ (\mathsf{winr}\ (gz))) = \mathsf{merid}\ z_0$, and $\mathsf{ap}_{\mathsf{into}\circ\mathsf{cfcod}}(\mathsf{glue}\ z) = \mathsf{merid}\ z$, this square serves to satisfy our requirement.

Left Inverse

We show $\mathsf{out} \circ \mathsf{into} \sim \mathsf{id}_{\mathsf{Cof}\ \mathsf{reglue}}$ by induction on the cofiber type. Observe that $X \vee Y$ is the pushout of the span $X \leftarrow 1 \rightarrow Y$, and that the right map of this span has a left inverse. Thus $\mathsf{Cof}\ \mathsf{reglue}$ satisfies the conditions of Theorem 4.7, and we may omit the highest coherence.

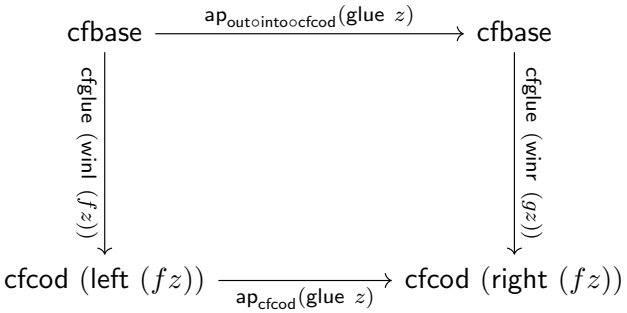
For the cfbase case we have $\mathsf{refl} : \mathsf{out}\ (\mathsf{into}\ \mathsf{cfbase}) = \mathsf{cfbase}$. For the cfcod case we will define $\mathsf{out\text{-}into\text{-}cod} : \prod_{\gamma : X \sqcup^Z Y} \mathsf{out}\ (\mathsf{into}\ (\mathsf{cfcod}\ \gamma)) = \mathsf{cfcod}\ \gamma$ by induction on $X \sqcup^Z Y$, with the point cases defined as follows:

$$\mathsf{out\text{-}into\text{-}cod}\ (\mathsf{left}\ x) :\equiv \mathsf{cfglue}\ (\mathsf{winl}\ x)$$
$$\mathsf{out\text{-}into\text{-}cod}\ (\mathsf{right}\ y) :\equiv \mathsf{cfglue}\ (\mathsf{winr}\ y)$$

For the path case we must show for every $z : Z$ that

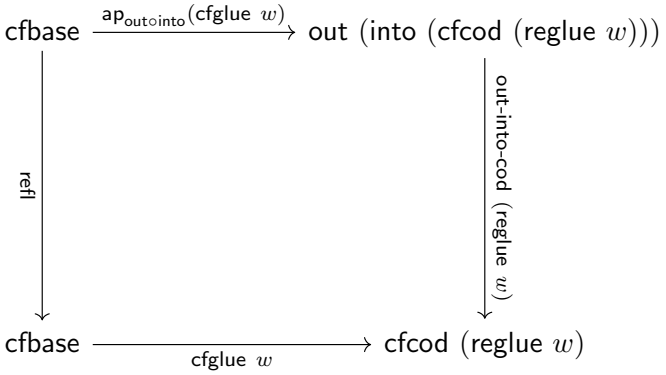
$$\mathsf{cfglue}\ (\mathsf{winl}\ (fz)) =^{\gamma.\mathsf{out}\ (\mathsf{into}\ (\mathsf{cfcod}\ \gamma)) = \mathsf{cfcod}\ \gamma}_{\mathsf{glue}\ z} \mathsf{cfglue}\ (\mathsf{winr}\ (gz))$$

By Lemma 2.4 we need a square

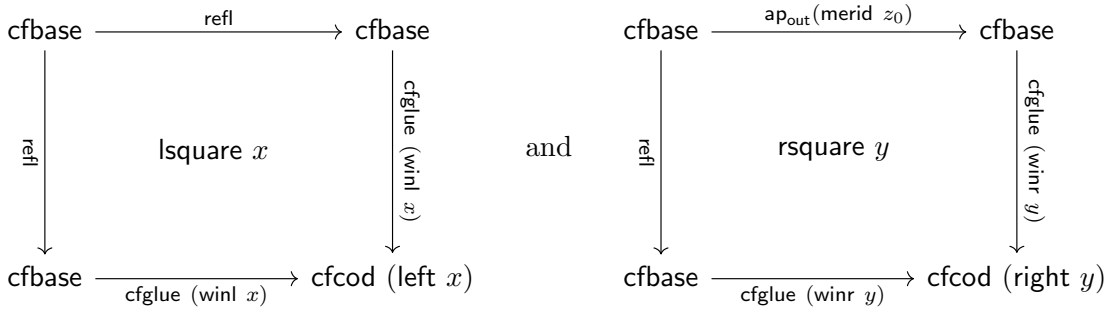


Since $\mathsf{out\text{-}square}\ z$ has the same left, bottom, and right sides as this square, it suffices to show that $\mathsf{ap}_{\mathsf{out} \circ \mathsf{into} \circ \mathsf{cfcod}}(\mathsf{glue}\ z) = \mathsf{out\text{-}glue}\ z$. For this we have $\mathsf{ap}_{\mathsf{out} \circ \mathsf{into} \circ \mathsf{cfcod}}(\mathsf{glue}\ z) :\equiv \mathsf{ap}_{\mathsf{out} \circ \mathsf{ext\text{-}glue}}(\mathsf{glue}\ z) = \mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ z) = \mathsf{out\text{-}glue}\ z$.

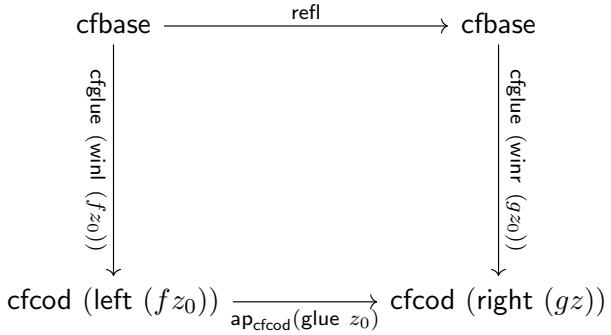
We have completed the cfbase and cfcod cases; we must now show that for any $w : X \vee Y$, we have $\mathsf{refl} =^{\kappa.\mathsf{out}\ (\mathsf{into}\ \kappa) = \kappa}_{\mathsf{cfglue}\ w} \mathsf{out\text{-}into\text{-}cod}\ (\mathsf{reglue}\ w)$. By Lemma 2.4, it is equivalent to show that we have a square



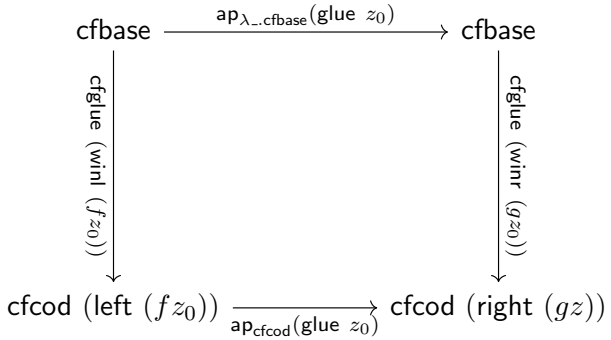
Note that $\mathsf{ap}_{\mathsf{out} \circ \mathsf{into}}(\mathsf{cfglue}\ w) = \mathsf{ap}_{\mathsf{out}}(\mathsf{into\text{-}glue}\ w)$. After making this simplification we proceed by induction on $w : W \vee Y$. Per our application of Theorem 4.7, we may ignore the wglue case. Thus we need only prove the winl and winr cases, which after reductions amount to inhabiting the following two squares:



The first of these clearly commutes. To show the second commutes, we must show that $\mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ z_0) = \mathsf{refl}$. We have $\mathsf{ap}_{\mathsf{out}}(\mathsf{merid}\ z_0) = \mathsf{out\text{-}glue}\ z_0$, so by uniqueness of square fillers (Theorem 2.6) it suffices to demonstrate that the square



is inhabited. Since natural-sq cfglue wglue gives us a square



and $\mathsf{ap}_{\lambda\_.\mathsf{cfbase}}(\mathsf{glue}\ z_0) = \mathsf{refl}$, we are done.

