Exactness of the Mayer-Vietoris Sequence in Homotopy Type Theory

e cavallo * (ecavallo@andrew.cmu.edu)

with Robert Harper, Daniel R. Licata, Carlo Angiuli, Ed Morehouse

1 Introduction

An *Eilenberg-Steenrod cohomology theory* consists of a family of contravariant functors $(C^n)_{n:\mathbb{Z}}$ from pointed types to abelian groups which satisfies the *Eilenberg-Steenrod axioms*.[1] Cohomology groups provide a means of classifying types which is coarser but simpler to compute than homotopy groups.

Given a span $X \overset{f}{\leftarrow} Z \overset{g}{\rightarrow} Y$ (of pointed types and basepoint-preserving maps), the Mayer-Vietoris sequence is a particular infinite sequence of maps

$$\cdots \longrightarrow C^n(\Sigma Z) \longrightarrow C^n(X \sqcup_Z Y) \longrightarrow C^n(X \vee Y) \longrightarrow C^{n+1}(\Sigma Z) \longrightarrow \cdots$$

(see Appendix A for definitions of Σ, $\sqcup$, and $\vee$ in homotopy type theory) which is *exact*: the kernel of each map is the image of the previous map. Since the Eilenberg-Steenrod axioms prescribe that $C^n(\Sigma Z) = C^{n-1}(Z)$ and $C^n(X \vee Y) = C^n(X) \times C^n(Y)$, this gives us a means, at least in some cases, of decomposing a pushout's cohomology in terms of its components' cohomology.

For our purposes, we will be interested in the exactness of a sequence

$$C^n(\Sigma Z) \overset{C^n(\mathsf{extract\text{-}glue})}{\longrightarrow} C^n(X \sqcup_Z Y) \overset{C^n(\mathsf{reglue})}{\longrightarrow} C^n(X \vee Y) \tag{1}$$

The maps $\mathsf{extract\text{-}glue}$ and reglue are defined recursively by

$\mathsf{extract\text{-}glue} : X \sqcup_Z Y \to \Sigma Z$
$\mathsf{extract\text{-}glue}\ (\mathsf{left}\ x) = \mathsf{north}$
$\mathsf{extract\text{-}glue}\ (\mathsf{right}\ y) = \mathsf{south}$

*This research was sponsored in part by the National Science Foundation under grant numbers CCF-1116703 and CCF-1445995 (REU). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

[1] See `http://homotopytypetheory.org/2013/07/24/cohomology/` for a formulation and exposition of these axioms in homotopy type theory. A specification of the axioms in Agda is available at `https://github.com/HoTT/HoTT-Agda/blob/master/cohomology/Theory.agda`.

$\mathsf{ap\ extract\text{-}glue}\ (\mathsf{glue}\ z) = \mathsf{merid}\ z$

$\mathsf{reglue} : X \vee Y \to X \sqcup_Z Y$
$\mathsf{reglue}\ (\mathsf{winl}\ x) = \mathsf{left}\ x$
$\mathsf{reglue}\ (\mathsf{winr}\ y) = \mathsf{right}\ y$
$\mathsf{ap\ reglue\ wglue} = \mathsf{ap\ left}\ {p_f}^{-1} \cdot \mathsf{glue}\ z_0 \cdot' \mathsf{ap\ right}\ p_g$

where $p_f : fz_0 = x_0$ and $p_g : gz_0 = y_0$ are the proofs that f and g preserve basepoint. (Here $\cdot$ is composition defined by induction on the left argument, and $\cdot'$ composition by induction on the right. This particular choice is not necessary, but it is convenient.)

The exactness of this sequence can be proven by way of the *Exactness axiom*:

Axiom 1 (Exactness). *Let pointed types X, Y and a (basepoint-preserving) map $f : X \to Y$ be given. Then the sequence $C^n(\mathsf{Cof}(f)) \overset{C^n(\mathsf{cfcod})}{\longrightarrow} C^n(Y) \overset{C^n(f)}{\to} C^n(X)$ is exact.*

($\mathsf{Cof}(f)$ is the cofiber space of f; see Appendix A.) By defining a path $\mathsf{mv\text{-}path} : \mathsf{Cof}(\mathsf{reglue}) = \Sigma Z$ and proving $\mathsf{cfcod} =^{V.\ X \sqcup_Z Y \to V}_{\mathsf{mv\text{-}path}} \mathsf{extract\text{-}glue}$, we can take the assertion that the sequence

$$C^n(\mathsf{Cof}(\mathsf{reglue})) \overset{C^n(\mathsf{cfcod})}{\longrightarrow} C^n(X \sqcup_Z Y) \overset{C^n(\mathsf{reglue})}{\longrightarrow} C^n(X \vee Y)$$

is exact and transport it along these equalities to prove that (1) is exact. In fact, a proof by this method can be extended to prove the exactness of the long Mayer-Vietoris sequence.[2] The existence of two such paths is therefore our main theorem.

A fully formalized proof, including those aspects which are omitted here, is available in the HoTT-Agda library at `https://github.com/HoTT/HoTT-Agda/blob/master/cohomology/MayerVietoris.agda`.

2 Proof

In what follows, we assume that the basepoint-preservation proofs for f and g are idp, and therefore that $x_0 \equiv f(z_0)$ and $y_0 \equiv g(z_0)$; this assumption can be justified by path induction. In this case, we have $\mathsf{ap\ reglue\ wglue} = \mathsf{glue}\ z_0$.