This completes the definition of the equivalence, which is definitionally basepoint-preserving.

Constructing the Paths (1) and (2)

By Assertion 1.8, $\mathsf{cfcod} =^{V.X \sqcup^Z Y \to V}_{\mathsf{ua}^\odot \circ \mathsf{mv\text{-}equiv}^\odot} \mathsf{into} \circ \mathsf{cfcod}$, and $\mathsf{into} \circ \mathsf{cfcod} \equiv \mathsf{ext\text{-}glue}$, so we have the path (1). For (2), Assertion 1.9 gives a path $\mathsf{diff} \circ \mathsf{into} =^{U.U \to \Sigma(X \vee Y)}_{\mathsf{ua}^\odot \circ \mathsf{mv\text{-}equiv}^\odot} \mathsf{diff}$, so it suffices to show $\prod_{\kappa : \mathsf{Cof}\ \mathsf{reglue}} \mathsf{diff}\ (\mathsf{into}\ \kappa) = \mathsf{ext\text{-}glue}\ \kappa$. With the help of Theorem 4.7, this is a straightforward proof by induction, which we leave as an exercise for the reader. □

Combining this result with Theorem 4.3 gives us an exact sequence

$$\cdots \longrightarrow H^{n\text{-}1}(X \vee Y) \xrightarrow{\mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^*} H^n(\Sigma Z) \xrightarrow{\mathsf{ext\text{-}glue}^*} H^n(X \sqcup^Z Y) \xrightarrow{\mathsf{reglue}^*} H^n(X \vee Y) \longrightarrow \cdots$$

We want to extract from this a sequence

$$\cdots \longrightarrow H^{n\text{-}1}(X) \times H^{n\text{-}1}(Y) \xrightarrow{f^* - g^*} H^{n\text{-}1}(Z) \xrightarrow{\partial} H^{n\text{-}1}(X \sqcup^Z Y) \xrightarrow{\langle \mathsf{left}^*, \mathsf{right}^* \rangle} H^{n\text{-}1}(X) \times H^{n\text{-}1}(Y) \longrightarrow \cdots$$

As before we have $\partial :\equiv \mathsf{susp\text{-}iso}^{n\text{-}1}_Z \circ \mathsf{ext\text{-}glue}^*$, now mapping out of a pushout rather than cofiber type. The main work here is in showing that $\mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^*$ corresponds to $f^* - g^*$. In the codomain we have the equivalence $\mathsf{susp\text{-}iso}^{n\text{-}1}_Z : H^n(\Sigma Z) \simeq H^{n\text{-}1}(Z)$, over which the former map corresponds to $\mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_Z)^{-1}$. In the domain we take the equivalence $H^{n\text{-}1}(X \vee Y) \simeq H^n(X) \times H^n(Y)$ given in Theorem 4.6. Over this equivalence, we have that projl^* and projr^* correspond over the equivalence to the injections into $H^{n\text{-}1}(X) \times H^{n\text{-}1}(Y)$. It therefore suffices to show that

$$\mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_Z)^{-1} \circ \mathsf{projl}^* = f^*,$$
$$\mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_Z)^{-1} \circ \mathsf{projr}^* = -g^*$$

We can compute the first easily:

$$\begin{aligned}
\mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_Z)^{-1} \circ \mathsf{projl}^* &= \mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ \mathsf{diff}^* \circ (\Sigma \mathsf{projl})^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_X)^{-1} \\
&= \mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ (\Sigma \mathsf{projl} \circ \mathsf{diff})^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_X)^{-1} \\
&= \mathsf{susp\text{-}iso}^{n\text{-}1}_{X \vee Y} \circ (\Sigma f)^* \circ (\mathsf{susp\text{-}iso}^{n\text{-}1}_X)^{-1} \\
&= f^*
\end{aligned}$$

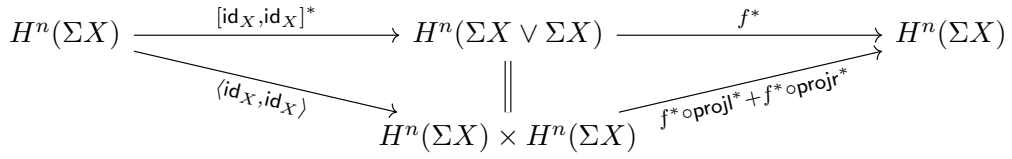
Here we use that $\Sigma \mathsf{projl} \circ \mathsf{diff} = \Sigma f$, which is easily evident. With the right projection, we instead have $\Sigma \mathsf{projr} \circ \mathsf{diff} = \Sigma f \circ \mathsf{flip}$; thus what remains is to show that flip^* is the group inverse. Hence:

Lemma 4.18. Let $X : \mathcal{U}_*$. The map $\mathsf{flip}^* : H^n(\Sigma X) \to H^n(\Sigma X)$ is the group inverse for $H^n(\Sigma X)$.

Proof. Define a map $f : \Sigma X \cdot\!\!\to \Sigma X \vee \Sigma X$ by

$$\begin{aligned}
&f\ \mathsf{north} :\equiv \mathsf{winl}\ \mathsf{south} \\
&f\ \mathsf{south} :\equiv \mathsf{winr}\ \mathsf{south} \\
&\mathsf{ap}_f(\mathsf{merid}\ x) = \mathsf{ap}_{\mathsf{winl}}((\mathsf{merid}\ x)^{-1}) \cdot \mathsf{wglue} \cdot \mathsf{ap}_{\mathsf{winr}}(\mathsf{merid}\ x)
\end{aligned}$$

Note that $[\mathsf{id}_X, \mathsf{id}_X] \circ f$ is equal to a constant function. If we transport these two functions across the equivalence given by Theorem 4.6, we have



Thus we have that $f^* \circ \mathsf{projl}^* = -(f^* \circ \mathsf{projr}^*)$. Finally, we observe that $f^* \circ \mathsf{projl}^* = \mathsf{projl} \circ f^* = \mathsf{flip}^*$ and $f^* \circ \mathsf{projr}^* = \mathsf{projr} \circ f^* = \mathsf{id}^* = \mathsf{id}$, which gives the desired result. □

4.6 Cohomology of Products of Spheres

We now compute the cohomology of arbitrary finite products of spheres in terms of the cohomology of spheres. While the route we take may not be the most straightforward, it gives us the opportunity to demonstrate some more involved equivalence proofs, in particular in Theorems 4.19 and 4.21. We obtain the result by more generally computing $H^n(S^k \times X)$ in terms of $H^n(X)$ and the cohomology of spheres. For this, we first use Corollary 4.16 to obtain

$$\Sigma(S^k \times X) \simeq S^{k+1} \vee \Sigma X \vee \Sigma(S^k \wedge X)$$

We then show that the suspension of a smash product is equivalent to a join, so that $\Sigma(S^k \wedge X) \simeq S^k * X$. Finally, we show that $S^0 * Z \simeq \Sigma Z$ and that the join commutes with suspension, in the sense that $(\Sigma Y * Z) \simeq \Sigma(Y * Z)$. From these it follows that $S^k * X \simeq \Sigma^{k+1} X$. Hence,

$$\Sigma(S^k \times X) \simeq S^{k+1} \vee \Sigma X \vee \Sigma^{k+1} X$$

and therefore $H^n(S^k \times X) \simeq H^{n-k}(S^0) \times H^n(X) \times H^{n-k}(X)$. This is enough to compute the cohomology of finite products of spheres in terms of the cohomology of spheres. Incidentally, it also shows that the suspension of any product of spheres is equivalent to a wedge of spheres.

Theorem 4.19. For any $X, Y : \mathcal{U}_*$, there is a pointed equivalence $\Sigma(X \wedge Y) \simeq X * Y$.

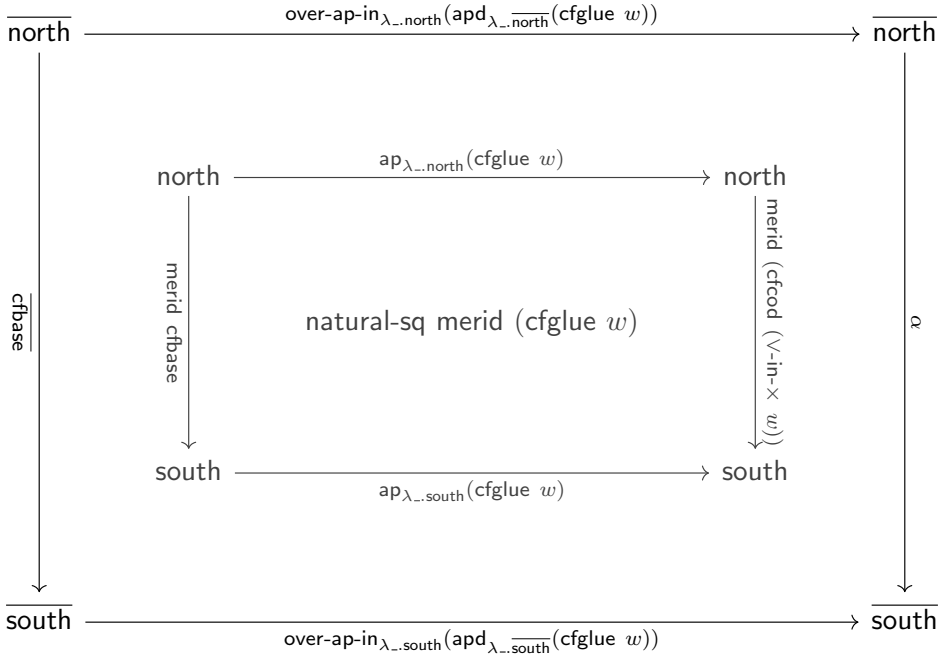
In order to simplify this proof, we first prove a lemma in the vein of Theorem 4.7 which enables automatic coherences for induction on $\Sigma(X \wedge Y)$.

Lemma 4.20. Let $P : \Sigma(X \wedge Y) \to \mathcal{U}$ be given. In order to construct a term of type $\prod_{\sigma:\Sigma(X\wedge Y)} P(\sigma)$, it suffices to give

- an element $\overline{\mathsf{north}} : P\ \mathsf{north}$,
- an element $\overline{\mathsf{south}} : P\ \mathsf{south}$,
- for $x : X$, $y : Y$, a path $\overline{\mathsf{cfcod}} : \overline{\mathsf{north}} =^P_{\mathsf{merid}(\mathsf{cfcod}(x,y))} \overline{\mathsf{south}}$.

Proof. We prove $\prod_{\sigma:\Sigma(X\wedge Y)} P(\sigma)$ by induction on σ. For the north and south cases we take $\overline{\mathsf{north}}$ and $\overline{\mathsf{south}}$ as given. We must then give, for $s : X \wedge Y$, a path $\overline{\mathsf{north}} =^P_{\mathsf{merid}\ s} \overline{\mathsf{south}}$. Since $X \wedge Y$

is the cofiber type of a pushout satisfying the conditions of Theorem 4.7, we may ignore the highest coherence (the wglue case). We define the cfbase case, $\overline{\mathsf{cfbase}} : \overline{\mathsf{north}} =^{P}_{\mathsf{merid}\ \mathsf{cfbase}} \overline{\mathsf{south}}$, by transporting $\overline{\mathsf{cfcod}}(x_0, y_0)$ along the path $\mathsf{cfglue}(\mathsf{winl}\ x_0)$. Before giving the cfcod case (which we will call $\overline{\mathsf{cfcod}}'$), we consider what is necessary for the $\mathsf{cfglue} \circ \mathsf{winl}$ and $\mathsf{cfglue} \circ \mathsf{winr}$ cases. For $w : X \vee Y$ in general, Lemma 2.7 asks that we provide a dependent square function, where α is $\overline{\mathsf{cfcod}}'(x, y)$:



Let us refer to this square type as $S(w, \alpha)$. Note that we cannot go as in Theorem 4.7 directly, because $\vee\text{-in-}\times : X \vee Y \to X \times Y$ is not a section. However, within the cfcod case, we have (x, y), so we *can* separately apply Lemma 4.8 and choose paths $p(x), q(y) : \overline{\mathsf{north}} = \overline{\mathsf{north}}$ such that $S(\mathsf{winl}\ x, p(x) \lhd \overline{\mathsf{cfcod}}(x, y_0))$ and $S(\mathsf{winr}\ y, q(y) \lhd \overline{\mathsf{cfcod}}(x_0, y))$ are satisfied. If we additionally knew that $p(x_0) = q(y_0) = \mathsf{refl}$, we could obtain our result by setting $\overline{\mathsf{cfcod}}'(x, y) :\equiv (p(x) \cdot q(y)) \lhd \overline{\mathsf{cfcod}}(x, y)$. While our current definition fails to ensure this, we can correct it with a further adjustment. First, take $r : \overline{\mathsf{north}} = \overline{\mathsf{south}}$ to be the path such that $S(\mathsf{winl}\ x_0, r \lhd \overline{\mathsf{cfcod}}(x_0, y_0))$ holds. We then choose $p'(x)$ and $q'(y)$ such that $S(\mathsf{winl}\ x, p'(x) \lhd r \lhd \overline{\mathsf{cfcod}}(x, y_0))$ and $S(\mathsf{winr}\ y, q'(y) \lhd r \lhd \overline{\mathsf{cfcod}}(x_0, y))$ are satisfied. By the uniqueness condition of Lemma 4.8, we get that $p'(x_0) = q'(y_0) = \mathsf{refl}$, and therefore we may take

$$\overline{\mathsf{cfcod}}'(x, y) :\equiv (p'(x) \cdot q'(y)) \lhd r \lhd \overline{\mathsf{cfcod}}(x, y)$$

to satisfy both the $\mathsf{winl}\ x$ and $\mathsf{winr}\ y$ square conditions. □

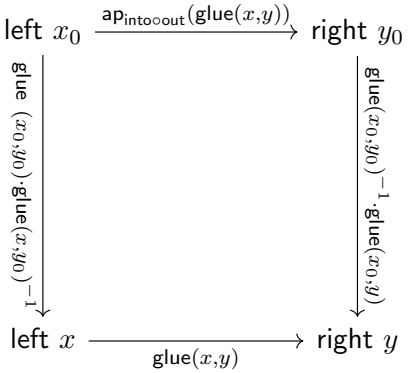
Proof (of Theorem 4.19). We begin by defining a map $\mathsf{out} : X * Y \to \Sigma(X \wedge Y)$. We set

$$\begin{aligned}
&\mathsf{out}\ (\mathsf{left}\ x) :\equiv \mathsf{north}\\
&\mathsf{out}\ (\mathsf{right}\ y) :\equiv \mathsf{south}\\
&\mathsf{out}\ (\mathsf{glue}\ (x, y)) := \mathsf{merid}\ (\mathsf{cfcod}\ (x, y))
\end{aligned}$$

For the inverse $\mathsf{into} : \Sigma(X \wedge Y) \to X * Y$, we set $\mathsf{into\ north} :\equiv \mathsf{left}\ x_0$ and $\mathsf{into\ south} :\equiv \mathsf{right}\ y_0$, and take $\mathsf{ap}_{\mathsf{into}}(\mathsf{merid}\ s) := \mathsf{into\text{-}glue}\ s$ where $\mathsf{into\text{-}glue} : X \wedge Y \to \mathsf{left}\ x_0 = \mathsf{right}\ y_0$ is defined by

$$\begin{aligned}
&\mathsf{into\text{-}glue\ cfbase} :\equiv \mathsf{glue}(x_0, y_0)\\
&\mathsf{into\text{-}glue}\ (\mathsf{cfcod}\ (x, y)) :\equiv \mathsf{glue}(x_0, y_0) \cdot \mathsf{glue}(x, y_0)^{-1} \cdot \mathsf{glue}(x, y) \cdot \mathsf{glue}(x_0, y)^{-1} \cdot \mathsf{glue}(x_0, y_0)
\end{aligned}$$