As an intuitive argument, consider the following picture:

[2] One must also consider the action of $\mathsf{mv\text{-}path}$ on the map $C^n(X \vee Y) \to C^{n+1}(\Sigma Z)$ – we omit this, since the proofs involved are straightforward given the ideas developed here.

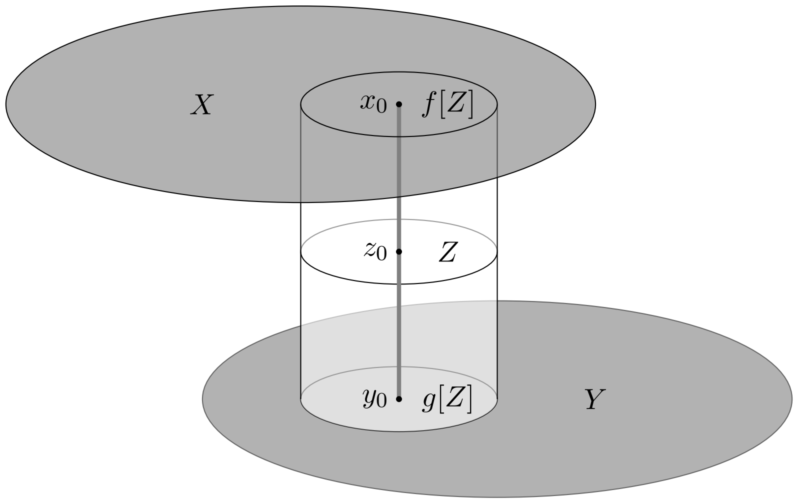


Figure 1: Image of reglue in $X \sqcup_Z Y$

We view the pushout $X \sqcup_Z Y$ as consisting of X and Y with glue connecting $f[Z]$ to $g[Z]$. The image of $X \vee Y$ by reglue in this space is shaded gray; it consists of X, Y, and a single path (the image of wglue) connecting x_0 to y_0. To obtain the cofiber space, $\mathsf{Cof}(\mathsf{reglue})$, we in essence contract the shaded subset to a point. Equivalently, we could merely contract X and Y to points, leaving the image of wglue (which is already equivalent to a point) intact. This leaves us with two endpoints and a family of paths indexed by Z between them, which is precisely the structure of ΣZ.

We prove the theorem by an equivalence $\mathsf{Cof}(\mathsf{reglue}) \simeq \Sigma Z$, following the high-level description above.

Equivalence Maps

First, we define a function $\mathsf{into} : \mathsf{Cof}(\mathsf{reglue}) \to \Sigma Z$ by recursion on the cofiber type. The point cases are simple:

$$
\begin{aligned}
&\mathsf{into} : \mathsf{Cof}(\mathsf{reglue}) \to \Sigma Z \\
&\mathsf{into}\ \mathsf{cfbase} = \mathsf{north} \\
&\mathsf{into}\ (\mathsf{cfcod}\ u) = \mathsf{extract\text{-}glue}\ u
\end{aligned}
$$

We now need a proof $\mathsf{into\text{-}glue} : \Pi w{:}X \vee Y.\ \mathsf{north} = \mathsf{extract\text{-}glue}\ (\mathsf{reglue}\ w)$, showing that $\mathsf{extract\text{-}glue}$ maps any point in the image of reglue to north. For this we go by induction on ΣZ. For the point cases, we can define

$$
\begin{aligned}
&\mathsf{into\text{-}glue} : \Pi w{:}X \vee Y.\ \mathsf{north} = \mathsf{extract\text{-}glue}\ (\mathsf{reglue}\ w) \\
&\mathsf{into\text{-}glue}\ (\mathsf{winl}\ x) = \mathsf{idp} \\
&\mathsf{into\text{-}glue}\ (\mathsf{winr}\ y) = \mathsf{merid}\ z_0
\end{aligned}
$$

For the coherence, we need a dependent path of type

$$\mathsf{idp} =^{w.\mathsf{north} = \mathsf{extract\text{-}glue}\ (\mathsf{reglue}\ w)}_{\mathsf{wglue}} \mathsf{merid}\ z_0$$

This is equivalent to proving the following square (for an overview of squares, see Appendix B):

$$\begin{tikzcd}
\mathsf{north} \arrow[r, "\mathsf{ap}\ (\lambda\ _ \to \mathsf{north})\ \mathsf{wglue}"] \arrow[d, "\mathsf{idp}"'] & \mathsf{north} \arrow[d, "\mathsf{merid}\ z_0"] \\
\mathsf{north} \arrow[r, "\mathsf{ap}\ (\mathsf{extract\text{-}glue} \circ \mathsf{reglue})\ \mathsf{wglue}"'] & \mathsf{south}
\end{tikzcd}$$

We observe that $\mathsf{ap}\ (\lambda\ \_ \to \mathsf{north})\ \mathsf{wglue} = \mathsf{idp}$ and

$$\begin{aligned}
\mathsf{ap}\ (\mathsf{extract\text{-}glue} \circ \mathsf{reglue})\ \mathsf{wglue} &= \mathsf{ap}\ \mathsf{extract\text{-}glue}\ (\mathsf{ap}\ \mathsf{reglue}\ \mathsf{wglue}) \\
&= \mathsf{ap}\ \mathsf{extract\text{-}glue}\ (\mathsf{glue}\ z_0) \\
&= \mathsf{merid}\ z_0
\end{aligned}$$

Applying these equalities, we are left to prove the square

$$\begin{tikzcd}
\mathsf{north} \arrow[r, "\mathsf{idp}"] \arrow[d, "\mathsf{idp}"'] & \mathsf{north} \arrow[d, "\mathsf{merid}\ z_0"] \\
\mathsf{north} \arrow[r, "\mathsf{merid}\ z_0"'] & \mathsf{south}
\end{tikzcd}$$

Since $\mathsf{connection} : \mathsf{Square}\ \mathsf{idp}\ \mathsf{idp}\ q\ q$ is provable for any path q, we are done.