We first prove $\mathsf{into} \circ \mathsf{out} \sim \mathsf{id}$. We have $\mathsf{glue}\ (x_0, y_0) \cdot \mathsf{glue}(x, y_0)^{-1}$ as a proof $\mathsf{into}\ (\mathsf{out}\ (\mathsf{left}\ x)) = \mathsf{left}\ x$, and $\mathsf{glue}(x_0, y_0)^{-1} \cdot \mathsf{glue}(x_0, y)$ as a proof $\mathsf{into}\ (\mathsf{out}\ (\mathsf{right}\ x)) = \mathsf{right}\ x$. For the glue case, we must construct for $x : X, y : Y$ a square



By computing $\mathsf{ap}_{\mathsf{into}\circ\mathsf{out}}(\mathsf{glue}(x, y))$ we see that this commutes.

We now show $\mathsf{out} \circ \mathsf{into} \sim \mathsf{id}$. Here we use Lemma 4.20. Since the north and south cases hold definitionally, we only need to show that $\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}}(\mathsf{merid}\ (\mathsf{cfcod}\ (x, y))) = \mathsf{merid}\ (\mathsf{cfcod}\ (x, y))$. We have

$$\mathsf{ap}_{\mathsf{out}\circ\mathsf{into}}(\mathsf{merid}\ (\mathsf{cfcod}(x, y))) = \mathsf{ap}_{\mathsf{out}}(\mathsf{glue}(x_0, y_0) \cdot \mathsf{glue}(x, y_0)^{-1} \cdot \mathsf{glue}(x, y) \cdot \mathsf{glue}(x_0, y)^{-1} \cdot \mathsf{glue}(x_0, y_0))$$

Now we simply use the fact that $\mathsf{ap}_{\mathsf{out}}(\mathsf{glue}\ (\vee\text{-}\mathsf{in}\text{-}\times\ w)) = \mathsf{merid}\ (\mathsf{cfcod}\ (\vee\text{-}\mathsf{in}\text{-}\times\ w)) = \mathsf{merid\ cfbase}$ to cancel the outer terms and reduce this to $\mathsf{merid}\ (\mathsf{cfcod}\ (x, y))$. □

Now we have reduced the problem to computing the cohomology of $S^n * X$ for $X : \mathcal{U}_*$. For this it suffices to show first that $S^0 * X \simeq \Sigma X$ for $X : \mathcal{U}_*$ and then that join is associative. In that case, we have for any $X, Y : \mathcal{U}_*$ that

$$\Sigma X * Y \ \simeq\ (S^0 * X) * Y \ \simeq\ S^0 * (X * Y) \ \simeq\ \Sigma(X * Y)$$

The first of these we leave as an exercise to the reader, as it is a straightforward equivalence proof. The second is more complicated. We go by direct equivalence proof; there is a formal proof due to

Brunerie [11, `homotopy.JoinAssoc3x3`] which uses the 3×3 lemma, but our proof is simpler for its specificity.

Theorem 4.21. For any $X, Y, Z : \mathcal{U}_*$, there is a pointed equivalence $(X * Y) * Z \simeq X * (Y * Z)$.

Proof. Since Lemma 4.2 tells us that join is commutative, it suffices to define for $X, Y, Z : \mathcal{U}_*$ a map $\mathsf{switch} : (X * Y) * Z \to (Z * Y) * X$ and then show that this map is involutive.

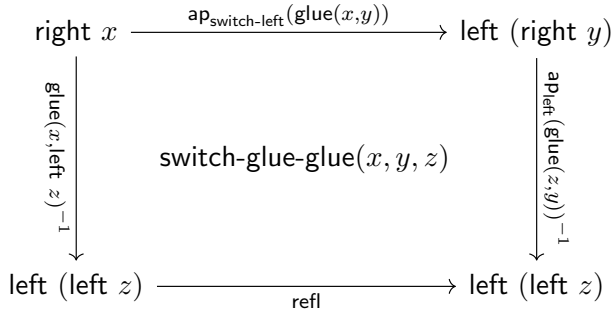
First we define a map $\mathsf{switch\text{-}left} : X * Y \to (Z * Y) * X$ for the left case:

$$\begin{aligned}
&\mathsf{switch\text{-}left}\ (\mathsf{left}\ x) :\equiv \mathsf{right}\ x\\
&\mathsf{switch\text{-}left}\ (\mathsf{right}\ y) :\equiv \mathsf{left}\ (\mathsf{right}\ y)\\
&\mathsf{ap}_{\mathsf{switch\text{-}left}}(\mathsf{glue}(x, y)) := \mathsf{glue}(\mathsf{right}\ y, x)^{-1}
\end{aligned}$$

We thus take $\mathsf{switch}\ (\mathsf{left}\ j) :\equiv \mathsf{switch\text{-}left}\ j$ and $\mathsf{switch}\ z :\equiv \mathsf{left}\ (\mathsf{left}\ z)$. It remains to give for $(j, z) : (X * Y) \times Z$ a path $\mathsf{switch\text{-}glue}(j, z) : \mathsf{switch\text{-}left}\ j = \mathsf{left}\ (\mathsf{left}\ z)$. For this we go by induction on j, first setting

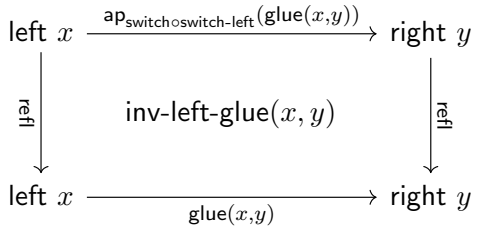
$$\begin{aligned}
&\mathsf{switch\text{-}glue}(\mathsf{left}\ x, z) :\equiv \mathsf{glue}(\mathsf{left}\ z, x)^{-1}\\
&\mathsf{switch\text{-}glue}(\mathsf{right}\ y, z) :\equiv \mathsf{ap}_{\mathsf{left}}(\mathsf{glue}(z, y))^{-1}
\end{aligned}$$

To define $\mathsf{ap}_{j.\mathsf{switch\text{-}glue}(j,z)}(\mathsf{glue}(x, y))$ would require giving a square

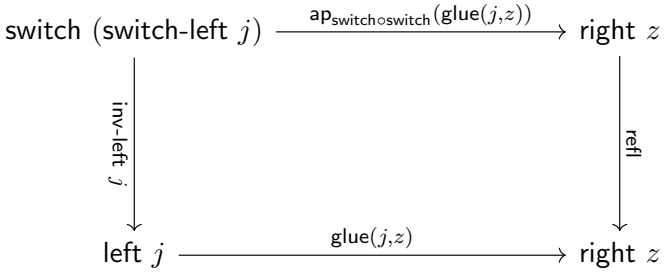


We defer the definition of $\mathsf{ap}_{j.\mathsf{switch\text{-}glue}(j,z)}(\mathsf{glue}(x, y))$ for now; the necessary choice will become clear later.