This completes our definition of the function into. We now define its inverse $\mathsf{out} : \Sigma Z \to \mathsf{Cof}(\mathsf{reglue})$. Ideally, $\mathsf{ap}\,\mathsf{out}\ (\mathsf{merid}\ z)$ should reduce to something like $\mathsf{ap}\,\mathsf{cfcod}\ (\mathsf{glue}\ z)$. But while $\mathsf{ap}\,\mathsf{cfcod}\ (\mathsf{glue}\ z)$ has type $\mathsf{cfcod}\ (\mathsf{left}\ (fz)) = \mathsf{cfcod}\ (\mathsf{right}\ (gz))$, the endpoints of $\mathsf{ap}\ \mathsf{out}\ (\mathsf{merid}\ z)$ are $\mathsf{out}\ \mathsf{north}$ and $\mathsf{out}\ \mathsf{south}$ and must be independent of z. We can correct for this using the paths

$$\begin{aligned}
&\mathsf{cfglue}\ (\mathsf{winl}\ (fz)) : \mathsf{cfbase} = \mathsf{cfcod}\ (\mathsf{reglue}\ (\mathsf{winl}\ (fz))) \equiv \mathsf{cfcod}\ (\mathsf{left}\ (fz)) \\
&\mathsf{cfglue}\ (\mathsf{winr}\ (gz)) : \mathsf{cfbase} = \mathsf{cfcod}\ (\mathsf{reglue}\ (\mathsf{winr}\ (gz))) \equiv\ \mathsf{cfcod}\ (\mathsf{right}(gz))
\end{aligned}$$

Thus we define the point cases of out as

$\mathsf{out} : \Sigma Z \to \mathsf{Cof}(\mathsf{reglue})$
$\mathsf{out}\ \mathsf{north} = \mathsf{cfbase}$
$\mathsf{out}\ \mathsf{south} = \mathsf{cfbase}$

We now need paths $\mathsf{out\text{-}glue} : Z \to \mathsf{cfbase} = \mathsf{cfbase}$; the above argument suggests we use

$$\mathsf{out\text{-}glue}\ z = \mathsf{cfglue}(\mathsf{winl}(fz)) \cdot \mathsf{ap}\ \mathsf{cfcod}\ (\mathsf{glue}\ z) \cdot \mathsf{cfglue}(\mathsf{winr}(gz))^{-1}$$

We will instead use another, propositionally equal path which is more convenient for our use: we define $\mathsf{out\text{-}glue}\ z$ by filling the following box, obtaining $\mathsf{out\text{-}square}\ z$ in the process:

$$\begin{tikzcd}
\mathsf{cfbase} \arrow[r, dashed, "\mathsf{out\text{-}glue}\ z"] \arrow[d, "\mathsf{cfglue}(\mathsf{winl}(fz))"'] \arrow[dr, phantom, "\mathsf{out\text{-}square}\ z"] & \mathsf{cfbase} \arrow[d, "\mathsf{cfglue}(\mathsf{winr}(gz))"] \\
\mathsf{cfcod}(\mathsf{left}(fz)) \arrow[r, "\mathsf{ap}\ \mathsf{cfcod}\ (\mathsf{glue}\ z)"'] & \mathsf{cfcod}(\mathsf{right}(gz))
\end{tikzcd}$$

This completes our definition of the equivalence maps. We now show these maps are mutually inverse.

Right Inverse

We first show out is a right inverse, that is, that $\mathsf{into}\ (\mathsf{out}\ \sigma) = \sigma$ for every $\sigma : \Sigma Z$; this is the simpler of the two proofs.

The proof is by suspension induction. At the endpoints we define:

$\mathsf{into\text{-}out} : \Pi \sigma{:}\Sigma Z.\ \mathsf{into}\ (\mathsf{out}\ \sigma) = \sigma$
$\mathsf{into\text{-}out}\ \mathsf{north} = \mathsf{idp}$
$\mathsf{into\text{-}out}\ \mathsf{south} = \mathsf{merid}\ z_0$

For the coherence, we need for every $z : Z$ a dependent path

$$\mathsf{idp} =^{\sigma.\mathsf{into}\ (\mathsf{out}\ \sigma) = \sigma}_{\mathsf{merid}\ z} \mathsf{merid}\ z_0$$

or equivalently, a square

$$\begin{tikzcd}
\mathsf{north} \arrow[r, "\mathsf{ap}\ (\mathsf{into} \circ \mathsf{out})\ (\mathsf{merid}\ z)"] \arrow[d, "\mathsf{idp}"'] & \mathsf{north} \arrow[d, "\mathsf{merid}\ z_0"] \\
\mathsf{north} \arrow[r, "\mathsf{ap}\ \mathsf{id}_{\Sigma Z}\ (\mathsf{merid}\ z)"'] & \mathsf{south}
\end{tikzcd}$$

Note that $\mathsf{ap}\ (\mathsf{into} \circ \mathsf{out})\ (\mathsf{merid}\ z) = \mathsf{ap}\ \mathsf{into}\ (\mathsf{ap}\ \mathsf{out}\ (\mathsf{merid}\ z)) = \mathsf{ap}\ \mathsf{into}\ (\mathsf{out\text{-}glue}\ z)$. By applying into to $\mathsf{out\text{-}square}$ (an operation analagous to $\mathsf{ap}\ \mathsf{into}$), we can obtain

$$\begin{tikzcd}
\mathsf{north} \arrow[r, "\mathsf{ap\ into\ (out\text{-}glue}\ z)"] \arrow[d, "\mathsf{ap\ into\ (cfglue(winl}(fz)))"'] \arrow[dr, phantom, "\mathsf{ap\text{-}square\ into\ (out\text{-}square}\ z)"] & \mathsf{north} \arrow[d, "\mathsf{ap\ into\ (cfglue(winr}(gz)))"] \\
\mathsf{north} \arrow[r, "\mathsf{ap\ into\ (ap\ cfcod\ (glue}\ z))"'] & \mathsf{south}
\end{tikzcd}$$

By definition of into, we know $\mathsf{ap\ into\ (cfglue(winl}(fz))) = \mathsf{idp}$ and $\mathsf{ap\ into\ (cfglue(winr}(gz))) = \mathsf{merid}\ z_0$. Finally, $\mathsf{ap\ into\ (ap\ cfcod\ (glue}\ z)) = \mathsf{ap\ (into} \circ \mathsf{cfcod)\ (glue}\ z)) \equiv \mathsf{ap\ extract\text{-}glue\ (glue}\ z) = \mathsf{merid}\ z = \mathsf{ap\ id}_{\Sigma Z}\ (\mathsf{merid}\ z)$. Thus we can construct the square we need from $\mathsf{ap\text{-}square\ into\ (out\text{-}square}\ z)$.

Left Inverse

We now show that out is a left inverse. We have $\mathsf{out\ (into\ cfbase)} \equiv \mathsf{out\ north} \equiv \mathsf{cfbase}$ definitionally. For the codomain, we go by induction on $X \sqcup_Z Y$. We can prove the point cases as follows:

$\mathsf{out\text{-}into\text{-}cod} : \Pi\gamma{:}X \sqcup_Z Y.\ \mathsf{out\ (into\ (cfcod}\ \gamma)) = \mathsf{cfcod}\ \gamma$
$\mathsf{out\text{-}into\text{-}cod\ (left}\ x) = \mathsf{cfglue\ (winl}\ x)$
$\mathsf{out\text{-}into\text{-}cod\ (right}\ y) = \mathsf{cfglue\ (winr}\ y)$

To complete this definition, we need for every $z : Z$ a dependent path

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

. We prove this via a square

$$\begin{tikzcd}
\mathsf{cfbase} \arrow[r, "\mathsf{ap\ (out} \circ \mathsf{into} \circ \mathsf{cfcod)\ (glue}\ z)"] \arrow[d, "\mathsf{cfglue(winl}(fz))"'] \arrow[dr, phantom, "\mathsf{out\text{-}into\text{-}cod\text{-}square}\ z"] & \mathsf{cfbase} \arrow[d, "\mathsf{cfglue(winr}(gz))"] \\
\mathsf{cfcod(left}(fz)) \arrow[r, "\mathsf{ap\ cfcod\ (glue}\ z)"'] & \mathsf{cfcod(right}(gz))
\end{tikzcd}$$

which we can build starting from $\mathsf{out\text{-}square}\ z$, since

$$\mathsf{ap\ (out} \circ \mathsf{into} \circ \mathsf{cfcod)\ (glue}\ z) = \mathsf{ap\ out\ (ap\ extract\text{-}glue\ (glue}\ z)) = \mathsf{ap\ out\ (merid}\ z) = \mathsf{out\text{-}glue}\ z$$

We have now given proofs that out is a left inverse in the cfbase and cfcod cases. To finish the induction, we need to give for every $w : X \vee Y$ a dependent path

$$\mathsf{idp} =^{\kappa.\mathsf{out(into}\ \kappa) = \kappa}_{\mathsf{cfglue}\ w} \mathsf{out\text{-}into\text{-}cod\ (reglue}\ w)$$

which we will do by giving a square

$$\begin{tikzcd}
\mathsf{cfbase} \arrow[r, "\mathsf{ap}\ (\mathsf{out} \circ \mathsf{into})\ (\mathsf{reglue}\ w)"] \arrow[d, "\mathsf{idp}"'] & \mathsf{out}\ (\mathsf{into}\ (\mathsf{cfcod}\ (\mathsf{reglue}\ w))) \arrow[d, "\mathsf{out\text{-}into\text{-}cod}\ (\mathsf{reglue}\ w)"] \\
\mathsf{cfbase} \arrow[r, "\mathsf{ap}\ \mathsf{id}_{X \vee Y}\ (\mathsf{cfglue}\ w)"'] & \mathsf{cfcod}\ (\mathsf{reglue}\ w)
\end{tikzcd}$$