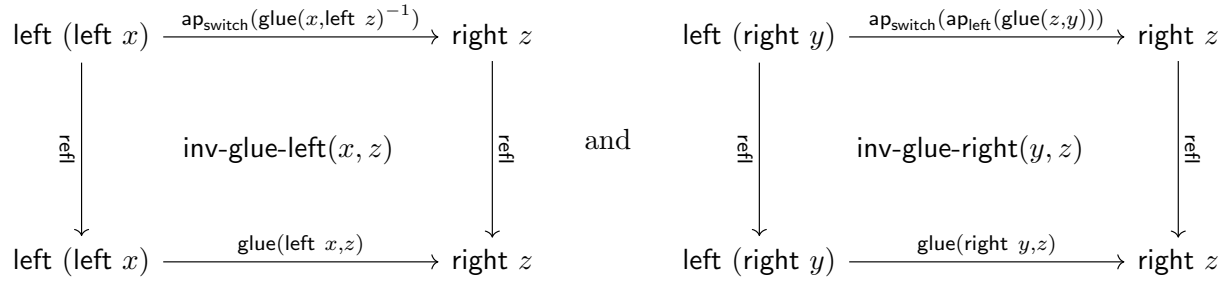
We now show that for $u : (X * Y) * Z$ we have $\mathsf{switch}\ (\mathsf{switch}\ u) = u$ (where the two applications of switch are instantiated at the appropriate types). First, we need a proof $\mathsf{inv\text{-}left} : \prod_{j:X*Y} \mathsf{switch}\ (\mathsf{switch\text{-}left}\ j) = \mathsf{left}\ j$. The point cases hold definitionally, leaving us to give a family of squares



Again, we defer this choice. For $z : Z$ we have $\mathsf{switch}\ (\mathsf{switch}\ (\mathsf{right}\ z)) = \mathsf{right}\ z$ definitionally. It remains to give for $(j, z) : (X * Y) \times Z$ a path $\mathsf{inv\text{-}glue}(j, z) : \mathsf{inv\text{-}left}\ j =^{u.\mathsf{switch}\ (\mathsf{switch}\ u)=u}_{\mathsf{glue}(j,z)} \mathsf{refl}$, that is, a square



Fix $z : Z$. We first observe that $\mathsf{ap}_{\mathsf{switch}\circ\mathsf{switch}}(\mathsf{glue}(j, z)) = \mathsf{ap}_{\mathsf{switch}}(\mathsf{switch\text{-}glue}(j, z))$, then proceed by induction on $j : X * Y$. For the left and right cases, this amounts to giving two families of squares



By Theorem 2.12, we may prove the glue case by giving for $(x, y) : X \times Y$ a cube of type

$$\begin{aligned}
&\mathsf{Cube}\ (\mathsf{inv\text{-}glue\text{-}left}(x, z))\ (\mathsf{inv\text{-}glue\text{-}right}(y, z)) \\
&\quad (\mathsf{natural\text{-}sq}\ \mathsf{inv\text{-}left}\ (\mathsf{glue}(x, y))) \\
&\quad (\mathsf{natural\text{-}sq}\ (\lambda j.\mathsf{ap}_{\mathsf{switch}}(\mathsf{switch\text{-}glue}(\mathsf{glue}(j, z))))\ (\mathsf{glue}(x, y))) \\
&\quad (\mathsf{natural\text{-}sq}\ (\lambda j.\mathsf{glue}(j, z))\ (\mathsf{glue}(x, y))) \\
&\quad (\mathsf{natural\text{-}sq}\ (\lambda_.\mathsf{refl})\ (\mathsf{glue}(x, y)))
\end{aligned}$$

In this case, however, it is worthwhile to instead use a more restrictive, specialized construction. The above suffers from a proliferation of terms of the form $\mathsf{ap}_{\lambda\_.\ldots}(\ldots)$, which are propositionally but not definitionally equal to refl; these present substantial complications later on. For this reason, we instead use the following constructions:

Lemma 4.22. Let $f : A \to B$ and $b : B$. For any $x, y : A$, $p : x = y$, $u : fx = b$, and $v : fy = b$, the type $u =^{z.fz=b}_{p} v$ is equivalent to $\mathsf{Square}\ u\ (\mathsf{ap}_f p)\ \mathsf{refl}\ (hy)$.

Proof. By induction we may assume $p \equiv \mathsf{refl}$, in which case the equivalence given in Lemma 2.4 serves. □

Lemma 4.23. Let $f : A \to B$ and $b : B$ be given along with $h : \prod_{a:A} fa = b$ be given. For any $x, y : A$ and $p : x = y$ there is a square $\mathsf{app\text{-}cst\text{-}sq}\ h\ p$ of type $\mathsf{Square}\ (hx)\ (\mathsf{ap}_f(p))\ \mathsf{refl}\ (hy)$. In particular, $\mathsf{app\text{-}cst\text{-}sq}\ h\ \mathsf{refl} = \mathsf{srefl\text{-}h}$.

Proof. We take the square corresponding to $\mathsf{apd}_h p$ over the previous lemma's equivalence. The latter requirement can be confirmed by inducting and tracing through definitions. □

Lemma 4.24. Let $f_{00}, f_{01} : A \to B$ and $b : B$ be given along with $p_{0\text{-}} : \prod_{a:A} f_{00}a = f_{01}a$, $p_{\text{-}0} : \prod_{a:A} f_{00}a = b$ and $p_{\text{-}1} : \prod_{a:A} f_{01}a = b$. For any $x, y : A$, $q : x = y$, and two squares $s_x : \mathsf{Square}\ (p_{0\text{-}}x)\ (p_{\text{-}0}x)\ (p_{\text{-}1}x)\ \mathsf{refl}$ and $s_y : \mathsf{Square}\ (p_{0\text{-}}x)\ (p_{\text{-}0}x)\ (p_{\text{-}1}x)\ \mathsf{refl}$, we can construct a term of type

$$s_x =_q^{z.\mathsf{Square}\ (p_{0\text{-}}z)\ (p_{\text{-}0}z)\ (p_{\text{-}1}z)\ \mathsf{refl}} s_y$$

from a term of type

$$\mathsf{Cube}\ s_x\ s_y\ (\mathsf{natural\text{-}sq}\ p_{0\text{-}}\ q)\ (\mathsf{app\text{-}cst\text{-}sq}\ p_{\text{-}0}\ q)\ (\mathsf{app\text{-}cst\text{-}sq}\ p_{\text{-}1}\ q)\ \mathsf{srefl}$$

Proof. By induction we may assume $q \equiv \mathsf{refl}$, in which case Lemma 2.11 applies. □

By applying the previous lemma, our obligation is reduced to providing a cube of type

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{inv\text{-}glue\text{-}left}(x,z))\ (\mathsf{inv\text{-}glue\text{-}right}(y,z))\\ &\quad(\mathsf{natural\text{-}sq}\ \mathsf{inv\text{-}left}\ (\mathsf{glue}(x,y)))\\ &\quad(\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{ap}_{\mathsf{switch}}(\mathsf{switch\text{-}glue}(\mathsf{glue}(j,z))))\ (\mathsf{glue}(x,y)))\\ &\quad(\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{glue}(j,z))\ (\mathsf{glue}(x,y)))\\ &\quad\mathsf{srefl}\end{aligned}$$

Note that $\mathsf{natural\text{-}sq}\ \mathsf{inv\text{-}left}\ (\mathsf{glue}(x,y)) = \mathsf{inv\text{-}left\text{-}glue}(x,y)$. To take inventory of the situation, the cube involves three squares each dependent on a pair of variables: $\mathsf{inv\text{-}glue\text{-}left}(x,z)$, $\mathsf{inv\text{-}glue\text{-}right}(y,z)$, and $\mathsf{inv\text{-}left\text{-}glue}(x,y)$. These three serve to mediate the relationship between the top and bottom squares. Thus our goal is to transform the top square, which is in some way determined by the definition of $\mathsf{switch\text{-}glue\text{-}glue}(x,y,z)$, into the bottom square via some cube with the right variable dependencies, at which point we will have in hand the necessary definitions for the first three squares.

We begin reducing the top square with the following lemma:

Lemma 4.25. Let $f : A \to B$, $g : B \to C$, and $b : B$ be given, along with $h : \prod_{a:A} fa = b$. For any $x, y : A$ and $p : x = y$, there is a cube of type

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\mathsf{ap}_g \circ h)\ p)\ (\mathsf{ap\text{-}square}_g(\mathsf{app\text{-}cst\text{-}sq}\ h\ p))\\ &\qquad\mathsf{srefl\text{-}h}\ (\mathsf{hsquare}\ \mathsf{ap\text{-}}\circ\ g\ f\ p)\ \mathsf{srefl}\ \mathsf{srefl\text{-}h}\end{aligned}$$

where $\mathsf{ap\text{-}}\circ\ g\ f\ p : \mathsf{ap}_{g\circ f}p = \mathsf{ap}_g(\mathsf{ap}_f p)$ is defined for $p : x = y$ by $\mathsf{ap\text{-}}\circ\ g\ f\ \mathsf{refl} :\equiv \mathsf{refl}$.