The top path is equal to $\mathsf{ap}\ \mathsf{out}\ (\mathsf{into\text{-}glue}\ w)$, and the bottom path to $\mathsf{cfglue}\ w$. From here we go by induction on $w : X \vee Y$. In the case $w \equiv \mathsf{winl}\ x$, the square simplifies to

$$\begin{tikzcd}
\mathsf{cfbase} \arrow[r, "\mathsf{idp}"] \arrow[d, "\mathsf{idp}"'] \arrow[dr, phantom, "\mathsf{out\text{-}into\text{-}sql}\ x"] & \mathsf{cfbase} \arrow[d, "\mathsf{cfglue}\ (\mathsf{winl}\ x)"] \\
\mathsf{cfbase} \arrow[r, "\mathsf{cfglue}\ (\mathsf{winl}\ x)"'] & \mathsf{cfcod}\ (\mathsf{left}\ x)
\end{tikzcd}$$

which we can prove with $\mathsf{connection}$. In the case $w \equiv \mathsf{winr}\ y$, the square simplifies to

$$\begin{tikzcd}
\mathsf{cfbase} \arrow[r, "\mathsf{ap}\ \mathsf{out}\ (\mathsf{merid}\ z_0)"] \arrow[d, "\mathsf{idp}"'] \arrow[dr, phantom, "\mathsf{out\text{-}into\text{-}sqr}\ x"] & \mathsf{cfbase} \arrow[d, "\mathsf{cfglue}\ (\mathsf{winr}\ y)"] \\
\mathsf{cfbase} \arrow[r, "\mathsf{cfglue}\ (\mathsf{winr}\ y)"'] & \mathsf{cfcod}\ (\mathsf{right}\ y)
\end{tikzcd}$$

We will prove this square by concatenating two squares, one of which we leave unspecified[3] for the moment:

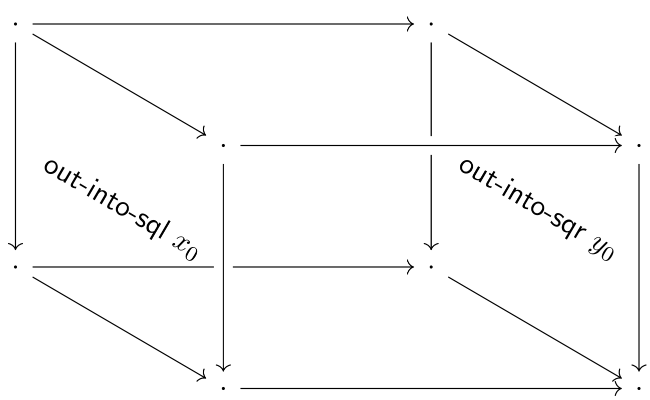
[3] Our proof of this square is somewhat indirect. To see why there ought to be a square with this type – equivalently, to see why $\mathsf{ap}\ \mathsf{out}\ (\mathsf{merid}\ z_0) = \mathsf{idp}$ – recall the definition of $\mathsf{out\text{-}glue}\ z_0$ as a filler. The square $\mathsf{natural\text{-}square}\ \mathsf{cfglue}\ \mathsf{wglue}$'s type (see Appendix B) has the same left, bottom, and right edges as $\mathsf{out\text{-}square}\ z_0$; since square fillers are unique, it follows that $\mathsf{out\text{-}glue}\ z_0$ is equal to the top edge in $\mathsf{natural\text{-}square}\ \mathsf{cfglue}\ \mathsf{wglue}$'s type, which is $\mathsf{ap}\ (\lambda\ \_ \to \mathsf{north})\ \mathsf{wglue}$. Thus $\mathsf{ap}\ \mathsf{out}\ (\mathsf{merid}\ z_0) = \mathsf{out\text{-}glue}\ z_0 = \mathsf{ap}\ (\lambda\ \_ \to \mathsf{north})\ \mathsf{wglue} = \mathsf{idp}$.

$$
\begin{tikzcd}
\mathsf{cfbase} \arrow[r, "\mathsf{ap\ out\ (merid}\ z_0)"] \arrow[d, "\mathsf{idp}"'] \arrow[dr, phantom, "?"] & \mathsf{cfbase} \arrow[d, "\mathsf{idp}"] \\
\mathsf{cfbase} \arrow[r, "\mathsf{idp}"] \arrow[d, "\mathsf{idp}"'] \arrow[dr, phantom, "\mathsf{connection}"] & \mathsf{cfbase} \arrow[d, "\mathsf{cfglue\ (winr}\ y)"] \\
\mathsf{cfbase} \arrow[r, "\mathsf{cfglue\ (winr}\ y)"'] & \mathsf{cfcod\ (left}\ x)
\end{tikzcd}
$$

Note that the type of the missing square is independent of y. In order to determine what this square should be, we consider the coherence condition. We need to prove a dependent path

$$\mathsf{out\text{-}into\text{-}sql}\ x_0 =^{w.\mathsf{Square\ idp\ (ap\ out\ (into\text{-}glue}\ w))\ (\mathsf{cfglue}\ w)\ (\mathsf{out\text{-}into\text{-}cod\ (reglue}\ w))}_{\mathsf{wglue}} \mathsf{out\text{-}into\text{-}sqr}\ y_0$$

We can prove a dependent path in a square fibration by giving a cube, in this case of the form



For our purposes, we only need to know the definitions of the left and right squares; the precise form is given in Appendix B, but we will not need it here.

Given five faces of a cube, we can find a cube filler, a sixth face such that the six together form a cube. We are almost in a position to use this fact: we have one missing square, the ? in our definition of out-into-sqr, which we want to choose so that we can prove the cube above. However, the ? is not a face of the cube, but only a part of a face. In order to fix this, we will shift the connection piece of out-into-sqr onto the bottom face of the cube, leaving only the ? on the right. We accomplish this with the following lemma:

Lemma 1. *Define the function*

$\mathsf{push} : \forall q \to \mathsf{Square}\ p_{0-}\ p_{-0}\ p_{-1}\ (p_{1-} \cdot q) \to \mathsf{Square}\ p_{0-}\ p_{-0}\ (p_{-1} \cdot' q^{-1})\ p_{1-}$
$\mathsf{push}\ \{p_{1-} = \mathsf{idp}\}\ \mathsf{idp}\ sq = sq$

In order to construct a cube of type $\mathsf{Cube}\ sq_{--0}\ (sq_{--1} \cdot^{v} sq')\ sq_{0--}\ sq_{-0-}\ sq_{-1-}\ sq_{1--}$*, where* sq' *has type* $\mathsf{Square}\ q_{0-}\ q_{-0}\ q_{-1}\ q_{1-}$*, it suffices to construct a cube of type* $\mathsf{Cube}\ sq_{--0}\ sq_{--1}\ (\mathsf{push}\ q_{0-}\ sq_{0--})\ sq_{-0-}\ (sq_{-1-} \cdot'^{h} (\mathsf{sym}\ sq')^{-1h})\ (\mathsf{push}\ q_{0-}\ sq_{1--})$*.*

(Here, $\cdot^{v}$ *is vertical composition of squares by induction on the first argument,* $\cdot'^{h}$ *is horizontal composition by induction on the second argument,* $^{-1h}$ *is horizontal inversion, and* sym *is the natural mapping from* $\mathsf{Square}\ p\ q\ r\ s$ *to* $\mathsf{Square}\ q\ p\ s\ r$*.)*

Proof. By square induction, we can assume $sq' \equiv \mathsf{ids}$. Generalizing slightly, we will instead assume $sq' \equiv \mathsf{vid}\ q_{-0}$ (where $\mathsf{vid}\ p : \mathsf{Square}\ \mathsf{idp}\ p\ p\ \mathsf{idp}$ is defined inductively by $\mathsf{vid}\ \mathsf{idp} = \mathsf{ids}$). This makes it possible to do a second induction on the square sq_{--1}, the upper square on the right face. In this case we observe that:

- $q_{-0} \equiv \mathsf{idp}$, so sq' reduces back to ids,
- $sq_{--1} \cdot^{v} sq' \equiv \mathsf{ids} \cdot^{v} \mathsf{ids} \equiv \mathsf{ids} \equiv sq_{--1}$,
- the right edges of the squares sq_{0--} and sq_{1--} (where they connect to sq_{--1}) are idp, which means that $\mathsf{push}\ q_{0-}\ sq_{0--} \equiv \mathsf{push}\ \mathsf{idp}\ sq_{0--} \equiv sq_{0--}$ and likewise $\mathsf{push}\ q_{1-}\ sq_{1--} \equiv sq_{1--}$,
- $sq_{-1-} \cdot'^{h} (\mathsf{sym}\ sq')^{-1h} \equiv sq_{-1-} \cdot'^{h} \mathsf{ids} \equiv sq_{-1-}$.

Thus, in this case both cubes have type $\mathsf{Cube}\ sq_{--0}\ sq_{--1}\ sq_{0--}\ sq_{-0-}\ sq_{-1-}\ sq_{1--}$, so that we can trivially construct one from the other. □

Now we can construct the cube we need: to get a cube of type

$$\mathsf{Cube}\ (\mathsf{out\text{-}into\text{-}sql}\ x_0)\ (\mathsf{out\text{-}into\text{-}sqr}\ y_0)\ (\ldots)\ (\ldots)\ (\ldots)\ (\ldots)$$

that is, of type

$$\mathsf{Cube}\ \mathsf{connection}\ (?\cdot^{v} \mathsf{connection})\ (\ldots)\ (\ldots)\ (\ldots)\ (\ldots)$$

we first choose ? to be the filler such that there is a cube of type

$$\mathsf{Cube}\ \mathsf{connection}\ ?\ (\mathsf{push}\ m\ (\ldots))\ (\ldots)\ (\ldots \cdot'^{h} \mathsf{sym}\ \mathsf{connection})\ (\mathsf{push}\ m\ (\ldots))$$

We can then use the lemma above to convert that cube into the form we need. This completes the definition of $\mathsf{out\text{-}into}$, and thus the definition of the equivalence $\mathsf{Cof}(\mathsf{reglue}) \simeq \Sigma Z$.

Properties of the Equivalence

The equivalence additionally shows that $\mathsf{Cof}(\mathsf{reglue})$ and ΣZ are equal as pointed types, since the equivalence is pointed: into sends the basepoint of $\mathsf{Cof}(\mathsf{reglue})$ (which is cfbase) to the basepoint of ΣZ (north) by definition.