Proof. By induction we may assume $p \equiv \mathsf{refl}$, so that $x \equiv y$. We are then obligated to give a cube $\mathsf{Cube}\ \mathsf{srefl\text{-}h}_{\mathsf{ap}_g(hx)}\ (\mathsf{ap\text{-}square}_g(\mathsf{srefl\text{-}h}_{hx}))\ \mathsf{srefl\text{-}h}_{\mathsf{ap}_g(hx)}\ \mathsf{srefl}\ \mathsf{srefl}\ \mathsf{srefl\text{-}h}_{\mathsf{ap}_g(hx)}$. If we generalize from hx to an arbitrary path and then induct on said path, we reach a case where crefl suffices. □

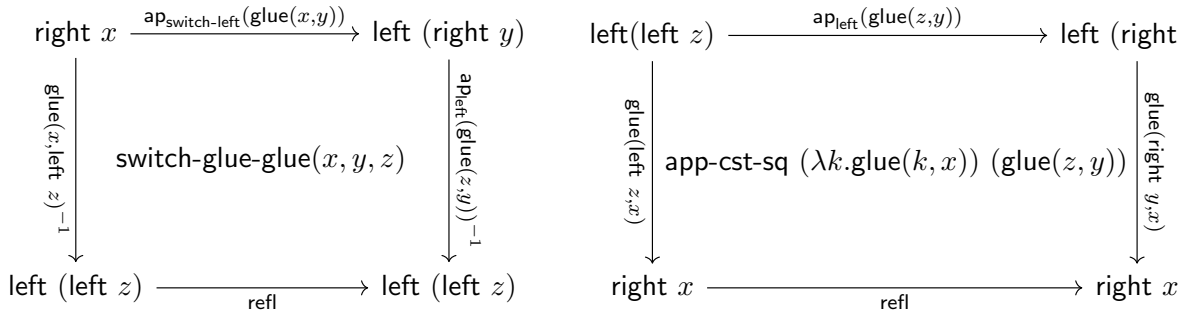
We are actually interested in a rotated form of this lemma, where $\mathsf{app\text{-}cst\text{-}sq}\ (\mathsf{ap}_g \circ h)\ p$ is the top face and $\mathsf{ap\text{-}square}_g(\mathsf{app\text{-}cst\text{-}sq}\ h\ p)$ is the bottom face; for readability, we stick to using left and right as primary faces henceforth. By instantiating with $g :\equiv \mathsf{switch}$, $h :\equiv \lambda j.\mathsf{switch\text{-}glue}(\mathsf{glue}(j, z))$, and $p :\equiv \mathsf{glue}(x, y)$, we obtain a cube of type

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{ap}_{\mathsf{switch}}(\mathsf{switch\text{-}glue}(\mathsf{glue}(j, z))))\ (\mathsf{glue}(x, y)))\\ &\quad(\mathsf{ap\text{-}square}_{\mathsf{switch}}(\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{switch\text{-}glue}(\mathsf{glue}(j, z)))\ (\mathsf{glue}(x, y))))\\ &\quad\mathsf{srefl\text{-}h}\ (\mathsf{hsquare}\ (\mathsf{ap}\text{-}\circ\ \mathsf{switch}\ \mathsf{switch\text{-}left}\ (\mathsf{glue}(x, y))))\ \mathsf{srefl}\ \mathsf{srefl\text{-}h}\end{aligned}$$

for the first step in our reduction. Assuming we define $\mathsf{switch\text{-}glue}$ such that

$$\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{switch\text{-}glue}(\mathsf{glue}(j, z)))\ (\mathsf{glue}(x, y)) = \mathsf{switch\text{-}glue\text{-}glue}(x, y, z)$$

we must choose a definition of $\mathsf{switch\text{-}glue\text{-}glue}$ to move forward. We would like it to reduce approximately to $\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{glue}(k, x))\ (\mathsf{glue}(z, y))$, modulo some involutive rearrangement, so that two applications compose to give $\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{glue}(j, z))\ (\mathsf{glue}(x, y))$. Compare the two square types:



Note that $\mathsf{ap}_{\mathsf{switch\text{-}left}}(\mathsf{glue}(x, y)) = \mathsf{glue}(\mathsf{right}\ y, x)^{-1}$. We see that the frame of $\mathsf{switch\text{-}glue\text{-}glue}(x, y, z)$, modulo the previous equality, is a rearrangement of the frame of $\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{glue}(k, x))\ (\mathsf{glue}(z, y))$. We define a function extending this rearrangement from types to terms.

Lemma 4.26. For $A : \mathcal{U}$, let points $a, b, c : A$ and paths $p : a = b$, $q : a = c$, $r : c = b$ be given. There is a map $\mathsf{massage} : \mathsf{Square}\ p\ q\ \mathsf{refl}\ r \to \mathsf{Square}\ p^{-1}\ r^{-1}\ \mathsf{refl}\ q^{-1}$.

Proof. By based square induction per Lemma 2.1, with the bottom (reflexive) face fixed. We define $\mathsf{massage}\ \mathsf{srefl} :\equiv \mathsf{srefl}$. □

This transformation has the following three properties, the first of which captures its "involutive" nature.

Lemma 4.27. For $A : \mathcal{U}$, points $a, b, c : A$, paths $p : a = b$, $q : a = c$, $r : c = b$, and square $s : \mathsf{Square}\ p\ q\ \mathsf{refl}\ r$, there is a cube of type

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{massage}\ (\mathsf{massage}\ s))\ s\\ &\quad((^{-1})\text{-}\mathsf{involutive}\ p)\ ((^{-1})\text{-}\mathsf{involutive}\ q)\ \mathsf{srefl}\ ((^{-1})\text{-}\mathsf{involutive}\ r)\end{aligned}$$

where $(^{-1})\text{-}\mathsf{involutive}\ t : t^{-1^{-1}} = t$ is defined by $(^{-1})\text{-}\mathsf{involutive}\ \mathsf{refl} :\equiv \mathsf{refl}$.

Proof. By based square induction on s. □

Lemma 4.28. Let $A, B : \mathcal{U}$ and $f : A \to B$ be given. For any points $a, b, c : A$, paths $p : a = b$, $q : a = c$, $r : c = b$, and square $s : \mathsf{Square}\ p\ q\ \mathsf{refl}\ r$, there is a cube of type

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{ap\text{-}square}_f(\mathsf{massage}\ s))\ (\mathsf{massage}\ (\mathsf{ap\text{-}square}_f s))\\ &\quad(\mathsf{hsquare}\ (\mathsf{ap}\text{-}(^{-1})\ f\ p))\ (\mathsf{hsquare}\ (\mathsf{ap}\text{-}(^{-1})\ f\ q))\ \mathsf{srefl}\ (\mathsf{hsquare}\ (\mathsf{ap}\text{-}(^{-1})\ f\ r))\end{aligned}$$

where $\mathsf{ap}\text{-}(^{-1})\ g\ t : \mathsf{ap}_g(t^{-1}) = (\mathsf{ap}_g t)^{-1}$ is defined by $\mathsf{ap}\text{-}(^{-1})\ g\ \mathsf{refl} :\equiv \mathsf{refl}$.