As for the effect of transporting cfcod along the equivalence, one can prove that for any function $f : A \to B$ and equivalence $e : B \simeq C$ that $f =^{D.A \to D}_{\mathsf{ua}\ e} e \circ f$. Writing $\mathsf{mv\text{-}equiv} : \mathsf{Cof}(\mathsf{reglue}) \simeq \Sigma Z$ for the equivalence, we thus have

$$\mathsf{cfcod} =^{V.X \sqcup_Z Y \to V}_{\mathsf{ua}\ \mathsf{mv\text{-}equiv}} \mathsf{into} \circ \mathsf{cfcod} \equiv \mathsf{extract\text{-}glue}$$

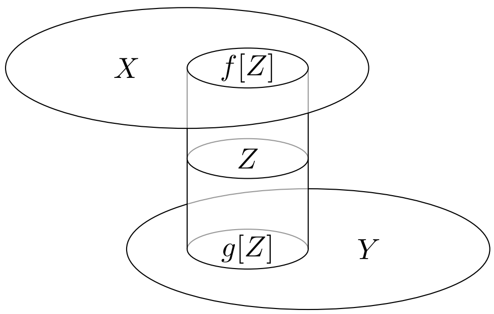
as desired. Furthermore, cfcod and $\mathsf{extract\text{-}glue}$ correspond as basepoint-preserving functions given the natural (and propositionally only) choice of basepoint-preservation proofs; we will not discuss this here, but the definitions and proofs are available in the Agda library.

A Types Involved

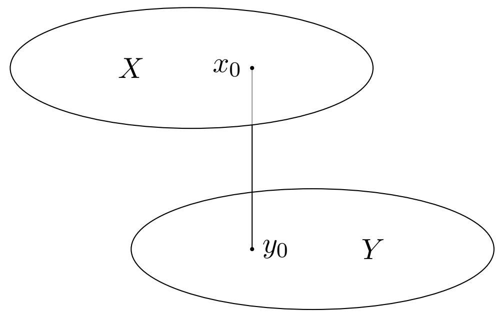
The main higher inductive type used here is the pushout of a span $X \xleftarrow{f} Z \xrightarrow{g} Y$, which is generated by the following constructors:

data $X \sqcup_Z Y$: Type where
  left : $X \to X \sqcup_Z Y$
  right : $Y \to X \sqcup_Z Y$
  glue : $\Pi z{:}Z.$ left $(f z) =$ right $(g z)$

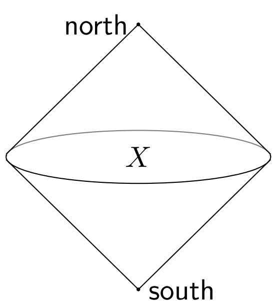
We assume that the computation rules for functions defined by higher induction hold definitionally for point cases and propositional for higher cases. The other higher inductive types we use are all special cases of the pushout:



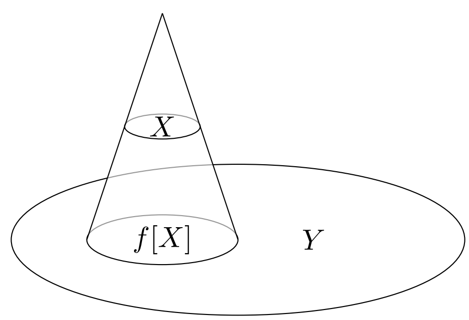
(a) The pushout type.



(b) The wedge type $X \vee Y$ is the pushout of the span $X \leftarrow \cdot \to Y$, with constructors written as winl : $X \to X \vee Y$, winr : $Y \to X \vee Y$, and wglue : winl $x_0 =$ winr y_0.



(c) The suspension type ΣX is the pushout of the span $\cdot \leftarrow X \to \cdot$, with constructors north : ΣX, south : ΣX, and merid : $X \to$ north $=$ south.



(d) The cofiber type Cof(f) of a function $f : X \to Y$ is the pushout of the span $\cdot \leftarrow X \xrightarrow{f} Y$, with constructors cfbase : Cof(f), cfcod : $Y \to$ Cof(f), and cfglue : $\Pi x{:}X.$ cfbase $=$ cfcod $(f y)$. If f is an "inclusion", the effect is to contract the image of X in Y to a point.

The basepoint of the pushout $X \sqcup_Z Y$ of a span of pointed functions is defined to be left x_0 in the Agda library, slightly arbitrarily; one could instead choose left $(f z_0)$, right $(g z_0)$, or right y_0, all of which are propositionally equal.

B Squares and Cubes

(This presentation is adopted from Dan Licata's cubical library at `https://github.com/dlicata335/hott-agda/tree/master/lib/cubical`.)