Proof. By based square induction on s. □

Lemma 4.29. Let a cube $c : \mathsf{Cube}\ s_{left}\ s_{right}\ s_{back}\ s_{top}\ \mathsf{srefl}\ s_{front}$ be given, where the types of the squares are constrained as necessary. Then there is a cube of type

$$\mathsf{Cube}\ (\mathsf{massage}\ s_{left})\ (\mathsf{massage}\ s_{right})\ ({s_{back}}^{-1^{\mathsf{v}}})\ ({s_{front}}^{-1^{\mathsf{v}}})\ \mathsf{srefl}\ ({s_{top}}^{-1^{\mathsf{v}}})$$

where $^{-1^{\mathsf{v}}} : \mathsf{Square}\ p_{0\text{-}}\ p_{\text{-}0}\ p_{\text{-}1}\ p_{1\text{-}} \to \mathsf{Square}\ ({p_{0\text{-}}}^{-1})\ p_{\text{-}1}\ p_{\text{-}0}\ ({p_{1\text{-}}}^{-1})$ is the vertical inverse for squares defined by $\mathsf{srefl}^{-1^{\mathsf{v}}} :\equiv \mathsf{srefl}$.

Proof. By based cube induction, with details left to the reader. □

We now define $\mathsf{switch\text{-}glue\text{-}glue}(x, y, z)$ to be the filler which satisfies the cube

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{switch\text{-}glue\text{-}glue}(x, y, z))\ (\mathsf{massage}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{glue}(k, x))\ (\mathsf{glue}(z, y))))\\ &\quad\mathsf{srefl\text{-}h}\ (\mathsf{hsquare}\ (u(x, y)))\ \mathsf{srefl}\ \mathsf{srefl\text{-}h}\end{aligned}$$

where $u(x, y)$ is the path $\mathsf{ap}_{\mathsf{switch\text{-}left}}(\mathsf{glue}(x, y)) = \mathsf{glue}(\mathsf{right}\ y, x)^{-1}$. We need one more lemma concerning $\mathsf{app\text{-}cst\text{-}sq}$ to complete the proof.

Lemma 4.30. Let $f : A \to B$ and $b : B$ be given along with $h_1, h_2 : \prod_{a:A} fa = b$ and a homotopy $\alpha : \prod_{a:A} h_1 a = h_2 a$. For any $a_1, a_2 : A$ and $p : a_1 = a_2$, there is a cube of type

$$\begin{aligned}&\mathsf{Cube}\ (\mathsf{app\text{-}cst\text{-}sq}\ h_1\ p)\ (\mathsf{app\text{-}cst\text{-}sq}\ h_2\ p)\\ &\quad(\mathsf{hsquare}\ (\alpha a_1))\ \mathsf{srefl\text{-}h}\ \mathsf{srefl\text{-}h}\ (\mathsf{hsquare}\ (\alpha a_2))\end{aligned}$$

Proof. First by induction on p, which leaves

$$\begin{aligned}&\mathsf{Cube}\ \mathsf{srefl\text{-}h}_{h_1 a_1}\ \mathsf{srefl\text{-}h}_{h_2 a_1}\\ &\qquad(\mathsf{hsquare}\ (\alpha a_1))\ \mathsf{srefl}\ \mathsf{srefl}\ (\mathsf{hsquare}\ (\alpha a_1))\end{aligned}$$

We may then generalize and induct on αa_1 and $h_1 a_1$, after which crefl serves. □

We therefore have a tower of cubes as follows, omitting all but the left and right faces:

$$\boxed{\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{ap}_{\mathsf{switch}}(\mathsf{switch\text{-}glue}(\mathsf{glue}(j,z))))\ (\mathsf{glue}(x,y))}$$

🧊 (Lemma 4.25)

$$\boxed{\mathsf{ap\text{-}square}_{\mathsf{switch}}(\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{switch\text{-}glue}(\mathsf{glue}(j,z)))\ (\mathsf{glue}(x,y)))}$$

= (definition of $\mathsf{switch\text{-}glue}$)

$$\boxed{\mathsf{ap\text{-}square}_{\mathsf{switch}}(\mathsf{switch\text{-}glue\text{-}glue}(x,y,z))}$$

🧊 (action of switch on the cube defining $\mathsf{switch\text{-}glue\text{-}glue}$)

$$\boxed{\mathsf{ap\text{-}square}_{\mathsf{switch}}(\mathsf{massage}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{glue}(k,x))\ (\mathsf{glue}(z,y))))}$$

🧊 (Lemma 4.28)

$$\boxed{\mathsf{massage}\ (\mathsf{ap\text{-}square}_{\mathsf{switch}}(\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{glue}(k,x))\ (\mathsf{glue}(z,y))))}$$

🧊 (Lemma 4.25, inverted)

$$\boxed{\mathsf{massage}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{ap}_{\mathsf{switch}}(\mathsf{glue}(k,x)))\ (\mathsf{glue}(z,y)))}$$

🧊 (Lemma 4.29 on Lemma 4.30 on definition of switch)

$$\boxed{\mathsf{massage}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\lambda k.\mathsf{switch\text{-}glue}(k,x))\ (\mathsf{glue}(z,y)))}$$

= (definition of $\mathsf{switch\text{-}glue}$)

$$\boxed{\mathsf{massage}\ (\mathsf{switch\text{-}glue\text{-}glue}(z,y,x))}$$

🧊 (Lemma 4.29 on the cube defining $\mathsf{switch\text{-}glue\text{-}glue}$)

$$\boxed{\mathsf{massage}\ (\mathsf{massage}\ (\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{glue}(j,z))\ (\mathsf{glue}(x,y))))}$$

🧊 (Lemma 4.27)

$$\boxed{\mathsf{app\text{-}cst\text{-}sq}\ (\lambda j.\mathsf{glue}(j,z))\ (\mathsf{glue}(x,y))}$$

In order for this to complete the proof, we must confirm that the back face is dependent only on (x,z), the top face only on (x,y), and the front face only on (y,z), with the bottom face being srefl. We leave this routine verification to the reader. Defining $\mathsf{inv\text{-}glue\text{-}left}(x,z)$, $\mathsf{inv\text{-}left\text{-}glue}(x,y)$, and $\mathsf{inv\text{-}glue\text{-}right}(y,z)$ appropriately and applying the necessary rotation to the completed cube gives the final result. □

Per the reasoning described at this section's introduction, we conclude with a theorem.

Theorem 4.31. For any $X : \mathcal{U}_*$ and $k : \mathbb{N}$, there is a pointed equivalence $\Sigma(S^k \times X) \simeq S^{k+1} \vee \Sigma X \vee \Sigma^{k+1} X$. As such, $H^n(X) \simeq H^{n-k}(S^0) \times H^n(X) \times H^{n-k}(X)$.

Corollary 4.32. If $X : \mathcal{U}_*$ is a finite product of spheres, then ΣX is homotopy equivalent to a finite wedge of spheres. There is thus a finite set of indices $m_1, \ldots, m_i : \mathbb{Z}$ such that, for any $n : \mathbb{Z}$,

$$H^n(X) \simeq \prod_{1 \leq j \leq i} H^{n+m_j}(S^0)$$

Proof. By iterating the previous theorem, coupled with the fact that $\Sigma(X \vee Y) \simeq \Sigma X \vee \Sigma Y$, which follows formally from the adjunction $\Sigma \dashv \Omega$ proved in Theorem 3.1. □

Chapter 5

Conclusions and Future Work

By now, we hope the reader has some understanding of the process of reasoning about higher inductive types in homotopy type theory. We can see that basic homotopy theory is possible and even accessible in HoTT, but there are practical limitations to its further development.

While cubical reasoning has proved valuable for simplifying previously infeasible proofs, standard homotopy type theory is *globular* by default, taking the concept of path rather than cube as primitive. While we were able to define 2- and 3-dimensional cube types within the theory, there is no known uniform definition of n-dimensional cube types. Moreover, it can be clumsy to prove results about cubes using only the standard eliminator which comes with an inductive definition. There is current work by Coquand et al. [4], Brunerie and Licata [3], and Altenkirch and Kaposi [1] on designing natively cubical type theories, which support various cubical operations as primitives and for which the induction principles are derived theorems. These have (or are expected to have) the additional benefit of computational interpretation, which, besides being a fundamental property of a type theory, could validate additional definitional equalities which only hold propositionally in standard HoTT. While many of these, such as the equality $\mathsf{ap}\text{-}\circ\ g\ f\ p : \mathsf{ap}_{g\circ f}p = \mathsf{ap}_g(\mathsf{ap}_f p)$ which we defined in the join associativity proof, can be glossed over in paper proofs, they introduce significant boilerplate to proof code, and in higher-dimensional proofs such as the aforementioned we may be forced to reason about them explicitly. It is still unclear which of subset of the several possible cubical operations is best suited for type theory, as well as which class of higher inductive types can be supported. Of course, no consensus on the nature of higher inductive types exists even for standard HoTT, and we may hope that the cubical work clarifies the issue in general.