A square is the two-dimsensional analogue of a path, defined by the inductive type

$$\begin{aligned}&\mathsf{data\ Square}\ \{A : \mathsf{Type}\}\ \{a_{00} : A\} : \{a_{01}\ a_{10}\ a_{11} : A\}\\ &\quad \to a_{00} = a_{01} \to a_{00} = a_{10} \to a_{01} = a_{11} \to a_{01} = a_{11} \to \mathsf{Type\ where}\\ &\quad\ \mathsf{ids} : \mathsf{Square\ idp\ idp\ idp\ idp}\end{aligned}$$

Exhibiting an element of type $\mathsf{Square}\ p_{0-}\ p_{-0}\ p_{-1}\ p_{1-}$ is equivalent to proving that $p_{0-} \cdot p_{-1} = p_{-0} \cdot p_{1-}$, i.e. that the following square commutes:

$$\begin{tikzcd} a_{00} \arrow[r, "p_{-0}"] \arrow[d, "p_{0-}"'] & a_{10} \arrow[d, "p_{1-}"] \\ a_{01} \arrow[r, "p_{-1}"'] & a_{11} \end{tikzcd}$$

One useful property expressible as a square is the naturality of homotopies: for functions $f, g : A \to B$, a homotopy $p : \Pi x{:}A.fx = gx$, and a path $q : a_1 = a_2$, we have a square

$$\begin{tikzcd} fa_1 \arrow[r, "\mathsf{ap}\ f\ q"] \arrow[d, "pa_1"'] \arrow[dr, phantom, "\mathsf{natural\text{-}square}\ p\ q"] & fa_2 \arrow[d, "pa_2"] \\ ga_1 \arrow[r, "\mathsf{ap}\ g\ q"'] & ga_2 \end{tikzcd}$$

In general, dependent paths in a fibration of the form $x.fx = gx$ are expressible as squares: the type $u =_p^{x.fx=gx} v$ is equivalent to the square type

$$\begin{tikzcd} fa_1 \arrow[r, "\mathsf{ap}\ f\ p"] \arrow[d, "u"'] & fa_2 \arrow[d, "v"] \\ ga_1 \arrow[r, "\mathsf{ap}\ g\ p"'] & ga_2 \end{tikzcd}$$

with $\mathsf{natural\text{-}square}\ p\ q$ corresponding to $\mathsf{apd}\ p\ q : pa_1 =_q^{x.fx=gx} pa_2$.

The three-dimensional analogue is, unsurprisingly, the cube, which is defined inductively as

$$\begin{aligned}&\mathsf{data\ Cube}\ \{A : \mathsf{Type}\}\ \{a_{000} : A\} : \{a_{010}\ a_{100}\ a_{110}\ a_{001}\ a_{011}\ a_{101}\ a_{111} : A\}\\ &\quad \{p_{0-0} : a_{000} = a_{010}\}\ \{p_{-00} : a_{000} = a_{100}\}\ \{p_{-10} : a_{010} = a_{110}\}\ \{p_{1-0} : a_{100} = a_{110}\}\\ &\quad \{p_{0-1} : a_{001} = a_{011}\}\ \{p_{-01} : a_{001} = a_{101}\}\ \{p_{-11} : a_{011} = a_{111}\}\ \{p_{1-1} : a_{101} = a_{111}\}\\ &\quad \{p_{00-} : a_{000} = a_{001}\}\ \{p_{01-} : a_{010} = a_{011}\}\ \{p_{10-} : a_{100} = a_{101}\}\ \{p_{11-} : a_{110} = a_{111}\}\\ &\quad (sq_{--0} : \mathsf{Square}\ p_{0-0}\ p_{-00}\ p_{-10}\ p_{1-0})\ (sq_{--1} : \mathsf{Square}\ p_{0-1}\ p_{-01}\ p_{-11}\ p_{1-1})\end{aligned}$$

$(sq_{0--} : \mathsf{Square}\ p_{0-0}\ p_{00-}\ p_{01-}\ p_{0-1})\ (sq_{-0-} : \mathsf{Square}\ p_{-00}\ p_{00-}\ p_{10-}\ p_{-01})$
$(sq_{-1-} : \mathsf{Square}\ p_{-10}\ p_{01-}\ p_{11-}\ p_{-11})\ (sq_{1--} : \mathsf{Square}\ p_{1-0}\ p_{10-}\ p_{11-}\ p_{1-1})$
$\to$ Type where
  idc : Cube ids ids ids ids ids ids

To clarify, we visualize sq_{--0} as the left face, sq_{--1} as the right, sq_{0--} as the back, sq_{-0-} as the top, sq_{-1-} as the bottom, and sq_{1--} as the front.

Just as a dependent path in a fibration $x.fx = gx$ can be expressed as a square, a dependent path in a fibration $x.\mathsf{Square}\ (p_{0-}x)\ (p_{-0}x)\ (p_{-1}x)\ (p_{1-}x)$ can be expressed as a cube. For a path $q : a_1 = a_2$ and squares $u : \mathsf{Square}\ (p_{0-}a_1)\ (p_{-0}a_1)\ (p_{-1}a_1)\ (p_{1-}a_1)$ and $v : \mathsf{Square}\ (p_{0-}a_2)\ (p_{-0}a_2)\ (p_{-1}a_2)\ (p_{1-}a_2)$, the type $u =_q^{x.\mathsf{Square}\ (p_{0-}x)\ (p_{-0}x)\ (p_{-1}x)\ (p_{1-}x)} v$ is equivalent to the cube type

$$\mathsf{Cube}\ u\ v\ (\mathsf{natural\text{-}square}\ p_{0-}\ q)\ (\mathsf{natural\text{-}square}\ p_{-0}\ q)\ (\mathsf{natural\text{-}square}\ p_{-1}\ q)\ (\mathsf{natural\text{-}square}\ p_{1-}\ q)$$

The most useful property of squares and cubes for our purposes is the existence of *fillers*. Given three consecutive edges (a frame for a square missing one edge), there exists a (propositionally) unique fourth edge such that the four form a square. Likewise, given five faces of a cube, there is a unique sixth face which completes the cube.