The second issue, which we touched upon in our discussion of sequential colimits in §3.3, is the lack of internal notion of definitional or exact equality in homotopy type theory. This is an old gripe with intensional type theory: because the elimination rule for the identity type is transportational, rather than substitutional, proofs become bogged down with transport operations. With homotopy type theory we have seen a new benefit to this extra work, the ability to work with types with nontrivial path structure. When it comes to "discrete" types, however, such as the natural numbers, we are still left carrying through transport operations which are morally trivial. The HoTT community has long been aware of this issue, as evidenced by a mention in [8, p.12]. There has been work on systems [25, 20] which fix the issue by introducing an internal notion of exact, substitutional

equality. This work is motivated by a desire to give a uniform definition of *semi-simplicial types* in HoTT. Much like the cube types above, we can define n-truncated semi-simplicial types for any particular n, but defining them uniformly as a family parameterized by n seems to require infinite coherence conditions which are inexpressible in HoTT. Our use case, on the other hand, only involves finite but overwhelming transport requirements. We might hope to define an algorithm which, by inserting appropriate transport operations, elaborates proofs from a theory where a particular propositional identification is assumed to be exact into standard HoTT.

These questions aside, what directions are open to us for formalizing cohomology in HoTT? There is the classical theorem that all Eilenberg-Steenrod cohomology theories arise from spectra; although we cannot prove this generally in HoTT for lack of additivity, we could prove that it holds when restricted to a smaller class of spaces. This would be the class of spaces whose cohomology is determined by that of the spheres, and another goal could be to compute said cohomology. It could be practically useful to give an algorithm for computing the cohomology of types presented as higher inductives. An alternative direction is the construction of cohomological spectral sequences as suggested by Shulman [23]. Finally, classical cohomology theories depend not only on homotopy type but on differential, smooth, or some other structure. While standard HoTT is ignorant to this structure, *cohesive homotopy type theory* can capture it by postulating *axiomatic cohesion* operators in the style of Lawvere [12]. Schreiber and Shulman have used this system to define differential cohomology as part of work on quantum field theory [21].

Appendix A

Formalization Reference

The Agda library in which this work was formalized is available at `github.com/HoTT/HoTT-Agda`. This thesis is current to the library as of December 16, 2015. This index does not cover all dependencies, but should be sufficient for the reader to trace them out.

Chapter	Section	Relevant Modules
1	1	`lib.Basics`, `cohomology.FunctionOver`
	2	`lib.Pointed`, `cohomology.FunctionOver`, `lib.types.LoopSpace`
	3	`lib.types.Types`
	4	`lib.NType`, `lib.types.Truncation`
	5	`lib.types.Group`, `lib.groups.Groups`
2	1	`lib.cubical.Square`, `lib.cubical.SquareOver`
	2	`lib.cubical.Cube`
3	1	`cohomology.Theory`, `cohomology.Exactness`
	2	`cohomology.SpectrumModel`, `cohomology.SuspAdjointLoopIso`, `cohomology.Choice`
	3	`cohomology.EMModel`, `homotopy.EilenbergMacLane`, `lib.types.EilenbergMacLane1`, `lib.types.NatColim`, `homotopy.SpaceFromGroups`
4	0	`cohomology.BaseIndependence`
	1	`cohomology.CofiberSequence`, `cohomology.LongExactSequence`
	2	`cohomology.ProductRepr`, `cohomology.WedgeCofiber`, `cohomology.Wedge`
	3	`homotopy.elims.CofPushoutSection`
	4	`cohomology.Unit`, `cohomology.Functor`, `cohomology.SplitExactRight`, `homotopy.SuspSectionDecomp homotopy.CofiberComp`, `homotopy.SuspProduct`
	5	`cohomology.MayerVietoris`, `cohomology.MayerVietorisExact`
	6	`homotopy.elims.SuspSmash`, `homotopy.SuspSmash`, `homotopy.JoinAssocCubical`, `cohomology.SphereProduct`

Bibliography

[1] Thorsten Altenkirch and Ambrus Kaposi. A syntax for cubical type theory. Available from http://www.cs.nott.ac.uk/ txa/publ/ctt.pdf, 2014.

[2] Steve Awodey. *Category Theory*. Oxford Logic Guides, Oxford University Press, 2006.

[3] Guillaume Brunerie and Daniel R. Licata. Cubical infinite-dimensional type theory. Talk at Oxford Workshop on Homotopy Type Theory, November 2014. Available from https://youtube.com/user/OxfordQuantumVideo.

[4] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical. Programming language implementation. Available from https://github.com/simhu/cubical.

[5] Eldon Dyer and Joseph Roitberg. Note on sequences of mayer-vietoris type. *Proceedings of the American Mathematical Society*, 80(4):pp. 660–662, 1980.

[6] Samuel Eilenberg and Norman E. Steenrod. *Foundations of Algebraic Topology*. Princeton University Press, 1952.

[7] Eric Finster. Cohomology in homotopy type theory. Video, March 2013. Available from https://video.ias.edu/univalent/1213/0306-EricFinster.

[8] The Univalent Foundations Program; Institute for Advanced Study. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Available from homotopytypetheory.com/book, 2013. References current as of December 16, 2015.

[9] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[10] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.

[11] HoTT-agda. Agda library. Available from https://github.com/HoTT/HoTT-Agda.

[12] F. William Lawvere. Axiomatic cohesion. *Theory and Applications of Categories*, 19(3):41–49, 2007.

[13] Daniel R. Licata. hott-agda. Agda library. Available from https://github.com/dlicata335/hott-agda.

[14] Daniel R. Licata and Guillaume Brunerie. A cubical approach to synthetic homotopy theory, January 2015. Available from http://dlicata.web.wesleyan.edu/pubs/lb15cubicalsynth/lb15cubicalsynth.pdf.

[15] Daniel R. Licata and Eric Finster. Eilenberg-MacLane spaces in homotopy type theory. *IEEE Symposium on Logic in Computer Science*, 2013.

[16] Peter LeFanu Lumsdaine. Weak ω-categories from intensional type theory. In Pierre-Louis Curien, editor, *Typed Lambda Calculi and Applications*, volume 5608 of *Lecture Notes in Computer Science*, pages 172–187. Springer Berlin Heidelberg, 2009.

[17] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

[18] Per Martin-Löf. Constructive mathematics and computer programming. In L. Jonathan Cohen, Jerzy o, Helmut Pfeiffer, and Klaus-Peter Podewski, editors, *Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. North-Holland, 1982.

[19] J.P. May. *A Concise Course in Algebraic Topology*. University of Chicago Press, 1999.

[20] Fedor Part and Zhaohui Luo. Semi-simplicial types in logic-enriched homotopy type theory. *CoRR*, abs/1506.04998, 2015.

[21] Urs Schreiber and Michael Shulman. Quantum gauge field theory in cohesive homotopy type theory, July 2014. arXiv:1408.0054.

[22] Michael Shulman. Cohomology, July 2013. Available from http://homotopytypetheory.org/2013/07/24/cohomology/.

[23] Michael Shulman. Spectral sequences, August 2013. Available from http://homotopytypetheory.org/2013/08/08/spectral-sequences.

[24] Benno van den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011.

[25] Vladimir Voevodsky. A simple type system with two identity types, 2013. Available from http://ncatlab.org/homotopytypetheory/files/HTS.pdf.